「マルチメディア，分散，協調とモバイル
(DICOMO2017) シンポジウム」 平成29年6月

車載ECU向けソフトウェア更新のためのデータ圧縮方式

小沼 寛[1]　中村 清花[2]　寺島 美昭[3]　清原 良三[2]

概要：車載システムの高度化，複雑化によりプログラミングエラーに起因するリコールが増加している．また，自動車がインターネットに接続することにより，自動車にもスマートフォンのようにアプリケーションの追加や機能の更新が行えるようになる．一方で，自動車がサイバー攻撃の対象となる恐れがあり，セキュリティリスクへの対策が求められる．これらの理由から，出荷後に自動車に搭載されているElectronic Control Unit(ECU) ソフトウェアの更新が必要となる．ECU ソフトウェアの更新では，更新時間の短縮が重要な課題となる．更新時間を短縮する手法として，携帯電話やカーナビゲーションシステムでは差分更新が用いられており，ECU ソフトウェア更新への適用も行われている．しかし，一部の ECU では RAM のサイズやフラッシュメモリの特徴から差分更新を適用できない場合がある．差分更新を適用できず更新データを送信する場合，差分更新適用時に比べ大幅に更新時間が増加する恐れがある．そこで本論文では，差分更新を適用できない ECU を対象としたソフトウェア更新に対し，対象 ECU 内で伸長可能な圧縮を更新データに施すことにより更新時間増加の軽減を行う．

A Compression Method to Update ECU Software

YUTAKA ONUMA[1]　SUMIKA NAKAMURA[2]　YOSHIAKI TERASHIMA[3]　RYOZO KIYOHARA[2]

1. はじめに

自動運転社会の実現に向け，自動車産業あるいは IT 産業の多くの企業が自動運転技術の開発に取り組んでいる．自動運転機能や Advanced Driver Assistance Systems(ADAS) の搭載により，自動車に搭載されている Electronic Control Unit(ECU) 数や ECU ソフトウェア規模が増大している．それに伴い，プログラミングエラーに起因するリコールが増加している [1]．不具合によっては事故に繋がるおそれがあるため，早急なソフトウェアの更新が重要となる．

コネクテッドカーの登場により，自動車にもスマートフォンのようなアプリケーションダウンロードサービスやトラッキングサービスなどが利用可能となり，ユーザの満足度向上が期待される．一方，自動車がインターネットに接続することでサイバー攻撃の対象となる恐れがあり，セキュリティリスクが高まっている．リスクへの対策として，自動車にも定期的なセキュリティアップデートが必要となる．実例として，FCA 社は，インターネット接続機能を搭載している車両に脆弱性が発見され，大規模なソフトウェアアップデートによるリコールを行っている [2]．

今後，車載システムの高度化やソフトウェア規模の増大化が進むにつれ，ソフトウェア更新の頻度が高まっていくことが予想される．

従来，車載 ECU ソフトウェアの更新はユーザがディーラに車両を持ち込むことで行われている．ディーラでは，エンジニアが診断ツールを用いて対象となる ECU の更新を行う．従来の方法では，ユーザはソフトウェア更新のためにディーラまで赴かなければならず，ユーザビリティが損なわれる．また，脆弱性の発見などにより緊急性の高い更新がリリースされた場合，更新を行うためにディーラに向かう間は危険な状態となる．ディーラの駐車可能量や診断ツールの数に制限があるため，リコール発生時など同時に複数の車両の対応に迫られた際，ユーザを長く待たせることになる．

これらの問題を解決するために，Over-The-Air(OTA) による車載 ECU ソフトウェア更新が開発されている．OTA

1 神奈川工科大学大学院
Graduate School of Kanagawa Institute of Technology
2 神奈川工科大学
Kanagawa Institute of Technology
3 創価大学
Soka University

による更新では，無線通信を受信可能な場所であればどこでも更新が可能となるため，ユーザのソフトウェア更新に対する負担を軽減することができる．また，緊急性の高い更新がリリースされた場合においても即座にに対応を行うことができる．特定の機器や場所を必要としないため，ディーラの駐車可能量や診断ツール数といった制限も受けずに済む．

一方で，ECU ソフトウェア更新には以下の要求が挙げられる．

- ソフトウェア更新時間の短縮
- ソフトウェア更新の完全性保証
- ソフトウェア更新のセキュリティ

第一に，更新時間の短縮化がある．更新時間を短縮する方法として，差分更新が用いられている．しかし，ECU の一部には RAM とフラッシュメモリの関係から差分更新を適用できない場合がある．第二に，更新の完全性保証がある．自動車という環境上，ソフトウェアの更新に問題が発生した場合，事故に繋がる恐れがある．そのため，更新を正しく完了したことを保証する必要がある．第三に，更新のセキュリティがある．ソフトウェア更新において，更新データの改ざんや，不正な更新データの受信，更新データの盗聴などのセキュリティリスクが考えられる．ECU ソフトウェアの更新を安全，安心に行うために，これらのリスクへの対策が必要である．

本論文では，RAM に制約があり差分更新を行えない ECU に対し，更新データを圧縮し更新時間の軽減を行うため，ECU ソフトウェアの更新データに最適な圧縮アルゴリズムの比較検討を行う．

2. 車載 ECU ソフトウェア更新システム

2.1 システム概要

車載 ECU ソフトウェア更新システムの概要を図 1 に示す．また，ECU ソフトウェアの更新の流れを図 2 に示す．OEM または自動車メーカによって開発された新しいソフトウェアは，ソフトウェア管理サーバに登録及びディーラに送られる．新しいソフトウェアの情報はユーザに通知され，通知を受け取ったユーザはディーラに自動車を持ち込む，または OTA により ECU ソフトウェアの更新を受けることができる．受信した更新データは，車載ネットワークを介して更新対象となる ECU に送信され，ROM の書き換えが行われる．

2.2 車載ネットワーク

ECU ソフトウェアの更新データは，車載ネットワークを介して各 ECU に送信される．車載ネットワークの業界標準として Controller Area Network(CAN) が使用されている．しかし，車載システムの高度化による通信量の増加により CAN の帯域が不足しつつある．

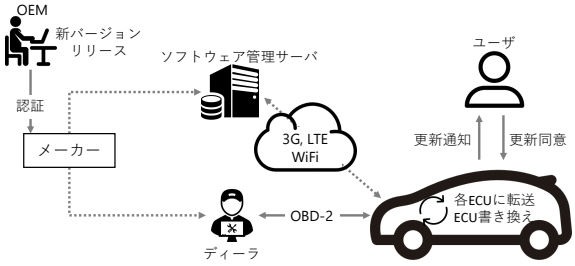

図 1　車載 ECU ソフトウェア更新システムの概要

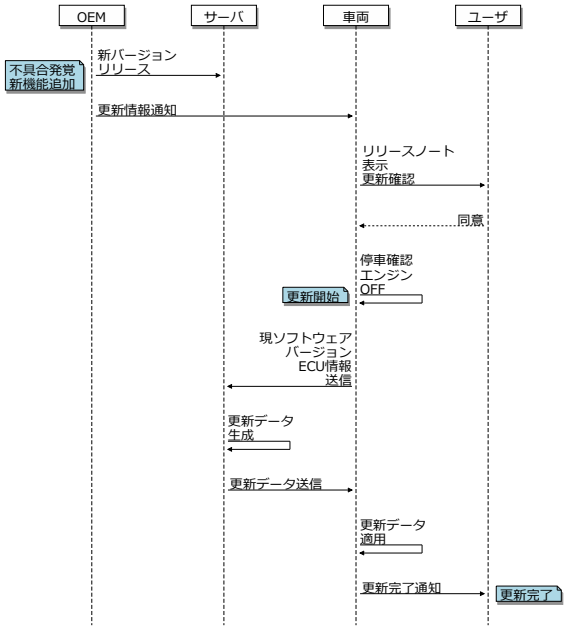

図 2　車載 ECU ソフトウェア更新の流れ

帯域不足解消のため，車載ネットワークに Ehternet の導入が進められている．車載ネットワークに Ethernet が導入されることにより，CAN よりも高速で柔軟なネットワークが実現可能となる．一方，自動車に搭載されている ECU は新旧様々なものが混在し，Ethernet に対応していない ECU も使用される場合がある．そのため，Etherent への置き換えは段階的に行われると考えられる．

本論文では，過渡期の車載ネットワークを対象とする．図 3 に想定される車載ネットワークを示す．想定される車載ネットワークでは，バックボーンや通信量の多い部分に関して Ethernet を使用し，一部のネットワークは従来の CAN を用いる．

2.3 課題

2.3.1 更新時間

走行中に ECU ソフトウェアを更新することは危険であ

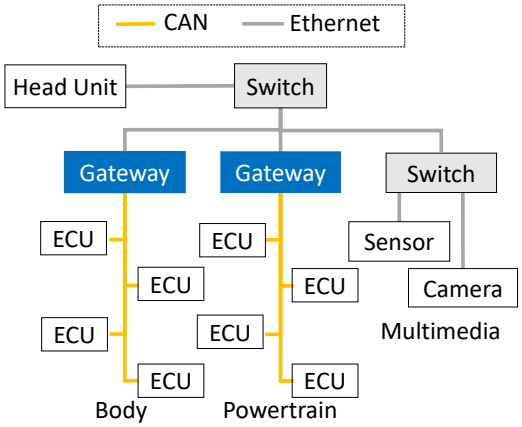

図 3　想定される車載ネットワーク

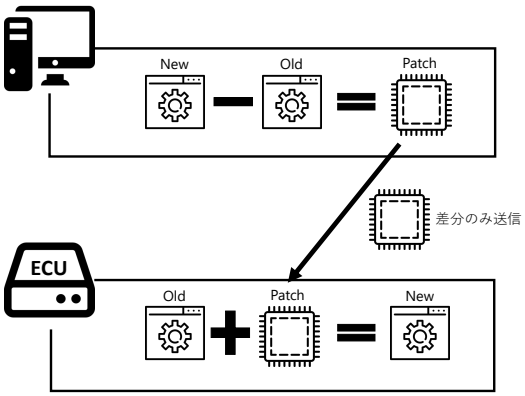

図 4　差分符号化によるソフトウェア更新

るため，更新は駐車中に行われることが前提となる．ユーザは ECU ソフトウェア更新中に自動車を使用することができないため，ソフトウェア更新時間を最小限にすることが要求される．また，ガソリン車においてはエンジン停止状態で更新を行う場合，ECU ソフトウェア更新を行うのに十分な電源を確報する必要がある．電力供給が可能な電気自動車や，アイドリング中に更新を行う場合は，環境やユーザへの負担を考慮する必要があるため，同様に更新に要する時間は短い方が望ましい．

ECU ソフトウェア更新時間は，更新データの転送時間と，フラッシュメモリの書き換え時間に大きく分けられる．走行中などに予めサーバから更新データをダウンロードしておくことで，その間の更新データの転送時間をユーザが待たなければならない更新時間から無視することができる．そのため，実際の更新にかかる転送時間は，車載ネットワークを介した更新データの転送時間のみとなる．

Ethernet を用いている部分の更新データの転送は高速に行えるが，CAN のデータ転送速度はフラッシュメモリの書き換え速度に比べ遅いため，CAN 上の更新データの転送時間がボトルネックとなり，更新データの転送時間が ECU ソフトウェア更新時間の大部分を占める．ECU ソフトウェア更新時間は車載ネットワークを介して転送される更新データのサイズに大きく依存するため，更新データサイズを減らすことが重要となる．

ソフトウェア更新時に転送するデータサイズを減らす技術として，差分符号化がある．差分符号化によるソフトウェアの更新を図 4 に示す．差分符号化はソフトウェアの新旧バージョン間の差分を抽出し，更新対象先の旧バージョンに差分を適用することで新バージョンのソフトウェアにアップデートする．差分のみを扱うため，ソフトウェア全体を送信する場合や通常の圧縮に比べ，大幅なデータ量の削減が可能となる．

差分符号化技術は携帯電話やカーナビゲーションシステムなどの分野で実用化されている．しかし，車載 ECU では，コストを抑えるため本来の機能を実行するために必要な最低限のハードウェア資源のみしか持っていない場合が多い．そのため，更新データを複合するために十分な RAM を持っておらず，差分符号化を用いた更新を実行できない場合がある．

2.3.2　更新の保証

自動車という基盤上，ソフトウェアの不具合は人命に関わる事故に繋がる恐れがある．OTA によるソフトウェア更新を行う場合，パケットロスにより更新データを正しく受け取ることができない可能性がある．また，更新中に急に電源が遮断されることも考えられる．その様な場合，ソフトウェアの更新が正しく行われず不具合や故障の原因と成り得る．そのため，更新データの完全性や正しくソフトウェアの更新が行われたことを保証することが必要である．

2.3.3　セキュリティ

自動車セキュリティに関する研究が多くされている [3][4]．ECU ソフトウェアの更新において，改ざんされた更新データの受信や更新データの盗聴などのセキュリティリスクが挙げられる．リスクへの対策として，更新データの暗号化や認証といった方法が考えらえる．しかし，暗号化や認証を行う場合，暗号化のためのオーバヘッドや複合化によりソフトウェアの更新時間へ影響を及ぼす．そのため，セキュリティを確保しつつオーバーヘッドを減らすことが重要となる．

本論文では，差分更新を行えない ECU のソフトウェア更新における更新データの削減について着目する．

3.　関連研究

ソフトウェア更新時間を短縮するための有効な手法として差分符号化がある．文献 [5][6] では，コンシューマ機器を対象として差分情報を用いた更新方法について提案している．

車載機器向けとして，文献 [7] では，車載 ECU のソフトウェア更新に対して差分更新を用いることで，更新時間をソフトウェア全体を送信する場合に比べて 90%減少可能であることを示している．文献 [8] では，汎用的なバイナリ

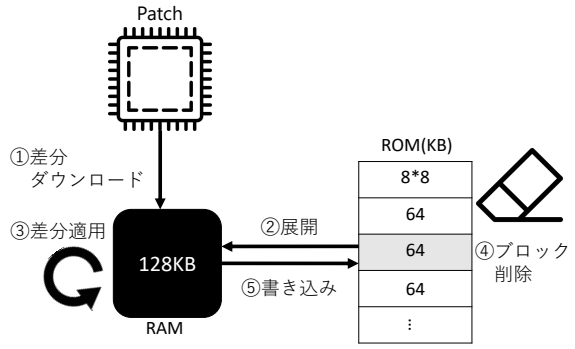

図 5 差分更新手順

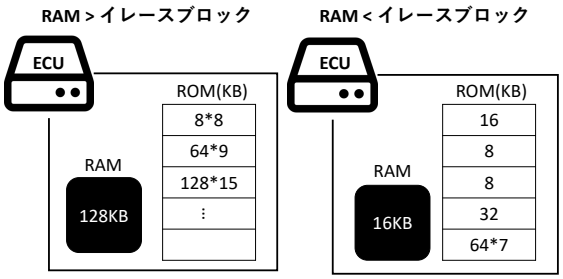

図 6 RAM とイレースブロックの関係

差分更新ツールの bsdiff[9] を車載機器向けに改良することで，圧縮率の向上を行っている．

文献 [10] では，bsdiff をインプレース型差分更新に改良することで，差分更新に必要なメモリを削減している．文献 [11] では，bsdiff をベースとし差分抽出方法とデータ構造，圧縮アルゴリズムを変更することで，圧縮率を維持しつつ省メモリ化を行っている．これらの手法は ECU のような小さい RAM で差分圧縮による更新を可能としている．しかし，NOR 型フラッシュメモリを採用している ECU の場合，RAM のサイズよりもフラッシュメモリの消去単位の方が大きく，RAM 上に差分更新の対象となるデータを展開することができない場合がある．

更新対象となる ECU のバスを分け，複数の ECU ソフトウェアの更新を並列に行うことで，全体の更新時間を短縮する手法が提案されている [12]．しかし，バスを分けるためのゲートウェイが追加で必要となり，コストを考えると更新のみのために導入することは難しいと考えられる．

4. 提案手法

4.1 想定環境

図 5 に，NOR 型フラッシュメモリの場合の差分更新手順を示す．ECU が NOR 型フラッシュメモリを採用している場合，フラッシュメモリを書き換える際にイレースブロック単位で消去を行う．そのため，差分更新を行う場合，少なくともイレースブロック分のデータを RAM 上に展開する必要がある．しかし，図 6 に示すように，一部の ECU では RAM のサイズがイレースブロックよりも小さく，差分更新を行えない場合がある．

差分更新が使用できない場合，ソフトウェア更新に必要なデータ転送量が増加し，ECU のソフトウェア更新時間が大幅に増加される．RAM のサイズがイレースブロックより小さい理由として，RAM 自体のサイズが小さい場合と，イレースブロックの単位が大きい場合が挙げられる．後者の場合，ある程度の RAM のサイズがあれば，圧縮されたデータの伸長が可能だと考えられる．そこで，データ転送量増加による更新時間増加を軽減するために，想定環境内で伸長が可能なデータ圧縮を行う．

表 1 圧縮アルゴリズムの比較

	bzip2	Deflate	LZMA2	BPE
圧縮率	中	低	高	低
メモリ使用量 (伸長)	高	低	中	低
伸長速度	低	高	中	高

4.2 圧縮アルゴリズムの選択

圧縮アルゴリズムは圧縮率向上やメモリ使用量抑制のため，多くのアルゴリズムが開発されている．本提案では，既存の圧縮アルゴリズムを想定環境向けに改良を行う．表 1 に圧縮アルゴリズムの比較を示す．

ECU ソフトウェアの更新で重要となるのはデータサイズと伸長時のメモリ使用量となるため，ここでは圧縮時間については考慮しない．比較対象として汎用的な差分更新ツールである bsdiff で使用されている bzip2[13]，高い圧縮率と速い伸長速度で知られている Deflate[14] と LZMA2[15]，他のアルゴリズム比べ高速な伸長を行える Byte Pair Encoding(BPE)[16] について比較を行う．以下にそれぞれの圧縮アルゴリズムの特徴について示す．

差分更新では，差扮符号化により抽出された差分を圧縮アルゴリズムにより圧縮している．カーナビや携帯電話で使用されている差分更新ツール bsdiff では，抽出された差分を bzip2 により圧縮されている．bzip2 はブロックソート法とハフマン符号化により圧縮を行う．bzip2 は，高い圧縮率を持つが，複合化に多くのメモリと時間を消費する．そのため，RAM の小さい ECU では bunzip2 による伸長が行えない場合がある．

Deflate は，LZ77 アルゴリズムとハフマン符号化により圧縮を行うアルゴリズムであり，ZIP や gzip[17] で使用されていることで知られている．bzip2 と比べ圧縮率は劣るが，伸長に使用するメモリ量は bunzip2 よりも少なく済む．

LZMA2 は，xz や 7zip[18] で使用されているアルゴリ

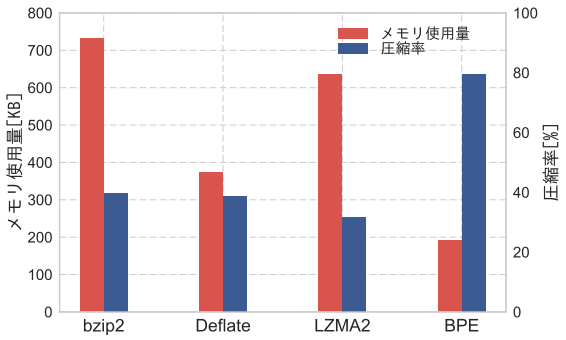

図 7　各圧縮アルゴリズムの圧縮率と伸長時のメモリ使用量

表 2　各圧縮アルゴリズムの圧縮後データサイズ [byte]

	圧縮前サイズ	圧縮後サイズ	圧縮率
bzip2	221,303	87,395	39.49%
Deflate		85,250	38.52%
LZMA2		69,632	31.46%
BPE		176,003	79.53%

ズムである．LZMA の拡張であり，LZ77 と Range Code アルゴリズムを基にしている．圧縮時に時間を要するが，bzip2 よりも高い圧縮と高速な伸長を行える．

BPE は，最も頻出に発生する 2 バイトを使用されていない 1 バイトに置き換えることによりデータを圧縮する．圧縮速度が遅い欠点があるが，伸張速度は高速である．

図 7 に，各圧縮アルゴリズムの圧縮率と伸長時のメモリ使用量のグラフを示す．また表 2 に各圧縮アルゴリズムの圧縮後のバイナリデータサイズを示す．圧縮対象にはリアルタイムカーネル TOPPERS/ASP[19] のバージョン 1.9.0 を ARM アーキテクチャ向けにビルドしたものを用いた．表 2 から，LZMA2 が最も圧縮できることがわかる．また，圧縮対象程度のサイズのバイナリデータであれば bzip2 と Deflate 間において圧縮率の差はほとんど見られない．図 7 から，BPE による圧縮が伸長時メモリ使用量において群を抜いて優位であることがわかる．しかし，BPE による圧縮の場合，他の圧縮アルゴリズムに比べ 1/2 程度の圧縮率となってしまっている．

以上の結果から，圧縮率と伸長時のメモリ使用量を考慮し，RAM に制約のある ECU においては Deflate による圧縮が適しているといえる．

4.3　メモリ使用量の制限

Deflate により圧縮されたファイルは Inflate により伸長される．伸長時のメモリ使用量は，圧縮されたデータサイズに依存する．そのため，図 8 圧縮するデータサイズを制限し，圧縮を分割して行うことにより伸長時の使用メモリ量を減らすことができる．一方で，圧縮を分割することによるオーバーヘッドが付加されるため，使用可能なメモリ量とのトレードオフを考慮する必要がある．

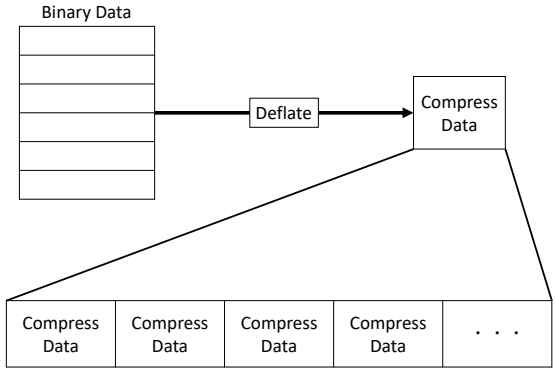

図 8　圧縮データサイズの制限

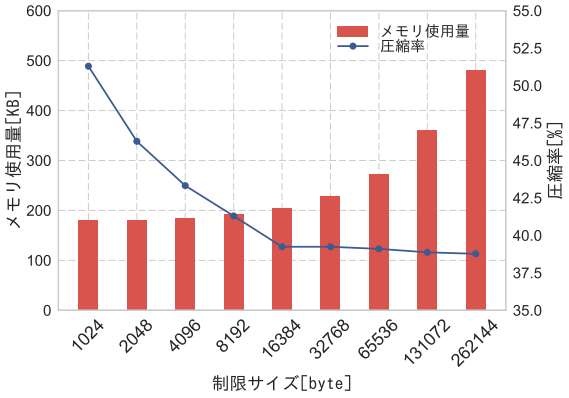

図 9　圧縮率と伸長時メモリ使用量 (TOPPERS/ASP 1.7.0)

表 3　圧縮後データサイズ (TOPPERS/ASP 1.7.0)[byte]

圧縮サイズ制限	圧縮前サイズ	圧縮後サイズ	圧縮率
1,024	221,178	113,438	51.29%
2,048		102,369	46.28%
4,096		95,825	43.32%
8,192		91,341	41.30%
16,384		86,780	39.24%
32,768		86,780	39.24%
65,536		86,472	39.10%
131,072		85,961	38.87%
262,144		85,742	38.77%

5.　評価

5.1　メモリ使用量と圧縮率

Deflate で圧縮するデータサイズを制限した場合の使用メモリ量と圧縮率について評価を行った．圧縮するデータサイズの制限を変更しつつ，対象バイナリに対して圧縮を行ったときの圧縮率と使用メモリ量を比較する．圧縮対象バイナリデータには TOPPERS/ASP のバージョン 1.7.0 と 1.9.0 を用いた．それぞれのバイナリデータに対する評価結果を図 9 と表 3，図 10 と表 4 に示す．

評価の結果，最もメモリ使用量が少ない場合，メモリ使

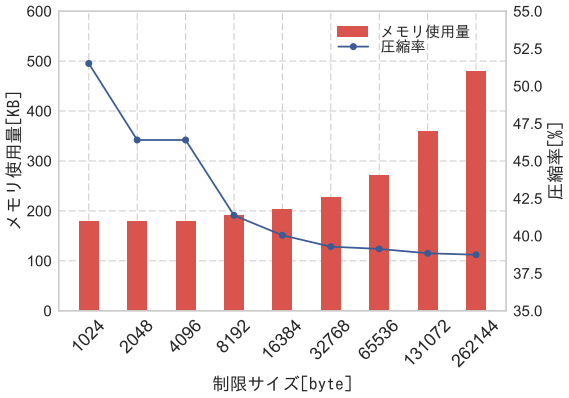

図 10 圧縮率と伸長時メモリ使用量 (TOPPERS/ASP 1.9.0)

表 4 圧縮後データサイズ (TOPPERS/ASP 1.9.0)[byte]

圧縮サイズ制限	圧縮前サイズ	圧縮後サイズ	圧縮率
1,024		113,994	51.51%
2,048		102,692	46.40%
4,096		102,692	46.40%
8,192		91,558	41.37%
16,384	221,303	88,599	40.04%
32,768		86,927	39.28%
65,536		86,597	39.13%
131,072		85,962	38.84%
262,144		85,733	38.74%

用量が 180KB のときに約 46%の圧縮率となった．また，圧縮率が最大の場合，メモリ使用量が 480KB のときに約 39%の圧縮率となった．

5.2 転送時間

更新データに対して圧縮を行った場合の更新データ転送時間への影響について評価する．評価では，10 台の ECU を更新すると想定したとき，差分更新を行えない ECU に対して更新データの圧縮をした場合の更新データの転送時間と，圧縮を行わない場合の更新データの転送時間の比較を行う．ここで，更新データサイズを $update_{size}$，CAN のビットレートを $CAN_{bitrate}$ としたとき，更新データ転送時間 T は以下の式で表される．

$$T = \frac{update_{size}}{CAN_{bitrate}}$$

評価は，TOPPERS/ASP のバージョン 1.7.0 から 1.9.0 へ更新を行うことを想定し，CAN のビットレート $CAN_{bitrate}$ を 500Kbps とした場合について行った．更新データ転送時間を表 5 に，比較を図 11 に示す．圧縮を行った場合と行わなかった場合の差は，差分更新を行えない ECU 数に対して線形的に増加する．評価環境では，全ての ECU に対して差分更新を行える場合に比べ，圧縮のみの場合の転送時間は約 5 倍となっている．また，圧縮を行わない場合と比べた場合の転送時間は約 1/2 程度となっている．

表 5 更新データ転送時間 [s]

差分更新可能 ECU 数	差分+新版	差分+圧縮
10	435.96	435.96
9	834.97	597.75
8	1233.98	759.54
7	1632.99	921.32
6	2032.00	1083.11
5	2431.01	1244.90
4	2830.02	1406.69
3	3229.03	1568.48
2	3628.04	1730.26
1	4027.05	1892.05
0	4426.06	2053.84

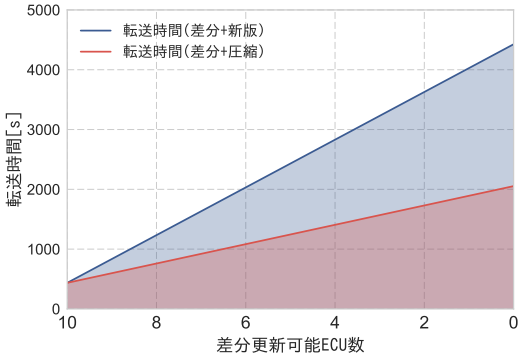

図 11 転送時間の比較

5.3 考察

差分更新を行えない場合に圧縮アルゴリズムを用いることで，180KB の RAM を持つ ECU であれば約 46%程度の更新データ転送時間の短縮が行えることを確認した．また，十分な RAM を持つ ECU であれば転送時間の短縮を向上することが可能である．但し，実際の環境においては CAN の全ての帯域を更新データ転送のために割り当てることはできないため，実時間は計算結果より増加すると考えられる．

評価環境の Deflate の実装では，圧縮を行うたびに辞書の生成を行っている．分割された各圧縮データがそれぞれの辞書を生成しているため，辞書間で重複が発生している場合がある．そこで，圧縮時の辞書を共有することで余分なデータを減らすことができると考えられる．

同一のソフトウェアであっても，異なる性能の ECU で使用される場合がある．差分更新の可否や，最適な圧縮サイズの制限を行うためには，対象となる ECU の RAM サイズやフラッシュメモリのイレースブロックを把握する必要がある．そのため，更新データの配信時に，ECU ソフトウェアに関連して対象となるハードウェア情報を管理し，対象となる ECU に合わせた更新データの配信を行うことが求められる．

しかし，図 12 に示すように，差分更新や圧縮を用いた

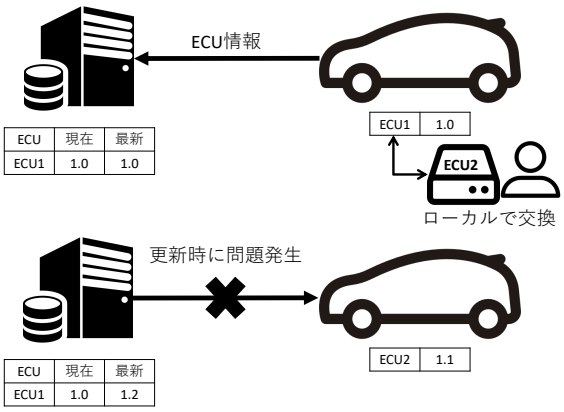

図 12 ECU のローカル変更時の問題

更新を行う際，リコール時やユーザの手によりローカルで ECU の交換，書き換えが行われていた場合，ソフトウェア更新時に問題が発生する可能性がある．そのため，ECU ソフトウェア更新時に配信元と自動車内の ECU 情報を共有する必要がある．

6. まとめ

本論文では，ECU ソフトウェア更新時に差分更新を適用できない ECU を対象とし，Deflate による圧縮を更新データに施すことで更新時間増加の軽減を行った．伸長時のメモリ使用量軽減のため，圧縮するデータサイズに制限を設けた．結果，180KB のメモリ使用量において約 50%の更新データの圧縮を確認した．

今回の手法では，最低でも 180KB の RAM が必要となる．しかし，多くの ECU にて圧縮を用いた更新を行うためには更なるメモリ使用量の削減が必要となる．

また，差分更新が行えない場合に，車載ネットワーク上を転送する更新データサイズを削減する代替手法として圧縮を行った．圧縮以外の更新データサイズを削減する方法として，ソースコードレベルで更新データを送信し，宛先でコンパイルを行う方法が考えられる．バイナリデータで送信する場合に比べ，送信するデータサイズを減らすことが可能であるが，メモリが制限されているためコンパイルに時間を要する可能性がある．そのため，バイナリデータで送信する場合との更新時間の比較が必要である．

今後の課題として，圧縮時の辞書を共有した場合との圧縮率の比較を行う．また，圧縮率を維持した伸長時のメモリ使用量の削減方法の検討や，他手法との更新時間の比較をする．さらに，更新時間の短縮以外の課題解決に向け，正確な更新の保証の方法や，配信元での ECU 情報管理方法の検討を行う．

参考文献

[1] 国土交通省　自動車局，平成 26 年度　リコール届出内容の分析結果について，http://www.mlit.go.jp/jidosha/carinf/rcl/common/data/h26recallbunseki.pdf （参照 2017-05-06）．

[2] FCA US Media website, http://media.fcanorthamerica.com/newsrelease.do?id=16849 （参照 2017-05-09）．

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham and S. Savage, "Experimental Security Analysis of a Modern Automobile," 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2010, pp. 447-462.

[4] C. Miller and C. Valasek, "Adventures in Automotive Networks and Control Units," DEF CON 21, 2013.

[5] R. Kiyohara, S. Mii, M. Matsumoto, M. Numao and S. Kurihara, "A new method of fast compression of program code for OTA updates in consumer devices," IEEE Transactions on Consumer Electronics, Vol. 55, Issue 2, pp. 812-817, 2009.

[6] R. Kiyohara, S. Mii, K. Tanaka, Y. Terashima and H. Kambe, "Study on Binary Code Synchronization in Consumer Devices," IEEE Transactions on Consumer Electronics, Vol. 56, Issue 1, pp. 254-260, 2010.

[7] D. Bogdan, R. Bogdan and M. Popa, "Delta flashing of an ECU in the automotive industry," 2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, 2016, pp. 503-508.

[8] Y. Onuma, M. Nozawa, Y. Terashima and R. Kiyohara, "Improved Software Updating for Automotive ECUs - Code Compression -," The 4th IEEE International Workshop on Consumer Devices and Systems conjunction with COMPSAC 2016.

[9] C. Percival, "Matching with Mismatches and Assorted Applications," doctoral thesis, Wadham College University of Oxford, http://www.daemonology.net/papers/thesis.pdf （参照 2017-05-06）．

[10] 施 欣漢，中西 恒夫，久住 憲嗣，福田 晃，"放送による車載機器向けソフトウェア差分更新方式，" 研究報告システム LSI 設計技術（SLDM），Vol. 2011-SLDM-149，No. 23，pp. 1-6 (2011)．

[11] 寺岡 秀敏，中原 章晴，黒澤 憲一，"車載 ECU 向け差分更新方式，" 研究報告コンシューマ・デバイス&システム（CDS），Vol. 2016-CDS-16，No. 5，pp. 1-8 (2016)．

[12] Y. S. Lee, J. H. Kim, S. J. Jang and J. W. Jeon, "Automotive ECU Software Reprogramming Method Based on Ethernet Backbone Network to Save Time," Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication, IMCOM 2016, Danang, Vietnam, January 4-6, 2016.

[13] bzip2, http://www.bzip.org/ （参照 2017-05-06）．

[14] Deflate, https://tools.ietf.org/html/rfc1951 （参照 2017-05-06）．

[15] XZ Utils, https://tukaani.org/xz/ （参照 2017-05-06）．

[16] P. Gage, "A new algorithm for data compression," C Users J., Vol. 12, No. 2, pp. 23-38, Feb. 1994.

[17] The gzip home page, http://www.gzip.org/ （参照 2017-05-10）．

[18] 7-Zip, http://7-zip.org/ （参照 2017-05-10）．

[19] TOPPERS プロジェクト/ASP カーネル，https://www.toppers.jp/asp-kernel.html （参照 2017-05-06）．