車載 ECU ソフトウェア更新向け圧縮アルゴリズムの比較評価

染谷一輝 [1]　杉本俊輔 [2]　寺島美昭 [3]　鈴木孝幸 [2]　清原良三 [2]

車載 ECU の OTA でのソフトウェア更新は，今後必須の機能となる．ソフトウェア更新の一連の処理プロセスでボトルネックとなるのは車載ネットワーク CAN 上の通信時間である．そのため，更新データを小さくする必要がある．我々は先行研究で LZ77 系の既存の圧縮アルゴリズムに辞書再利用をすれば圧縮効果があることを示した．この方式は他の圧縮アルゴリズムでも有効ではないかと考えられるため，他の圧縮アルゴリズムへの適用を想定して比較評価の解析をした．結果として，RAM 使用量を考慮すると BPE が一番良く，時点で gzip であった．一方，さらに圧縮率を考慮すると，Zstandard の学習辞書圧縮が車載 ECU に向いていることがわかった．

1. はじめに

様々な機器がネットワークに接続されるようになり，ソフトウェアで制御されるようになりつつある．例えば，セットトップボックス（STB），スマートフォン，PC などである．これらのソフトウェアは高機能になればなるほど不具合なしでの出荷は難しく，ネットワーク経由でのソフトウェアの更新が必須になっている[1].

一方，人の命がかかるようなエレベータや自動車などは，ネットワーク経由ではなく，更新後の確認なども含めて個別に機器を接続して更新することが多い[1]．しかし，自動車では自動運転の普及とともに，セキュリティ的な危険性，およびコストと利便性を考えるとネットワーク経由でのソフトウェア更新（Over-the-Air 更新，OTA 更新）が今後は必須になる．

また，衛星などでは，ソフトウェアの伝送可能な時間が限られ，地球の裏側などでサービスのできない時間に書き換えをするような制約の厳しいケースもある．

また，書き換え対象も様々である．起動時間を最優先とするため，ランダムアクセス可能な NOR 型フラッシュメモリを採用している場合はソフトウェアの更新時には一旦サービスを停止する必要がある．また，コスト最優先で NAND 型フラッシュメモリやハードディスクを利用している場合は，ファイルシステム上のファイルを書き換えるので，再起動時に構成をするように，ダウンロード時間などを気にする必要がない場合もある．

このようにソフトウェアの更新はその機器に特有の制約条件を満たしながら適切な方式を採用する必要がある．

自動車の ECU に関しても自動運転自動車の搭乗とともに，ネットワークに接続され，個数が増え，ソフトウェアの規模が増大し，複雑化している．ソフトウェアの不具合によるリコールの数も増えている[2]．また，セキュリティ的な脆弱性なども報告されており[3][4]，OTA でのソフトウェア更新の必要性は高まっている．

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

OTA でのソフトウェア更新システムのイメージを**図 1** に示す．車載 ECU は通常部品メーカ(サプライヤ)が開発していることが多く，サプライヤにてソフトウェアの新しい版を作成し，メーカにて試験など確認をして認証する．新版のソフトウェアは, サプライヤのサーバ経由で OTA を通して車両に伝送する．もしくは従来のようにディーラで OBD2 ポートから車両に伝送する．車両内は車載ネットワークを通じて各 ECU に伝送し，ソフトウェアを書き換える．車載ネットワークは一般的には CAN(Controller Area Network)[5]が利用されている. 帯域が 500Kbps 程度であり，更新時間のほとんどを伝送時間で占めることがわかっている[6]．FlexRay[7]など高速な車載ネットワークが提案され 2009 年に標準化がされているが，コストの関係で普及していない．将来的には車載イーサネットが組み合わさるものの，末端の ECU は CAN で接続されることになる．即ち，車載ネットワーク上を伝送するデータサイズで更新時間が決まる．

車載機器は，エンジンをオンにしたら，瞬時に動作可能になることが求められる．例えば，米国では後退時の事故防止の観点からバックモニタの映像はギアをバック（R）に入れてから 2 秒以内に表示されることが必要である[8]．駐車場から出る際などのエンジンを入れてすぐに出たい場合

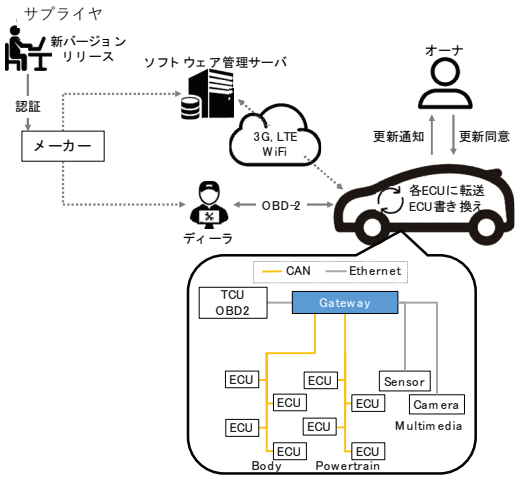

図 1　車載 ECU ソフトウェア更新

などを考慮するとできる限り早い起動が求められる．このようにECUによっては瞬時の起動を求められることから，NOR型フラッシュメモリを利用しているケースが多い．

逆に変数領域とスタック程度のRAMを搭載すれば良いので，RAMは少ないケースも多い．このような機器の構成は，OTAによる更新方法に大きく影響する．どのような場合に，どのような手法が適切なのかを整理しておく必要がある．

そこで，先行研究では，NOR型フラッシュメモリにおいてRAM容量が少ない場合に適用可能なソフトウェア更新向けのデータ圧縮方式として伸長用辞書の再利用方式の有効性を確認した[9]．本論文では，先行研究の方式を適切に様々なアルゴリズムに適用しても有効であることを示し，かつアルゴリズム選択するための指標を作ることを目的に，複数のアルゴリズムを比較評価する．

2. 関連研究

ソフトウェア更新はPCソフトウェアの更新などで身近なものになっている．一方で時間がかかるため困る時もあることがよく知られている．そこで更新時間を短くするための研究開発も多く実施されている．ソフトウェアの更新フローで時間がかかるのは以下の2点である．

- デバイスへの更新データの伝送
- デバイス上でのソフトウェアの書き換え

ここで，PCやスマートフォンでは回線速度も速く，デバイスの保存領域も十分あることが多いため，バックグラウンドで伝送する後者のソフトウェアの書き換え時間を短くすることを考えなければならない．一方，車載ECUなどでは個々のソフトウェアの規模は，PCやスマートフォンほど大きいわけではないため，書き換え時間は短く，遅いネットワーク上の伝送時間が問題となる．このように適用する対象の特徴や制約に応じた処理が必要になる．

前者のデータ伝送量の削減の研究は，著者らも長年研究しており，携帯電話のソフトウェアの書き換えで実用化もされている[10][11]．また，車載ECU向けのソフトウェア更新に関してはbsdiff[12]の活用が有効であることが示されている[6].

文献[11]はソフトウェアの差分が出にくいソフトウェアの構成法に関するものである．プログラム全体をモジュール化し，不具合などの更新でもプログラム上のアドレス参照部分の変化が少ないようにする手法で，ソフトウェアの書き換え量の削減も狙っている．大規模なソフトウェアでかつ，フラッシュメモリの書き換え時間がボトルネックとなるような場合に有効な手法である．しかし，車載ECUのソフトウェアは個々のECUを見ると大規模ではない．また，フラッシュメモリの書き換え時間がボトルネックになるわけでもない．

文献[1]は携帯電話に限らず組み込み機器を対象とした差分更新方式である．また[13]は差分そのものを小さく表現するための工夫であり，車載ECUのソフトウェア車載ECUの更新には有効な手法であるが，RAM使用量が多いため，RAMに余裕のない場合には適用できない問題がある．

適用時にRAMが不足する場合を想定したアルゴリズムが提案されている[14]．しかし，この方式においても消去ブロックサイズ以上のRAMは必要であり，消去ブロックのサイズに満たないRAMしかない場合には適用できない．

文献[15]は，車載ECUでRAMが小さい場合を想定しているが，汎用的な圧縮方式のRAM使用量や伸長時間を評価しており，どのような汎用圧縮方式が適切かを示しているにすぎない．

さらに，汎用的な可逆圧縮方式は様々提案されている[16]．これらはそれぞれ圧縮が高速なアルゴリズムや，伸長が高速なアルゴリズム，省メモリアルゴリズムなどがある．多くの圧縮アルゴリズムは，同じ記号列が連続していることに着目するランレングス法を基盤とし，同じ記号列が分散してあったとしても一致位置と一致長で表すLZ法を採用している．LZ法の代表的なアルゴリズムは，探索範囲を限定しスライディングウインドウで探索するLZ77系[17]と，辞書を用いて同じ記号列を探すLZ78系[18]に分かれるが，どちらも実装の違いということができる．また，同じ記号列がなかった部分の符号化により，さらに圧縮率を高める手法がLZSS[19]である．

さらに，このLZSSに，ハフマン符号化[20]を組み合わたのが，Deflate[21]であり，現在普及している圧縮ツールに最も利用されている．

この手法の他に，BPE[22]という手法がある．これは2bytes列を1byteデータに置きかえる処理を繰り返し圧縮する方法である．圧縮に時間がかかるものの，伸長速度が速く，RAM使用量が少ないことに特徴がある．

いずれの方法も辞書または辞書に準じたデータを伸長時に利用することになる．

本論文では，LZ系の代表的なアルゴリズムのDeflate，まったく違うアルゴリズムのBPE，差分圧縮で圧縮率が高いbsdiff，Deflateよりも伸長速度が速いZstandard[28]の通常圧縮，筆者らが着目しているZstandardの学習辞書圧縮に5つのアルゴリズムを複数の観点から比較評価することとした．

3. 車載ECUソフトウェア更新

3.1 ソフトウェア更新

システムソフトウェアの更新は，**図1**に示すようにまずサプライヤにて不具合や脆弱性に対する対策など，各ECU向けに修正をする．本論文の前提となるソフトウェア更新のシステムに関して以下に説明する．出荷元は，ECUごとに次に示す情報はわかっている．

- CPU の種類•RAM 容量
- NOR 型フラッシュメモリの消去ブロックサイズ
- フラッシュメモリの総容量

ここで，ある版から新しい版にソフトウェアを更新するために必要な更新データを生成する．部品メーカはこのデータの正しさなどを出荷元で適用試験を実施するとともに，カーメーカも適用試験をした上でリリースする．車両側は適切な更新プロトコルに基づいて，車両のソフトウェアのバージョンや ECU の種類といった情報を更新データ伝送サーバと交換することにより適切なデータを OTA でカーナビや TCU(Telematics Control Unit)と呼ばれるゲートウェイなどにダウンロードし，そこから車両が停止している時に，車載ネットワークを利用してデータを ECU に伝送し更新する．

データを生成する際には，差分更新に必要な RAM が確保できることが分かっていれば差分更新を，そうでなければ汎用のデータ圧縮方式などで RAM 容量などを考慮した上で､適切なアルゴリズムでのデータを圧縮して伝送する．このようなソフトウェア更新システムを前提とする．

3.2 ECU の更新

ECU 上で差分更新の場合は，**図 2** に示すフローで更新する．最初に RAM 上に更新のためのデータをダウンロードするとともに，次にフラッシュメモリの消去ブロック分のデータを RAM に読み込む．さらに，消去ブロックを消すとともに旧版のデータに差分を適用することにより RAM 上に新版のイメージを作成する．最後に新版のイメージを書き込む．このようなフローを実行する．差分データは消去ブロック単位で車載ネットワークを利用してダウンロードすれば良いが，それでも RAM 容量は消去ブロックのサイズの倍以上は最低でも必要となる．

そのため，十分な RAM 容量が搭載されていなければ差分適用は難しい．あるいは，フラッシュメモリに予備の消去ブロック 1 つ分を用意することができるのであれば，旧版データは RAM ではなく，予備の消去ブロックにコピーすることで RAM を節約することができる．しかし，フラッシュメモリの書き換え量は 2 倍になり，書き換え時間も無視できなくなるかもしれない．

一方，汎用的な圧縮を利用して更新する場合は，**図 3** に示すフローになる．まずデータは車載ネットワークを利用してダウンロードし，RAM 上に置き，圧縮を伸長し，フラッシュメモリを消去してから書き込む．RAM が小さい場合は，何度もダウンロード展開し，書き込む操作を繰り返す．フラッシュメモリでは消去操作は消去ブロック単位でまとめて消すが，書き込む操作は逐次できる．本論文では，この処理のための車載ネットワーク上のダウンロードするデータサイズの削減を目的としている．

4. 圧縮アルゴリズム

4.1 データ圧縮

関連研究で述べたように，データ圧縮は古くから研究され多くのアルゴリズムやツールが存在する．その基本となるのはハフマン符号化，および LZ77 系や LZ78 系である．原理としては，既に出てきたデータ列の並びが再度出てきたら短い符号に置き換える手法である．

先行研究[23]で比較評価をした結果，車載 ECU へのソフトウェアの配布においては，圧縮率，伸長時の RAM 使用量，伸長速度の観点から LZ77 系の圧縮が良いことが示されている．工場出荷時にかかる圧縮の時間は，ECU 上の処理と比べて高速な実行環境が準備でき，個々の ECU ソフトウェアの規模は大きくないことから無視してよい．差分適用に関しては RAM 使用量が多く圧縮率は高い．高速な実行環境でプログラムメモリの読み出し時間の方が伸長時間より気になるようなケースに利用する圧縮方式の BPE の RAM 使用量は小さいが，圧縮率は LZ77 系に比べ良くない．そこで，本論文では，車載 ECU に向いている伸長時の RAM 使用量が小さいアルゴリズムを再選定し，伸長時の RAM 使用量を比較評価する．

4.2 LZ77

LZ77 は記号列を一致位置，一致長，次の不一致記号という 3 種類の値に，順にデータをスキャンしては置き換える

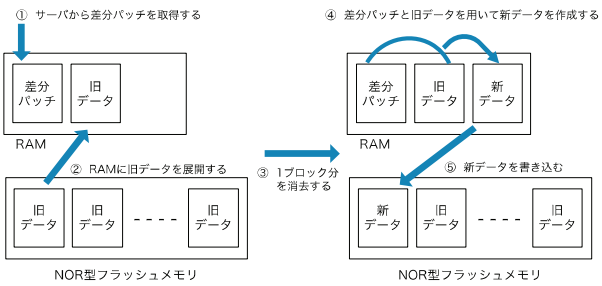

図 2 差分更新の場合の更新フロー

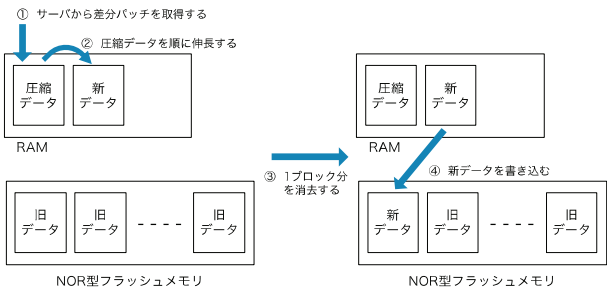

図 3 圧縮データによる更新の場合の更新フロー

ことにより圧縮する方式である．記号列を探す範囲をスライドウィンドウとして，置き換えパターンを辞書とする．処理の例を**図 4** に示す．赤色の文字列が入力であり，青色の文字列が出力である．(一致位置,一致長,次の不一致記号)と表している．圧縮する際には，データ全体を検索しやすいように工夫することが多く，これを圧縮用辞書と呼ぶ．一方，一致位置や一致長が指すデータを伸長用辞書と呼ぶ．圧縮時には，データをスキャンしながら圧縮用の辞書を引き，同じ記号列を見つけると置き換える操作を繰り返し，最終的には初めて出た記号列が辞書のデータとなり，2 回目以降に現れた同じ記号列が一致位置，一致長という形で表現される．なお，不一致記号は，同じデータが辞書中になかったデータなので，圧縮データ中にそのまま埋め込まれる．短い記号列で，一致長と一致位置で示す表現の方がより多くのデータサイズとなる場合も不一致記号として扱う．

4.3 ハフマン符号化

ハフマン符号化はエントロピー符号化のデータ圧縮方式である．出現頻度が高い文字に対しては短いビット列を割り当て，出現頻度が低い文字に対しては長いビット列を割り当てることでデータを圧縮する．「ADBCBABCBBCE」というデータをハフマン符号化する．このデータは A ~ E の 5 種類の文字があるため 2 進数で表現する場合，1 文字当たり最大 3bit となる．A を 000，E を 100 と順当に割り当てた場合「000 011 001 010 001 000 001 010 001 001 010 100」となり無圧縮の場合は 36bit となる．ハフマン符号化では出現頻度が高い文字を短いビット列に置き換えるため，B を 0，C を 10，A を 110，D を 1110，E を 1111 と置き換える．このようにすると「ADBCBABCBBCE」は「110 1110 0 10 0 110 0 10 0 0 10 1111」となりハフマン符号化の場合は 25bit となる．ハフマンツリーを**図 5** に示す．末端である子ノードが頻出の個数であり，親ノードは子ノードの総和である．最下段にある青いアルファベットへ辿り付くように

圧縮

1. AABCBBABCABCC

⋮

n. (0,0,A) (1,1,B) (0,0,C) (2,1,B) (5,3,A) (3,2,C)

伸長

1. (0,0,A) (1,1,B) (0,0,C) (2,1,B) (5,3,A) (3,2,C)
2. A (1,1,B) (0,0,C) (2,1,B) (5,3,A) (3,2,C)
3. AAB (0,0,C) (2,1,B) (5,3,A) (3,2,C)
4. AABC (2,1,B) (5,3,A) (3,2,C)
5. AABCBB (5,3,A) (3,2,C)
6. AABCBBABC (3,2,C)
7. AABCBBABCABCC

図 4　LZ77 の処理例

した赤い数値がハフマン符号化で置き換わるビット列となる．

4.4 Deflate

Deflate は辞書式のデータ圧縮方式である．LZ77 とハフマン符号化を用いて圧縮している．zlib[24]や gzip[25]などに使われているアルゴリズムである．LZ77 でデータを一致位置，一致長に置き換え，その後，ハフマン符号化でデータを符号化している． LZ77 のスライドウィンドウ毎にブロックを作成しており，ブロック毎にハフマン符号化をしている．また，誤り検出符号となるヘッダーとフッターがブロック毎に付けられる．

4.5 BPE

BPE(Byte-Pair-Encoding)は辞書式のデータ圧縮方式である．2bytes ずつデータを読み込み，出現頻度が高い 2bytes をデータ内で使われていない 1byte に置き換えることを繰り返すことでデータを圧縮する．置き換えは出現頻度が高い 2bytes がなくなる，または 1byte に置き換えるための 256 個分の未使用データがなくなるまで続ける．この性質上，圧縮する場合はデータを何度も繰り返し処理するため時間がかかる．一方，伸長の場合は置き換わった 1byte を 2bytes に置き換える処理だけであるため，短時間かつ少ない RAM で済む．処理の例を**図 6** に示す．上から下へ順に処理している．元のビット列「aabddaaababa」を 2 ビットずつ順に読み込み，頻出が高い文字から順に 1 ビットへと置き換えている．

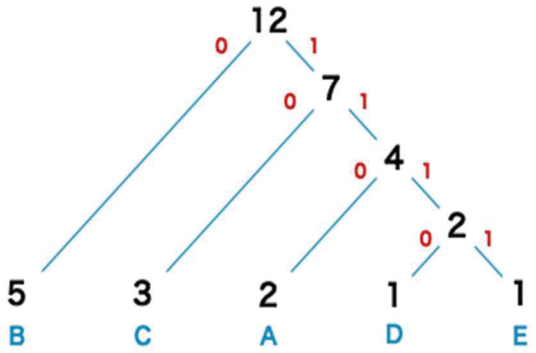

図 5　ハフマンツリー

ビット列	辞書
1. aabddaaababbd	**Z = aa**
2. ZbddZababbd	**Y = ab**
3. ZbddZYYbd	**X = bd**
4. ZXdZYYX	

図 6　BPE 圧縮の処理例

4.6 bsdiff

bsdiff はバイナリファイル用に差分更新をするソフトウェアある．旧データと新データの差分パッチを作成することで，その差分パッチを旧データに適用し，新データへと更新する．bsdiff で差分パッチを作成し，bspatch で差分パッチを旧データに適用して新データを作る．bsdiff を使用することで伝送する必要があるデータは差分パッチのみとなるため，変更点が少ない程，小さなデータ伝送で済む．bsdiff が作成する差分パッチは xdelta[26]と比較すると 50% ~ 80%小さく，RTPatch[27]と比べると 15%小さくなることが確認されている．

4.7 Zstandard

Zstandard は辞書式のデータ圧縮方式である．LZ77 とハフマン符号化を用いて圧縮している．Deflate との相違点はハフマン符号化の復号時にデータの先頭から読み取るのではなく，データの終端から読み取ることである．また Deflate よりも圧縮・伸長速度が速いことが特徴である．

Zstandard の圧縮方法は 2 つある．

1. 他の圧縮アルゴリズムと同様にデータの先頭から圧縮する方法である通常圧縮
2. 事前にデータを学習することで通常圧縮よりも高圧縮が可能な方法である学習辞書圧縮

通常圧縮と学習辞書圧縮を実行する手順を**図 7** に示す．図に示したよう学習辞書圧縮は 2 つのステップから成り立つ．ステップ 1 では圧縮したいデータを入力データとして Zstandard を用いて学習させ，そのデータに最適な圧縮が可能となる学習辞書を作成する．ステップ 2 では圧縮したいデータと学習辞書を Zstandard に入力することで圧縮データを作成する．また，伸長時には伸長したいデータと学習辞書を入力データとすることでオリジナルのデータへと伸長される．

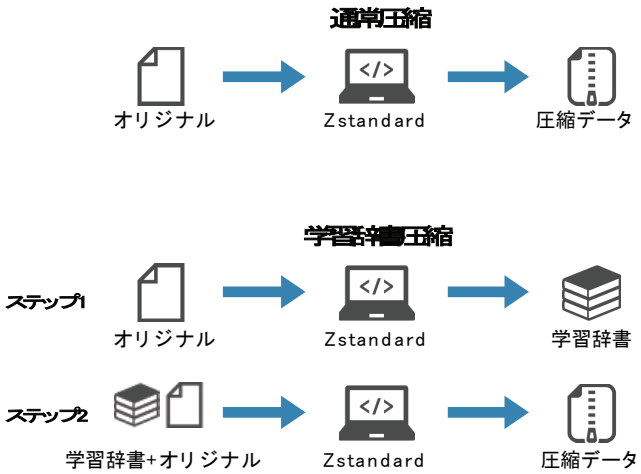

図 7　Zstandard の圧縮方法

5. 実験手法

車載 ECU ではデータを圧縮する必要はなく，圧縮されたデータを伸長する場合のみ圧縮アルゴリズムを用いる．そのため，伸長時の RAM 使用量を評価する．

RAM 使用量は変数宣言や機械語コードなどのプログラムから計算でき，プログラム実行時に確保する量が分かる静的な部分，およびプログラム実行時に malloc で動的に確保する量を決めるヒープ領域とローカル変数やメソッドを格納するスタック領域という動的な部分がある．そのため，静的な RAM 使用量は size コマンドで計測する．動的な RAM 使用量はアルゴリズムがどの程度 RAM を使用するか計算する．また，実際にアルゴリズムを実行する場合は Valgrind[29]を用いて動的な RAM 使用量を計測する．Valgrind は動的にメモリ管理をデバッグするソフトウェアである．malloc でどの程度の RAM を確保したか，RAM の解放忘れが起きていないか，RAM 解放済のポインタにアクセスしてメモリリークが起きていないかを調べることができる．

size コマンドでは text，data，bss，dec，hex の 5 つのデータが取得できる．text は機械コードと定数データが格納される空間である．data は初期値が入力されているグローバル変数が格納される空間である．bss は初期値が入力されていないグローバル変数が格納される空間である．dec は text，data，bss が合計されたデータであり，hex は dec を 16 進数で表したデータである．評価で必要となる静的な RAM 使用量を計測するためには data と bss を確認する必要がある．text はフラッシュメモリに格納されるためで対象外である．一方，動的な RAM 使用量はプログラムを実行するまでは分からない．アルゴリズムから分かる RAM 使用量を計算式で算出する評価とプログラム実行時に使用される RAM 使用量の評価の実験をする．

6. 評価

6.1 評価環境

アルゴリズムを比較するために使うプログラムは Deflate を用いて開発された gzip，BPE，bspatch，Zstandard の通常圧縮，Zstandard の学習辞書圧縮の 5 つを対象とする．BPE は文献[22]に付録としてある C 言語プログラムを用いた．bspatch は bsdiff の伸長用プログラムである．伸長用プログラムのデータサイズを**表 1** に示す．gzip は圧縮用プログラムと伸長用プログラムがあるためデータサイズが大きくなっている．また，圧縮レベルが設定できるアルゴリズムでは，RAM 使用量が少なくなる圧縮レベルで圧縮した．

本論文では，車載 ECU のソフトウェア更新を対象としているため，実際に ECU として使われているマイコンを想定する．想定する ECU 環境，および伸長プログラムや size コマンド，valgrind を実行した PC 環境を**表 2** に示す．ECU の想定環境より，伸長プログラムを実行するためにはピーク時の RAM 使用量が 32Kbytes 以下である必要がある．

6.2 静的 RAM 使用量

プログラムから RAM 使用量を計算する．size コマンドを実行した結果を**表 3** に示す．RAM に格納される空間は data と bss である．gzip は 333,776bytes，BPE は 656bytes，bsdiff は 760bytes，Zstandard(通常圧縮)は 912bytes，Zstandard(学習辞書圧縮)は 912bytes であった． gzip が大きく RAM を必要とする理由として，伸長用アルゴリズムだけでなく圧縮用アルゴリズムもプログラムに組み込まれているからだと考えられる．また，プログラムサイズも大きいため伸長時には使われないグローバル変数の存在が考えられる．

以上のことから伸長用プログラムは必要最低限な実装をするように作成すれば静的な RAM 使用量は無視できるサイズに抑えることができる．

6.3 動的 RAM 使用量

RAM 使用量が計算できる部分である一度にどの程度のデータを読み込むかのウィンドウサイズを計算する．

gzip の計算式を(1)に示す．*windowBits* は最大で 15，最小 8 である．*sizeof(int)*は 4bytes である．そのため，最大で 44,288bytes，最小で 11,776bytes の RAM 使用量となる．また，*1 << windowBits* が LZ77 で説明したスライドウィンドウのサイズとなる．

$$RAM = (1 \ll windowBits) + 1440 * 2 * sizeof(int) \quad \cdots \quad (1)$$

BPE の計算式を(2)に示す． 2bytes を 1byte に置き換えた文字(最大 256 文字)を元のデータに戻すため，置き換えられたデータと元のデータを保持する用に 2 つ分の領域が必要となる．また一度の置き換えだけではなく伸長したビット列を何度か伸長する場合がある．その置き換えを保持する領域で 30bytes 確保している．残りの 8bytes は添字などに使われており，shot 型のローカル変数を 4 つ宣言している．256*2 + 30 + 2*4 = 550bytes となる．計算式に変数がないことからデータサイズに関係なく伸長時の RAM 使用量は一定である．fgetc 関数を用いて 1 文字ずつ読み込んでいるため，上述の通り少ない RAM 使用量で動作するが，データサイズに比例して伸長時間が長くなる．

$$RAM = 550 \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (2)$$

bsdiff の計算式を(3)に示す．N は旧データサイズ，M は新データサイズである．ROM に存在する旧データを RAM に展開し，差分パッチを適用した新データを作成するためである．プログラムを見ると bsdiff の伸長機能ではローカル変数として 56bytes のみ RAM を確保して動作している．これらより，旧データと新データを RAM に展開することが RAM 使用量を占めていることが考えられる．

$$RAM = N + M \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (3)$$

表 1　伸長プログラムのサイズ比較

アルゴリズム名	データサイズ(bytes)
gzip	101,560
BPE	12,720
bsdiff	10,232
Zstandard(通常圧縮)	225,892
Zstandard(学習辞書圧縮)	225,924

表 2　車載 ECU 実行 PC のスペック

	車載 ECU	実行 PC
名称	Renesas RH850/F1L	Ubuntu(64bit) 18.04.4
CPU	RH850G3 (120MHz)	Intel Core i7-7567U(3.5GHz)
フラッシュメモリ	256Kbytes	20Gbytes
RAM	32Kbytes	4Gbytes

表 3　size コマンドの実行結果

	text(bytes)	data(bytes)	bss(bytes)	dec(bytes)	hex
gzip	93,342	3,688	330,088	427,118	6846e
BPE	5,001	648	8	5,761	1681
bsdiff	3,103	752	8	3,759	eaf
Zstandard(通常圧縮)	417,366	896	16	418,278	661e6
Zstandard(学習辞書圧縮)	418,286	896	16	419,198	6657e

表 4　アルゴリズム実行後のデータサイズ

	圧縮前	gzip	BPE	bsdiff	Zstandard (通常圧縮)	Zstandard (学習辞書圧縮)
v1.0 (bytes)	27,256	16,895	20,789		17,486	132
v1.1 (bytes)	27,664	17,075	21,054	5,389	17,711	6,030

表 5　Valgrind の実行結果

	malloc 回数	合計 RAM 使用量(bytes)	RAM 使用量/1malloc 当たり (bytes)
gzip	40	14,016	339
BPE	4	9,296	2,324
bsdiff	19	11,081,258	583,224
Zstandard (通常圧縮)	6	211,953	35,325
Zstandard (通常圧縮)	11	280,242	25,476

Zstandard の計算式を(4)に示す. 通常圧縮と学習辞書圧縮どちらも同じである. E は最大 5bit, M は最大 3bit となる. そのため，ウィンドウサイズの RAM 使用量は最小で 1Kbytes，最大で 3.75Tbytes となる.

$$RAM = (1 \ll (10 + E)) + ((1 \ll (10 + E)) \ / \ 8) * M \cdots (4)$$

6.4 動的 RAM 使用量の実測値

理論値としてはどのプログラムも小さな RAM 使用量で伸長することができる．一方，他の要因でヒープ領域やスタック領域が使われ，理論よりも RAM 使用量が多いことが考えられる．Valgrind を用いて伸長プログラムを実行した時にヒープ領域がどの程度，RAM を使用するか評価した．伸長するファイルは Toppers[30]より，マイコンの OS ファイルを用いた．また bsdiff は差分更新のため，v1.0 と v1.1 の 2 つを用いて結果を評価した.Zstandard の学習辞書圧縮は v1.0 で作成した学習辞書を用いて v1.1 のデータを圧縮した.

プログラムを実行して得た結果を**表 4** に，Valgrind の結果を**表 5** に示す. malloc で動的に確保した RAM は使用後，解放されるため合計 RAM 使用量がピーク時の使用量ではない．そのため，1 度の malloc で確保される RAM を比較評価する．本論文では，RAM 使用量の上限値を 32Kbytes としていた．bsdiff は上限値を大幅に上回っており，RAM に余裕がない車載 ECU には使用することが困難であることが確認できる． bsdiff の RAM 使用量は計算式(3)を見るとデータサイズ 2 つ分が必要となるため，伸長に必要な最低限のプログラムに改良しても実行できないことがわかる.

また，プログラムが一度に扱うデータサイズをウィンドウサイズから計算したが, それと比べてかなり多くの RAM 使用量となっている．原因として if 文などのプログラムが RAM を使用していることが確認できた．BPE はウィンドウサイズと伸長プログラムのデータサイズが小さいにも関わらず計算式とはかなり違う結果となった．valgrind では全体の動的 RAM 使用量と malloc の回数だけが取得できるため，プログラム実行中のピークを知るためには他の手法が必要となることがわかった.

7. まとめ

本論文では，車載 ECU 向けに伸長時の RAM 使用量を考慮したアルゴリズムの比較評価をした．結果として，静的 RAM と動的 RAM の使用量を考慮すると BPE が車載 ECU に向いている結果となった．一方，gzip のプログラム内容を見るとアルゴリズム自体は BPE と遜色ない RAM 使用量で伸長できる．そのため，伸長に必要な最低限のプログラム実装することで車載 ECU でも動作できる可能性があることを示した．また, 圧縮率を考慮に入れると Zstandard の学習辞書圧縮が適している結果となった. BPE と gzip に比べると RAM 使用量は多いが, ECU の RAM 容量は 32Kbytes と想定したので問題なく動作が可能である.

今後の課題として, RAM 使用量はピーク時にどれだけ使用するかが重要であるため，そのデータを取得できる提案をする必要がある．また，本論文の目的としては車載 ECU のソフトウェア更新にかかる時間を，データサイズを削減することで時間短縮することを目指している．RAM 使用量は bsdiff を除き，他は問題がないため，Zstandard の学習辞書圧縮に着目して車載 ECU に向けた評価をしていく.

参考文献

1) 清原良三，三井聡，木野重徳, “組込みソフトウェア向けバイナリー差分抽出方式, ”電子通信学会論文誌 Dm Vol.J90-D, No.6. pp.1375-1382 (2007)
2) Ministry of Land, Infrastructure, Transport and Tourism, ”Vehicle Recall,” http://www.mlit.go.jp/en/jidosha/vehicle_recall_17.html (accessed 16-March 2020)
3) S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank and T. Engel, ”A Car Hacking Experiment: When Connectivity Meets Vulnerability,” 2015 IEEE Globecom Workshops (GC Wkshps), San Diego, CA, pp.1-6, 2015
4) Amara Dinesh Kumar, Koti Naga Renu Chebrolu, Vinayakumar R, Soman KP, ”A Brief Survey on Autonomous Vehicle Possible Attacks, Exploits and Vulnerabilities,” arXiv:1810.04144, 2018
5) Bosch, R., ”CAN specification version 2.0, ” Rober Bousch GmbH, Postfach, 300240 (1991)

6) 寺岡秀敏，中原章晴，黒澤憲一，”車載 ECU 向け差分更 新方式,” 情報処理学会論文誌コンシューマ・デバイス＆ システム（CDS），Vol.7, No.2, pp.41-50, 2017
7) S. Lorenz, “The flexray electrical physical layer evolution,” SP ECIAL EDITION HANSER automotive FLEXRAY, pp.14-16, 2 010
8) NHTSA,”MVSS111,” https://www.nhtsa.gov/sites/nhtsa.dot.gov files/documents/fp-111-v01-final_tag.pdf
9) 染谷一輝，寺島美昭，清原良三.“圧縮用辞書の再利用による車載 ECU 向けデータ圧縮方式,”情報処理学会マルチメディア，分散，協調とモバイル(DICOMO)2020, pp.748-754
10) 星誠司，一瀬晃弘，野瀬康弘，細川篤司，武市真知，矢野英司,”無線通信を利用した「ソフトウェア更新」システム“, NTTDoCoMo テクニカルジャーナル，Vol.11, No.4, pp.36-41, 2004
11) 清原良三，栗原まり子，小宮章裕，高橋清，橘高大造,” 携帯電話ソフトウェアの更新方式，情報処理学会論文誌，Vol.46, No.6, pp.1492-1500, 2005
12) Colin Percival, “Matching with Mismatches and Assorted Applications,” Doctoral thesis, Wadham College University of Oxford, 2006
13) 清原良三，栗原まり子，三井聡，木野茂徳,”携帯電話ソ フトウェア更新のためのバージョン間差分表現方式,”電子情報通信学会論文誌 B, Vol. J89-B, No.4, pp.478-487, 2006
14) R. Burns, L. Stockmeyer and D. D. E. Long, ”In-place recons truction of version differences,” IEEE Transactions on Knowle dge and Data Engineering, vol. 15, no. 4, pp.973-984, 2003
15) T. Nakanishi, H. Shih, K. Hisazumi and A. Fukuda, ”A software update scheme by airwaves for automotive equip ment,” International Conference on Informatics, Electronics and Vision, pp.1-6, 2013
16) J. Uthayakumar, T. Vengattaraman, P. Dhavachelvan, ”A survey on data compression techniques: Fr9om the perspective of dat a quality coding schemes data type and applications”, J. King Saud Univ. Comput. Inf. Sci., 2018
17) J. Ziv and A. Lempel, ”A universal algorithm for sequential d ata compression,” IEEE Transactions on Information Theory, v ol. 23, no. 3, pp.337-343, 1977
18) J. Ziv and A. Lempel, ”Compression of individual sequences via variable-rate coding,” IEEE Transactions on Information Th eory, vol. 24, no. 5, pp.530-536, 1978
19) J. A. Storer, T. G. Szymanski, “Data Compression via Textual Substitution,” JACM, Vo.29, Issue 4, pp. 928-951, 1982
20) D. A. Huffman, ”A Method for the Construction of Minimum-Redundancy Codes,” Proceedings of the IRE, vol. 40, no. 9, p p.1098-1101, 1952
21) P. Deutsch, “DEFLATE Compressed Data Format Specification Version 1.3,” RFC 1951 (1996)
22) Gage P., “A New Algorithm for Data Compression,” The C U sers Journal, Vol.12, No.2, pp.23-38, 1994
23) Y. Onuma, M. Nozawa, Y. Terashima and R. Kiyohara, "Impro ved Software Updating for Automotive ECUs: Code Compressi on," IEEE 40th Annual Computer Software and Applications C onference (COMPSAC), pp. 319-324, 2016
24) zlib, https://tools.ietf.org/html/rfc1951https://tools.ietf.org/html/rfc 1950 (accessed 2020/7)
25) gzip, https://www.ietf.org/rfc/rfc1952.txt (accessed 2020/7)
26) xdelta, http://xdelta.org/ (accessed 2020/7)
27) RTPatch, https://www.pocketsoft.com/ (accessed 2020/7)
28) Zstandard, https://facebook.github.io/zstd/ (accessed 2020/09)
29) Valgrind, https://valgrind.org/ (accessed 2020/7)
30) Toppers, https://toppers.com (accessed 2020/5)