Vol.2021-ITS-84 No.2
2021/3/4

共有辞書を用いた車載 ECU 向けソフトウェア更新方式

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

概要: コネクテッドカーの登場により自動運転車両や車車間，路車間の通信を利用した高度な制御のできる車両が増加しつつある．高度な制御をするために，車載 ECU の数やソフトウェアの規模は増大している．ネットワークに接続可能であるがゆえのセキュリティ的な脅威も存在する．よって，車載 ECU のソフトウェアの更新機能は必須である．早急なバグ対応を考慮すると，携帯電話網などのネットワークを経由して更新することが望ましい．車外と車両を結ぶネットワークに比べ，車載ネットワークの主流である CAN は伝送速度が遅く，ソフトウェア更新の時間全体から見ると車載ネットワーク内の伝送時間がボトルネックとなる．そのため，車載 ECU のソフトウェア更新では差分更新が基本となる．しかし，差分更新を適用するには各車載 ECU に十分な RAM 容量が必要であり，RAM 容量に余裕がない場合は利用できない課題がある．本論文では，従来の汎用的なデータ圧縮方式を改良した．圧縮用に作成される辞書を再利用することにより，ソフトウェア更新が可能な方式を提案する．提案する Zstandard の共有辞書圧縮により，車載 ECU の小さな RAM 容量でも伸長プログラムを実行でき，差分更新で圧縮するデータサイズに近い圧縮が実現できた．これによりソフトウェア更新の時間短縮ができる．

キーワード: データ圧縮，ソフトウェア更新，ECU

Software update method for in-vehicle ECU using shared dictionary

Kazuki Someya[1] Shunsuke Sugimoto[3] Yoshiaki Terashima[2] Takayuki Suzuki[3] Ryozo Kiyohara[3]

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．ソフトウェアは高機能に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る. また，研究開発がさかんであるがゆえに，機能も日々向上していくことが想定される．さらに，車載機器の制御をする車載 ECU(Electronic Control Unit)に関しても自動運転車両の登場とともに，ネットワークに接続され，車載 ECU の個数が増え，ソフトウェアの規模が増大・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 [1]. また，セキュリティ的な脆弱性なども報告されており [2] [3]，OTA でのソフトウェア更新の必要性は高まっている.

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．車両内では車載ネットワークを通じて各車載 ECU に更新データを伝送し，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） [4] が利用されており，帯域が 500 Kbps 程度しかなく，車載ネットワーク上の伝送時間が更新時間のほとんどを占める [5]．FlexRay [6] など高速な車載ネットワークが提案されている．しかし，コストの関係でそれほど普及していない．将来的には車載 Ethernet の組み合わされるものの，末端の車載 ECU は CAN，または CAN をコストダウンした LIN(Local Interconnect Network)で接続されると想定できる．実際にトヨタが 2021 年春ごろに実用化する運転支援機能に車載 Ethernet を採用することが発表されている [7].

すなわち，車載ネットワーク上を送信するデータサイズで更新時間が決まる．OTA 更新では，車載ネットワークを利用するため，更新中は車両を利用できない． そのため，更新時間を短くするために伝送するデータサイズをいかに小さくするかが課題となる.

データ量を削減するためには，差分更新が適している [5]．新版と旧版の差分のみを伝送し，車載 ECU 上で旧版に差分データを適用して更新する技術である．一方，車載 ECU には様々な種類があり，コストの関係で RAM 容量が少ないものがある．文献 [5]でも示されているように差分更新ではフラッシュメモリのイレースブロックサイズ以上の RAM 容量が必要であり，差分更新が適用できない場合もある． そこで，本論文では，差分更新が適用できないような RAM 容量が少ない場合でも，データサイズを削減することを目的とする．筆者らは，受信側のデータを活用することにより，従来からあるデータ圧縮方式の改良を提案した [8]. 具合的には, データを圧縮するための圧縮用に作

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

成される辞書をあらかじめ出荷時にフラッシュメモリに記憶し，新版を作成する際にもこの旧版の辞書を再利用して圧縮する．

本論文では，圧縮用辞書を再利用する方式を用いて，車載 ECU 向けの運用を評価した．また，様々なアルゴリズムと比較して圧縮率と RAM 使用量の有用性を示す．

2. 差分更新とデータ圧縮のフロー

車載 ECU 上で差分更新をする場合は，**図 1** に示すフローで更新する．最初に差分データは車載ネットワークを利用してフラッシュメモリにダウンロードされる．差分データは RAM 上に展開され，旧版データを読み込んで新版データを作成する．次にフラッシュメモリのイレースブロック 1 つ分のデータを RAM に読み込む．さらに，イレースブロックを 1 つ消すとともに旧版データに差分データを適用することにより RAM 上に新版データの一部を作成する．これを新版データが復元できるまで繰り返す．差分データはイレース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．しかし，旧版データと新版データを保持しておくために，イレースブロック 2 つ分以上の RAM 容量が最低限必要となる．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イレースブロック 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 に展開するのではなく，予備のイレースブロックにコピーすることで RAM 容量を節約できる．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と想定される．

汎用的な圧縮アルゴリズムを利用して更新する場合は，**図 2** に示すフローになる．最初に圧縮データは車載ネットワークを利用してフラッシュメモリにダウンロードされる．圧縮データは逐次 RAM 上に展開され，データを伸長し，フラッシュメモリから旧版データを消してから書き込む．RAM 容量が小さい場合は，何度も RAM 上に展開を繰り返し，書き込む操作を繰り返す．NOR 型フラッシュメモリでは消去操作はイレースブロック単位でまとめて削除するが，書き込む操作は逐次できる．最後にダウンロードしてきた圧縮データを削除する．データ圧縮は差分更新と違い，RAM に展開するデータが少ないため，小さな RAM 容量で新版データに更新することができる．

本論文では，RAM 容量が小さい車載 ECU に対してソフトウェア更新を時間短縮することを目的としている．そのため，車載 ECU のスペックでは差分更新ができないことが多いため，データ圧縮に注目した．

3. 関連研究

ソフトウェア更新は PC やスマートフォンのソフトウェア更新などで身近なものになっている．一方，ソフトウェア更新には時間がかかるため，更新中であると使用したいときに使用できず，困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は多く実施されている．ソフトウェア更新の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，ネットワーク上の伝送時間が遅いことが問題となる．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である．

前者のデータ伝送量の削減の研究は，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 [9] [10]．また，車載 ECU 向けのソフトウェア更新に関しては bsdiff [11]の活用が有効であることが示されている [5]．

文献 [10]はデータ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，個々の車載 ECU のソフトウェアは大規模ではない．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．

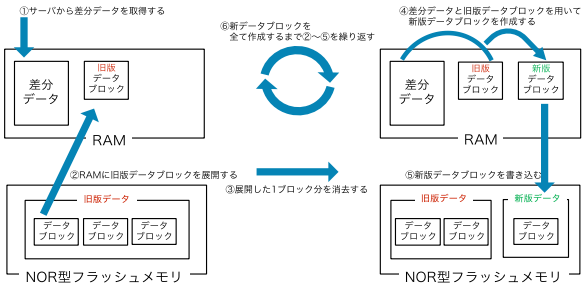

図 1 差分更新のデータフロー

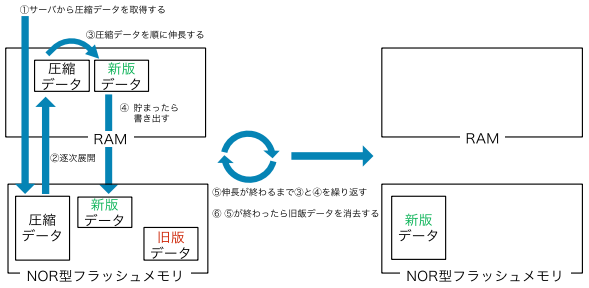

図 2 データ圧縮のデータフロー

文献 [12]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，文献 [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 のソフトウェア更新には有効な手法であるが，RAM 使用量が多いため RAM 容量が小さい場合には適用できない問題がある．

適用時に RAM 容量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] [15]．しかし，この方式においてもフラッシュメモリのイレースブロックサイズ以上の RAM 容量は必要であり，イレースブロックのサイズに満たない RAM 容量しかない場合には適用できない．

文献 [16]は, 車載 ECU で RAM 容量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．汎用的な圧縮方式は様々提案されている [17]．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．先行研究である文献 [16]で評価した結果，L77 系アルゴリズムが車載 ECU のソフトウェア更新には適していることが示されている．

ソフトウェアの更新はデータのダウンロード処理以外に，完全性・安全性の保証が必要である．チェックサムや CRC といった従来手法が多くあり，完全性・安全性の保証の研究がされている．

車載ネットワークの脆弱性をついた攻撃が報告されている．CAN はブロードキャストによる通信をしており，CAN バスに接続を行えば容易に通信メッセージの盗聴が可能である．また，暗号化や認証の機能を持っていないため，悪意のある送信者から不正なメッセージを受信してしまう恐れがある．

車載ソフトウェアの標準化を進めている AUTOSAR [18]では, CAN メッセージにパケットカウンタと MAC(Message Authentication Code)を付与することで，メッセージの完全性を保証する方式が策定されている [19]．MAC を付与することで, CAN メッセージの認証やリプレイ攻撃の阻止が可能となる．しかし， CAN メッセージのペイロードを圧迫してしまうため, データの伝送効率は低下する．MAC による負荷を軽減するために，軽量な MAC 認証手法が提案されている [20]．カウンタ値をデータとは別のメッセージで送信することで，ペイロードの負荷を軽減している．

各車載 ECU がバス上のメッセージを監視し，自身の ID を持つメッセージが不正に送信されていたときに偽装メッセージを破棄する方法が提案されている [21]. 車載 ECU に簡易な変更を加えることで実現可能な手法であり，手法によるトラフィックの増加もない．しかし，対象となる車載 ECU を取り外されてしまうと機能しないという問題がある．

セキュアエレメントをサポートした車載 EUC を用いることで，安全性のハードウェアレベルの信頼性を実現する手法が提案されている [22]. 車両制御に関する車載 ECU のセキュアブート，CAN のメッセージ単位の認証，更新データの署名と検証することで，エンジン始動から走行時，メンテナンス時の安全性を確保している. CAN メッセージの認証のために，分割した MAC を付加している．MAC サイズによりセキュリティ強度が変わるため，セキュリティ強度とペイロード負荷を加味して MAC サイズを決定することを勧めている．

CAN バスを監視しルールベースによる不正検知と無効化をする車載 ECU と，遠隔監視サーバによる機械学習を用いた異常検知による多層化した攻撃検知システムが提案されている [23]．教師なし学習による異常検知アルゴリズムを用いており，ソフトウェアアップデートにおいても更新データ伝送の正常モデルを生成することで適用可能であると考えられる．

また，CAN 上の通信を安全にするために CAN メッセージの周期性を用いた異常検知手法が提案されている [24]. CAN 上の通信が一定の周期的性質を持つことに着目し，メッセージの周期性と出現順序の特徴から正常状態を抽出し異常を検知している．

コンシューマ機器向けのソフトウェア更新を車載機器向けに採用することで，インフォテインメントシステムソフトウェアのセキュアなソフトウェア更新を提案している [25]. その提案では, インフォテインメントシステムの製造者により作成されていないソフトウェアでシステムを更新できないようにすることで，安全なソフトウェアアップデートを提供している．

車載 ECU の認証と車載ネットワークの暗号化を行うために，ハッシュアルゴリズムと共通鍵暗号方式を採用したシステムが提案されている [26]．シミュレーションにより，改ざんやなりすまし，リプレイ攻撃に対し低コストで対処することができることを示している．

このようにセキュリティに関しては多数の研究がある状況であるものの，いずれも更新時間を気にするものはなく，データの圧縮の重要性がより高いと分かる．また，書き換え後の完全まで含めたシステムに関しても，多数研究されている [27] [28].

本論文では，車載機器までのダウンロードにおいて保証されているものとする．

4. 提案手法

4.1 圧縮アルゴリズム

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 [16]で比較評価をした結果，車載 ECU へのソ

フトウェアの更新においては，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，車載 ECU 上の処理と比べて高速な実行環境が準備でき，個々の車載 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 に適している伸長時の RAM 使用量が小さいことと，伝送速度が遅い CAN でも短時間で伝送できるように圧縮率を考慮してアルゴリズムを再選定し，比較評価した．

4.1.1 LZ77

LZ77 [29]は記号列を一致位置，一致長，次の不一致記号の 3 種類の値に，順にデータを読み込んでは置き換えることで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 5** 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．例では 2 文字以上が一致する場合に圧縮するようにした．

圧縮時は，データ全体を検索しやすいように工夫することが多い．データを読み込みながら圧縮用に辞書を作り，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．

4.1.2 BPE

BPE(Byte-Pair-Encoding) [30]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．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 個の未使用データがなくなるまで続ける．この性質上，圧縮時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，伸長時は置き換わった 1byte を 2bytes に戻す処理だけであるため，短時間かつ，少ない RAM 使用量で済む．処理の例を**図 3** に示す．上から下へ順に処理している．元のビット列「aabddaaababa」を 2bytes ずつ順に読み込み，頻出が高い文字から順に 1byte へと置き換えている．

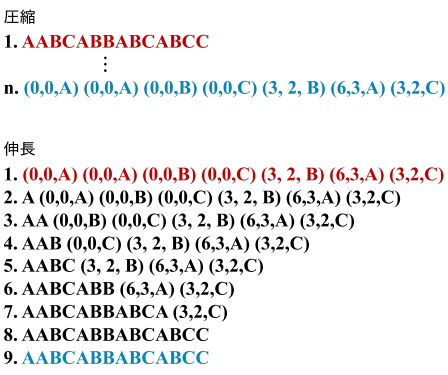

図 3 LZ77 の処理例

4.1.3 Zstandard

Zstandard [31]は辞書式のデータ圧縮方式である．LZ77 とハフマン符号化 [32]を用いて圧縮している．Deflate [33]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．

4.2 辞書再利用方式

圧縮アルゴリズムでは，圧縮時に辞書を作成し，それを元にオリジナルデータを圧縮している．一方，伸長時には既に最適化がされ，圧縮された状態からオリジナルデータに戻すだけである．

本論文では，辞書と圧縮データをそれぞれ別のファイルに取り出す辞書再利用方式を提案する．取り出した辞書を用いて圧縮することで，既存の圧縮アルゴリズムで課題となっているデータの先頭が圧縮されないことを回避できると考える．また，辞書のデータサイズ分，伝送するデータが削減できることからソフトウェア更新の時間短縮にな

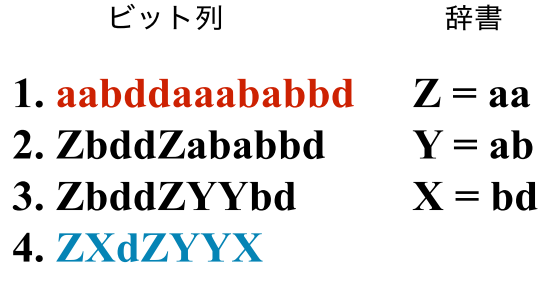

図 5 BPE の処理例

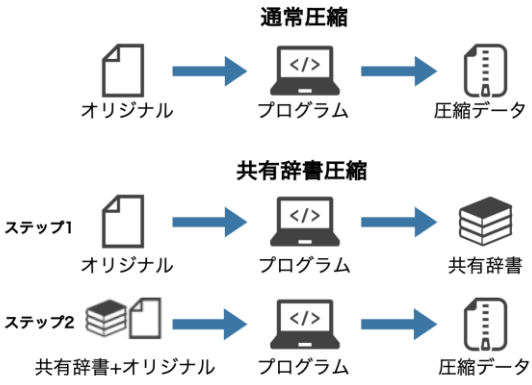

図 4 辞書再利用方式

ると想定できる．辞書を取り出すことで，圧縮時と伸長時の両方に使えること，バージョン間の変更が少なければ他のバージョンでもその辞書を用いて圧縮・伸長ができることから，取り出した辞書を本論文では，共有辞書と呼ぶ．また，共有辞書を用いて圧縮することを共有辞書圧縮と呼ぶ．**図 4** に共有辞書圧縮の流れを示す．通常の圧縮では，圧縮時にオリジナルデータをプログラムに渡すことで圧縮データを作成する．共有辞書圧縮では，最初に，共有辞書を作成する．その後，オリジナルデータと共有辞書をプログラムに渡すことで圧縮データを作成する．伸長時はオリジナルデータと共有辞書をプログラムに渡すことでオリジナルデータを作成する．

共有辞書の考えられるメリット 2 つとデメリット 2 つを下記に示す．

メリット

- データの先頭を圧縮できるため，通常圧縮よりも圧縮率が向上する
- 辞書が無い分，圧縮データのサイズが小さくなる

デメリット

- 圧縮・伸長する場所に共有辞書がないと操作ができない
- 共有辞書のサイズ分のフラッシュメモリが余分に必要となる

本論文では，既存のプログラムを改良，またはコマンドラインのオプションを使用することで共有辞書を作成した．共有辞書圧縮として評価するアルゴリズムは Zstandard と BPE である．Zstandard を選んだ理由は，広く普及している LZ77 をベースとしているためである．BPE を選んだ理由は，LZ 系とは異なるアルゴリズムで動いており，伸長時に RAM 使用量が少なく動作するためである．同じ系統ではなく，異なる動作をするアルゴリズムを比較することにより，辞書を取り出した時の効果がよりわかる．

提案手法である共有辞書圧縮が効果的であるか既存アルゴリズムと比較評価する．

4.2.1 共有辞書圧縮

共有辞書を用いて Zstandard と BPE を圧縮する手順を述べる．Zstandard の場合はコマンドラインより，train オプションを用いて辞書を取り出した．train オプションにより作成した共有辞書とオリジナルデータを入力することで圧縮する．伸長の場合も同様の手順で，共有辞書と圧縮データを入力することで伸長する．

BPE の場合はアルゴリズムの特性上，オリジナルデータのデータサイズによっては複数個の辞書が取り出せる．16 進数で 1byte 表記できる数値 0~255 の 256 個のデータが全て使用されるとそこまでを 1 つのブロックとして区切り，以降のデータを新たに圧縮できないか探索するためである．そのため，区切られたブロックに対応する共有辞書を用いて圧縮をする．伸長する場合も同様の手順で，ブロックごとに対応する共有辞書を用いて伸長する．

5. 評価

5.1 環境

アルゴリズムを比較するために使うプログラムは gzip，BPE の通常圧縮，BPE の共有辞書圧縮，bsdiff，Zstandard の通常圧縮，Zstandard の共有辞書圧縮の 6 つを対象とする．BPE の通常圧縮は文献 [30]に付録としてある C 言語プログラムを用いた．圧縮レベルが設定できるアルゴリズムでは，プログラム実行時の RAM 使用量が少なくなるように最小の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，圧縮・伸長するデータは Toppers [34]より，マイコン用 OS データを用いた．データサイズを**表 1** に示す．想定する車載 ECU の環境，および伸長プログラムや size コマンド，Valgrind を実行したコンピュータの環境を**表 2** に示す．車載 ECU の環境より，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

表 2 実験用 OS データのサイズ

	v1.0	v1.1	v1.2	v1.3
データ A (bytes)	28,742	28,742	28,746	28,746
データ B (bytes)	27,256	27,664	26,852	

表 1 車載 ECU と実行 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

要がある．

5.2 圧縮率

データ A は 4 つのバージョンがあり，データ B は 3 つのバージョンがある．**図 6** に圧縮率の結果を示す．bsdiff は v1.0 から v1.1 に更新に必要な差分データを使用した．Zstandard の共有辞書圧縮は v1.0 で共有辞書を作成し，v1.1 のデータを圧縮した．他のアルゴリズムでは v1.0 を圧縮した．BPE の共有辞書圧縮は伸長の問題より辞書再利用方式には適さないことがわかった．そのため，結果から除いている．図にある BPE は通常圧縮した結果である．

BPE に共有辞書圧縮が適さない理由は，アルゴリズムの特性にある．伸長の場合は置き換えた 1byte を元の 2bytes に戻す．v1.0 で共有辞書を作成し，v1.1 に対して共有辞書で圧縮・伸長をすると，伸長時に置き換えていない 1byte が共有辞書を参照すると 2bytes に置き換わってしまう．そのため，正常に伸長することができない．**図 7** に動作例を示す．元のビット列「aabcbbabcabccZ」があったとする．共有辞書圧縮すると bc を Z に，aZ を Y に置き換えるため圧縮データは「aYbbYYcZ」となる．この圧縮データを共有辞書で伸長すると「aabcbbabcabccbc」となる．最後の Z は本来ならば bc に伸長したくない．しかし，共有辞書圧縮をするとこのような問題が起こる．主な原因は共有辞書に置き換えたい文字が v1.1 のデータでは既に使われていることである．この問題を避けるために 3 つの対策を実施した．

① 置き換えない文字を選び出した例外リストを作る
② 置き換わった文字を選び出した展開リストを作る
③ v1.0 の共有辞書を v1.1 でも使えるように再構築する

①は伸長しない文字を決めることで余計に伸長しない対策をした．しかし，圧縮で置き換わった文字が伸長しない文字と指定されていると，正常に 1byte から 2bytes に伸長ができなくなるため，どの位置のどの文字なのか，より詳細な情報が必要となり，データサイズが増えてしまう．②は圧縮しない文字を決めることが異なるだけで，①と同様である．③は v1.1 でも使用できる共有辞書を再構築するため，RAM を多く使うことがプログラム実装から予測できた．共有辞書の再構築は車載 ECU 内でするため，RAM 容量が少ない環境では再構築は難しい．以上の理由より BPE の共有辞書圧縮は効果的でない．

他のアルゴリズムの結果を述べる．データ B の方がデータ A よりもバージョン間のデータサイズに違いがあるため，bsdiff と Zstandard の共有辞書圧縮の圧縮率はデータ A に比べると低い．しかし，他の圧縮アルゴリズムに比べると bsdiff と Zstandard の共有辞書圧縮の圧縮率は高い．

5.3 RAM 使用量

Valgrind [35]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. 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．また，bsdiff は差分更新のため，v1.0 と v1.1 の 2 つを用いて評価した．Zstandard の共有辞書圧縮は v1.0 で作成した共有辞書を用いて v1.1 のデータを圧縮した．

Valgrind の結果を**表 3** に示す．malloc で動的に確保した RAM は使用後に解放されるため，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，表 2 の車載 ECU

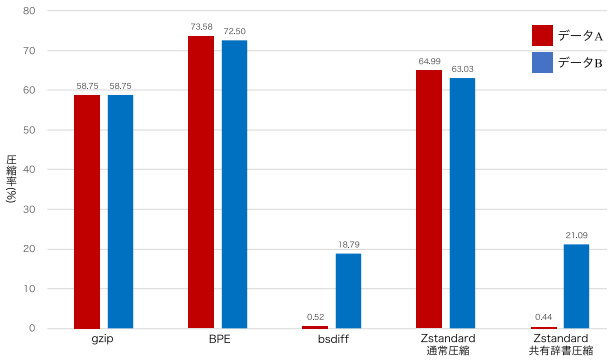

図 6 各アルゴリズムの圧縮率比較

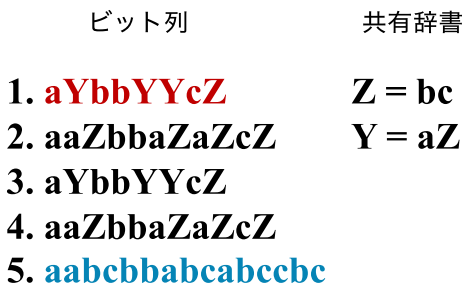

図 7 BPE の共有辞書圧縮を伸長する

表 3 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 共有辞書圧縮	11	280,242	25,476

のスペックより，RAM 使用量の上限値が 32Kbytes となっている．bsdiff は上限値を大幅に上回っており，RAM 容量に余裕がない車載 ECU には使用することが困難であることが確認できる．

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．

5.4 共有辞書の運用

圧縮率と RAM 使用量を考慮した場合，評価したアルゴリズムの中で Zstandard の共有辞書圧縮が車載 ECU のソフトウェア更新に一番適している．本章では，Zstandard の共有辞書圧縮に注力し，共有辞書をどのように運用すれば，より車載 ECU に適応するか評価した．

5.4.1 共有辞書のサイズ変更

共有辞書のサイズを変更することでどう変化するか評価した．データ A の結果を**図 11** に，データ B の結果を**図 10** に示す．どちらも v1.0 を用いた．共有辞書の最大サイズがデータ A とデータ B で違う理由はその最大サイズが最適な共有辞書のサイズである．

共有辞書のサイズが大きいほど圧縮データを小さくすることができる．しかし，CAN に伝送する合計サイズは，共有辞書のサイズが大きいほどデータサイズが大きくなってしまう．そのため，新版バージョンに最適な共有辞書を作成し，圧縮データと共に毎回 CAN で伝送することは難しい．そのため，共有辞書はメジャーバージョンアップデートの時に更新する運用を考える．バグ修正などの早急な対応が必要なマイナーバージョンアップデートの時に，共有辞書の効果が発揮される．

5.4.2 複数回更新の評価

圧縮データに最適な共有辞書を毎回伝送することは難しい．複数回更新でも，共有辞書が効果的に働くか評価した．データ A の結果を**図 9** に，データ B の結果を**図 8** に示す．v1.0 のデータで作成した共有辞書で v1.1 や v1.2 のデータを圧縮した．縦軸の圧縮率はオリジナルデータが圧縮データとなった時の割合である．

バージョン間の変化がデータ A とデータ B 程度であれば，共有辞書のサイズはオリジナルの 8~9 割程度のサイズが複数回更新では適している．早急なソフトウェア更新が求められる場合はバグ修正などの軽微なプログラム変更が多いため，問題ないと判断する．

5.4.3 bsdiff と共有辞書圧縮の比較評価

圧縮率だけを比較すると bsdiff が優れている，複数回更新で bsdiff と Zstandard の共有辞書圧縮を比較評価する．データ A の結果を**図 12** に，データ B の結果を**図 13** に示す．共有辞書のサイズは大きいほど効果がある．そのため，共有辞書のサイズが 20,480bytes 以上を対象として評価した．

Zstandard の共有辞書圧縮は，bsdiff の差分データよりも圧縮データを小さくすることはできなかった．しかし，bsdiff に匹敵する圧縮ができた．また，フラッシュメモリが小さく，RAM 容量に余裕がある場合は共有辞書のサイズを小さくすれば良いため，フラッシュメモリと RAM でトレードオフの関係が成り立つ．

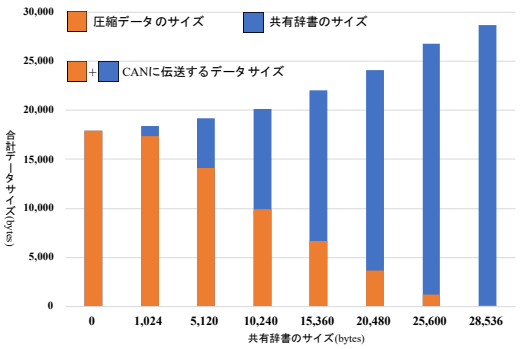

図 9 データ A の合計サイズ

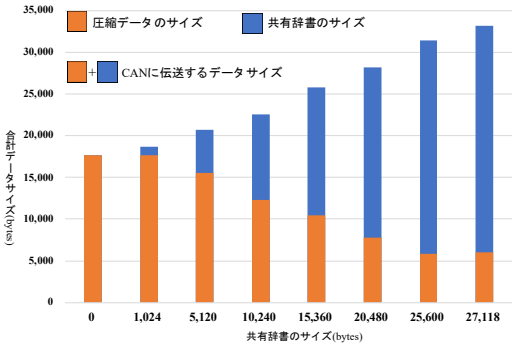

図 8 データ B の合計サイズ

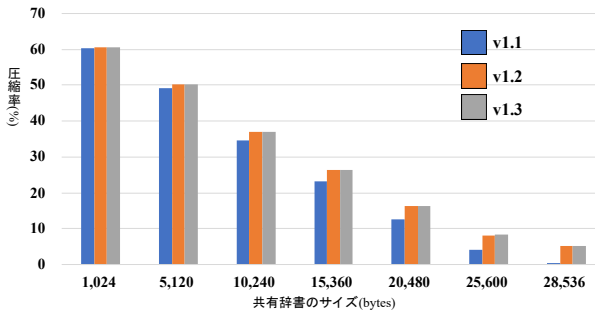

図 11 データ A の複数回更新

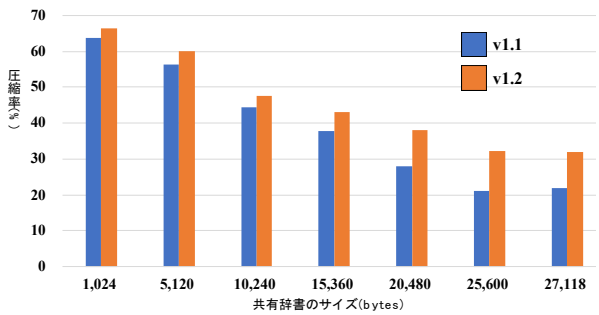

図 10 データ B の複数回更新

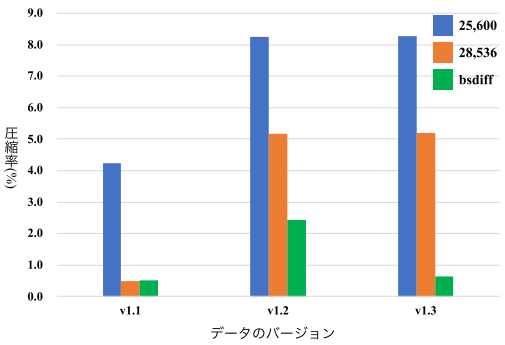

図 12 データ A の圧縮率

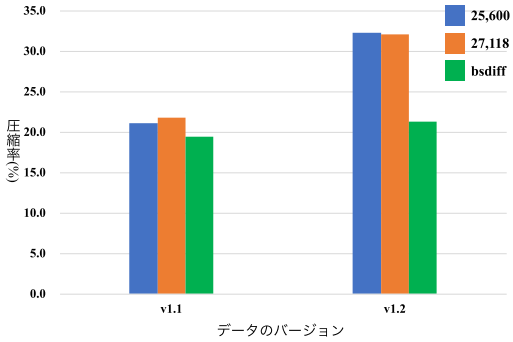

図 13 データ B の圧縮率

6. まとめ

本論文では，CAN の伝送時間を短縮するためにアルゴリズムの比較評価した．また，データサイズ削減のために辞書再利用方式を提案した．圧縮率と伸長時の RAM 使用量を考慮すると，Zstandard の共有辞書圧縮が車載 ECU のソフトウェア更新に適している．共有辞書自体はサイズが大きいため，ソフトウェア更新の度に共有辞書を更新することは難しい．しかし，バージョン間の変更データが評価したデータサイズ程度であれば，v1.0 の時点で作成した共有辞書でも効果があることを示した．フラッシュメモリと RAM 容量を考慮し，共有辞書のサイズを変更することで，トレードオフの関係が成り立つことを示した．

参考文献

[1] Ministry of Land, Infrastructure, Transport and Tourism, “Vehicle Recall,” http://www.mlit.go.jp/en/jidosha/vehicle_recall_17.html, (accessed 2020/5).

[2] 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.

[3] A mara 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.

[4] R. Bosch, “CAN specification version 2.0,” Rober Bousch GmbH, Postfach, 300240, 1991.

[5] 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.

[6] S. Lorenz, “The flexray electrical physical layer evolution,” SPECIAL EDITION HANSER automotive FLEXRAY, pp.14-16, 2010.

[7] 日経 xTECH, “日産「スカイライン」が車載 Ethernet を採用、「手放し運転」向けで,” https://xtech.nikkei.com/atcl/nxt/column/18/00001/02959/, (accessed 2021/1).

[8] 染谷一輝，鈴木直希，寺島美昭，清原良三，“車載 ECU 向け省メモリソフトウェア更新方式,” 情報処理学会論文誌，Vol.62, No.1, pp.254-262, 2021.

[9] 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,” NTT DoCoMo テクニカルジャーナル，Vol.11, No.4, pp.36-41, 2004.

[10] 清原良三，栗原まり子，古宮章裕，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.

[11] Colin Percival, “Matching with Mismatches and Assorted Applications,” Doctoral thesis, Wadham College University of Oxford, 2006.

[12] 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 D，Vol.J90-D, No.6. pp.1375-1382, 2007.

[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 reconstruction of version differences,” IEEE Transactions on Knowledge 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 equipment,” International Conference on Informatics, Electronics and Vision, pp.1-6, 2013.

[16] Y. Onuma, M. Nozawa, Y. Terashima and R. Kiyohara, “Improved Software Updating for Automotive ECUs: Code Compression,” IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), pp. 319-324, 2016.

[17] J. Uthayakumar, T. Vengattaraman and P. Dhavachelvan, “A survey on data compression techniques: Fr9om the perspective of data quality coding schemes data type and applications,” J. King Saud Univ. Comput. Inf. Sci., 2018.

[18] AUTOSAR, https://www.autosar.org/, (accessed 2020/1).

[19] AUTOSAR, “Requirements on Secure Onboard Communication,” https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SRS_SecureOnboardCommunication.pdf, (accessed 2020/1).

[20] 中野将志，久保田貴也，汐崎充，藤野毅，“先進運転支援 システムを搭載した自動車に対する制御乗っ取り攻撃の脅威分析,” 2016.

[21] 畑正人，田邉正人，吉岡克成，大石和臣，松本勉，“不正送信阻止：CAN ではそれが可能である,” 情報処理学会セキュリティシンポジウム 2011 論文集，pp.624-629, 2011.

[22] 竹森敬祐，溝口誠一郎，川端秀明，窪田歩，“セキュアブート+認証による車載制御システムの保護,” 情報処理学会研究報告高度交通システムとスマートコミュニティ (ITS)，Vol.2014-ITS-58, No.8, pp.1-8, 2014.

[23] 芳賀智之，岸川剛，鶴見淳一，松島秀樹，高橋良太，佐々木 崇光，“機械学習による車載ネットワーク攻撃検知システム,” パナソニック技報，Vol.63, No.1. pp.16-21, 2017.

[24] 栗田萌，渡辺俊貴，山野悟，“ログの順序パターンに基づく異常検知手法の提案と CAN のログへの適用,” 情報処 理学会研究報告マルチメディア通信と分散処理 (DPS)，Vol.2017-DPS-170, No.28, pp.1-7, 2017.

[25] Gandhi Kapilan Kulayan Arumugam and Arumugam Chamundeswari, “An Approach for Secure Software Update in Infotainment System,” Proceedings of the 10th Innovations in Software Engineering Conference (ISEC), pp.127-131, 2017.

[26] Zhou Qin, Fei Li, Yi-Huai Wu and Chao Wang, “New ECU Attestation and Encryption Mechanism for In-Vehicle Communication,” DEStech Transactions on Engineering and Technology Research, (ssme-ist), 2016.

[27] Marco Steger, Carlo Alberto Boano, Thomas Niedermayr, Michael Karner, Joachim Hillebrand, Kay Roemer and Werner Rom, “An Efficient and Secure Automotive Wireless Software Update Framework,” IEEE Transactions on Industrial Informatics, Vol.14, No.5, pp.2181-2193, 2018.

[28] Thomas Chowdhury, Eric Lesiuta, Kerianne Rikley, Chung-Wei Lin, Eunsuk Kang, BaekGyu Kim, Shinichi Shiraishi, Mark Lawford and Alan Wassyng, “Safe and Secure Automotive Over-the-Air Updates,” Computer Safety, Reliability and Security pp.172-187, 2018.

[29] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” IEEE Transactions on Information Theory, Vol. 23, No. 3, pp.337-343, 1977.

[30] P. Gage, “A New Algorithm for Data Compression,” The C Users Journal, Vol.12, No.2, pp.23-38, 1994.

[31] Zstandard, https://facebook.github.io/zstd/, (accessed 2020/9).

[32] D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” Proceedings of the IRE, Vol.40, No.9, pp.1098-1101, 1952.

[33] P. Deutsch, “DEFLATE Compressed Data Format Specification Version 1.3,” RFC 1951, 1996.

[34] Toppers, https://toppers.com/, (accessed 2020/5).

[35] Valgrind, https://valgrind.org/, (accessed 2020/7).