Vol.2019-CDS-25 No.6
2019/5/31

車載 ECU 更新向けデータ圧縮方式の評価

鈴木直希[†1]　林敏生[†1]　清原良三[†1]

概要：近年自動車の電子部品の増加に伴って ECU の搭載数や ECU のソフトウェアの規模が増加している．その結果自動車の出荷前に不具合を発見することが困難になり，ソフトウェアの不具合を修正するためのリコールが増加傾向にある．ソフトウェア更新はディーラで行っており，更新が増えるとディーラに負担がかかるため，こ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．ユーザの待ち時間を最小限にするため，ソフトウェア更新時間を最小限にすることが目標となっている．更新時間を最小限にするために差分更新技術が開発されているが，ECU の RAM サイズの関係で適用できないケースもある．本論文では，このような差分更新を適用できない場合のデータ圧縮方式に関して新たな方式を提案し評価を実施し，効果を確認した．

キーワード：圧縮，車載更新

Evaluation of Software Updating for ECU in Vehicle Networks

NAOKI SUZUKI[†1]　TOSHIKI HAYASHI[†1]　RYOZO KIYOHARA[†1]

1. はじめに

自動運転機能の開発や，Advanced Driver Assistance Systems(ADAS)の高度化などにより，自動車に搭載されている Electronic Control Unit(ECU)数や ECU ソフトウェアの規模が増大している．それに伴い，ソフトウェアの不具合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，アプリケーションレベルではソフトウェアの更新も可能になっている．また盗難車両の遠隔操作などのために，車載ネットワークにも外部からアクセス可能になる場合もある．

一方，自動車が外部のネットワーク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の必要性がある．

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た．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，頻繁には更新はできない．

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，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 ソフトウェア更新機能が開発されている．

OTA 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．

2. 車載ソフトウェア更新の課題

走行中に ECU ソフトウェアを更新することは危険であるため，更新は停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．そのため ECU ソフトウェア更新を行うために十分な電源を確保する必要がある．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．

車載機器のソフトウェアは，多くの場合，NOR 型フラッシュメモリに格納され，直接フラッシュメモリ上のコードが実行される場合が多い．

ECU ソフトウェアの更新は車載ネットワークを介して行われる．車載ネットワークの業界標準である CAN (Controller Area Network) は，最大ビットレートが 1Mbps, 最大データ長は 8bytes となっており大量のデータを転送す

神奈川工科大学
Kanagawa Institute of Technology

るには時間を要する[2].

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[3]． 転送時間は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．差分符号化を用いた際の伸長時にはフラッシュメモリの消去ブロックを削除してから適用をしていくため，RAM 上に旧バージョンのイメージが必要である．そこで，RAM メモリ使用量の概念を図 1 に示す．

伸長の際，差分更新では受信した差分データ量，書き換え対象となる消去ブロック，復元した新プログラムの格納領域，ワークメモリ分のメモリ量が必要となる[3]．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，更新データを復号するために十分な RAM を持っておらず，差分更新を実行できない場合がある．

3. 関連研究

車載 ECU の主流である NOR 型のフラッシュメモリのソフトウェア更新の研究は，ガラ携と言われていた時代の携帯電話のソフトウェア更新や，定期的に更新データを送信する STB などで研究されてきた差分更新技術がある[4].

また，車載 ECU にターゲットを絞った差分更新技術もある[5].

しかし，これらの研究は NOR 型フラッシュメモリの消去ブロックのサイズに比べ，RAM 容量が十分に大きいことを前提としている．

車載 ECU のソフトウェア更新システムに関しても差分更新をベースとして様々な ECU や車載機器がある中で更新処理の自動化に関する研究もある[6]．しかしいずれも差分更新できることが前提である．

そこで，我々た，今まで単なる汎用圧縮方式の適用を想定していた NOR 型フラッシュメモリの消去ブロックのサイズに比べ RAM が小さく差分更新できない場合の圧縮データの削減方式に関して研究を実施し，圧縮辞書の再利用によりデータ削減が可能であることを既に示してきた[7].

本論文ではこの手法が，複数回のソフトウェア更新にも適用した場合を評価し，有用性を示す．

4. 提案方式

4.1 Deflate アルゴリズム

本論文では，先行研究[8]で載 ECU の圧縮には適切であると判断した Deflate アルゴリズム[9]に改良を加える方式を提案する．Deflate 圧縮とはハフマン符号化と LZ77 を組み合わせた可逆データ圧縮アルゴリズムである．図 2 にこのアルゴリズムの簡単な流れを示す．Deflate 圧縮では以下のような流れで圧縮を行う．

① 圧縮の際，まず任意の分割サイズごとに圧縮対象データを読込み，その後ハフマンツリーの生成する．

② ハフマンツリーの生成では読み込んだ圧縮対象データを先頭から順に呼び出し，呼び出した位置から前方 32768 以内に長さが 3 以上 256 以内で呼び出した文字から始まるものと一致する文字列があるか探索し(以下この範囲をスライド窓と呼ぶ)，一致長と距離，または文字の記録を行う．その後読み込ん

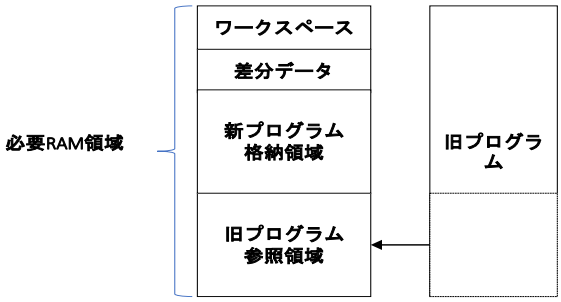

図 1 差分での伸長の際必要なメモリ量

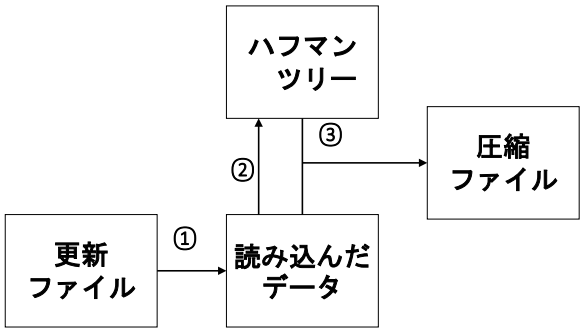

図 2 Deflate アルゴリズムでの圧縮の流れ

だブロックの最後まで探索を終えた後，記録した値を頻度順に並び変え，頻度の多い順に小さい符号の割り当てを行う．

③ そこで生成した符号を用いて読み込んだデータへの割り当てを行いの圧縮を行う．そして分割ブロック分の圧縮データの生成が完了したらハフマンツリーを格納してひとブロック分の圧縮を完了させる．これを更新データ/分割ブロックサイズ分行う．

また伸長の際は圧縮時に用いたハフマンツリーと同じ辞書を用いて圧縮データの伸長を行う．そのため一般的には生成した辞書を圧縮ファイル内に格納して圧縮データとして生成する．

4.2 ファイルの内部構造

Deflate 圧縮を用いた際の圧縮ファイルの内部構造は図 3 に示すようになっておりブロックごとに圧縮データとその圧縮データを生成する際に作られたハフマンツリー，そしてヘッダー，フッターが格納されている．このデータのまとまりが，分割ブロック分存在する．本研究ではこの内部構造に着目して 3 つの提案手法を比較検討しデータ更新時間の削減を図る．

4.3 ハフマンツリーの取り出し

ソフトウェアの更新はちょっとした修正が主なマイナーバージョンアップが多いと仮定し，図 4 に示すように．各ブロック内のハフマンツリーを取り除いて送信をすることによって圧縮ファイルからハフマンツリー分のデータサイズの短縮を図る手法を提案する．

前節でも述べたように通常の圧縮では読み込んだデータを元にハフマンツリーを生成し，そこで生成したハフマンツリーを用いて圧縮・伸長を行う．本提案手法では図 5 に示すように，圧縮の際にハフマンツリーを生成するのではなく外部のハフマンツリーを参照して圧縮･伸長を行う．

自動車の出荷時，車両内にあるデータを用いてハフマンツリーを生成しておく．更新の際，そのハフマンツリーを用いて圧縮・伸長を行う．また 2 回目以降の圧縮では前回のデータ更新が完了した後そのデータを用いて消去ブロック分ごとハフマンツリーを生成し，そこで生成したハフマンツリーを用いて圧縮・伸長を行う．

本来の圧縮では圧縮の際読み込んだデータに対して最適なハフマンツリーを生成し，そこで生成したツリーを用いて圧縮する．しかし，この提案手法で使用しているハフマンツリーは更新前の書き換え対象データに対しての最適なハフマンツリーであり，更新データに対して最適なハフマンツリーを使用しているわけではない．このため，ハフマンツリー分のデータ量を削減できる代わりに各ブロック内部の圧縮データ量が増幅するといったデメリットがある．

4.4 圧縮データの圧縮率の向上

ハフマンツリーを生成する際，前節でも述べたように圧縮データを先頭から呼び出し，前方に一致する文字列があるかを判別し探索を行う．そのためデータの前半に格納されているデータは圧縮されずそのまま格納されることが考

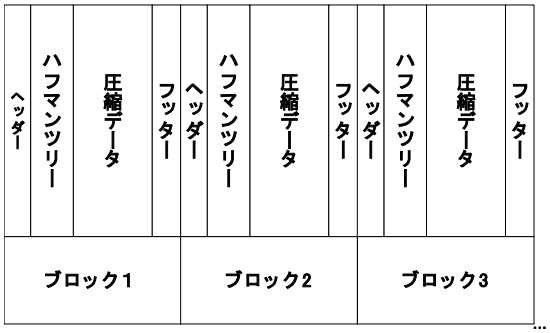

図 3 Deflate 圧縮を用いた際の圧縮ファイルの内部構造

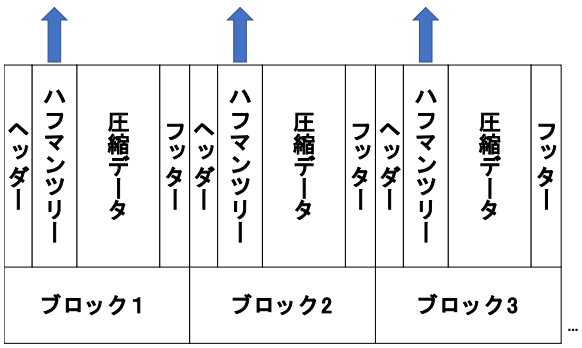

図 4 ハフマンツリーの取り出しによる圧縮

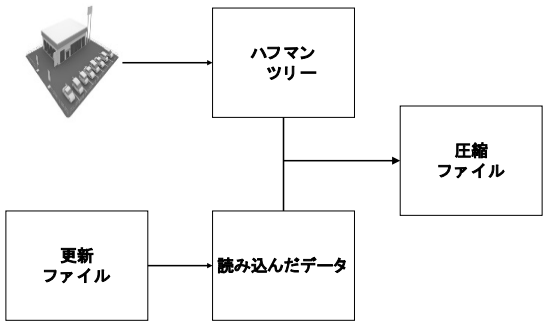

図 5 ハフマンツリーの取り出しによる圧縮方法

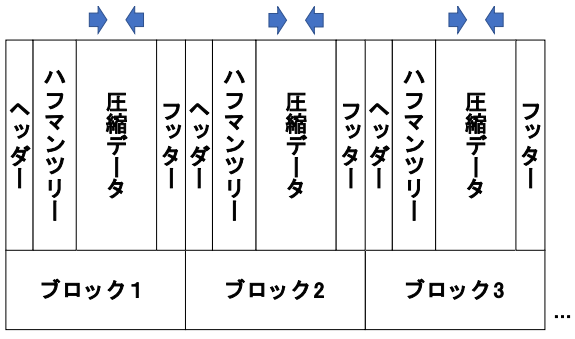

図 5 圧縮データの圧縮率の向上

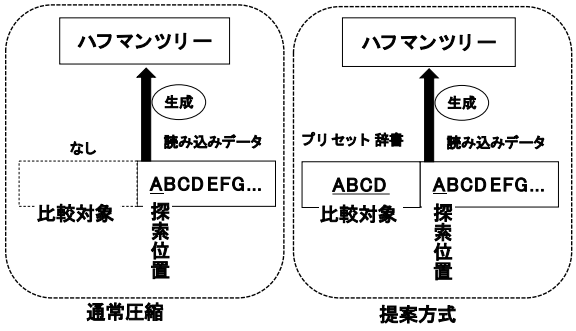

図 6 プリセット辞書を用いたハフマンツリー生成

えられる．そのためハフマンツリーを生成する際，データの前半に擬似的なデータをつなげてデータを拡張することによって前半の探索範囲を広げ，データの圧縮を可能にすることで図 6 に示すようにデータ量を削減する手法を提案する．

具体的な圧縮方法を図 7 に示す．分割ブロックサイズ分のデータを読み込む前に探索対象とするデータを読み込み（以下このファイルをプリセット辞書と呼ぶ),プリセット辞書としてセットする．次に，このプリセット辞書の後に続けて更新ファイルを分割サイズ分読み込み更新ファイルの先頭から一致する文字列の探索を行う．プリセット辞書には更新前の旧データを用いる．

また，この方法でもデータの伸張の際，プリセット辞書が必要になるので，圧縮の際に用いたプリセット辞書は更新データとして送信せず，自動車内にある同じ旧データの同じブロックのデータをプリセット辞書として用いて更新する．

4.5 ハフマンツリーの取り出しと圧縮データ量の削減

さらに，2 種類の提案手法を組み合わせた実装を提案する。この手法では圧縮の際，プリセット辞書を読み込むことでブロック内データの前半部分のデータ量を削減しつつ,その後圧縮データを生成する際，外部のハフマンツリーを用いることで,ハフマンツリー分のデータ量を削減しつつ、圧縮データ量削減を図る．これらの提案手法を比較検討し更新時間の短縮を図る．

5. 評価

5.1 評価環境

本実験ではまず前節で述べた 3 つの提案手法の圧縮率の比較し．そしてその後適切である提案手法を用いて更新した際の更新時間を測定した．また，適切である提案手法を用いて複数回におけるバージョンアップを行い圧縮率の変化を確認した．

なお，更新ファイルは deflate 圧縮で使用している zlib ライブラリと自動車制御様リアルタイム OS である TOPPERS を対象として評価を行った．圧縮の際追加データ量の大きさによって圧縮率，更新時間にばらつきが出るため，追加データ量が多いものを A,B，追加データ量の少ないものを C,D として評価した．評価に用いた更新ファイルと更新内容を表 1 に示す.

5.2 圧縮率の比較

本論文ではまず先ほど提示した 3 つの提案手法の中でどの提案手法の一番圧縮率が高くなるのか調査した．なお通常の圧縮とハフマンツリーの取り出しによる圧縮では分割はせずに一括で圧縮を行い，圧縮データ量の削減による圧縮と，圧縮データ量削減方式とハフマンツリー取り出し方式の組み合わせた圧縮方式ではアルゴリズムの性質上スライド窓が 32KB までしか適用できないため[8]分割サイズを 32KB，プリセット辞書のサイズを 32KB と仮定して実験を行った．

結果を表 2 に示す．ハフマンツリー取り出し方式より圧縮データ量削減方式が有効であることが確認できた．この結果よりハフマンツリーを取り出すよりファイル内圧縮データの圧縮率を向上する方がデータ量削減する上で効果的であることが確認できた．

また，ハフマンツリー取り出し方式とプリセット辞書適用方式の合わせた提案方式よりプリセット辞書適用方式の方が圧縮率は上回る結果となった．理由として，取り除いたハフマンツリーのサイズよりも圧縮データに対して最適ではない他のハフマンツリーを圧縮の際に用いることによって圧縮データ量が増加した量が大きくなったため，このデータ増加分がプリセット辞書を用いて短縮したコードを増幅させてしまい，圧縮データ量の削減とハフマンツリーの取り出しを組み合わせた方式が圧縮データ量削減方式に比べて圧縮率が劣ったと考えられる．この結果より本論文では圧縮データ量削減方式を中心とした提案手法として用いる．

表 1 今回使用した更新ファイル[KB]

	プログラムサイズ	更新内容
A	76,2→82,2	libz1.0.1.a→libz1.1.1.a
B	82,2→103,3	libz1.1.1.a→libz1.2.1.a
C	150,6→150,6	libz1.2.4.a→libz1.2.5.a
D	153,8→155,6	libz1.2.10.a→libz1.2.11.a

表 2 圧縮率の比較[KB]

	通常	ハフマンツリー取り出し	圧縮データ量削減
A	29,9	29,5	24,6
B	43,3	42,8	38,3
C	66,0	65,4	4,8
D	67,4	66,8	5,3

5.3 複数回のバージョンアップ

次に,本論文では，提案方式での複数回でのバージョンアップに関して評価を実施した．表 3 に評価対象となる各バージョンのデータサイズを示す．各バージョンを単純圧縮したデータ削減率と辞書を再利用した場合のの削減率を表 4,表 5 に示す．また２つの方式の比較グラフに示す．

7 回の更新を行った結果ではプリセット辞書を毎回作る

表 3　プログラムサイズ[KB]

	プログラム名	プログラムサイズ
A	libz1.0.1.a	76.28
B	libz1.1.1.a	82.26
C	libz1.1.4.a	85.62
D	libz1.2.1.a	103.39
E	libz1.2.4.a	150.62
F	libz1.2.5.a	150.68
G	libz1.2.10.a	153.8
H	libz1.2.11.a	153.87

表 4 データを単純圧縮した場合

	更新内容	削減量	削減率
A	libz1.0.1.a→libz1.1.1.a	24.60	70.095%
B	libz1.1.1.a→libz1.1.4.a	8.96	89.535%
C	libz1.1.4.a→libz1.2.1.a	38.50	62.762%
D	libz1.2.1.a→libz1.2.4.a	60.63	59.746%
E	libz1.2.4.a→libz1.2.5.a	4.84	96.788%
F	libz1.2.5.a→libz1.2.10.a	56.81	63.062%
G	libz1.2.10.a→ libz1.2.11.a	5.36	96.517%

表 5　プリセット辞書を再利用した場合[KB]

	更新内容	削減量	削減率
A	libz1.0.1.a→libz1.1.1.a	24.60	70.095%
B	libz1.1.1.a→libz1.1.4.a	25.46	70.264
C	libz1.1.4.a→libz1.2.1.a	38.67	62.598%
D	libz1.2.1.a→libz1.2.4.a	63.36	57.934%
E	libz1.2.4.a→libz1.2.5.a	63.41	57.917%
F	libz1.2.5.a→libz1.2.10.a	64.63	57.978%
G	libz1.2.10.a→ libz1.2.11.a	64.59	58.023%

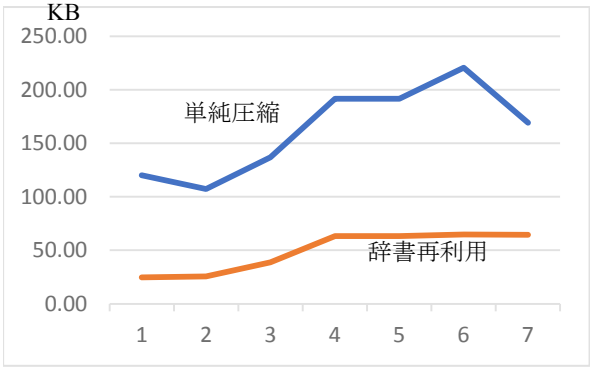

図８　単純圧縮と辞書再利用方式での複数回更新のデータ量比較

ことになる単純圧縮よりもはじめに作ったプリセット辞書を用いて圧縮をしていく辞書再利用方式の方がが,データ量が少なことがわかる．

バージョンアップにより変化があるのは，新規のコードの部分と若干の修正によるリンク先アドレスの変化などである．アドレス変化に関しては，ハフマン符号化の辞書には変化があるが，ハフマン符号化辞書を毎回送るようにしているため，プリセット辞書に対する変化が少なく，複数回の更新にも耐えられていると考えられる．

しかしながら，徐々に実行プログラムの内容も変わっていき，最終的には大きく変わるのであれば，辞書再利用による方式では線形的にデータ量が多くなるため，どの程度複数回の更新があるのかを見極める必要がある．

6. まとめ

本論文では Deflate アルゴリズムをベースに圧縮方法をことで消去ブロックサイズより RAM サイズが小さい ECU を対象とした更新時間の軽減を行った．更新対象の書き換え対象データをプリセット辞書として用いて 16KB 分のデータ量をセットし圧縮を行うことによって，普通の Deflate 圧縮と比べて 1 つのファイル当たり約 1 から 30 秒程度の更新時間の削減を確認した．

しかしプリセット辞書として用いたデータが書き換え前の同じブロックのデータを用いて行ったため追加量の多いファイルの場合だと用いた辞書と読み込んだ更新データで一致する文字列が少なくデータ量の削減にあまり効果が得られないことが考えられる．

そこで今後の課題として，プリセット辞書として用いるデータの取り出し方を検討することや探索文字の比較対象の範囲の定め方を変更するなどといったハフマンツリーの生成方法の変更を行う．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．

また，実際の実行用バイナリファイルでの評価が必要で

あり，ECU 全体を見た差分更新を含む更新システムとしての評価も予定している．システムとしてとらえた場合は，ドライバのコンテキストに応じて，自動車の停車時間がかわるため，コンテキストを推定または指定させることにより，今回の方式における辞書の更新も可能であるかもしれない．こういった運用も含めた検討および評価を実施する予定である．

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] 佐藤道夫，”車載ネットワーク・システム徹底解説，” CQ 出版社，　ISBN978-4-7898-3721-7
[3] 寺岡秀敏他，”車載 ECU 向け差分方式，” 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, pp.41-50, 2017
[4] 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
[5] Tsuneo Nakanishi, Hsin-Han Shih, Kenji Hisazumi, Akira Fukuda, “Software update scheme by airwaves for automotive equipment,” International Conference on Informatics, Electronics and Vision 2013.
[6] 寺岡秀敏他，”軽量スクリプト言語を用いた自動車ソフトウェア遠隔更新制御方式の検討，”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.8, No.3 pp,.32-42, 2018
[7] 林敏生, 清原良三，”車載 ECU ソフト更新のためのデータ圧縮方式,”研究報告マルチメディア通信と分散処理（DPS）,2019-DPS-177(3)
[8] 小沼寛他，”車載 ECU 向けソフトウェア更新のためのデータ圧縮方式，“ 情報処理学会 DICOMO 2017, pp.761 – 767, 2017
[9] DEFLATE, https://tools.ietf.org/html/rfc1951