Vol.2019-DPS-177 No.3
2019/1/31

車載 ECU ソフト更新のためのデータ圧縮方式

林敏生　清原良三

概要：近年自動車の電子制御部品の利用，増加により，ECU の搭載数や ECU ソフトウェアの規模が増大しており，自動車の出荷前に不具合や欠陥を発見することが困難になっている．そのため，プログラムの不具合を起因とするリコールが増加傾向にある．しかし毎回カーディーラでソフトウェア更新を行うとディーラのキャパシティの問題などで多大なコストが発生する．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．ユーザの待ち時間を最小限にするためにもソフトウェアの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 の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

Compression for Software Updating of ECU on Vehicles

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

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，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．ECU ソフトウェア更新を行うため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

神川話工科大学
Kanagawa Institute of Technology

ⓒ 2019 Information Processing Society of Japan

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]．

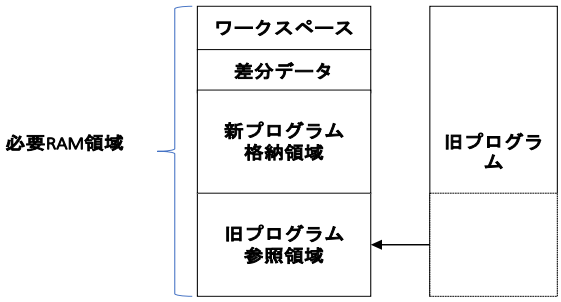


図 1 差分での伸長の際必要なメモリ量

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，本論文では，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を提案する．

4. 提案方式

4.1 Deflate アルゴリズム

本論文では，先行研究[7]で車載 ECU の圧縮には適切であると判断した Deflate アルゴリズム[8]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 3276 以内に長さが 3 以上 286 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辞書と呼ぶ)，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．

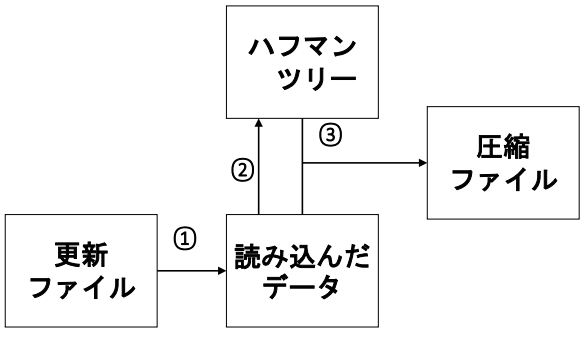


図 2 Deflate アルゴリズムでの圧縮の流れ

ⓒ 2019 Information Processing Society of Japan

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 圧縮を行った際の圧縮ファイルの内部構造は図 3 に示すようにブロックごと圧縮データとその圧縮データを生成する際に用いたハフマンツリー辞書，それにヘッダーとフッターを加えたデータが生成した分割ブロック分格納されている．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，前節で述べたように探索文字より前に一致する文字があるか判別し生成を行う．そのため読み込んだ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えられる．そのためハフマンツリーを生成する際，読み込んだ更新データの前に疑似的にデータを繋げてデータの拡張することによって前半部分の探索範囲を増やす手法を提案する．それによって読み込みデータの前半部分の圧縮を行い，図 6 に示すように，圧縮データの圧縮率の向上を図る．

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)，スライド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

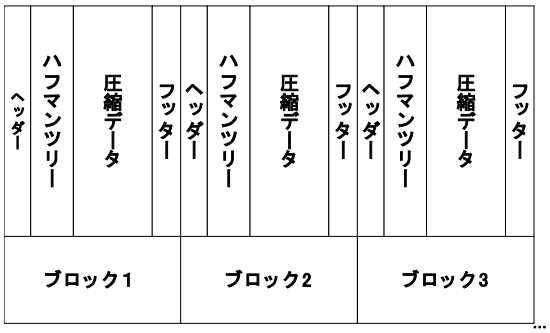


図 3 Deflate 圧縮を用いた際の圧縮ファイルの内部構造

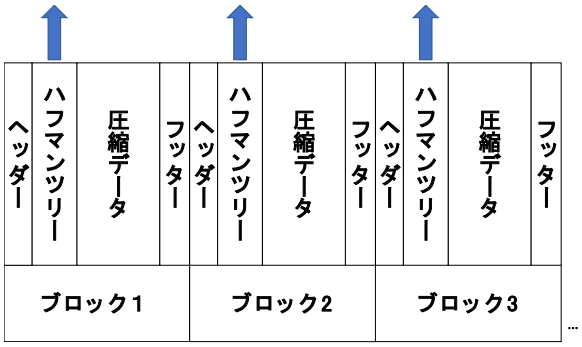


図 4 ハフマンツリーの取り出しによる圧縮

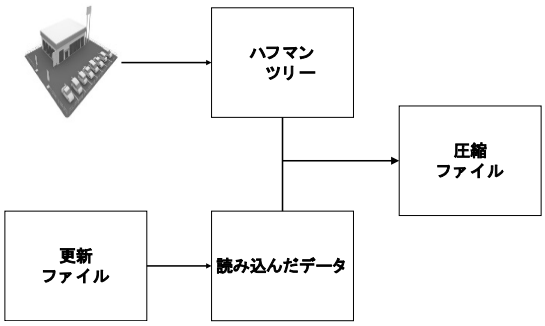


図 5 ハフマンツリーの取り出しによる圧縮方法

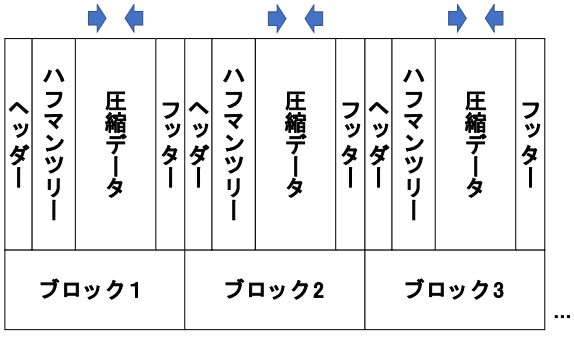


図 5 圧縮データの圧縮率の向上

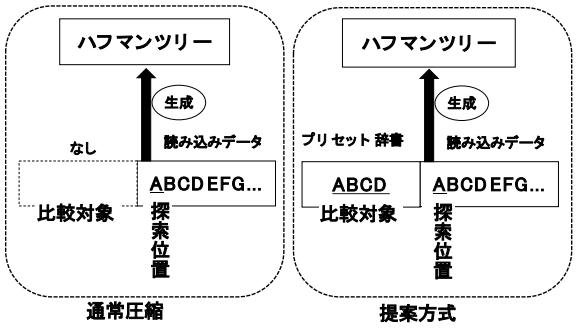


図 6 プリセット辞書を用いたハフマンツリー生成

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文字列の探索を行う．プリセット辞書には書き換え対象となる旧データを用いる．

またこの方法でも伸長の際，同じプリセット辞書が必要になる．圧縮の際に用いたプリセット辞書は更新データとして一緒に送信せずに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．

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.

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

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辞書が 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．

5.3 メモリ使用量

しかし圧縮率の比較の際用いた圧縮データ量削減方式で用いたデータファイルは旧データを 1 ブロック分丸ごと用いている．前節でも述べたようにプリセット辞書を用いて圧縮をした場合，伸長の際にプリセット辞書も展開しないといけない．そのため伸長時，ハフマンツリー用のメモリ量や圧縮データの展開用のメモリ量とは別にプリセット辞書用に消去ブロック分のメモリ量を用意しなければならない．しかし本実験では利用できる RAM サイズが消去ブロックよりも小さく差分が出来よう出来ない場合に対応したいため，付属のデータサイズを小さくしないといけない．そのため付属ファイルのデータ量を小さくしていった場合

表 2 圧縮率の比較[KB]

	通常	辞書取り出し	圧縮データ量削減
A	28,6	29,5	24,4
B	41,9	42,8	40,6
C	62,2	65,4	4,3
D	63,8	66,8	4,9

ⓒ 2019 Information Processing Society of Japan

表 3 A に対しての圧縮率[KB]

		分割サイズ					
		1	2	4	8	16	32
プリセット辞書	分割サイズ分	35.4	30.7	27.4	21.3	17.4	24.4
	16						19.8
	8					20,7	23.1
	4				24.3	25.5	26.3
	2			30.3	27.6	28.0	27.6
	1,		33.7	32.7	30.5	29.8	29.1
	0.5	36.6	35.3	33.7	31.7	30.5	29.5

表 4 B に対しての圧縮率[KB]

		分割サイズ					
		1	2	4	8	16	32
プリセット辞書	分割サイズ分	51.2	46.1	42.6	41.6	40.3	38.2
	16						37.1
	8					40.9	37.1
	4				43.9	43.1	40.5
	2			44.5	45.0	43.2	42.1
	1		48.1	46.0	46.3	45.1	42.9
	0.5	52.3	49.2	46.7	46.5	45.3	43.1

表 5 C に対しての圧縮率[KB]

		分割サイズ					
		1	2	4	8	16	32
プリセット辞書	分割サイズ分	7.1	5.0	3.8	3.1	2.7	4.3
	16						30.1
	8					33.1	47.0
	4				34.4	48.4	55.7
	2			36.5	50.9	57.2	60.3
	1		38.1	52.4	59.2	62.1	63.1
	0.5	42.6	55.5	60.9	63.7	64.9	64.6

表 6 D に対しての圧縮率[KB]

		分割サイズ					
		1	2	4	8	16	32
プリセット辞書	分割サイズ分	12.1	7.7	5.5	4.4	4.1	4.9
	16						28.8
	8					35.7	48.0
	4				35.9	50.6	57.6
	2			39.5	53.6	59.8	62.5
	1		41.1	56.0	62.2	64.3	65.1
	0.5	46.6	58.9	64.6	66.5	66.7	66.2

の圧縮ファイルのデータ量の測定を行った．

表 3 から表 6 に評価結果を示す．表の網掛けの部分は通常の圧縮よりも圧縮ファイルのデータ量が小さいことを表している．追加データ量が多い場合だと通常の圧縮より圧縮率を上回るためにはプリセット辞書のサイズが 8KB は必要となることが確認できた．また追加データ量が少ない場合だとプリセット辞書用に 0.5KB 以上メモリ量を確保できていれば,圧縮率が通常圧縮を上回ることが確認できた．この結果よりプリセット辞書に割り当てることが出来るメモリ量として約 8KB 以上 RAM が確保されていれば通常の圧縮より圧縮率が向上することが確認できた．

5.4 更新時間

これらの結果より，圧縮データ量削減方式を提案方式として用いて実際の ECU(RX110)を想定して消去ブロックを 32KB, RAM サイズを 16KB とおいて更新時間を評価した．

更新時間の評価の仕方として文献[3]を参考に，シミュレータによる復元時間の計測と，卓上計算による通信時間の積算を組み合わせて行った．通常の圧縮での更新時間の評価を表 7 に示す．提案手法での更新の評価を表 8 に示す．そして通常の圧縮と提案手法を用いた圧縮の更新時間の比較評価を表 9 に示す．

表 8 より A，B の追加データ量の多い更新では通常の圧縮に比べ通信時間が約 1 秒から 5 秒程度短縮されているこ

表 7 通常の Deflate 圧縮を用いた際の更新時間[s]

	総更新時間	通信時間	リクエスト	レスポンス	伸長時間
A	14.260	14.260	9.097	5.163	0.000
B	20.862	20.862	13.309	7.554	0.000
C	30.933	30.933	19.733	11.200	0.000
D	31.722	31.722	20.237	11.486	0.000

表 8 提案手法を用いた際の更新時間[s]

	総更新時間	通信時間	リクエスト	レスポンス	伸長時間
A	8.667	8.667	5.529	3.138	0.000
B	20.075	20.075	12.806	7.268	0.000
C	1.367	1.367	0.872	0.495	0.000
D	2.053	2.053	1.310	0.743	0.000

表 9 更新時間の比較[s]

	秒数(秒)	通常の圧縮に対する削減率
A	5.593	61%
B	0.787	96%
C	29.567	4%
D	29.669	6%

ⓒ 2019 Information Processing Society of Japan

とを確認した．また，追加データ量の少ない C，D に関しては通常の圧縮に対し，通信時間が約 30 秒程度と大幅に短縮されていることを確認した．また伸長時間に関しては通常の圧縮でも提案手法での圧縮でも 0 秒と変化はなかった．

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．

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] 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] 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
[8] DEFLATE, https://tools.ietf.org/html/rfc1951
[9] zlib 1.2.11 Manual，https://zlib.net/manual.html

ⓒ 2019 Information Processing Society of Japan