Vol.2016-DPS-167 No.13
Vol.2016-MBL-79 No.13
Vol.2016-ITS-65 No.13
2016/5/26

車載 ECU 向けデータ差分圧縮方式

小沼寛[†1]　寺島美昭[†2]　清原良三[†3]

概要 : 自動車では様々な電子制御を行うようになり，搭載される ECU(Electronic Control Unit)の数が年々増加の一途をたどっている．運転支援機能など，システムの高機能化により，組み込まれるソフトウェアは大規模，複雑化し，その結果，出荷前に可能な試験の工数やタイミング試験の数の発散によりバグや欠陥を発見しきれず，出荷後に不具合が発覚し，リコールとなる更新が必要な場合がある．また，自動運転車両やコネクテッドカーの登場によりソフトウェアの更新機能をネットワークで実現することが技術的に可能になりつつある．ECU のソフトウェアの更新は，低速な車載ネットワークを介して行うことになるため，大量のデータの転送には車両内で時間を要する．即ち，エンジンをかけたまま車両を動かせない時間が発生することが想定される．そこで本論文では，更新に必要なデータ量を削減することでダウンロード時間を短縮する手法を提案，評価する． 提案手法では，差分符号化ツール bsdiff に対し ECU ソフトウェア向けに改良提案を行い，実装した提案手法では僅かではあるが若干の効果があることを示す.

キーワード : ソフトウェア更新，差分符号化，ECU，bsdiff

1. はじめに

車載システムは ADAS（Advanced Driver Assistance System），ABS（Antilock Brake System）などの多くの機能を持っている．それに伴い搭載される ECU（Electronic Control Unit）は年々増加し，1 台当たり 70 個以上に上るものも存在する[1]．また，システムの高度化によりソフトウェアは大規模，複雑化し，不具合を修正しきれず出荷後に発覚する場合がある．そのような場合，不具合が発見され次第，早急に修正を行うことが重要となる．

不具合が発覚した例として FCA 社は，車載システムのソフトウェアに存在した脆弱性により大規模なリコールを行っている[2]．また，フォルクスワーゲンのディーゼル車には不正なソフトウェアが組み込まれており，ソフトウェアのアップデートの必要性がある．

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るには時間を要する[3]．そのため，更新データのサイズを削減することが重要である．また，コネクテッドカーや自動運転車の登場により，インターネットを介した更新データのダウンロードの増加が予想される．トラフィックについて考慮した場合，データ量はより小さい方が望ましい．

本論文では，バイナリコード向け差分符号化方式の中でも圧縮率が高い方式の一つである bsdiff[4]を ECU ソフトウェア向けに改良し，更新に必要なデータ量を削減する提案を行う．

†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

2. 関連研究

ECU ソフトウェアの更新中は，自動車の運転は行えないが，エンジンをかけておく必要あるいは，更新中に給電がなくならないことを保証する必要がある．そのため，更新時間を出来るだけ短くすることが求められる．ソフトウェアの更新時間を短縮するための要点として以下の 3 点がある．

(1) **ソフトウェア構造**
(2) **ダウンロード時間**
(3) **ROM 書き換え時間**

2.1 ソフトウェア構造

ソースコード上では一部分のみの変化であるが，図 1 のようにバイナリコード上では命令の追加や削除などにより

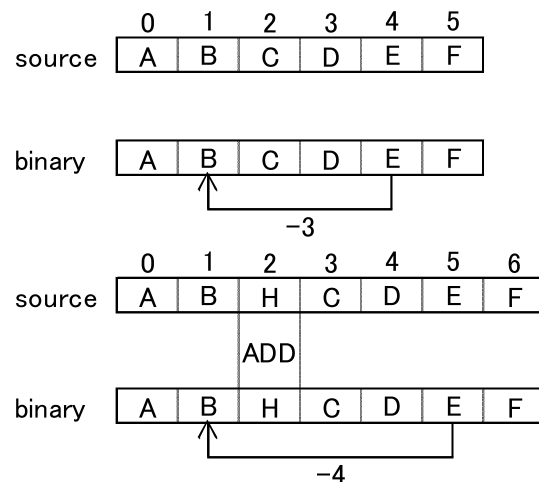


図 1　追加による相対参照のずれ

ⓒ2016 Information Processing Society of Japan

ずれが生じ，参照先のアドレス値が変わることや，レジスタのアサインが変わることで直接変更が加わっていない部分にも差分が生じる場合がある．その結果，ソースコード上の一部の変化により，バイナリコード全体に差分が生じ，多くの差分が発生する可能性がある．

ソフトウェアの構造を工夫することでバイナリコード上の差分を減らす研究がされている[5]．この研究では，コードを一定のモジュール毎に区切り，モジュール間に空き領域を設け，アップデートによる追加や削除の影響を吸収することで，他の領域への影響を無くし差分を生じにくくしている．

しかしながら，携帯電話のソフトウェアは一定の少数の会社が作成しており巨大な大きなソフトウェアとして扱うことができるのに対して，車載ソフトウェアは部品メーカごとに作成する分散ソフトウェアと考えるべきで，多数のメーカの各 ECU 向けのソフトウェアを合わせることにより巨大化していると考える．そのため，一つの ECU 上のソフトウェアはそれほど大きいわけではなく，本方式のような工夫は必要がないと考えられる．

2.2 ダウンロード時間

更新時間は，更新データのダウンロード時間と，ROM への書き込み時間からなる．データ量を D，転送速度を V，書き換えが必要な領域を S，S による書き換え時間を R(S) としたとき，更新時間 T(D,S)は以下の式からなる．

$$\mathrm{T(D,S)} = \frac{\mathrm{D}}{\mathrm{V}} + R(S) \quad (1)$$

ここで，PC やスマートフォンなど広帯域の通信が行える場合 V が大きくなるため，更新時間は書き換え時間 R(S)への依存度が大きくなる．しかし，CAN のような低帯域通信の場合，データ量による要因が大きくなる．そのため，データ量を削減することが重要となる．

更新に必要なデータ量を削減する有効な手法として差分符号化がある．差分符号化については様々な方式が研究されている[4][6][7][8][9]．

また文献[10]では，本論文の先行研究として，汎用的な差分符号化の一つである bsdiff[4]を ECU ソフトウェア向けに改良を行っている．ECU の多くは RISC プロセッサであるため，固定長命令のバイナリコードとなる．本先行研究[10]では，bsdiff を固定長命令用に改良を行うことと，命令の挿入や削除による参照アドレスの変化や，レジスタアサインのずれを把握することにより差分情報の削減を行っている．

2.3 ROM 書き換え時間

使用されている ROM の種類により書き換え時間を考慮する必要がある．SSD や USB メモリで使用されている NAND 型フラッシュメモリは，プログラムの実行中でも更新を行うことができる．しかし，マイコンや車載機器で使用されている NOR 型フラッシュメモリは実行中に更新が行えないため書き換え時間を考慮する必要がる．

書き換え時間は，書き換えが必要なブロック数に依存する．書き換えが必要な個所を局所的にする研究がされている[1]．

ECU ソフトウェアのダウンロード時間は，低帯域の車載ネットワークを介して行われるため大きな要因となる．そのため，更新に必要なデータ量を削減することが重要となる．

本論文では，汎用的な差分符号化の一つである bsdiff に改良を行うことでデータ量の削減を行う．

3. bsdiff/bspatch

bsdiff は差分符号化アルゴリズムの一つである．差分符号化とはデータの転送や保存を完全な状態ではなく，バージョン間の差分情報のみを用いることでデータ量を削減する手法である．差分情報のみを用いて更新を行うため，大幅なデータ量の削減が期待できる．bsdiff は他の差分符号化と比べ高い圧縮率で知られている．

bsdiff のアルゴリズムを図 2 に示す. bsdiff による差分符号化は以下の動作を繰り返すことで行われる．

(1) **共通文字列の検出**
(2) **検出した文字列の拡張**
(3) **差分判定**

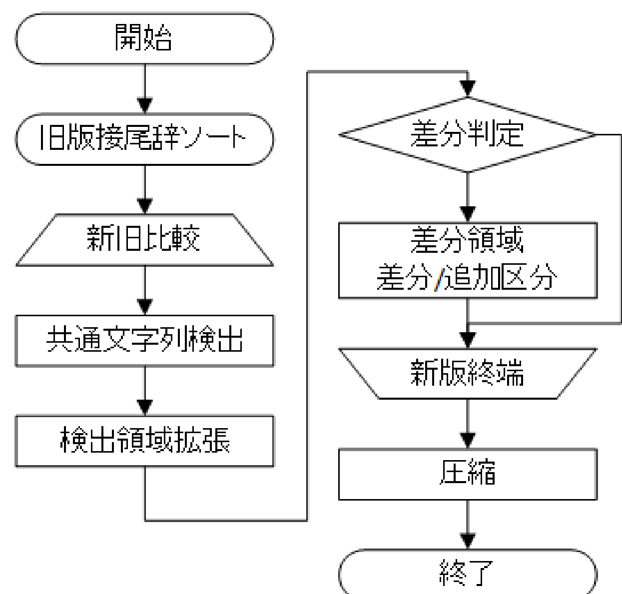


図 2 bsdiff アルゴリズム

(1) 共通文字列の検出

始めに新版と旧版を比較し共通文字列を検出する．共通文字列の検出は，接尾辞配列と二分探索を用いることで高速に行われる．

(2) 検出した文字列の拡張

その後，検出した共通文字列の領域を一定以上の差分がでるまで拡張していく．一定以上の差分が出たとき，その領域を差分領域と見なす．

(3) 差分判定

差分領域は，図 3 に示すように差分情報と追加情報に判別され，差分情報は旧版と新版の差を，追加情報はそのまま保存される．差分が無い部分の差分情報は 0 となり，バイナリコード上で差分が無い部分では 0 が連続する．この様な同じ文字の連続は，bzip2 アルゴリズムにより効率良く圧縮することができる．差分情報と追加情報は，図 4 に示すフォーマットのパッチファイルとして各ブロックに保存する．同時に，パッチファイルには複合化時に使用する命令として，符号化の編集順序を保存する．

最後に差分情報，追加情報，編集命令に対して圧縮が行われる．通常の bsdiff では bzip2 アルゴリズムにより圧縮が行われるが，他の圧縮アルゴリズムによる実装も可能である．ソフトウェアの更新には，bspatch により，パッチファイルの編集命令を順次読み込み，各情報を旧版ファイルに適用することで行われる．

本論文では既存の bsdiff に改良を加えることにより，パッチファイルサイズを削減することで，ダウンロード時間の短縮を図ることを目指している．

4. 提案手法

本論文では，ソースコード上では表れなくても，バイナリコード上では繰り返し同様の処理が行われている部分が多数存在することに着目する．

例として，関数の呼び出し規約について

表 1 に示す．関数が呼び出される際に，サブルーチンプロローグと呼ばれる一連の処理が行われる．この処理は関数毎に同様の処理が行われるが，局所変数の有無やサイズなどにより即値が異なる．そのため，単純にバイナリ差分抽出を行うと差分領域として扱われることが多くなると想定できる．

そこで, 車載 ECU の多くが固定長命令アーキテクチャであることを利用し，bsdiff による差分符号化の前処理とし

表 1 サブルーチンプロローグ

アセンブラ			概要
1:	push	{fp, lr}	ベースポインタの保存
2:	add	fp, sp, #4	ベースポインタの変更
3:	sub	sp, sp, #imm	局所変数の領域確保(imm:即値)

	0	1	2	3	4	5	6	7	8	9
old	t	o	x	x	x	x	x	x		
new	t	o	x	x	t	s	x	x	y	y
差分	0	0	0	0	-4	-5	0	0		
追加									y	y

図 3 差分領域の判別

内容		サイズ
ヘッダー	”BSDIFF40”	8
	命令ブロック長	8
	差分ブロック長	8
	新版ファイルサイズ	8
命令ブロック	ADD	8
	COPY	8
	SEEK	8
	…	
差分ブロック		
追加ブロック		

図 4 bsdiff パッチファイルフォーマット

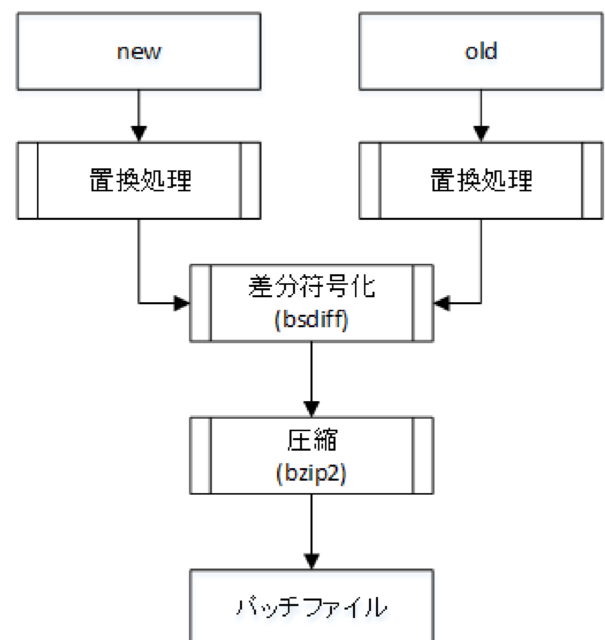


図 5 提案手法

Ⓒ2016 Information Processing Society of Japan

て，一連の処理を単一の命令に置き換える．

提案手法の流れを図 5 に示す．一連の処理を一つの命令に置き換え，即値となる部分はオペランド部に保持することで命令量の削減を行う．命令量を削減し，バイナリコードを短縮することで，差分として扱う領域が短縮され，更新に必要なデータ量も小さくなると考える．

4.1 エンコード

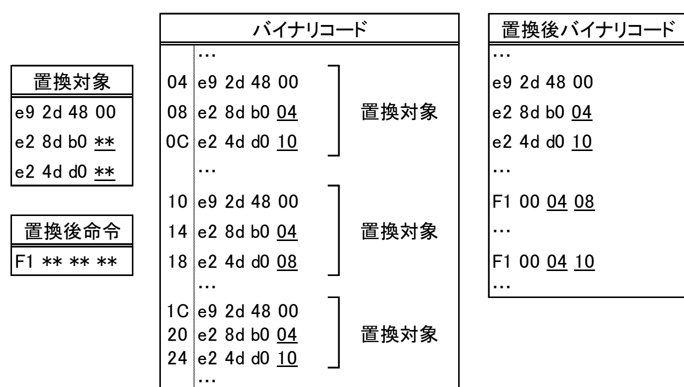
置換処理の例を図 6 に示す．初めに，対象とする命令列のパターンと，変換後の架空の命令を生成する．ここで架空の命令コードには，バイナリコード上で使用されていない命令コードを使用する．

次に，バイナリコードをパターン長ごとに順次読み込み，置換対象であるか判定を行う．置換対象であれば，架空の命令に置き換え，オペランド部に即値を記録する．なお，復元用に最初に検出された置換対象には置換を行わず，その位置を記録する．以上の処理をバイナリコード終端まで繰り返す．

最後に，置換情報として置き換えた架空の命令コード，置換対象の長さ，即値となるインデックス，最初に検出された置換対象の位置を保存する．

4.2 デコード

置換情報を読み込むことで置き換えた命令コードを，元の命令列に復元する．符号化されたバイナリから順に一命令ずつ読み込み，架空の命令が検出されたら，記録されている置換対象の位置から命令列を読み出し，架空の命令と置換情報を用いて元の命令列へと復元する．



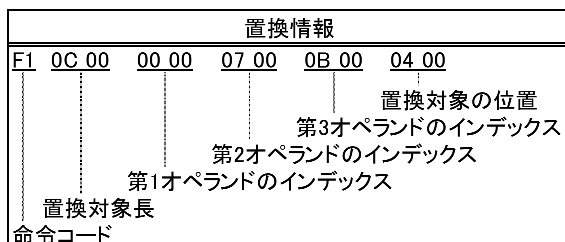


図 6 エンコード例

5. 評価

5.1 サブルーチンプロローグの置換

提案手法の評価を行う．評価は，複数のバイナリコードと，それらの異なるバージョンを用意し，それぞれのバージョン間毎の差分データサイズにより行った．バイナリには，ELF フォーマット，命令セットは 32bit 固定長 ARM のものを使用した．

評価にあたって，内部の構造をすべて把握できる程度の実験用のバイナリファイル Ax, Bx と，実際のソフトウェアサイズに匹敵するバイナリファイル Cx で評価を実施した．

表 2 に評価に使用したバイナリコードのサイズと，提案手法による命令置換を行った後のサイズを示す．バイナリ A はバージョン間でのサイズ差はあまりないが，ずれによる差分が生じている．また，複数の関数に機能が分けられた構造になっている．バイナリ B はバージョン間でのサイズ差が大きく，機能がほとんど分けられていない．

バイナリ C は A，B に比べサイズが大きく，多くの関数からなっている．A，C では置換により数バイト削減がされたが，僅かな効果しか見られなかった．B においては置換による削減より，置換情報によるオーバーヘッドが上回ってしまったため，元よりもサイズが大きくなった．

表 2 バイナリサイズ

バイナリ	サイズ (byte)	手法適用 (byte)	置換回数	比率
A1	3,176	3,167	5	99.72%
A2	3,168	3,159	5	99.72%
B1	2,060	2,067	1	100.34%
B2	4,140	4,147	1	100.17%
C1	513,465	512,708	96	99.85%
C2	513,638	512,881	96	99.85%
C3	514,545	513,788	96	99.85%

表 3 パッチファイルサイズ

パッチファイル	bsdiff (byte)	提案手法 (byte)	比率
A1 → A2	559	563	100.72%
B1 → B2	1,577	1,560	98.92%
C1 → C2	6,763	6,776	100.19%
C1 → C3	25,386	25,413	100.11%
C2 → C3	25,431	25,433	100.01%

ⓒ2016 Information Processing Society of Japan

Vol.2016-DPS-167 No.13
Vol.2016-MBL-79 No.13
Vol.2016-ITS-65 No.13
2016/5/26

表 4　A の各ブロックサイズ

A1 → A2	bsdiff(byte)	提案手法(byte)	比率
命令ブロック	109	107	98.17%
差分ブロック	330	325	98.48%
追加ブロック	88	88	100.00%

表 5　B の各ブロックサイズ

B1 → B2	bsdiff(byte)	提案手法(byte)	比率
命令ブロック	256	249	97.27%
差分ブロック	507	478	94.28%
追加ブロック	782	790	101.02%

表 6　C1→C2 の各ブロックサイズ

C1 → C2	bsdiff(byte)	提案手法(byte)	比率
命令ブロック	320	317	99.06%
差分ブロック	6,067	6,038	99.52%
追加ブロック	344	378	109.88%

表 7　C1→C3 の各ブロックサイズ

C1 → C3	bsdiff(byte)	提案手法(byte)	比率
命令ブロック	1,307	1,331	101.84%
差分ブロック	20,994	20,976	99.91%
追加ブロック	3,053	3,063	100.33%

表 8　C2→C3 の各ブロックサイズ

C2 → C3	bsdiff(byte)	提案手法(byte)	比率
命令ブロック	1,173	1,169	99.66%
差分ブロック	21,491	21,486	99.98%
追加ブロック	2,735	2,735	100.00%

表 3 に bsdiff により生成されたパッチファイルのサイズを示す．パッチファイルでは，B だけ bsdiff のみの場合と比べ僅かであるがサイズが小さくなった．

それぞれで生成されたパッチファイルについて，各ブロックにおける影響を表 4 から表 8 に示す．A の場合と C2 から C3 の場合，削減量よりも置換情報によるオーバーヘッドが上回っているためパッチファイル全体のサイズが大きくなっている．C1 から C2 と，C1 から C3 の場合，圧縮の効きやすい差分ブロックが減っているが，圧縮の効きにくい他のブロックが増えているため全体のサイズが増大していると考えられる．

表 9　複数回の置換によるサイズ(A)

対象数	A1	A2	A1 → A2
0	3,176	3,168	559
1	3,076	3,068	564
2	3,007	2,999	581
3	2,939	2,915	618

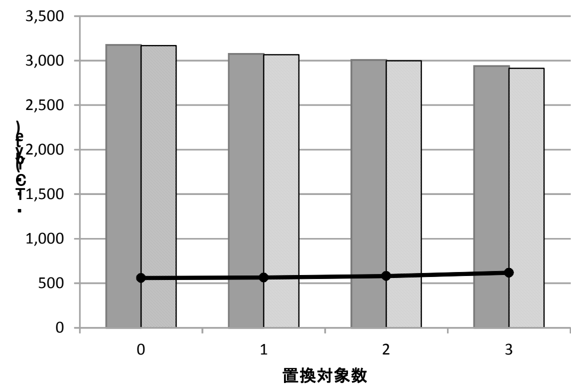


図 7　複数回の置換によるサイズ推移(A)

表 10　複数回の置換によるサイズ(B)

対象数	B1	B2	B1 → B2
0	2,060	4,140	1,577
1	1,992	3,896	1,545
2	1,964	3,868	1,489
3	1,916	3,820	1,503

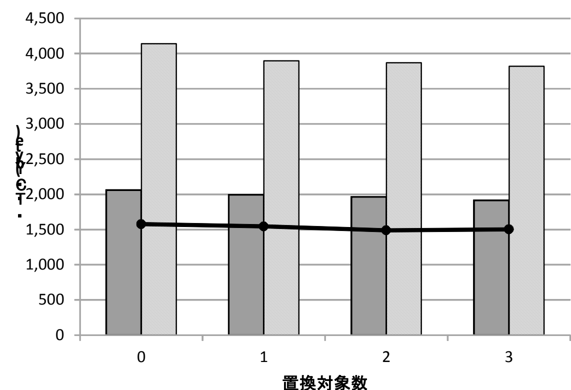


図 8　複数回の置換によるサイズ推移(B)

5.2 他パターンへの適用

置換対象としてサブルーチンプロローグについて述べたが，同様に繰り返し行われている処理をバイナリコード上から探索することでも提案手法の適用が可能である．バイナリ A，B に対してバイナリコード上から置換による削減

Ⓒ2016 Information Processing Society of Japan

量が最大となるパターンを探索し，提案手法を適用した．また，パターン生成，架空の命令の用意ができれば複数対象への置換も可能である．複数回置換を行った場合のバイナリサイズとパッチファイルのサイズを表 9，表 10 に，グラフを図 7，8 に示す．A，B 共に置換対象数を増やすごとにバイナリサイズは削減されている．しかし A のパッチファイルは，置換対象が増えるごとにサイズが大きくなっている．これは置換対象を増やすことで削減量は大きくなっているが，追加で記録する置換情報も共に大きくなっているためである．B のパッチファイルは 2 回まではサイズが小さくなっているが，3 回目で 2 回目のサイズを上回っている．以上のことから，置換によりある程度の削減の効果が得られている場合，複数回置換を行うことで効果を上げることができることがわかる．

A，B 程度のサイズのバイナリから全パターンを生成し，最適なパターンを選択することは可能であるが，C ほどのサイズのバイナリの場合，莫大な量のパターンが存在するため最適なパターンを探索することは難しい．そのため，繰り返し行われている処理がどのようなものかある程度の見当をつけた後に探索するなどの工夫が必要だと考えられる．

5.3 バイナリの分割

提案手法ではバイナリコード全体に対し差分符号化を行い，パッチファイルを生成している．しかしバイナリコード全体に対して圧縮を行った場合，すべてのデータを受信するまで展開を行うことができない．車載ネットワークは基本的にバス型トポロジであり, 同時に複数の ECU に送信することは出来ない．そのため受信，または展開をしていない ECU には待機時間が発生する．そこで，バイナリを分割し，図 9 のように部分ごとに送信を行い，展開中に別の待機中である ECU に送信することで, 待機時間を減らし効率よく更新を行うことが出来ると考えられる．

A，B においてバイナリを分割した場合の影響を調べた．セグメント毎にバイナリを分割し，それぞれのセグメントで差分符号化を行った．結果を図 10, 11 に示す．A，B 共に分割を行ったことにより，通常の差分符号化よりも必要なデータサイズが増えている．データ量増加の要因として，それぞれに bsdiff によるヘッダーが付加されたこと，分割したことによる圧縮率の低下が挙げられる．

よって，分割数が多くなるにつれ，オーバーヘッドが増加すると考えられる．また，A においては差分が存在しないセクションも確認された．差分が存在しない部分については送信する必要が無いため，データ量の削減が期待できる．提案手法を適用した後に各セクションで差分符号化をしたところ，それぞれのセクションで行われる置換は 1 回程度となり，効果は得られなかった．

6. まとめ

複数の命令からなる一連の処理を，一つの命令に置き換えることで，バイナリコードを小さくし，差分更新に必要なデータ量の削減を行う手法の提案と評価を行った．提案手法の評価の結果，僅かではあるが，効果が得られた．いかに圧縮効率の良い形で削減することができるかが課題となる．

今回の提案手法では，既に効果的な圧縮方法をさらに改良しようとしているので，更新時間短縮に対し十分な効果を得ることは難しいが，間欠的に転送更新を繰り返す処理

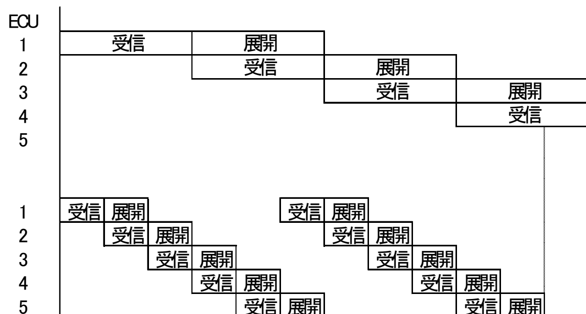


図 9　効率の良い転送

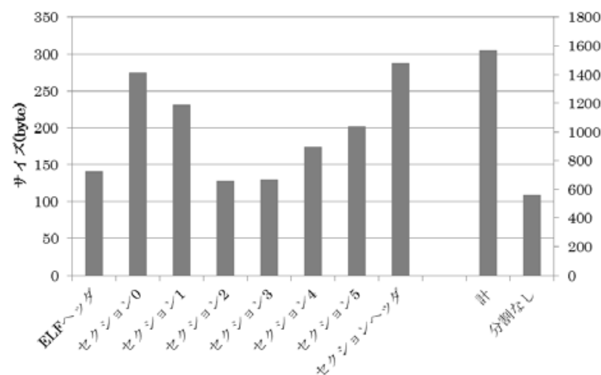


図 10　バイナリ分割による影響(A)

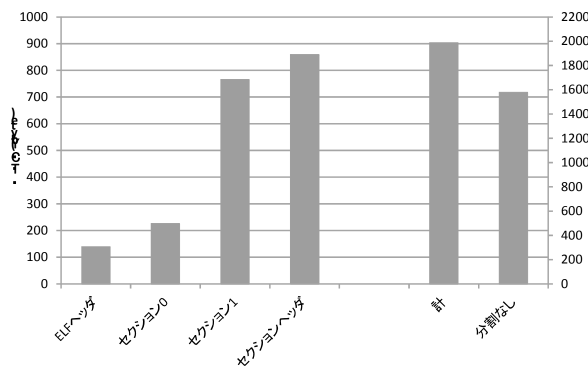


図 11　バイナリ分割による影響(B)

ⓒ2016 Information Processing Society of Japan

においてもパターンを辞書的に保持することにより，有効に扱える可能性はあると考えている．

今後は，手法の改善を行うと共に，効率的な更新を行うためのバイナリ分割方法や，転送制御方式の策定など，更新データ量削減以外からのアプローチも行う．

また，部品メーカごとに異なるであろうセキュリティ問題，即ち不正なソフトにおける更新や，情報の漏えいなどの点に関しても検討を行い，それらの機能との整合性をとっていく必要があると考えている．

参考文献

[1] 堀川健一，目崎元太，渡辺章代，加櫓武，松本達治：自動車ソフトウェアの標準仕様”AUTOSA”の評価，SEI テクニカルレビュー，175 号，pp.92－97（2009）.

[2]“Chrysler，車の遠隔操作問題で 140 万台のリコール発表,” http://www.itmedia.co.jp/enterprise/articles/1507/27/news038.html <accessed 2016/04/18>

[3] 佐藤道夫,”車載ネットワーク・システム徹底開設,” CQ 出版社，ISBN978-4-7898-3721-7

[4] Colin Percival, “Matching with Mismatches and Assorted Applications ,” doctoral thesis, Wadham College University of Oxford http://www.daemonology.net/papers/thesis.pdf <accessed 2016/04/18>

[5] 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）.

[6] The VCDIFF Generic Differencing and Compression Data Format，http://www.ietf.org/rfc/rfc3284.txt.

[7] xdelta，http://xdelta.org/.

[8] Tridgell, A.and Mackerras, P.: The rsync algorithm, https://rsync.samba.org/tech_report/.

[9] Ryozo Kiyohara, Satoshi Mii, Koichi Tanaka, Yoshiaki Terashima, and Hidetoshi Kambe、Study on Binary Code Synchronization in Consumer Devices、IEEE Transactions on Consumer Electronics, Vol.56, Issue 1,pp.254-260, 2010

[10] 野沢優尚，小沼寛，清原良三：車載 ECU 向け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），2015-DPS-165，1 ‐ 6（2015）.