Vol.2016-ARC-219 No.2
Vol.2016-SLDM-175 No.2
Vol.2016-EMB-40 No.2
2016/3/24

車載ネットワーク向けTCP/IPプロトコルスタックの開発

本多 一樹[1,a)]　松原 豊[1,b)]　高田 広章[1,c)]

概要：近年，自動車の機能の高度化・複雑化に伴い，車載ネットワークにEthernetを採用する検討が進んでいる．Ethernetが注目される理由として，車載ネットワークで伝送するデータ量の増加が挙げられる．従来の車載ネットワークは，データ量に応じた配線数の増加により，コストや設置空間，重量の点で不利となってきている．Ethernetは，従来の方式と比較して，同じデータ転送量の場合に配線数を大幅に削減することができる．
また，車載制御システムは，汎用的なシステムとは異なりCPU性能やメモリ容量などハードウェア資源に対する制約が厳しいため，メモリ不足を防ぐことが重要となる．しかし，Ethernetはハードウェア資源に余裕のあるシステムで利用されてきたため，既存のプロトコルスタックの多くが内部で動的メモリ管理を利用している．メモリ容量の限られた組込みシステムで動的メモリ管理を行った場合，動作中にメモリ不足になる可能性がある．このため，車載Ethernet向けに動的メモリ管理を削除したプロトコルスタックが必要になる．
本研究では，TOPPERSプロジェクトで開発されている，AUTOSAR OS仕様準拠の自動車制御用リアルタイムOSであるTOPPERS/ATK2[1]上に，動的メモリ管理を排除したプロトコルスタックを構築することを目指す．
まず，動的メモリ管理を用いない組込みシステム向けのTCP/IPプロトコルスタックであるTINET[2]を，TOPPERS/ATK2上に移植した．TINETはμITRON仕様のOS上で動作するプロトコルスタックであるため，AUTOSAR OSの機能を利用し，同等の動作を行うよう変更を加えた．この結果，AUTOSAR OS上に動的メモリ管理を削除したプロトコルスタックを実装することができた．

Development of TCP/IP protocol stack for automotive network

Honda Kazuki[1,a)]　Matsubara Yutaka[1,b)]　Takada Hiroaki[1,c)]

1. はじめに

近年，車載ネットワークにEthernetを採用する機運が高まっている．Ethernetが注目される理由としては，従来の方式と比べてケーブルのコストを大幅に削減できること，外部機器との連携の容易さが挙げられる．自動車の周辺監視用に多数のカメラを接続した場合，従来のディスプレイ/映像インターフェースであるLow Voltage Differential Signaling（LVDS）では，扱うデータ量の増加に応じてケーブル本数が増え，コストや設置空間，重量の面で不利となる．Ethernetは，同じデータ転送量の場合従来の方式と比較して配線数を大幅に削減することができ，ケーブルコストの削減だけでなく設置空間，重量の面でも有利になる．また，Ethernetはすでにオフィスや一般家庭に広く普及しているオープン規格のため対応製品が多く，外部機器との情報連携を行いやすい．これにより，Webを用いた各種サービスを車載システムに構築しやすくなる．

車載制御システムは，汎用的なシステムとは異なりCPU性能やメモリ容量などハードウェア資源に対する制約が厳しい．このため，通信に使用するプログラムやデータのサイズを小さく抑え，またメモリ不足を防ぐ必要がある．さらに，車載システムの不具合は人の命に直接関わる問題で

[1] 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University
[a)] honda-kazuki@ertl.jp
[b)] yutaka@ertl.jp
[c)] hiro@ertl.jp

あるため，車載システムを構築するネットワークには高い信頼性も求められる．しかし，Ethernet は PC 等ハードウェア資源に余裕のあるシステムで利用されてきたため，既存のプロトコルスタックの多くが内部で動的メモリ管理を利用しているという現状がある．メモリ容量の限られた組込みシステムで動的メモリ管理を行った場合，メモリ不足になる状況が避けられなくなる．

現在，車載ソフトウェアの標準規格 AUTOSAR の採用活動が活発化している．AUTOSAR の目的は，車載ソフトウェアをモジュール化し，再利用性を高めることで，開発規模の増大に対処しようというものであり，車載 Ethernet の実装に関する仕様も存在する．ところが，AUTOSAR の TCP/IP プロトコルスタックの仕様 [3] にも，通信に使用するリソースの獲得，開放を行う関数の仕様が存在し，動的メモリ管理を排除できていないという問題が存在する．このため，車載 Ethernet 向けに動的メモリ管理を排除したプロトコルスタックが必要とされている．

すでに，組込み向けの TCP/IP プロトコルスタックとして，TOPPERS プロジェクトから公開されている TINET が存在する．TINET は，標準的に利用されている BSD の TCP 制御アルゴリズムをベースとした TCP/IP プロトコルスタックで，動的メモリ管理の排除など組込み用途に合わせた設計がなされている．しかしながら，TINET は µITRON 仕様 OS 上で動作させることを想定しているため，AUTOSAR OS 仕様の OS で動作させるためには，OS 機能を利用する部分を修正する必要がある．

本研究では，TOPPERS プロジェクトで開発されている AUTOSAR OS 仕様準拠の自動車制御用リアルタイム OS である TOPPERS/ATK2（以下 ATK2）上に，動的メモリ管理を排除した TCP/IP プロトコルスタックを構築することを目指す．開発には，アルテラ社の FPGA ボードである DE2-115 を用いた．まず，開発するプロトコルスタックのベースとなる TINET には，使用される OS 機能に関する設計書が存在しないため，新規に作成する．次に，作成した設計書を元に，µITRON 仕様 OS の機能を，AUTOSAR OS の機能を用いて実装を行う．そして，実装した機能を利用し，TINET を ATK2 上に移植することで，AUTOSAR OS 上に動的メモリ管理を用いないプロトコルスタックを実現する．

2. TCP/IP プロトコルスタックの開発

本章では，本研究で行ったプロトコルスタックの開発について説明する．まず，TINET が利用するµITRON 仕様 OS の機能の調査と，ATK2 上への機能の実現方法について説明する．次に，開発したスタックの実装対象とした環境と，使用したツールについて説明する．最後に，テストの方法と性能評価方法について説明する．

表 1　SEM_TCP_POST_OUTPUT
Table 1　SEM_TCP_POST_OUTPUT

SEM_TCP_POST_OUTPUT	
利用目的	イベント通知
送信元	アプリケーションタスク，Ethernet 出力タスク，Ethernet 入力タスク，ネットワークタイマタスク，TCP 出力タスク，送信完了割込みハンドラ
受信先	TCP 出力タスク

表 2　SEM_TCP_CEP
Table 2　SEM_TCP_CEP

SEM_TCP_CEP	
利用目的	排他制御
排他対象	アプリケーションタスク，TCP 出力タスク

2.1 TINET の設計情報の調査

TINET は，µITRON 仕様の RTOS カーネルの機能を利用している．しかし，TINET 内で利用される OS 機能について記述した設計書は存在せず，AUTOSAR OS 仕様等，異なる仕様の RTOS で動作させるために必要となる OS 機能の置き換え方法を決める必要がある．このため，まず TINET のタスク構成とタスク間の関連について調査する．次に，利用される API について，調査したタスク間の関連をもとに利用目的を明確化する．

TINET において利用されるタスクを以下に示す．

- アプリケーションタスク
- Ethernet 出力タスク
- Ethernet 入力タスク
- ローカルループバックインタフェース・出力タスク
- ローカルループバックインタフェース・入力タスク
- ネットワークタイマタスク
- PPP 出力タスク
- PPP 入力タスク
- TCP 出力タスク
- UDP 出力タスク

タスク間の関連について，TCP 出力タスクを例にあげて述べる．TCP 出力タスクは，セマフォ SEM_TCP_POST_OUTPUT が返却されることで起床する．起床後，出力すべき TCP 通信端点が存在するか探索し，TCP 出力関数を呼び出して送信を行う．また，通信端点の探索は，セマフォ SEM_TCP_CEP を獲得することで排他制御を行う．TCP 出力タスクにおいて利用される OS オブジェクトを **表 1**，**表 2** にまとめる．

同様に，TINET において利用される 54 の OS オブジェクトについて，利用目的とオブジェクトに関連付けられたタスクを明確化した．

2.2 AUTOSAR OS 仕様上での実現方法

前節で調査した OS オブジェクトについて，それらを利用するための機能を ATK2 上で実現する方法について説明を行う．

2.2.1 セマフォ

µITRON 仕様におけるセマフォは，資源の数を表すカウンタ（資源数）を介し，排他制御やイベント通知を行うオブジェクトである．セマフォの資源数から 1 を減ずることを資源の獲得，資源数に 1 を加えることを資源の返却と呼ぶ．

各セマフォが持つ情報は次の通りである．

- セマフォ属性（待ち行列の順序）
- 資源数（の現在値）
- 待ち行列（セマフォの資源獲得待ち状態のタスクのキュー）
- 初期資源数
- 最大資源数
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

AUTOSAR OS 仕様において，排他制御とイベント通知の両方を行う機能は存在しない．このため，各機能の利用目的に応じ，異なる方法で ATK2 上に機能を実現させる．

2.2.1.1 排他制御

セマフォによる排他制御は，資源の獲得により資源数を 0 にし，他のタスクが資源を獲得できなくなることで排他区間を実現する．排他区間の終了時に資源の返却を行い，資源獲得待ち状態のタスクは待ち状態を解除され，資源の獲得が可能になる．単一のタスクのみが排他区間に入ることができるようにするため，初期，最大資源数は 1 として宣言される．

AUTOSAR OS 仕様において，排他制御を行うための OS オブジェクトはリソースである．リソースを用いた排他区間の開始をリソースの獲得と呼び，排他区間の終了をリソースの解放と呼ぶ．タスクがリソースを獲得した時，タスクの優先度が上限優先度に引き上げられることで，複数のタスクが同時に排他区間に入らないようにする．

2.2.1.2 イベント通知

セマフォによるイベント通知は，資源の返却により資源数を増やし，資源獲得待ち状態となっていた通知先のタスクが起床されることで実現される．イベント通知が行われるまで通知先のタスクを待ち状態とするため，初期資源数は 0 として宣言される．

AUTOSAR OS 仕様において，イベント通知を行うための OS オブジェクトはイベントである．イベント通知はイベントのセットと呼び，イベントの待ちが解除されたタスクは通知済みのイベントをクリアする必要がある．イベントのセットは休止状態ではない拡張タスクに対して行うことができる．イベントがセットされると，イベント待ちタスクは実行可能状態に遷移する．

2.2.2 データキュー

µITRON 仕様におけるデータキューは，1 ワードのデータをメッセージとして，FIFO 順で送受信を行うオブジェクトである．より大きいサイズのメッセージを送受信するには，メッセージを置いたメモリ領域へのポインタを 1 ワードのデータとして送受信する．

各データキューが持つ情報は次の通りである．

- データキュー属性
- データキュー管理領域
- 送信待ち行列（データキューへの送信待ち状態のタスクのキュー）
- 受信待ち行列（データキューからの受信待ち状態のタスクのキュー）
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

データキュー管理領域は，データキューに送信されたデータを，送信された順に格納しておくためのメモリ領域である．

AUTOSAR OS 仕様において，データキューに対応する機能は存在しない．このため，µITRON 仕様の RTOS である TOPPERS/ASP カーネル（ASP カーネル）[4] の実装をもとに，ATK2 上に機能を実現させる．データキューが利用する管理領域等のオブジェクトの静的な生成は，コンフィギュレータが宣言を読み取ることで行われる．ただし，今回開発したスタックにおいてはコンフィギュレータは用いず，直接オブジェクトを記述した．

2.2.3 固定長メモリプール

µITRON 仕様における固定長メモリプールは，生成時に決めたサイズのメモリブロック（固定長メモリブロック）を動的に獲得・返却するためのオブジェクトである．

各固定長メモリプールが持つ情報は次の通りである．

- 固定長メモリプール属性
- 待ち行列（固定長メモリブロックの獲得待ち状態のタスクのキュー）
- 固定長メモリプール領域
- 固定長メモリプール管理領域
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

固定長メモリプール領域は，固定長メモリブロックを割り付けるためのメモリ領域である．固定長メモリプール管理領域は，固定長メモリプール領域中の割当て済みの固定長メモリブロックと未割当てのメモリ領域に関する情報を格納するためのメモリ領域である．

AUTOSAR OS 仕様において，固定長メモリプールに対

Vol.2016-ARC-219 No.2
Vol.2016-SLDM-175 No.2
Vol.2016-EMB-40 No.2
2016/3/24

表 3　OS 機能の設計方針
Table 3　Design policy of OS function

OS 機能	μITRON	AUTOSAR
排他制御	セマフォ	リソース
イベント通知	セマフォ	イベント
タスク間通信	データキュー	ミドルウェアとして独自に実装
メモリ管理	固定長メモリプール	ミドルウェアとして独自に実装
カーネルオブジェクトのコンフィギュレーション	コンフィギュレータ	コンフィギュレータ（未対応）
API のイベント待ち	する	しない（ポーリング）

表 4　実装環境
Table 4　Implementation environment

ハードウェア名	情報
FPGA	Cyclone IV EP4CE115F29C7
SDRAM	128MByte
SRAM	2MByte
Flash	8MByte
Ethernet	2 つ (10/100/1000 Mbps)

応する機能は存在しない．このため，ASP カーネルの実装をもとに，ATK2 上に機能を実現させる．固定長メモリプールが利用する固定長メモリプール領域等のオブジェクトの静的な生成は，コンフィギュレータが宣言を読み取ることで行われるが，データキューと同様，直接オブジェクトを記述した．

OS オブジェクトの各機能の実現方法を **表 3** にまとめる．

2.3　ターゲット環境

本節では，開発したプロトコルスタックを実装するにあたってターゲット環境としたボード，OS について述べる．

2.3.1　ボード

本研究では，プロトコルスタックの実装を行うボードとして DE2-115 を採用した．DE2-115 は，Cyclone IV EP4CE115F29C7 FPGA を搭載するアルテラ社の開発・学習ボードである．Cyclone IV は組み込み用の RISC プロセッサ・コアである Nios II，およびメモリや Ethernet といった機能 IP を組み合わせることで，回路構成を変更できる FPGA である．DE2-115 ボード上には，Ethernet ポートが搭載されており，Cyclone IV と組み合わせることで Ethernet 通信が可能となる．**表 4** に DE2-115 の情報をまとめる．

2.3.2　OS

本研究では TOPPERS/ATK2-SC1 Release 1.3.2（AUTOSAR Release4.0 Revision3 準拠）を使用した．

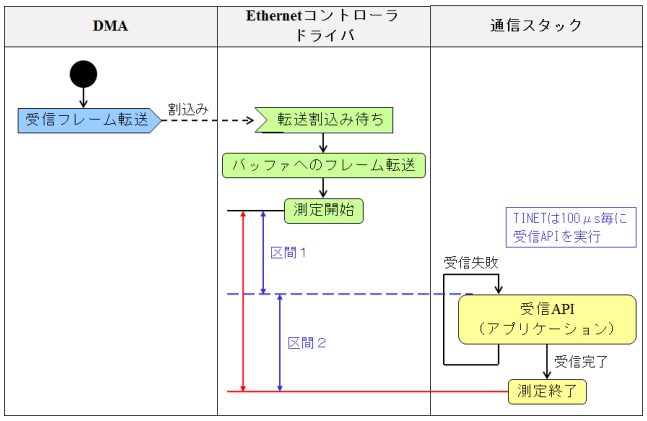

図 1　測定範囲
Fig. 1　Range of measurement

3.　評価

本章では，ATK2 上へ実装したプロトコルスタックの性能の比較と，その結果について述べる．評価としては，Ethernet フレームの転送完了時から，受信 API が完了するまでの平均実行時間，最悪実行時間の測定，およびその標準偏差の評価を行う．

3.1　実行時間の計測

プロトコルスタックの性能を評価するため，DMA による Ethernet ドライバのバッファへの受信フレーム転送完了割込みから，スタックの受信 API が実行完了するまでの時間を測定する．**図 1** に，時間を測定した範囲を示す．計測に使用した Ethernet フレームは UDP/IP パケットで，計測回数はそれぞれ 10000 回である．ATK2 上で，ローエンドのマイコン向けに開発されたプロトコルスタックである uIP[5] を実行した場合を比較対象として比較を行う．実装するターゲットのハードウェアタイマの制限により，開発したスタックの受信 API の呼び出し間隔は 100μs として計測を行う．時間の計測には，リアルタイム性能評価のために用いられる ATK2 の実行時間分布集計モジュールを利用する．実行時間はマイクロ秒単位で記録される．実行時間分布集計モジュールの計測オーバーヘッドは，4μs である．

3.2　実行時間の比較

開発したプロトコルスタック，uIP それぞれの受信処理の平均実行時間，最悪実行時間，標準偏差の比較を行った．各スタックの実行時間の測定結果を **図 2**，**図 3** に，比較結果を **表 5** に示す．図の横軸は実行時間 [μs] であり，縦軸は実行時間で受信処理が完了した回数である．

表 5 に示す結果から，開発したスタックの実行時間および標準偏差は uIP と比較して大きかった．しかし，平均実行時間と最悪実行時間の比は，開発したスタックがより小

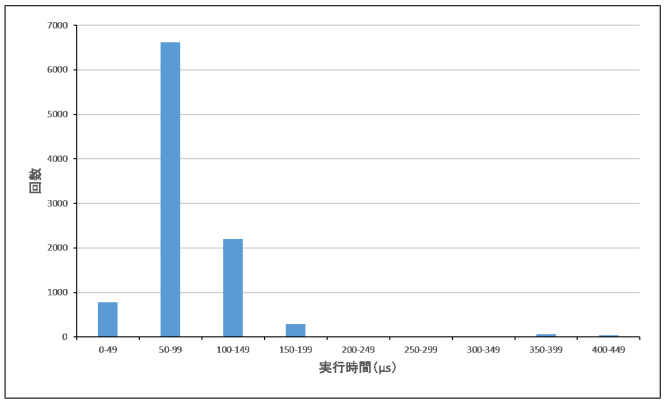

図 2　uIP の実行時間
Fig. 2　Execution time of uIP

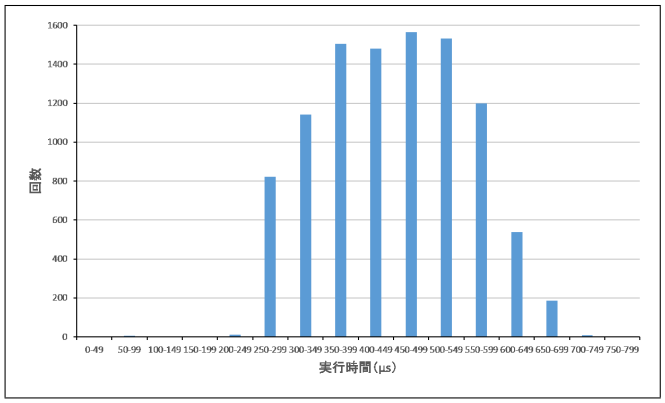

図 3　開発したプロトコルスタックの実行時間
Fig. 3　Execution time of developed protocol stack

表 5　実装方法ごとの比較結果
Table 5　Comparison result of each implementation method

スタック	平均実行時間 [μs]	最悪実行時間 [μs]	標準偏差
uIP	81	434	43
開発成果物	449	773	104

さかった．開発したスタックの実行時間，および標準偏差が増加した原因として，次に挙げる 3 つが考えられる．

- タスクの切り替えによるオーバーヘッド
- 100μs の API 呼び出し周期
- バッファのコピー回数

これらの点について調査するため，DMA による Ethernet ドライバのバッファへの受信フレーム転送完了割込みから，受信 API 開始までの時間（図 1 の区間 1），および受信 API 開始からスタックの受信 API が実行完了するまでの時間（図 1 の区間 2）を測定する．区間 1 での各スタックの実行時間の測定結果を **図 4**，**図 5** に，比較結果を **表 6** に示す．また，区間 2 での測定結果を **図 6**，**図 7** に，比較結果を **表 7** に示す．

表 6，表 7 に示す結果より，開発したスタックの実行時間の分散のほとんどは API 開始前で発生している．実行時間の標準偏差が大きくなる原因として，API の呼び出し間隔が 100μs であること，および DMA 転送完了割込みの

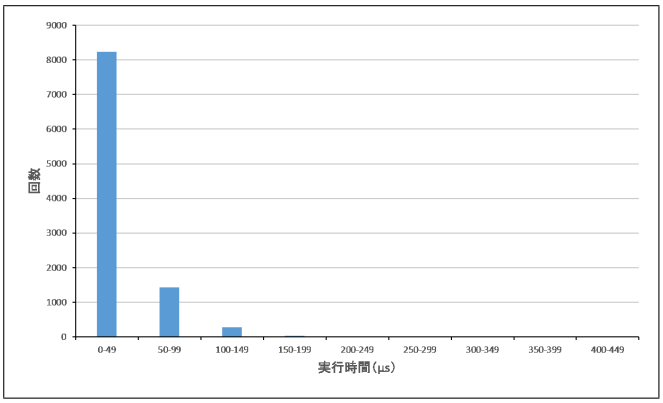

図 4　uIP の実行時間（区間 1）
Fig. 4　Execution time of uIP (Section 1)

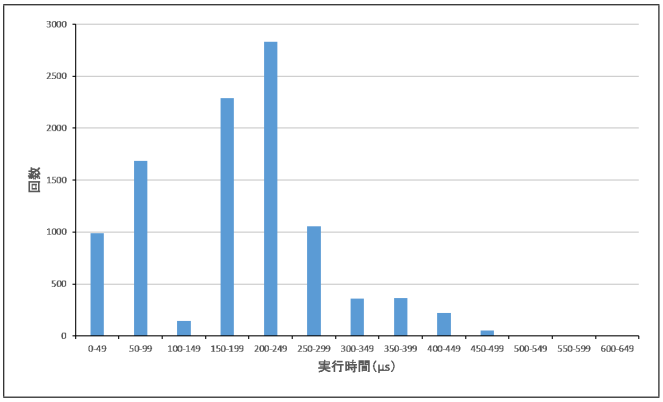

図 5　開発したプロトコルスタックの実行時間（区間 1）
Fig. 5　Execution time of developed protocol stack (Section 1)

表 6　実装方法ごとの比較結果（区間 1）
Table 6　Comparison result of each implementation method (Section 1)

スタック	平均実行時間 [μs]	最悪実行時間 [μs]	標準偏差
uIP	38	422	26
開発成果物	185	620	98

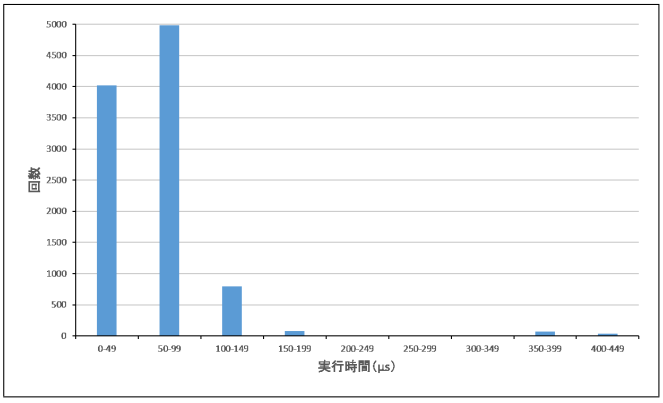

図 6　uIP の実行時間（区間 2）
Fig. 6　Execution time of uIP (Section 2)

発生から API 開始（アプリケーションタスクの起床）までに，ネットワークタイマタスク等の TINET の他のタスクが，タスクの起床を妨げていることが考えられる．このた

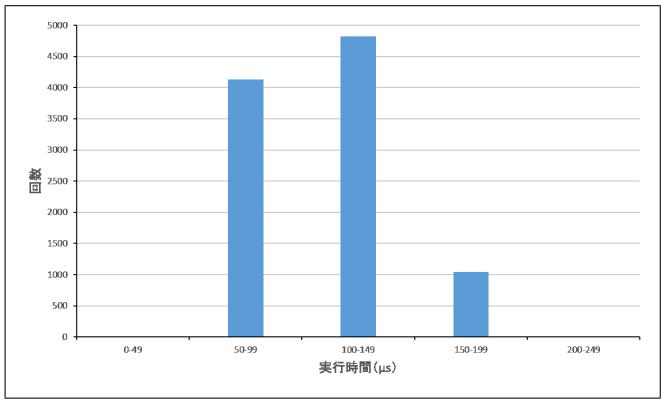

図 7　開発したプロトコルスタックの実行時間（区間 2）
Fig. 7　Execution time of developed protocol stack (Section 2)

表 7　実装方法ごとの比較結果（区間 2）
Table 7　Comparison result of each implementation method (Section 2)

スタック	平均実行時間 [μs]	最悪実行時間 [μs]	標準偏差
uIP	62	405	32
開発成果物	108	160	30

め，割込みの発生時にアプリケーションタスクを起床する仕組みを追加することで，リアルタイム性が向上すると考えられる．また，もう 1 つの要因にスタックとアプリケーションのバッファを共有していないことが挙げられる．このため，バッファのコピー回数が uIP と比べ 1 回多くなっている．スタックのバッファをアプリケーション側から利用可能とするように修正を加えることで，同様にリアルタイム性が向上すると考えられる．

uIP の平均実行時間と最悪実行時間の比が大きい原因としては，受信確認とタイムアウト確認を交互に毎回行うループ処理により，タイムアウト処理を行う場合と行わない場合での実行時間の差が大きくなったためと考えられる．タイムアウト処理を行わない場合，ほとんどオーバーヘッドなしに受信 API を呼び出すことができ，実行時間が非常に短くなるが，タイムアウト処理を行った場合との差が大きくなる．

4. おわりに

4.1 まとめ

本研究では，自動車制御用リアルタイム OS である TOPPERS/ATK2 上に，動的メモリ管理を削除した TCP/IP プロトコルスタックの構築を行った．動的メモリ管理の削除は，TINET をベースとして，μITRON 仕様の機能を AUTOSAR OS 上に実装することで実現した．

評価として，受信 DMA の完了割込みに対する受信 API の実行完了までの平均実行時間，最悪実行時間の測定，標準偏差の計算を行った．その結果，既存の uIP に対する利点も確認できた．しかし，DMA 転送完了割込み発生時のアプリケーションタスクの起床，およびアプリケーションとスタックのバッファ共有等，リアルタイム性をより高めるために変更すべき点があることも確認できた．

4.2 今後の課題

本研究で開発したプロトコルスタックは，TINET の機能の一部を AUTOSAR OS 上に実装したのみであり，バッファのコピー回数が多いこと，AUTOSAR OS の機能に最適化あるなど，不完全な部分が存在する．スタックのバッファとアプリケーションのバッファの共有，および受信割込みのアプリケーションタスクへの通知を追加することによる，よりリアルタイム性の高い実装を目指すことが課題といえる．

また，TINET で使用する静的 OS オブジェクトファイルの記述では，コンフィギュレータを用いない実装としたが，通信端点の数や TCP，UDP を切り替えた際の負担が増えるという問題がある．本来，実装するアプリケーションにより利用量が変動する要素を静的に定義するため，TINET ではコンフィギュレータを使用する．TINET もカーネルとは別に独自のコンフィギュレータをを持ち，静的なメモリ管理の簡易な利用を実現している．プロトコルスタックを使用する際，より使用者に扱いやすい実装のためにも，コンフィギュレータの実装は必要不可欠である．

参考文献

[1] TOPPERS: TOPPERS プロジェクト／ATK2, TOPPERS Project（オンライン），入手先 ⟨https://www.toppers.jp/atk2.html⟩（参照 2016-02-15）.
[2] TOPPERS: TOPPERS プロジェクト／TINET, TOPPERS Project（オンライン），入手先 ⟨https://www.toppers.jp/tinet.html⟩（参照 2016-02-15）.
[3] AUTOSAR: Specification of TCP/IP Stack, AUTOSAR (online), available from ⟨http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_TcpIp.pdf⟩ (accessed 2016-02-15).
[4] TOPPERS: TOPPERS プロジェクト／ASP カーネル，TOPPERS Project（オンライン），入手先 ⟨https://www.toppers.jp/asp-kernel.html⟩（参照 2016-02-15）.
[5] Dunkels, A.: adamdunkels/uip: The historical uIP sources, Swedish Institute of Computer Science (online), available from ⟨https://github.com/adamdunkels/uip⟩ (accessed 2016-02-15).