Vol.2012-EMB-26 No.1
2012/9/10

組込み向けTCP/IPプロトコルスタックのコンポーネント設計

原　拓[1,a)]　石川　拓也[1]　大山　博司[2]　高田　広章[1]

概要：本研究では，組込み向け TCP/IP プロトコルスタックの UDP 機能を，TECS を用いてコンポーネント化する．TECS は組込み向けコンポーネントシステムであり，小さいオーバヘッドでコンポーネント化できる．コンポーネント化により，拡張性や変更容易性といったコンフィギュラビリティのあるプロトコルスタックを実現することを本研究の目的としている．開発したプロトコルスタックを既存のものと比較した結果，小さいオーバヘッドでコンポーネント化ができ，コンフィギュラビリティが向上していることを確認できた．

キーワード：TECS，TCP/IP プロトコルスタック，組込みシステム，コンポーネントシステム

Component base development of TCP/IP protocol stack for embedded systems

Abstract: We developed a component based UDP function for the TCP/IP protocol stack that can be used in embedded systems inside the TECS framework. By doing this, the factored protocol stack will get more configurability, such as extendability and changeability. TECS is a component system for embedded systems, thus the component based UDP function can be implemented with low overhead inside the TECS framework. We evaluated the factored protocol stack by comparing it with the original protocol stack, we confirmed that the factored protocol stack is implemented with low overhead and the configurability is obtained.

Keywords: TECS，TCP/IP protocol stack，embedded systems，component system

1. はじめに

TCP/IP プロトコルスタックは, 組込みシステムの多様化に伴い多くの製品で必要とされている．TINET（Tomakomai InterNETworking）[1] は TOPPERS プロジェクト [2] において開発された TCP/IP プロトコルスタックである．TINET はリアルタイム OS 上でミドルウェアとして動作し，内部でのバッファコピー回数を最小にしている，動的メモリ管理を排除しているなど，組込み向けの特性を備えたプロトコルスタックである．

組込みシステムはリソース制約が厳しく，ターゲットの目的に合わせて TCP/IP プロトコルスタックの構造を変えなければならない場合がある．しかし TINET のソースコードは複雑に多くのファイルやマクロにより構成されており，IPv4 と IPv6 を共存させる，ネットワークバッファの数や個数を変更するといった，機能拡張，内部構造変更には詳細なコード解析が必要であり，容易ではない．

機能拡張や内部構造変更を容易化するための手法として，コンポーネントシステムを用いてソフトウェアを開発する手法がある．コンポーネントシステムとは，ソフトウェアの部品（コンポーネント）の集合によりソフトウェアを構築する開発環境であり，コンポーネント図によるソフトウェアの可視化，コンポーネントの交換によるソフトウェア構造の変更容易化といった，ソフトウェアのコンフィギュラビリティの向上が期待される．

組込み向けのコンポーネントシステムとして，TECS（TOPPERS Embedded Component System）がある [3][4]．汎用向けコンポーネントシステムが動的なコンポーネント

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University
[2] 株式会社オークマ
OKUMA Corporation
[a)] hrtk53@ertl.jp

の生成や結合をサポートしているのに対し，TECS は静的なコンポーネントの生成や結合を採用している．これによりコンポーネントの生成や結合を実現するためのモジュールをターゲットシステムに含める必要がなく，さらに，結合の最適化も行うことができるため，実行時のオーバヘッドを小さく抑えることができる [5].

本研究では，TINET の UDP 機能を TECS でコンポーネント化する．TECS を用いて TINET をコンポーネント化することで，コンポーネント化によるオーバヘッドを抑えながら，コンフィギュラブルな TCP/IP プロトコルスタックを実現する．また，コンポーネント化による影響を示すため，実行時間とメモリ使用量のオーバヘッド，およびコンフィギュラビリティを評価する．

本論文の構成は次のとおりである．まず 2 章で，背景技術となる TECS について述べる．3 章で既存の TINET について述べ，その問題点を述べる．4 章で TINET のコンポーネント設計について要件を述べ，その要件を満たすコンポーネント設計について述べる．5 章でコンポーネント化による影響を評価する．6 章で関連研究を述べる．7 章で本論文をまとめる．

2. TECS

本章では，本研究に用いるコンポーネントシステムである TECS について解説する．TECS は TOPPERS プロジェクトにより開発された，組込み向けのコンポーネントシステムである．TECS では静的なコンポーネントの生成や結合を採用しており，実行時のオーバヘッドを抑えることができる．また，組込みシステム開発において広く用いられる C 言語によるコンポーネント開発を採用しており，組込み向けに特化したコンポーネントシステムである．

2.1 コンポーネントモデル

図 1 に TECS のコンポーネント図の例を示す．TECS では，インスタンス化されたコンポーネントをセルと呼ぶ．セルは，自身が提供する機能のインタフェースである受け口，他のセルの機能を利用するためのインタフェースである呼び口，セルの情報を表す属性，セルの内部状態を表す変数で構成される．受け口は，セルの機能を提供する関数の集合であり，呼び口は，他のセルの利用可能な関数の集合である．1 つのセルは，複数の呼び口や受け口を持つことができる．

受け口と呼び口の型は，シグニチャと呼ばれる関数ヘッダの集合で定義される．セルの呼び口は，同一のシグニチャを持つ他のセルの受け口と結合できる．これにより，前者のセルから後者のセルの関数群を呼び出すことが可能になる．図 1 はセル UDPOutput の呼び口 cIPv4Output とセル IPv4Output の受け口 eIPv4Output が結合していることを表している．

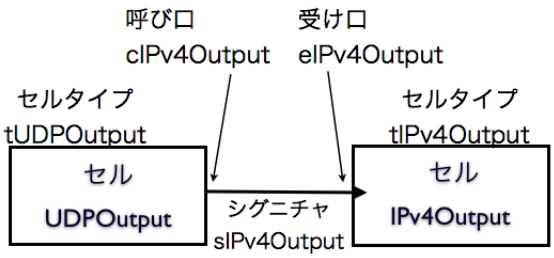

図 1 コンポーネントモデル
Fig. 1 Component model

```
1 signature sIPv4Output{
2   ER IPv4Output([in,size_is(len)]const int8_t *addr,[in]
        int32_t len,...);
3   ER getFlag([inout]uint32_t *flag);
4 };
```

図 2 シグニチャ記述
Fig. 2 Signature defenition

セルの型，つまり，セルが持つ受け口，呼び口，属性，変数の組を，セルタイプと呼ぶ．1 つのセルタイプから，複数のセルをインスタンス化することができる．

2.2 コンポーネント記述

コンポーネントの各要素は，TECS 独自のコンポーネント記述を用いて表される．コンポーネント記述は，シグニチャ記述，セルタイプ記述，組上げ記述に分類される．図 1 を例として，以下でそれぞれのコンポーネント記述を説明する．

シグニチャ記述

シグニチャ記述は，シグニチャを定義するために用いられる．signature キーワードに続けて，シグニチャ名を記述し，そのシグニチャが持つ関数ヘッダを中括弧内に列挙する．図 2 に示すシグニチャ記述の例では，関数 IPv4Output と getFlag を持つシグニチャ sIPv4Output を定義している．

TECS では，引数の入出力を明確にするために，図 2 の 2 行目の [in] に見られるような，指定子を用いる．指定子の例として，in，out，send，receive，inout，size_is などがある．in と send が入力，out と receive が出力，inout が入出力，size_is(len) が大きさ len の配列であることをそれぞれ示す．in と send の違い，および out と receive の違いについては 4.4 節で解説する．

セルタイプ記述

セルタイプ記述は，呼び口，受け口，属性，変数の集合を 1 つのセルタイプとして定義するために用いられる．celltype キーワードに続けてセルタイプ名を記述し，中括弧内にセルタイプが持つ要素を列挙する．呼び口は call キーワード，受け口は entry キーワード，属性は attr キーワー

```
1  celltype tIPv4Output{
2    /* 呼び口の宣言 */
3    call sEthernetOutput cEthernetOutput;
4    /* 受け口の宣言 */
5    entry sIPv4Output eIPv4Output;
6
7    attr{ /* 属性 */
8      uint16_t fragInit;
9    };
10   var{ /* 変数 */
11     uint16_t fragId = fragInit;
12   };
13 };
```

図 3 セルタイプ記述
Fig. 3 Celltype defenition

```
1  cell tIPv4Output IPv4Output{
2    /* 属性の設定 */
3    fragInit = 0;
4  };
5
6  cell tUDPOutput UDPOutput{
7    cIPv4Output = IPv4Output.eIPv4Output;
8  };
```

図 4 組上げ記述
Fig. 4 Cell defenition

ド，変数は var キーワードを用いてそれぞれ列挙する．

図 3 に示すセルタイプ記述の例では，シグニチャが sEthernetOutput の呼び口 cEthernetOutput と，シグニチャが sIPv4Output の受け口 eIPv4Output ，属性 fragInit，変数 fragId を持つセルタイプ tIPv4Output を定義している．

特別なセルタイプ記述として，呼び口定義の直前に optional 指定子を記述することができる．optional 指定子は，対象の呼び口が未結合のままにしてもよいことを表すための指定子である．optinal 指定子が用いられていた場合，コンポーネントの C 言語ソース内部から呼び口が結合されているかどうかを確認する仕組みも提供されている．

組上げ記述

組上げ記述は，セルタイプをインスタンス化してセルを生成し，セル同士の結合関係を定義してアプリケーションを構築するために用いられる．cell キーワードに続けてセルタイプ名，セル名を記述し，中括弧内には，自身の呼び口と他のセルの受け口との結合，および属性の設定を列挙する．図 4 の例では，セルタイプ tUDPOutput のセル UDPOutput とセルタイプ tIPv4Output のセル IPv4Output がそれぞれインスタンス化され，IPv4Output の属性 fragInit が初期化されている．また，セル UDPOutput の呼び口 cIPv4Output は，セル IPv4Output の受け口 eIPv4Output と結合している．

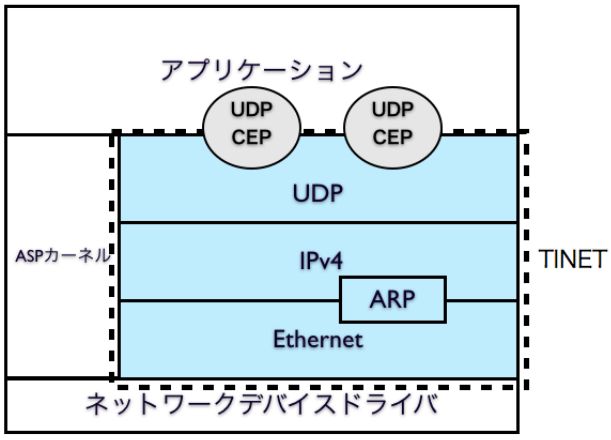

図 5 UDP の構成
Fig. 5 Structure of UDP protocol stack

3. TINET

本研究では，TINET の UDP 機能を TECS によりコンポーネント化する．本章では既存の TINET の概要と UDP プロトコルスタックについて解説する．

3.1 TINET 概要

TINET は TOPPERS プロジェクトにより開発された，組込み向けの TCP/IP プロトコルスタックである．TINET は，リアルタイム OS である TOPPERS/ASP カーネル上でミドルウェアとして動作する．また，最小コピー回数や動的メモリ管理の排除など，組込み向けの特性を備えている．

3.2 TINET の UDP 機能

TINET における UDP 機能を有するプロトコルスタックの構成を図 5 に示す．ユーザは，通信端点（以降 CEP）と呼ばれるソケットに似たインタフェースを用いて，UDP データの送受信を行う．送信処理では，CEP に渡されたデータ本体に各プロトコル層でヘッダが添付され，ネットワークデバイスドライバからデータが送信される．受信処理では，ネットワークデバイスドライバから受け取ったデータについて各プロトコル層でヘッダが解析され，データ本体が目的の CEP に渡される．

TINET におけるプロトコル層間のデータの移動は，コピー回数が最小となるように実現されている．汎用システム向けの TCP/IP プロトコルスタックでは，プロトコル層間で送受信データのコピーが行われるが，データのコピー処理は実行時間とメモリ使用量のオーバヘッドが大きい．そのため，TINET においてはデータのコピーを行わず，データが格納されているバッファのポインタのみをプロトコル層間で受け渡しすることで，コピー回数を最小化している．

3.3 TINET の問題点

TINET の問題点の 1 つ目は，多く複雑なソースファイルやマクロによる，可読性の低さである．そのため，IPv4 と IPv6 の共存やファイアウォールの実装など，TINET でサポートしていない機能拡張を行う場合は，詳細なコード解析が必要であり，容易ではない．

TINET の問題点の 2 つ目は，内部構造を決定するパラメータが容易に変更できないことである．例としてプロトコルスタックで用いるバッファを考える．組込みシステムでは，RAM サイズが小さく，RAM 使用量を抑える必要があるため，動作するアプリケーションに応じてバッファサイズを変更したいという要求が発生する．しかしバッファサイズの変更は容易には行えず，要求に合わないメモリ領域をバッファとして消費してしまう．

4. TINET のコンポーネント設計

本研究では 3.3 節で述べた問題点を解決するために，TINET をコンポーネント化する方法を採用した．本章ではまずコンポーネント化の要件について述べる．次にその要件を満たすように，コンポーネント化した UDP プロトコルスタックの全体像を述べる．最後にコンフィギュラビリティと最小コピー回数をどのように実現したのかを述べる．

4.1 コンポーネント化の要件

コンポーネント化した TINET には，1 つ目に本来の TINET の利点を失っていないこと，2 つ目にコンポーネント化によるメリットが引き出されていることが求められる．そこで TINET の UDP 機能コンポーネント化に当たり，以下の 4 つを要件とした．

(1) 組込み向け特性の継承
TINET にもともと備わっている，組込み向けの特性をコンポーネント化した後も維持できるように設計する．この組込み向けの特性とは，送受信データのコピー回数最小化や動的メモリ管理の排除があげられる．

(2) TINET の構造の継承
コンポーネント化により，著しくプロトコルスタックの挙動や使い方が変わることを防ぐ必要がある．そこで，内部で使われているタスクやリソース，TINET を操作するための API などの点を大きく変更しないようにコンポーネント化する．

(3) 拡張性の向上
コンポーネントの独立性を高め，どのコンポーネントがどの機能を実現しているかをわかりやすく設計し，可視性を向上させる．これにより，TINET でサポートしていないプロトコルの機能拡張が容易に行えるようにする．IPv4 と IPv6 の共存も，この要件に含まれる．

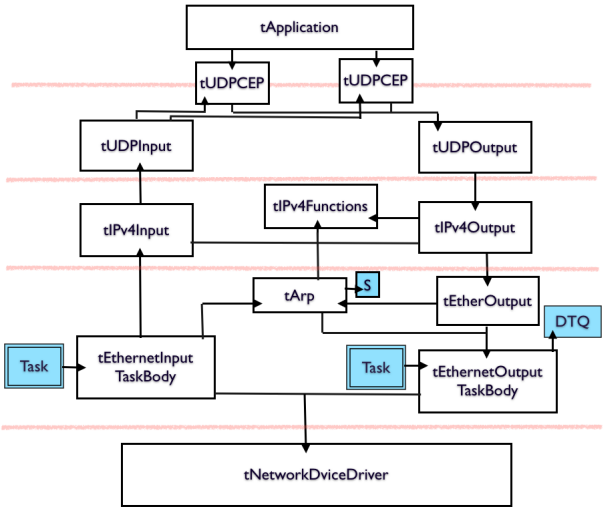

図 6 コンポーネント化した TINET
Fig. 6 Component model of TINET

(4) 内部構造変更の容易化
コンポーネントの生成や結合，属性などへの操作により，プロトコルスタックの構造を容易に変更できるようにする．ネットワークバッファのサイズ変更や，機能取外しなどの容易化も，この要件に含まれる．

4.2 全体像

コンポーネント化した UDP プロトコルスタックを図 6 に示す．明確な機能分割のために，プロトコル層ごとにコンポーネントを分割し，さらに，送信処理と受信処理を別のコンポーネントとする．また，特定の機能は 1 つのコンポーネントとして分離する．アプリケーションコンポーネントは，CEP をコンポーネント化した tUDPCEP コンポーネントに対して API を発行することでデータの送受信を行う．API や CEP の仕様に大きい変更はないため，アプリケーションは既存の TINET と同様にデータ送受信を行うことができる．

TINET 内部に存在する，タスクやセマフォ，データキューといったカーネルオブジェクトは TINET の動作に強く影響するため，既存の TINET と同じになるように設計する必要がある．先行研究 [7] により，TOPPERS/ASP カーネルのカーネルオブジェクトはコンポーネント化されているため，プロトコルスタック内部で用いる各カーネルオブジェクトをコンポーネントとして扱うことができる．図 6 中にある DTQ や S とあるコンポーネントは，それぞれデータキューとセマフォを表す．

動的メモリ管理の排除を行うために，プロトコルスタックで統一したメモリアロケータが必要である．プロトコルスタック全体で用いるアロケータを定義することで，どのコンポーネントでもメモリの確保と解放が行える．アロケータのコンポーネント図を図 7 に示す．このコンポーネントは，ネットワークバッファの移動が必要となるすべ

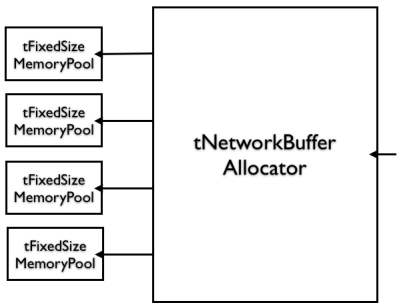

図 7 アロケータのコンポーネント図
Fig. 7 Component model of allocator

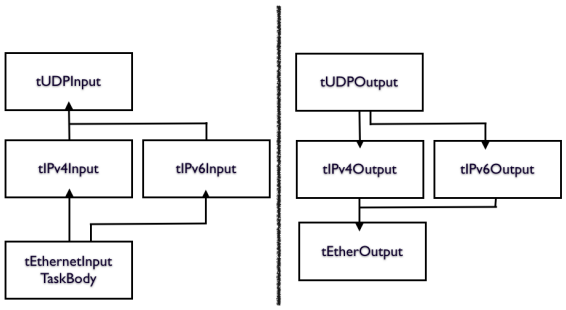

図 8 IPv6 拡張コンポーネント図の一部
Fig. 8 Part of component model with IPv6 protocol

てのコンポーネントと結合する．アロケータに用いられるカーネルオブジェクトであるメモリプールも，先行研究 [7] によりコンポーネント化されている．このアロケータには様々なサイズや数のメモリプールが接続されており，転送するデータサイズに合わせて用いるメモリプールを切り替える．

4.3 コンフィギュラビリティ

図 6 に示すように，コンポーネント化した UDP プロトコルスタックの内部処理は明確に分割されているので，各コンポーネントの独立性が高く，あるコンポーネントの処理が，他のコンポーネントの処理に影響することは少ないと考える．例えば図 6 の tIPv4Input や tEthernetInputTaskBody コンポーネントの処理に変更があっても，tUDPInput コンポーネントの処理には影響しない．これにより，新しいプロトコルに対応するなどの機能拡張が容易にできると考えられる．例えば新たに IPv6 の機能を追加したいと考えた場合，送受信データに対して IPv6 用の処理を行うコンポーネントを開発し，それらを UDP コンポーネントや Ethernet コンポーネントと接続するだけで拡張可能である．IPv6 機能を追加したプロトコルスタックのコンポーネント図の一部を図 8 に示す．

TECS では，2.2 節で述べたように，optional 指定子が提供されている．UDP 入出力，IPv4 入出力，ARP などのコンポーネントはこの指定子を用いて結合している．そのため，ターゲットに不必要な機能が存在した場合，組上げ記述において結合の記述を削除するだけで，プロトコルスタックから該当の機能を削除することができる．

また，内部構造を決定するパラメータは，コンポーネントの属性や変数を用いて定義する．バッファのサイズや数，ARP キャッシュ数，プロトコルスタックに割り当てる IPv4 アドレスなどがこのパラメータに含まれる．コンポーネント化された TINET では，組上げ記述への数行の変更で，これらの値を操作することができる．

コンフィギュラビリティについては 5.3 節の評価で詳細を述べる．

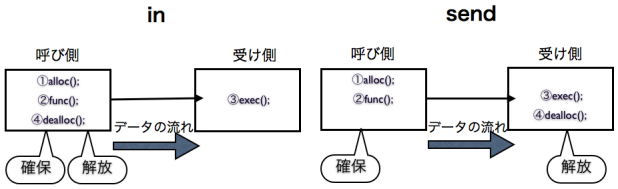

図 9 in と send の違い
Fig. 9 Difference between in and send

4.4 最小コピー回数

コンポーネント設計における最小コピー回数の実現方法を以下に述べる．

4.4.1 send，receive 指定子の利用

TECS はインタフェース指定子として send，receive を提供している [6]．send は in 指定子と同様に呼び側から受け側にデータを渡す際に使用される．in と send の違いを図 9 に示す．in の場合は引数で渡すメモリ領域の確保と解放のいずれも呼び側で行うのに対し，send の場合は解放を受け側で行う点が異なっている．receive の場合も同様に，受け側でメモリ確保を行う点が out と異なっている．

in，out 指定子を用いた場合，データの移動には双方のコンポーネントでメモリ領域を確保し，中身をコピーしなければならない．そこで send，receive 指定子を用いることにより，呼び側で確保したメモリ領域を受け側でそのまま利用できるため，コピー回数を抑えた実装が可能である．コンポーネント化した TINET おいても最小コピー回数の実装にするために，図 6 におけるプロトコル層をまたぐシグニチャについては，send，receive 指定子を用いている．

4.4.2 引数拡張

既存の TINET では，ネットワーク層以下のプロトコルは 1 種類に限定されており，コンパイル時に下位層のヘッダ長を決定できた．しかしコンポーネント化された TINET では，IPv4 と IPv6 の共存を考慮しているため従来の手法は使えない．最小コピー回数を実現するためには，下位層のプロトコルを知る必要がある．例えば，同じ tUDPInput コンポーネントに届いたデータでも，下位層で IPv4 パケットとして受信したのか IPv6 パケットとし

```
1  ER Output_eUDPOutput(...,uint32_t flag)
2  {
3    int offset = 0;
4    ...
5    if(flag & FLAG_USE_IPV4) /* IPv4 のビットが有効 */
6      offset += IPV4_HDR_SIZE;
7    if(flag & FLAG_USE_IPV6) /* IPv6 のビットが有効 */
8      offset += IPV6_HDR_SIZE;
9    if(flag & FLAG_USE_ETHER)/*ether のビットが有効
         */
10     offset += ETHER_HDR_SIZE;
11   ...
12 }
```

図 10 プロトコルフラグの利用例
Fig. 10 Use example of protocol flag

て受信したのかで，IP 層のヘッダ長が異なるため，バッファ中のどの位置から UDP ヘッダを読み込めばよいか異なる．また，tUDPOutput におけるデータの送信時にも，バッファのどの位置から UDP ヘッダを書き込み始めればよいか知る必要がある．

この問題への対応として，今まで通ってきた（送信の場合はこれから通る）プロトコルを管理するビットパターン（プロトコルフラグと呼ぶ）を引数に追加する方法を採用した．例えば IPv4 のパケットならプロトコルフラグの 9 ビット目が 1 になっており，IPv6 のパケットならプロトコルフラグの 10 ビット目が 1 になっている．データの送受信時に下位層のヘッダ長が必要な場合は，プロトコルフラグを参照することで下位層のプロトコルに対応したヘッダ長を決定することができる．例として tUDPOutput におけるプロトコルフラグの利用方法を図 10 に示す．図中では，引数として渡されたプロトコルフラグ flag の有効ビットを確認し，バッファ中における UDP ヘッダの書き込み位置 offset の値を決定している．

5. 性能評価

本章では，TINET をコンポーネント化したことによる影響を評価する．コンポーネント化による影響は，コンポーネント化によるオーバヘッド及びコンフィギュラビリティの向上を，既存の TINET と比較することで評価した．まず，コンポーネント化によるオーバヘッドを評価するために，データ送信に必要な実行サイクル数とメモリ使用量をそれぞれ比較した．次にコンフィギュラビリティの向上を評価するために，ICMPv4 機能の追加，および，ネットワークバッファのパラメータ変更を行った．本章では，既存の TINET のことを，単に TINET と呼び，コンポーネント化した TINET のことを，TINET+TECS と呼ぶ．

表 1 評価用ボード環境
Table 1 Evaluation board enviroment

ボード名	AP-SH4AD-0A
CPU	SH7786 533MHz
FlashROM	16MB
SDRAM	256MB
LAN コントローラ	LAN9221

表 2 実行時間計測結果
Table 2 Evaluation result of execution time

	平均実行サイクル	最悪実行サイクル
TINET	119	159
TINET+TECS	121	164

5.1 評価環境

評価用のボードとして，SH7786 搭載 CPU ボードである株式会社アルファプロジェクトの AP-SH4AD-0A を用いた．評価用ボードとホスト PC を LAN ケーブルで接続し，データ送受信を行う環境で評価を行った．評価用ボードの詳細な性能を表 1 に示す．コンパイラは sh-elf-gcc4.3.2，最適化オプションは O2 を用いた．また，メモリ使用量の評価については Cygwin の size コマンドを用いて計測した．

5.2 オーバヘッド評価

5.2.1 実行時間

実行時間のオーバヘッドを計測した．今回計測したのは，UDP パケットを送信するための API である，udp_snd_dat の発行から送信完了までにかかるサイクル数である．1 度に送信するデータは 8bit データを 300 個とした．これを 1 万回試行し，TINET と TINET+TECS で実行サイクルの変化を計測した．表 2 にその計測結果を示す．評価結果は平均実行サイクルが約 1.6%増，最悪実行サイクルが約 3.1%増であり，小さいオーバヘッドでコンポーネント化できているといえる．

コンポーネント化による実行サイクル数の増加分は，各種の値へのアクセスが容易でなくなったことが原因であると考えられる．TINET ではグローバル変数やマクロで定義していたような値を，TINET+TECS では，機能の明確な分割のために，コンポーネントの属性や変数として定義した．そのため TINET では変数を直接参照できていた箇所が，TINET+TECS では，その変数を保持しているコンポーネントに対して，変数にアクセスするための関数を呼び出さなければならないことがある．また，4.4.2 項でも述べたように，コンポーネント化された TINET ではプロトコルに拡張性を持たせるため，プロトコルフラグを用いて下位層のオフセットを取得している．これらの処理の実行時間が，オーバヘッドになっていると考えられる．

5.2.2 メモリ使用量評価

メモリ使用量のオーバヘッドを計測した．表 3 にその評

表 3　メモリ使用量計測結果
Table 3　Evaluation result of memory consumption

	テキスト (byte)	データ (byte)
TINET	51,336	10,786
TINET+TECS	51,852	10,926

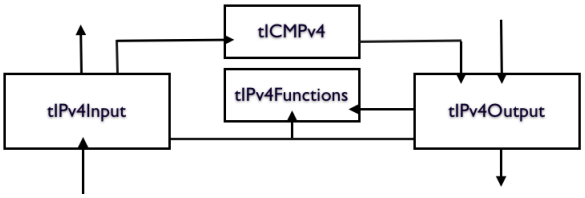

図 11　ICMPv4 拡張コンポーネント図の一部
Fig. 11　Part of component model with ICMPv4 protocol

価結果を示す．データ領域からは，カーネルオブジェクトであるメモリプールやタスクスタックに必要とされるメモリ使用量は除いてある．合計のメモリ使用量の増加分は1%以下であり，小さいオーバヘッドでコンポーネント化できているといえる．

テキスト領域の増加分は，コンポーネントの初期化などのコンポーネントシステム導入に必要な処理分が原因と考えられる．データ領域の増加分は，コンポーネントのコントロールブロックなどの生成が原因であると考えられる．

5.3　コンフィギュラビリティ評価

5.3.1　拡張性

拡張性の向上を評価するために，ICMPv4 の機能拡張を行った．拡張後のネットワーク層におけるコンポーネント図の一部を図 11 に示す．ICMPv4 は IPv4 の上に位置するプロトコルであるので，まず tICMPv4 コンポーネントを作成し，これを tIPv4Input から呼び出せるように接続した．この接続には optional 指定子を用いており，取り外し可能である．tIPv4Input では，受信した IPv4 パケットが ICMP のものであれば tICMPv4 へデータを渡す処理を追加した．tICMPv4 では ICMP パケットを返信する処理が必要な場合があるので，tIPv4Output コンポーネントに返信用の機能を実装した受け口を追加し，tICMPv4 とそれを接続した．ソフトウェアの可視性があがったことと，機能がコンポーネントで明確に分割され，それぞれが独立していることにより，このように容易に機能拡張を行うことができた．

5.3.2　変更容易性

変更容易性評価のために，ICMPv4 機能の取り外しとネットワークバッファのパラメータ変更を行った．

まず ICMPv4 の取り外しを行った．図 12 はコンポーネント記述ファイル内の変更が必要な記述部分である．ICMPv4 機能を無効にするには，1〜3 行目の ICMPv4 コンポーネントの組上げ記述と，7 行目に見られる IPv4Input の呼び口と ICMPv4 の受け口の接続記述を削除すればよい．

```
1  cell tICMP4 ICMP4{
2      cIPv4Reply = IPv4Output.eReply;
3  };
4
5  cell tIPv4Input IPv4Input{
6      cFunctions = IPv4Functions.eFunctions;
7      cICMP4 = ICMP4.eICMP4;
8      cUDPInput = UDPInput.eInput;
9  };
```

図 12　ICMPv4 取外しに必要な記述部分
Fig. 12　Part of defenition to remove ICMPv4

```
1  cell tFixedSizeMemoryPool NetworkBuffer_000 {
2      blockSize = 1520;
3      blockCount = 5;
4  };
5  cell tFixedSizeMemoryPool NetworkBuffer_001 {
6      blockSize = 1030;
7      blockCount = 1;
8  };
9
10   /*〜〜 中略　〜〜*/
11
12 cell tFixedSizeMemoryPool NetworkBuffer_005 {
13     blockSize = 70;
14     blockCount = 2;
15 };
```

図 13　バッファの変更に必要な記述部分
Fig. 13　Part of defenition to change buffer

表 4　メモリ使用量の変化
Table 4　Change of memory consumption

	テキスト (byte)	カーネルオブジェクト (byte)
変更前	53,024	13,824
変更後	51,852	7,744

次にネットワークバッファのパラメータを変更した．図 13 が変更が必要な記述部分である．4.2 節で述べたように，TINET には複数のサイズや数のメモリプールが，ネットワークバッファとして提供されている．例えば図 13 の 2，3 行目に記されるように，サイズが 1,520 バイトのバッファの数は 5 つある．この数を 1 つまで減らしたい場合，3 行目の値を 1 に書き換えるだけで変更が可能である．

上記の 2 点を行った場合のメモリ使用量の変化を表 4 に表す．カーネルオブジェクトは，利用するタスクスタックやメモリプールのために必要なメモリ使用量を表す．ソースコードにはまったく手を加えず，わずかな変更でメモリ使用量を合計で 7,252 バイト減少させることができた．

5.4 考察

TINET+TECS が，4.1 節で述べた要件を満たせているかどうか考察する．4.2 節や 4.4 節で述べた設計方法や，5.2 節の評価結果から，TINET の組込み向けの特性は継承できており，要件（1）を満たせている．4.2 節で述べた設計方法から，TINET の内部構造や操作方法に大きい変更はないため，要件（2）を満たせている．4.3 節で述べた内容や 5.3.1 項の評価結果から，コンポーネント化されたプロトコルスタックには拡張性があり，要件（3）を満たせている．4.3 節で述べた内容や 5.3.2 項の評価結果から，コンポーネント化されたプロトコルスタックには変更容易性があり，要件（4）を満たせている．以上により，すべての要件を満たすようにプロトコルスタックのコンポーネント化ができたといえる．

6. 関連研究

6.1 AUTOSAR BSW のコンポーネント化

Dietmar Schreiner らは，車載向けプラットフォームである AUTOSAR の BSW（Basic SoftWare）を，COMPASS（Component Based Automotive System Software）を用いてコンポーネント化する手法を提案した [8][9]．COMPASS は静的なコンポーネント結合を採用しており，コンポーネント化によるオーバヘッドを小さく抑えることができる．また，AUTOSAR のアプリケーションコンポーネントとの接続もサポートされているため，AUTOSAR への適用が容易である．これは車載向けプラットフォームについてのみ対象とした研究であり，組込みシステムにおいてより一般的に用いられる TCP/IP プロトコルスタックについては考慮されていない．

6.2 携帯端末向け TCP/IP プロトコルスタックのコンポーネント化

Nancy Alonistioti らは，携帯端末に搭載される TCP/IP プロトコルスタックを動的に切り替えるためのコンポーネントシステムを提案した [10]．この手法では，コンポーネントを切り替えるためのモジュール，コンポーネントダウンロード用モジュール，インストールモジュールなどが搭載され，動的に切り替え可能なコンフィギュラブルなプロトコルスタックを実現している．その反面，実現のためにはターゲットに多くのソフトウェアモジュールを搭載する必要があり，また，コンポーネント間の通信にオーバヘッドが大きいなどの理由から，より細かい粒度の機能切替えや，携帯端末よりもリソース制約の厳しい組込みシステムへの適用は難しいと考えられる．

7. おわりに

本研究では，組込み向けコンポーネントシステムである TECS を用いて，TCP/IP プロトコルスタックの UDP 機能をコンポーネント化した．組込み向けの特性を無くさないことと，もともとの構造に大きな変更を加えないこと，拡張性を持たせること，変更容易性を持たせることの 4 点を必要要件としてコンポーネント化した．評価の結果，実行サイクルにかかるオーバヘッドが約 1.6%，メモリ使用量のオーバヘッドが 1%未満と，オーバヘッドが少ないことが確認できた．また，コンフィギュラビリティの向上も確認できた．

今後の課題として，TCP/IP プロトコルスタックとして必須の機能である，TCP や IPv6 プロトコルの設計を考えている．これにより，コンポーネント設計の正当性を示す．またファイアウォールの設計も考えている．TINET でサポートしていない機能を追加設計することで，コンポーネント設計の拡張性を示す．

参考文献

[1] TOPPERS プロジェクト/TINET: ”http://www.toppers.jp/tinet.html”
[2] TOPPERS プロジェクト: ”http://www.toppers.jp/index.html”
[3] TOPPERS プロジェクト/TECS: ”http://www.toppers.jp/tecs.html”
[4] 安積卓也，山本将也，小南靖雄，高木信尚，鵜飼敬幸，大山博司，高田広章：組込みシステムに適したコンポーネントシステムの実現と評価，コンピュータ ソフトウェア, Vol.26, No.4, pp.39-55, Nov. 2009.
[5] Takuya Azumi, Hiroshi Oyama and Hiroaki Takada：*Optimization of Component Connections for an Embedded Component System*, Proceedings of the IEEE/IFIP International Conference on Embedded and Uniquitous Computing, pp. 182-188, Vancouver, Canada, Aug. 2009.
[6] 安積卓也，大山博司，高田広章：メモリ共有を考慮した RPC システム, 情報処理学会論文誌プログラミング, Vol.2, No.2, pp.37-53, Mar. 2009.
[7] 石川拓也, 安積卓也, 一場利幸, 柴田誠也, 本田晋也, 高田広章：TECS 仕様に基づいた NXT 用ソフトウェアプラットフォームの開発, コンピュータソフトウェア， Vol.28, No.4, pp.158-174, 2011.
[8] Thomas M. Galla, Dietmar Schreiner, Wolfgang Forster, Christoph Kutschera, Karl M. Gschka, Martin Horauer：*Refactoring an Automotive Embedded Software Stack using the Component-Based Paradigm*, Proceedings of the IEEE Second International Symposium on Industrial Embedded Systems , ISBN 1-4244-0840-7, pp. 200 - 208, IEEE, 2007
[9] Dietmar Schreiner, Markus Schordan, Karl M. Gschka：*Component Based Middleware-Synthesis for AUTOSAR Basic Software*, Proceedings of the 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing , IEEE, 2009
[10] Nancy Alonistioti, Eleni Patouni, Vangelis Gazis: *Generic Architecture and Mechanisms for Protocol Reconfiguration*, Mobile Networks and Applications Journal, Special Issue on Reconfigurable Radio Technologies in Support of Ubiquitous Seamless Computing, Vol. 11, No. 6. pp.917-934,Dec. 2006