TINET+TECS: Component-~~based~~Based TCP/IP Protocol Stack for Embedded Systems

Comment [Editor1]: Remark: Please check with the journal if abbreviations are permitted in the title. If not, please define these abbreviations.

Takuro Yamamoto*, Takuma Hara† · Takuya Ishikawa† · Hiroshi Oyama‡ · Hiroaki Takada† and Takuya Azumi*

* Graduate School of Engineering Science, Osaka University † Graduate School of Information Science, Nagoya University ‡ OKUMA Corporation

***Abstract*—~~Embedded systems are applied to Internet of Things (IoT), and the high~~ High-productivity ~~of~~ embedded network software is required~~.~~ to run embedded systems within the Internet of Things (IoT). Tomakomai InterNETworking (TINET) is a Transmission Control Protocol/Internet Protocol (TCP/IP) protocol stack for use in embedded systems. Although TINET is a compact ~~TCP/IP~~ protocol stack, it ~~consists of~~comprises many complex source codes~~. Therefore, it~~ and is therefore difficult to maintain, extend, and ~~analysis the software.~~analyze. To improve ~~the~~ scalability and configurability, this paper proposes TINET componentized with the Toyohashi Open Platform for Embedded Real-time Systems (TOPPERS) embedded component system (TINET+TECS~~,~~), a component-based TCP/IP protocol stack for embedded systems~~: TINET componentized with TOPPERS embedded component system (TECS). The~~. This component-based TINET ~~provides~~offers software developers high productivity ~~such as change of~~through variable network buffer ~~size~~sizes and ~~adding/removing~~the ability to add or remove TCP (or UDP) ~~function.~~functionality. TINET+TECS utilizes a dynamic TECS component connection method ~~of TECS components~~ to satisfy the original TINET ~~specification. We evaluate the~~ specifications. The results of an experimental comparison between the proposed component-based ~~TINET compared with the~~ and original ~~TINET. Experimental results~~ TINETs show that the ~~overheads of~~ execution time and memory consumption overhead are ~~low,~~reduced and ~~that~~ the configurability is improved.**~~[1]~~

~~individual component diagrams enable the visualization of an entire system.~~

I. Introduction

The Internet of Things (IoT) is an essential next evolutionary step for the Internet [1][2] in which various items and platforms, for example, wearable devices, smart devices, and smart homes, will be connected via the Internet to further enrich people's lives. However, as the IoT uses embedded systems such as data sensors and controlling actuators as elemental constituents, it is often not practical to implement the same Transmission Control Protocol/Internet Protocol (TCP/IP) protocol stacks used by traditional computing systems because embedded systems face restrictions in terms of, for example, low memory capacity.

Tomakomai InterNETworking (TINET) is a compact TCP/IP protocol stack for embedded systems [3]. As TINET supports functionalities such as a minimum copy frequency and the elimination of dynamic memory control, it requires significantly reduced memory for its TCP/IP protocol stack and is therefore suitable for embedded systems. However, TINET comprises many complex source codes, i.e., it contains many files and defines many macros, which can be problematic for software developers seeking to maintain, extend, and analyze the software. Thus, embedded network software is required for high productivity and quality.

One approach to improving software productivity is component-based development, a design technique that can be applied in reusable software development for embedded systems [4] [5] such as TECS [6] [7], AUTOSAR [8], or SaveCCM [9]. Component-based systems are flexible to software extension and specification changes.

This paper proposes a component-based TCP/IP protocol stack for embedded systems~~, i.e.,~~ —TINET componentized with the Toyohashi Open Platform for Embedded Real-time Systems (TOPPERS) embedded component system (TINET+TECS~~,~~)—to improve the ~~configurabilit~~configurability and scalability of TCP/IP software. ~~TECS (TOPPERS Embedded Component System) [6] [7] is utilized to componentize TINET, because TECS~~Because it is a component system suitable for embedded systems~~.~~, TECS [6] [7] is used to componentize TINET in the proposed protocol stack. As TECS supports static ~~configuration which~~ configurations that statically define component behaviors and interconnections~~. Thus, TECS~~, it can optimize the ~~overhead of~~ componentization overhead.

In addition~~,~~ to satisfying the original TINET ~~specification~~specifications, the proposed framework utilizes the TECS dynamic connection~~, a method of TECS,~~ capability to dynamically switch component bindings. ~~A~~Although general TCP/IP protocol ~~server~~servers

---

~~1 Introduction~~

~~Internet of Things (IoT) is an essential keyword for the next era [1][2]. Various things, such as wearable devices, smart devices, and smart homes, connected to the Internet will enrich our lives. Embedded systems are the elements constituting IoT, e.g., sensing data and controlling actuators. It is not practical to implement the same TCP/IP protocol stack as a general computer because embedded systems have several restrictions such as low memory capacity.~~

~~TINET (Tomakomai InterNETworking) is a compact TCP/IP protocol stack for embedded systems [3]. TINET supports the ability such as minimum copy frequency and elimination of dynamic memory control. TINET needs only small memory for its TCP/IP protocol stack; therefore it is suitable for embedded systems. However, there are several issues that TINET consists of many complex source codes. In other words, TINET is composed of many file and define many macros. This may take a lot of time for software developers to maintain, extend, and analysis the software. Embedded network software is required for the high productivity and quality.~~

~~An approach to improve software productivity is component-based development, which is a design technique that can be applied to reusable software development for embedded systems [4] [5], such as TECS [6] [7], AUTOSAR [8], and SaveCCM [9]. Component-based systems are fl xible to software extension and specificatio changes. In addition,~~

dynamically ~~processes the~~process requested ~~port~~ports, i.e., HTTP (port: 80) and HTTPS (port: 443~~). However,~~), embedded systems are restricted in their dynamic processing ~~due~~ability owing to ~~the~~ strict memory ~~restriction~~constraints. TINET supports the static generation of Communication Endpoints (CEPs) and Reception Points (REPs), which are ~~like~~similar to sockets. As TINET+TECS, like TINET, statically generates components and dynamically combines them ~~as well as TINET~~in the same manner, TINET+TECS reduces ~~dynamically increasing~~ the dynamic increase of memory consumption.

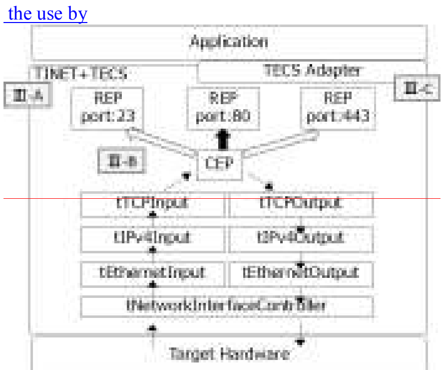
In the proposed framework, software applications can be developed ~~by not only~~using both the TECS method ~~but also~~and existing ~~method. The software~~ methods. Software applications can be developed as ~~a~~ TECS ~~component~~components because TINET+TECS is a component-based framework ~~using TECS. Moreover~~. Furthermore, TECS supports the use of an adapter to call TECS component functions ~~of TECS components~~ from non-TECS codes~~. The adapter~~, which allows for the use of existing TCP/IP applications without modification.

This paper evaluates the overheads of execution time and memory consumption and the amount of code line ~~change for adding/removing the~~ changes needed to add and remove functionalities~~, which demonstrates~~ in order to determine the ability of TINET+TECS ~~can~~to improve ~~the configurabilit~~configurability with small overheads. ~~Moreover~~Furthermore, the advantages of dynamic connection in terms of the memory consumption and ~~the~~ low overhead of the TECS adapter are demonstrated.

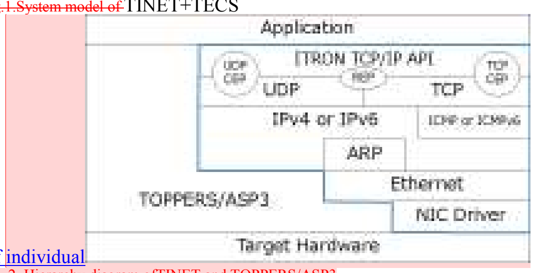
Contributions: This paper provides the following contributions~~.~~:

1) **Improve configurability**

~~Since~~Because TINET+TECS is a component-based system, ~~the~~its software ~~is fl xible to~~can flexibly change ~~the configuratio such~~ as system's configuration by, for example, resizing network ~~buffer~~buffers, adding/removing TCP (or UDP) ~~functions, and~~functionality, or supporting ~~both~~either IPv4 ~~and~~or IPv6. In addition, the use by



~~Fig.1.System model of~~ TINET+TECS



of individual

~~Fig. 2. Hierarchy diagram ofTINET and TOPPERS/ASP3~~

~~a~~ component ~~diagram provides~~diagrams enables visualization of ~~TINET, a complicated~~an entire system.

2) **Dynamic connection method**

Dynamically switching the binding of components, that is, switching between a TINET communication endpoint and REP~~reception point of TINET~~, realizes a TCP/IP protocol stack for an embedded ~~systems~~system.

3) **Support of legacy codes**

TINET+TECS can be applied to ~~an~~ existing ~~application~~applications because TECS supports the ability of the adapter to call TECS functions from C codes.

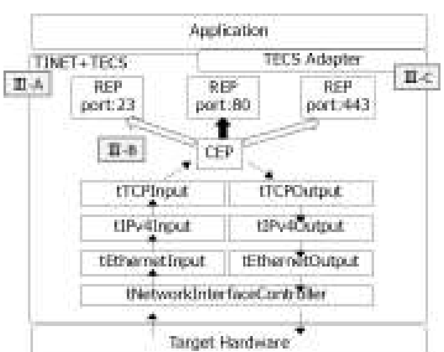


Fig. 1. System model of TINET+TECS

Comment [Editor2]: Remark: Please note that we have moved this fragment here from the beginning of this section; please check that this conforms to your intended meaning.

Comment [Editor3]: Remark: Please note abbreviations must be defined in each caption followed by the abbreviation in parenthesis. Please verify this and revise for all captions

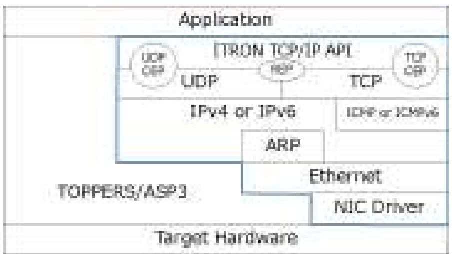


Fig. 2. TINET and TOPPERS/ASP3 hierarchy diagrams

Organization: The remainder of this paper is organized as follows. Section II introduces the system model and its basic technologies, i.e., TINET and TECS. Section III describes the design and implementation of the proposed framework. Section IV evaluates the proposed framework and demonstrates ~~the~~its advantages. Related work is discussed in Section V. Conclusions and suggestions for future work are presented in Section VI.

II. System Model

This section describes the system model of TINET+TECS, including ~~the~~ basic technologies such as TINET and TECS. ~~System~~A system model of the proposed framework is shown in Fig. 1. TINET+TECS is a component-based TCP/IP protocol stack~~, and~~ in which the TCP output task (tTCPOutput) and Ethernet input task (tEhternetInput) are implemented as TECS components. CEPs and REPs (Section II-A), which are also implemented as TECS components, dynamically switch bindings ~~by~~using the TECS method. Moreover, the TECS adapter supports the legacy codes for existing TCP/IP applications.

Comment [Editor4]: Tip: definite article (the) + noun. Some singular nouns refer to one specific thing (the only one of its kind) and therefore "the" is placed before the noun.

A. TINET

TINET is a compact TCP/IP protocol stack for embedded systems based on the ~~ITRON~~[1]ITRON [2] TCP/IP API ~~Specificatio~~Specification [10], developed by ~~TOPPERS (~~the TOPPERS~~oyohashi OPen Platform for~~ Project [11]. TINET has been released as an open-source tool.

~~[1]ITRON is a realtime operating system (RTOS) developed by TRON project.~~

~~Embedded Real-time Systems) Project [11].TINET has been released as open source.~~

To satisfy restrictions for embedded systems ~~such as~~in terms of, for example, memory capacity, size, and power consumption, TINET supports the following functions:

- ~~Minimum~~minimum copy frequency,
- ~~Elimination~~elimination of dynamic memory control,
- ~~Asynchronous interface~~
- ~~Error detailed~~asynchronous interfacing,
- error detailing per API.

1) Overview: TINET runs as middleware on TOPPER-~~-~~S/ASP3 [12] [13], ~~which is~~ a ~~realtime~~real-time kernel based on μITRON [14]. As it is compatible with TOPPERS RTOS, TINET also supports other RTOSs such as TOPPERS/ASP and TOPPERS/JSP ~~because TINET is compatible with TOPPERS RTOS~~.

Fig. 2 shows the hierarchy diagram of TINET and ~~TOP- PERS~~TOPPERS/ASP3. Users transmit and receive ~~the~~ data using a Communication End Point (CEP~~) which is~~), an interface that functions like a socket. In the transmission process, headers are attached to the data body passed to the CEP at each protocol layer~~, and~~ before the data ~~is~~are transmitted from the network device. In the reception process, the headers of the data ~~body~~bodies received ~~in~~by the network device are analyzed at each protocol layer, and the data ~~is~~are then passed to the CEP.

A TCP reception point called the REP~~eception Point (REP) is prepared~~stands by to ~~wait for a~~receive connection ~~request~~requests from the partner side. ~~An~~The REP has an IP address *(myaddr)* and a port number (my^or/^o) as attributes~~,~~ and performs functions ~~like~~such as ¿iwd() and *listenQ*.

In TINET, the ~~number~~amount of ~~the~~ data ~~copy~~copying between each protocol layers is minimized. ~~A~~ In standard computing systems, the TCP/IP protocol stack ~~for general computers~~ has large overheads in terms of execution time and memory consumption because the data ~~is~~are copied at each protocol ~~layers~~layer. To solve ~~the~~this problem, TINET ~~does pass~~passes the pointer of the data buffer between each protocol ~~layers, not perform~~layer instead of performing data ~~copy~~copying.

B. TECS

[2] ITRON is a real-time operating system (RTOS) developed by the TRON project.

TECS is a component system suitable for embedded systems~~. TECS~~ that can increase productivity and reduce development costs ~~owing to~~based on the improved reusability of software components. TECS also provides component diagrams~~, which~~ that can help developers visualize the overall structure of a system.

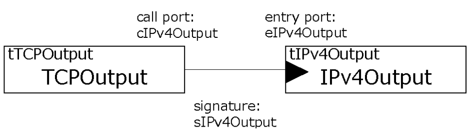


Fig. 3. Component Diagram

```
signature sIPv4Output {
  T_IN4_ADDR getIPv4Address(void);
  ER getOffset([inout]T_OFF_BUF *offset);
  ER setHeader([inout,size_is(size)]
                     int8_t *outputp,
                     [in]int32_t size,
                     [in]T_IN4_ADDR dstaddr,
                     [in]T_IN4_ADDR srcaddr);
    /* Omit: other functions */
};
```

Fig. 4. Signature description

In TECS, component deployment and composition are performed statically~~. Consequently, connecting components~~, and as a consequence, component connection does not incur ~~significan~~significant overhead, and memory requirements can be reduced. TECS ~~can be implemented in C, and demonstrates~~has various ~~feature~~features such as source level portability and fine-~~graine~~grained components and can be implemented in C.

~~1~~*1) Component* ~~*Model*~~*model*: Fig. 3 shows a component diagram. ~~A *cell*~~*Cells*, which ~~is an~~are instance ~~of a component~~components in TECS, ~~consists~~consist of *entry* ports, *call* ports, attributes, and internal variables. An *entry* port is an interface that provides functions to other cells, ~~and~~whereas a *call* port is an interface that enables the use of other ~~*cell's*~~*cells'* functions. ~~A~~Each *cell* has one or more *entry* ports and *call* ports. *Cell* functions are implemented in C.

The type of *entry/call* port is ~~define~~defined by a *signature,* which is a set of functions~~. A *signature* is~~ that defines the interface ~~definitio~~definition of ~~a~~the *cell.* The *cell's call* port can be connected to the *entry* port of another *cell* by the same *signature.* Note that *celltype* ~~define~~defines one or more *call/entry* ports, attributes, and internal variables of a *cell*.

2) Component ~~*Description*~~*description:* In TECS, components are described ~~with~~using component description language (CDL). CDL can be divided into three categories: *signature, celltype,* and build description. These components are described as follows.

Signature Description

The *signature* ~~define~~defines a *cell* interface. The *signature* name follows the keyword *signature* and takes the ~~prefi~~prefix “s~~”, e.g.,~~”–for instance, sIPv4Output (Fig. 4). ~~In TECS, to~~To clarify the function of an interface, ~~specifier~~specifiers such as [in], [out], and [inout] are used~~, which~~ in TECS to represent the input, output, and input/output, respectively. Similarly, [size_is(len)] represents an array of size len.

Celltype Description

The *celltype* ~~define~~defines the *entry* ports, *call* ports, attributes, and variables. A *celltype* name with the ~~prefi~~prefix “t” follows the keyword *celltype,* e.g., tIPv4Output (Fig. 5). To ~~defin~~define *entry* ports, a *signature,* e.g., sIPv4Output, and an *entry* port name, e.g., eIPv4Output, follow

```
celltype tIPv4Output {
   /* Entry port */
   entry sIPv4Output eOutput;

   /* Call port */
   call sEthernetOutput cEthernetOutput;
   /* Omit: other call ports */

   attr { /* Attribute */
      uint16_t fragInit = 0;
   };
   var { /* Variable */
      uint16_t fragId = fragInit;
   };
};
```

Fig. 5. *Celltype* description

```
cell tIPv4Output IPv4Output {
   /* Omit: other build description */

   fragInit = 0; /* Attribute */
};
cell tTCPOutput TCPOutput {
   cIPv4Output = IPv4Output.eOutput;
   /* Omit: other build description */
};
```

Fig. 6. Build description

the keyword *entry*. *Call* ports are ~~define~~defined similarly. Attributes and variables follow the keywords *attr* and *var,* respectively.

Build Description

The build description is used to instantiate and connect *cells.* Fig. 6 shows an example of a build description. A *celltype* name and *cell* name, e.g., tIPv4Output and IPv4Output, respectively, follow the keyword *cell.* ~~A *call* port, cell's name, and an *entry* port are described in that order to compose *cells,*~~ In ~~Fig. 6~~the figure, *entry* port eIPv4Output in *cell* IPv4Output is connected to *call* port cIPv4Output in *cell* ~~TCPOut- put.~~TCPOutput. Finally, *C_EXP* ~~calls~~ can be used to call macros ~~define~~defined in C files

III. Design and Implementation

This section describes the design and implementation of the proposed ~~framework,~~ TINET+TECS framework. The proposed framework is a component-based TCP/IP protocol stack for embedded systems~~, i.e.,~~; in other words, a componentized TINET using TECS. In ~~addition, A TECS~~ this section, a novel TECS functionality~~,~~—the dynamic connection method~~,~~—and the TECS adapter to support legacy codes are described ~~with the~~via a use case of the proposed framework.

A. TINET+TECS

TINET+TECS, the proposed componentized TCP/IP protocol stack, ~~consists~~comprises a number of ~~some~~ TECS components. This section describes the components of the TINET+TECS framework with the aid of component diagrams.

1) Components of a protocol stack: The components of a TINET+TECS protocol stack ~~for TINET+TECS is~~are shown in Fig. 7. Note that some small particle components, such as a kernel object, ~~dataqueques~~data queues, and semaphores, are omitted to simplify the component diagram. In TINET+TECS, the components are

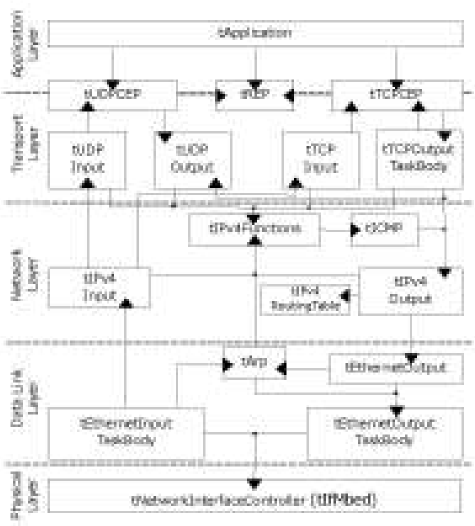


Fig. 7. Component diagram of a protocol stack

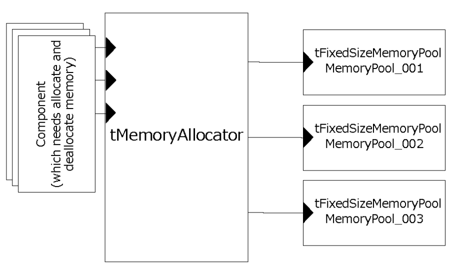


Fig. 8. Component diagram of tMemoryAllocator

divided for each protocol, and ~~the~~ functionalities such as input ~~function~~ and output ~~function~~functions are ~~define~~defined as ~~each a component. Therefore~~respective components. By using such small grain components, software visibility is ~~improving because of small grain components~~improved.

The components of each protocol are described ~~below~~in the following.

Application layer: An application in TINET+TECS is implemented as a component such as tApplication. Software with TINET uses ITRON TCP/IP API [10] such as *tcp_s-nd_dat* and *tcp_rcv_dat*. In TINET+TECS, the application component calls TECS functions such as cTU_E4P/_sendData and *cTCPAPI_receiveData*. Moreover, in TINET+TECS supporting a TECS adapter (III-C), an existing application with TINET can run on the TINET+TECS framework without transporting~~. Therefore~~, and therefore, software can be developed either ~~with~~using existing methods or as ~~a~~ TECS ~~component~~components.

Transport layer: tTCPCEP (tUDPCEP) ~~is a~~and tREP are, respectively, CEP ~~component~~and REP components for TCP (UDP~~), and tREP is a REP component.~~). For example, a server program supporting multiple clients can be developed by preparing ~~the~~ multiple tTCPCEP components. tTCPInput (tUDPInput) and tTCPOutput (tUDPOutput) are components for performing ~~the~~, respectively, receiving and sending processing ~~respectively~~ in the transport layer.

Network layer: tIPv4Input and tIPv4Output are components ~~performing the~~that perform, respectively, receiving and sending processing ~~respectively~~ in the network layer. The tIPv4Functions component performs ~~some~~ functions such as checksum, the tICMP component is for implementing the ICMP protocol, and the tIPv4RoutingTable component operates a routing table.

Data link layer: tEthernetInputTaskBody and ~~tEthernetOut- putTaskBody~~tEthernetOutputTaskBody (tEthernetOutput) are components for performing ~~the~~, respectively, receiving and sending processing ~~respectively~~ in the data link layer. The tArp component is used for the ARP protocol.

Physical layer: ~~tNetworkInterfaceContoroller~~The tNetworkInterfaceController component implements a network device driver. Software can be run on other devices by replacing the component because only the component depends on the target device.

To utilize the protocol stack ~~as~~in the same manner as in the original TINET, communication object components such as tTCPCEP,

Comment [Editor5]: Remark: Please note that abbreviations should be usually defined at first use in the abstract as well as in the text. Once defined, they should be used consistently in place of the spelled-out term. Please check for similar instances throughout the document.

tUDPCEP, and tREP are ~~define~~defined as an interface between TINET+TECS and an application. The communication object component ~~is a component corresponding~~corresponds to a CEP or REP of the original TINET. Application developers can utilize the TINET+TECS functionalities by generating and combining as many components as necessary.

The protocol stack of TINET+TECS supports the coexistence of multiple protocols. ~~By developing the~~ Through its use of IPv6 and Point-to-~~-~~Point Protocol (PPP) components, TINET+TECS can make IPv4 and IPv6 coexist and support PPP without ~~a~~ ~~modificatio~~modification of component implementation.

2) Memory allocator component: The original TINET eliminates dynamic memory control to meet the severe memory ~~restriction~~restrictions of embedded systems. A memory area for sending/receiving data in the protocol stack is allocated and released within a predetermined area. ~~Memory~~The memory allocator component ~~performs the~~allows for elimination of dynamic memory control in TINET+TECS~~. The component provides~~ by providing a requested memory area from the ~~memory area~~ statically allocated memory area.

The memory allocator component connects to as many tFixedSizeMemoryPool as ~~needed~~required, shown in Fig. 8. tFixedSizeMemoryPool is a componentized kernel object of ~~TOP- PERS~~TOPPERS/ASP3 ~~to alloc~~for allocating and release ~~a~~ memory ~~area~~areas of ~~the~~a requested size. tFixedSizeMemoryPool components ~~with~~of various sizes are prepared, and an appropriate memory area can be allocated according to the used data size. On the other ~~hands~~hand, all components ~~which~~that need to allocate ~~and~~or deallocate memory, e.g., tTCPInput and tEthernetOutput, connect to the memory allocator component~~, e.g., tTCPInput and tEthernetOutput.~~.

In addition, TINET+TECS utilizes the TECS *send/receive* ~~specific of TECS~~specifier to minimize the memory copy frequency, which is a functionality supported by TINET.

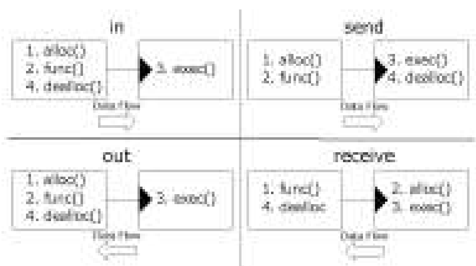


Fig. 9. Differences between in/out and send/receive

```
signature sNicDriver {
  void start(
    [send(sNetworkAlloc),size_is(size)]
        int8_t *outputp,
    [in]int32_t size,
    [in]uint8_t align);
  void read(
    [receive(sNetworkAlloc),size_is(*size)]
        int8_t **inputp,
    [out]int32_t *size,
    [in]uint8_t align);
    /* Omit: Other functions */
};
```

Fig. 10. Signature description of the nic driver (An example of send/receive)

Fig. 11. Dynamic connection

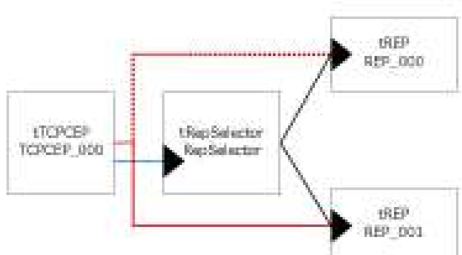


Fig. 12. Dynamic connection between CEP and REP

~~**send**~~

Send/receive ~~specifier~~specifiers: TECS supports send/receive ~~specifiers which are~~ interface ~~specifier~~specifiers [15]. TINET+TECS uses *send* and *receive* ~~specifier~~specifiers instead of *in* and *out* to reduce the number of copies~~.~~:

- *in* is a ~~specific~~specifier for input arguments. A callee side uses the memory of arguments with *in* ~~during~~when executing the callee function. When the processing returns to the caller side, the caller can reuse and deallocate the memory.
- *send* is ~~also a specific~~another specifier for transferring data to a callee from a caller such as *in*. The difference between *in* and *send* is whether ~~to deallocate~~ the data memory is deallocated in ~~a~~the caller or callee, as shown in Fig. 9. In the case of the *in* ~~specific~~ specifier, both allocating and deallocating of the data memory are performed in the caller. ~~On~~By contrast, in the ~~other hand, in~~ case of *send*, the caller allocates the data memory, and the callee deallocates it.
- *out* is a ~~specific~~specifier for output arguments~~. A~~ through which a callee writes data in the memory allocated by a caller~~, and~~ while the caller receives the data.
- *receive* is ~~also a specific~~another specifier for a caller receiving data from a callee such as *out*. The difference between *out* and *receive* ~~is~~lies in whether ~~to allocate~~ the data memory is allocated in ~~a~~the caller or callee, as shown in Fig. 9. ~~While, in~~In the

case of *out*, ~~a~~the callee writes data in the memory allocated by a caller, whereas in ~~case of~~the *receive* case, the callee allocates the data memory. Deallocating of the memory is performed in the caller in both cases.

As shown in Fig. 10, ~~the arguments of~~data sending and receiving ~~data~~arguments such as *outputp* and *inputp* are ~~define with~~defined using, respectively, the *send*/~~*re- ceive*~~*receive* specifier in the signature description.

B. Dynamic ~~*Connection of*~~ connection in TECS

TECS supports dynamic connection~~ as a new functionality. Dynamic connection is a ~~, a method ~~to switch~~for switching the binding of components at runtime ~~as shown in~~ (Fig. 11~~.~~) as a new functionality. Note that all components are statically generated in TECS~~. TECS~~, which can optimize the overhead of componentization because components are statically configured. Dynamically generating ~~the~~ components causes a ~~lot~~good deal of memory consumption, which is a serious problem for embedded systems with strict memory ~~constraint~~constraints. The proposed framework ~~realizes~~ can take advantage of the componentization ~~of~~in TINET while satisfying the memory constraint~~,~~ because components are statically ~~generate~~generated and dynamically ~~connect~~connected in TECS.

TINET+TECS utilizes ~~the~~ dynamic connection to switch between CEP and REP components, as shown in Fig. 12. In a server application, CEP is associated with REP in the state of waiting for connection request from clients[2]. For example, when processing with the HTTP protocol, CEP passively opens with a REP of port number 80.

To utilize dynamic ~~connection, the~~ connectivity, a selector should be defined~~ The~~. A selector connects all ~~the~~ components that can be dynamically ~~connnected, to refer the~~connected under a common descriptor ~~of them. Descriptor is~~that serves as an ~~identifie~~identifier to access ~~the~~each component [16]. The cREP ports ~~are~~form a call ~~ports~~port array~~, which connect the number of~~ connecting to all tREP cells (Line 9 in Fig. 13). *[ref_desc]* is ~~described~~used to identify ~~the~~ call ~~port refers~~ports referring to descriptors. In the case ~~of~~shown in Fig. 12, the tRepSelector cell connects all tREP cells.

A CEP component has two call ports~~.~~: the cRepSelector port, which connects to the eRepSelector port of the tRepSelector cell, and the cREP4

[2]ITRON TCP/IP API ~~Specificatio~~Specification [10] tcp_acp_cep(ID cepid, ID repid, T_IPV4EP *p_dstaddr, TMO tmout~~)~~).

```
1  signature sRepSelector {
2     void getRep([out]Descriptor(sREP4) *desc,
3               [in]int_t i);
4  };
5
6  celltype tRepSelector {
7     entry sRepSelector eRepSelector;
8     [ref_desc]
9        call sREP4 cREP[NUM_REP];
10 };
11
12 celltype tTCPCEP {
13    call sRepSelector cRepSelector;
14    [dynamic]
15       call sREP4 cREP;
16    /* Omit: other call/entry ports */
17    /* Omit: attributes and variables */
18 };
```

Fig. 13. Signature and celltype description for dynamic connection

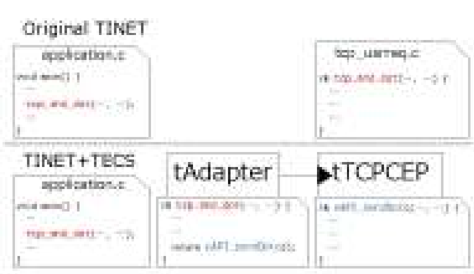


Fig. 15. TECS adapter

```
1  eAPI_accept (.., ..) {
2     /* Get a descriptor of intended REP cell */
3     cRepSelector_getRep(&desc, repid);
4     /* Set the descriptor */
5     cREP_set_descriptor(desc);
6     /* Call the function of intended REP cell */
7     cREP_getEndpoint();
8  }
```

Fig. 14. Accept function (~~a~~ dynamic connection example)

TABLE I

EVALUATION BOARD ENVIRONMENT

Board	GR-PEACH
CPU	Cortex-A9 RZ/A1H 400MHz
Flash ROM	8 MB
RAM	10 MB
LAN Controller	LAN8710A

~~IV. EVALUATION~~

~~port~~

Port, which connects to either of the tREP cells (Lines 13~~–~~–15 in Fig. 13). The cREP port is ~~define with~~defined using *[dynamic]* to identify the call port used to dynamically switch the components. The call port with the *[dynamic]* ~~specifie~~specifier is not optimized and is allocated in RAM ~~by~~using a plug-in.

Fig. 14 shows a sample code of dynamic connection. The eAPLaccept function is the function wrapping *tcp_acp_cep* ~~with~~under TECS, which is ~~utilized to be~~set as the state waiting for connection request. ~~In~~ Dynamic connection in the function~~, dynamic connection~~ is performed as shown in Fig. 14. First, ~~get~~ the descriptor of REP to be joined is obtained (Line 3 in Fig. 14). The ~~firs~~first argument, *&desc*, is a variable used to store the descriptor information, ~~and~~whereas the second argument, *repid*, is the index of tREP cells. Next, ~~set~~ the ~~descritptor~~descriptor is set (Line 5 in Fig. 14~~).~~), and the cREP port combines the tREP cell ~~that~~specified by the descriptor~~ specify. Thus,~~, enabling the tCEP cell ~~can~~to call the function of the tREP cell to be joined (Line 7 in Fig. 14).

C. TECS ~~*Adapter*~~adapter

TECS supports *Adapter* functionality, which enables ~~to call~~calling a function in TECS from existing C codes. ~~An~~The adapter is implemented between C codes and a TECS component~~,~~ and links a C function to a TECS function as shown in Fig. 15. In TINET+TECS, when ~~the~~an application calls an API such as tcp_snd_dat, the adapter component calls ~~the~~a function of tTCPCEP such as *eAPI_sendData*. Note that *tcp_s^d_dat* is ~~define with~~defined under the name *eAPI_sendData* in TINET+TECS. The adapter wraps the APIs used in the existing applications into TECS functions~~. Therefore,~~, enabling software developers ~~can~~to utilize an existing TCP/IP application ~~by using~~via the adapter.

IV. Evaluation

This section describes ~~our~~the experimental evaluation used to demonstrate the effectiveness of the proposed framework.

A. Evaluation ~~*Environment*~~environment

GR-PEACH ~~is~~was employed as the evaluation board. ~~We connected the board and the host PC with a LAN cable, and evaluated the data transmission and reception. The detail specificatio~~Detailed specifications of the board ~~is~~are shown in TABLE I. We also employ TINET 1.5.4 and the compiler arm-none-eabi-gcc 5.2. To pretest the system, we connected the board to a host PC via a LAN cable and evaluated the data transmission and reception.

B. Performance of TINET+TECS

To demonstrate the low overhead of TINET+TECS, we ~~evaluated the~~ compared its execution time and ~~the~~ memory consumption ~~of TINET+TECS compared~~ with that of TINET~~.~~

~~The~~, producing the comparison ~~of execution time between TINET and TINET+TECS is~~results shown in Fig. 16. ~~The APIs used for the evaluation are~~

The *tcp_s^d_dat* ~~to send TCP data~~ and *tcp_rc^_dat* ~~to~~ APIs were used in the evaluation to, respectively, send and receive TCP data. For *tcp_s^d_dat*, we measured the executing time starting from the API ~~calling by~~call through the application ~~to~~until the return of the processing result. In TINET+TECS, ~~the~~this process is performed in the order ~~of~~ tApplication, tTCPCEP, ~~tTCPOutput-TaskBody~~tTCPOutputTaskBody, tIPv4Output, tEthernetOutput, tArp, ~~tEthernetOut- putTaskBody~~tEthernetOutputTaskBody, and tlfMbed ~~of~~, as shown in Fig. 7. For *tcp_rc^_dat*, we measured the execution time from the data ~~receiving~~receipt in the LAN driver ~~to the~~until data acquisition in the application. In TINET+TECS, the process is performed in the order ~~of~~ tlfMbed, tEthernetInputTaskBody, tIPv4Input, tTCPInput, tTCPCEP, and tApplication ~~of~~ , as shown in Fig. 7. The execution time of TINET+TECS is ~~as well as~~close to that of TINET~~; the~~, with an overhead ~~is~~of about 3 ~~usec.~~ μs. As the use of the *send/receive* specifier ~~enable to access~~enables accessing of the buffer address without data ~~copies. Therefore~~copying, the ~~overhead of~~ componentization overhead does not ~~effect~~affect the execution time.

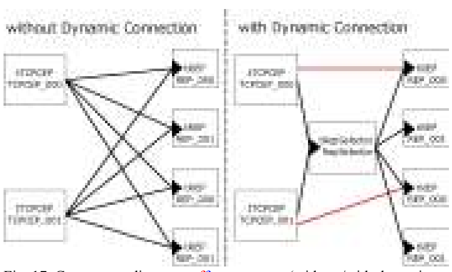


Fig. 17. Component diagrams ~~of~~for two cases (without/with dynamic connection)

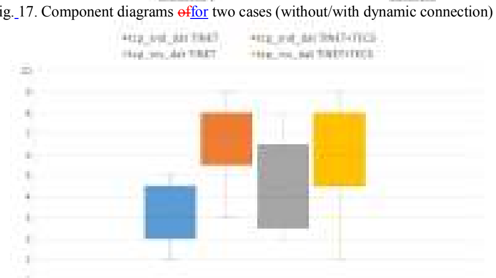


Fig. 16. Execution ~~time~~times of TINET and TINET+TECS

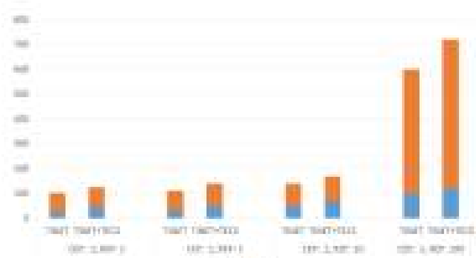


Fig. 18. Memory consumption ~~of~~in two cases (with/without dynamic connection)

TABLE II
MEMORY CONSUMPTION OF TINET AND TINET+TECS

	TINET	TINET+TECS
ROM	74.93 KB	76.62 KB
RAM	27.36 KB	28.24 KB
ROM + RAM	102.29 KB	103.86 KB

TABLE III
MODIFIED CODE LINES OF CDL

	Size	Size (- Default)	CDL
Default	104.86 KB	0 KB	0
I	80.79 KB	~~-~~–24.07 KB	18
I + II	80.50 KB	~~-~~–26.35 KB	26
I + II + III	72.03 KB	~~-~~–32.83 KB	31

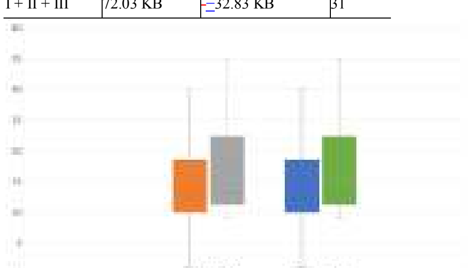


Fig. 19. Execution ~~time of~~times in two cases (without/with TECS adapter)

The memory consumptions of TINET and TINET+TECS are compared in TABLE II. The memory consumption of TINET+TECS ~~increases~~is about 2.5% ~~compared with~~ higher than that of TINET~~. The~~, as the data and processes ~~and data to manage TECS components,~~ such as initialization of cells, descriptors, function tables, and skeleton functions~~, cause the~~ needed to manage TECS components increase memory consumption.

As shown in TABLE ~~??,~~III, the code lines for ~~modificatio~~modification were measured to demonstrate the improved ~~configurabilit . We can~~configurability. This demonstrated the ability to change the composition of the protocol stack ~~with~~using a small workload~~. Thus,~~, confirming that the proposed framework improves ~~the configurabilit~~ configurability.

C. Dynamic ~~*Connection*~~connection

~~The memory~~Memory consumption without and with TECS dynamic connection was then evaluated. As shown in the left panel of Fig. 17, ~~a~~each CEP component should be statically ~~connect~~connected to all REP components ~~in the case~~if dynamic connection is not used. As the number of REPs increases, additional call ports of CEP are required ~~for that. Thus, it consumes a lot~~, in turn increasing the consumption of memory. Dynamic connection reduces ~~the~~ memory consumption because only one CEP-to-REP call port ~~of CEP for REP, which is drawn~~is required per CEP, as illustrated with red lines in the right panel of Fig. 17~~, is required per CEP.~~. Even ~~when~~if the number of REPs increases, ~~a~~additional call ~~port of~~ports can be joined through the selector~~, not~~ instead of the ~~CEP, is joined~~CEPs.

~~The memory~~Memory consumption ~~of two cases,~~ without~~/~~ and with dynamic connection~~,~~ is shown in Fig. 18. The ~~case of~~ dynamic connection case consumes ~~the~~more RAM memory ~~more than the other case~~ because~~ the~~, as mentioned in Section III-B, call ports with *[dynamic]* ~~is~~are not optimized and allocated in RAM areas ~~as mentioned in Section III-B. The total~~. However, the overall memory consumption is ~~fewer in~~lower under the proposed framework.

D. ~~*Overhead of*~~ Adapter overhead

The API execution time ~~of API with the adapter~~when using adapters such as *tcp_snd_dat/eAPI_sendData* was used to analyze the overhead of the TECS adapter ~~which supports~~supporting existing applications. As shown in Fig. 19, the overhead of the TECS adapter is very small because the adapter only passes ~~the~~ parameters from C codes to TECS components ~~from C codes. Therefore,~~; thus, the TECS adapter overhead does not affect the system.

V. Related Work

Open-source TCP/IP protocol stacks that have been developed for embedded systems ~~have been developed such uIP~~include uIP [17~~], lwIP~~] and lwIP [18].

uIP: ~~uIP (microIP) isavery~~uIP (microIP) is a very small TCP/IP stack intended for tiny 8-~~bit~~ and 16-bit microcontrollers. ~~uIP only~~uIP requires only about ~~f ve~~5 KB of code size and several ~~hundreds~~hundred bytes of RAM. uIP has been ported to various systems and has found its way into many commercial products. ~~After~~ Following the release of ver. 1.0~~ is released~~, later versions of uIP, including uIPv6, ~~are~~have been integrated with Contiki OS [19], [20], an operating system to connect tiny ~~microcontroller~~microcontrollers to the Internet.

lwIP: ~~lwIP~~lwIP (lightweightIP) is a small TCP/IP implementation for embedded systems~~. The focus of lwIP~~ that is intended to reduce memory resource usage while still ~~having~~maintaining a full~~-~~-scale TCP. lwIP requires about 40 KB of ROM and tens of KB of RAM. lwIP is larger than uIP, but provides better throughput.

VI. Conclusion

This paper proposed TINET+TECS, a component-based TCP/IP protocol stack for embedded systems. TINET+TECS is a componentized version of TINET, ~~which is~~ a compact TCP/IP protocol stack~~,~~ ~~using~~that uses TECS. Because TINET ~~consists of~~comprises many macros and complicated codes~~ and the~~, its software productivity ~~are~~is low. The proposed framework improves ~~the configurabilit~~on TINET's configurability while suppressing the overhead of componentization. ~~The proposed framework~~Scalability is also ~~improves the scalability~~improved because the component-based framework ~~simplifie~~simplifies to add/remove and change protocols such as TCP/UDP, IPv4/IPv6, and ~~Ether- net~~Ethernet/PPP.

~~In addition, this~~This paper ~~presents~~also presented dynamic connection, a new TECS method ~~of TECS,~~ to enable dynamic processing while reducing memory consumption. ~~To~~ TINET+TECS utilizes dynamic connection to satisfy ~~TINET specificatio that~~ the TINET ~~support~~specification for supporting static generation of CEPs and REPs~~, TINET+TECS utilizes dynamic connection.~~. As the TECS adapter supports legacy codes, existing TCP/IP applications can run without ~~modificatio~~modification in the proposed framework.

In ~~the~~ future~~,~~ work, we will adapt the proposed framework ~~will~~to cooperate with mruby~~ on TECS [21] to easily manage IoT devices. Note that mruby is~~, a scripting language for embedded systems ~~[22].~~ on TECS [21][22], to more easily manage IoT devices. We will support ~~the~~ functionalities ~~that~~in which TINET functions can be utilized from mruby programs as an extension of mruby-socket [23].