Vol.2013-SLDM-160 No.32
Vol.2013-EMB-28 No.32
2013/3/14

An Evaluation for RAM Usage of TCP/IP Protocol Stack on Embedded Systems

NOBUHITO MIYAUCHI[†1]

It is quite common to connect to IP network for embedded systems such as consumer electronics appliances, automobile electronics devices, security communication devices, etc. in recent years. These systems need a single-chip microcomputer with internal RAM and don't have usually external RAM to reduce hardware cost. Quite a few TCP/IP protocol stacks including open source software have been developed for usage of a small size RAM.

We investigated the RAM usage in detail by using "lwIP" named for a lightweight TCP/IP protocol stack, an embedded real time ITRON-based OS, and evaluation boards for a 32-bit ARM® Cortex™ M3 microprocessor. We recognized that their RAM usage has a very large proportion of storing buffer for packet data and the buffer space increases more than a twofold by congested network. And we are researching buffer management mechanism among an IP network interface and other interfaces, based on the evaluation data of IP network congestion and stagnant data among such interfaces.

1. Introduction

Internet technology allows a lot of electronic devices in offices, homes, and factories to exchange data and work in close cooperation with each other nowadays. Although many kinds of interfaces such as RS-232C serial, parallel, USB and so on have been already utilized for network communication among devices since the 1980s, Ethernet plays a leading role for high performance, versatility, availability, and interoperability.

These devices are usually manufactured by embedding a single-chip microcomputer with internal RAM. The cost reduction of hardware for mass production demands limitation of available RAM, which size is smaller than a hundred of kilobytes, but traditional implementations of a TCP/IP protocol stack required several hundreds of kilobytes of RAM.

Therefore, for almost a decade, not a few TCP/IP protocol stacks for embedded systems have been designed to execute application programs using a small size RAM. One of such TCP/IP protocol stacks is "lwIP" named for a lightweight TCP/IP protocol stack originally designed by Adam Dunkels at the Swedish Institute of Computer Science and used by a lot of manufacturers of embedded systems [10][11][12]. lwIP is available for an embedded system with tens of kilobytes of free RAM. He also designed another implementation named "uIP."[13]

We investigated basic situation of RAM usage for TCP/IP communication devices to establish implementation techniques of application programs. Assuming development of embedded systems in our manufacturing fields, we prepared an experimental system consisting of an evaluation board for a 32-bit ARM® Cortex™ M3 microprocessor[16], an embedded real time ITRON-based OS "Cros", and OS porting lwIP. Cros (Configurable Real－time Operating System) was developed originally by Mitsubishi Electric corporation [17].

2. Related Work

2.1 BSD-based TCP/IP protocol stack

Most of current widespread implementations of TCP/IP protocol stacks are based on some stacks of Berkeley Software Distribution (BSD)[1][2][3], which support full specification of internet protocol and are approved of for its reliability. These are not oriented toward embedded devices but personal computers, workstations, office computers and server systems. For example, these have a function of virtual memory. However, some BSD-based TCP/IP protocol stacks have been ported to embedded systems, and a porting case reported that FreeBSD-based stack used 580 kilobytes of RAM[4].

2.2 TCP/IP protocol stack for embedded systems

There are some design policies for a TCP/IP protocol stack with less memory consumption, for example, cutbacks in various functions such as flow control, improvement of memory management, imposing resource requirements on application programs, and so on.

Some stacks with minimum function and resource limitation were developed and reported [4][5][6][15].

TINET[7][8] is a FreeBSD-based stack developed as a part of TOPPERS project (Toyohashi OPen Platform for Embedded Real-time Systems)[9], which is a Japanese open source software project.

Other stacks as well as TINET were designed not to copy received data in data buffers [10][11][12][13]. TINET adopted mainly ITRON TCP/IP API to avoid dynamic memory allocation in handling protocol process of packet data except initial network buffer allocation. In case of lwIP, BSD socket API was minimized and new API was designed for similar purposes.

2.3 lwIP protocol stack

lwIP supports congestion control, IP fragment reassembly and multiple simultaneous connection, while a lot of porting cases to various microprocessors and embedded real time OS were reported [14].

In memory management, uIP uses a single global buffer for holding network packets with repetition of overwriting and a secondary buffer for application process. On the other hand, lwIP uses a dynamic memory allocation mechanism to be able to store by splitting an incoming larger packet than one allocated buffer. Furthermore, in outgoing case, lwIP copies application data to the buffer with queuing mechanism for retransmission.

†1 Information Technology R&D Center, Mitsubishi Electric Corporation

On the other hand, uIP uses the same buffer as a receiving buffer without queuing.

lwIP avoids copying of memories by sharing buffer memories between applications and the internal stack process as well as other protocol stacks [4]. Therefore, such shared buffer memories are allocated and released by management of reference counters for garbage collection.

The memory management mechanism allocates various types of memory cells from a large memory pool (heap), as shown in table 1.

Table 1 Types of memory cells in lwIP

Index	Type	Description
1	RAW_PCB	Protocol Control Block for RAW connection
2	UDP_PCB	Protocol Control Block for UDP connection
3	TCP_PCB	Protocol Control Block for simultaneously active TCP connection
4	TCP_PCB_LISTEN	Protocol Control Block for listening TCP connection
5	TCP_SEG	Simultaneously queued TCP segments on the unsent and unacked queues
6	REASSDATA	Used for simultaneously IP packets queued for reassembly
7	NETBUF	Linkage information for network buffer cells (only needed if you use the sequential API)
8	NETCONN	Network connection information (only needed if you use the sequential API)
9	TCPIP_MSG_API	TCP/IP message information for callback/timeout API communication
10	TCPIP_MSG_INPKT	TCP/IP message information for incoming packets
11	ARP_QUEUE	Used for simultaneously queued outgoing packets that are waiting for an ARP request (to resolve their destination address) to finish
12	IGMP_GROUP	Used for multicast groups whose network interfaces can be members et the same time
13	SYS_TIMEOUT	Used for simultaneously active timeouts (for porting of OS)
14	PBUF_POOL	Used for buffers in the memory pool

3. Implementation

3.1 System overview

We prepared the evaluation system consists of evaluation boards for a 32-bit ARM® Cortex™ -M3 microprocessor, an embedded real time OS (ITRON-based “Cros”), a lwIP protocol stack, and evaluation application programs, as shown in fig. 1.

The Cortex™-M is a group of 32-bit RISC ARM® processor cores developed by ARM limited. Processing including OS and interrupt can be executed mostly by 16-bit operations, because the Cortex™-M3 supports Thumb2 instructions that was extended from Thumb ones, which were adopted to ARM7T® core. The Cortex™-M can attain a compact program size by minimizing the number of embedded functions. It is expected to be the direct descent of ARM7TDMI core which came into wide use as a low-cost embedded microprocessor.

We chose one of Stellaris® ARM® Cortex™-M3 Microcontrollers (LM3S9U90: table 2) [20] manufactured by Texas Instruments Incorporated considering a mass production.

We experimented and collected memory consumption data using the evaluation board connected to a personal computer (PC) by a 100BASE-TX Ethernet cable and a USB cable to control an In-Circuit-Emulator (ICE).

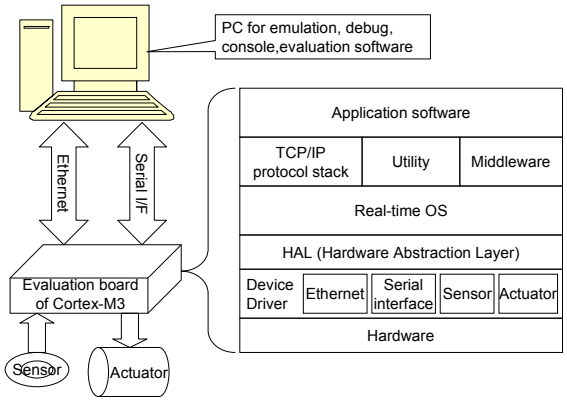

Fig. 1 Hardware and software structure of evaluation system

Table 2 Microprocessor Primary Specification (LM3S9U90)

Index	Item	Value
1	Flash	384KB
2	SRAM	96KB
3	DMA	32-channel configurable controller
4	Max Speed	80MHz
5	Ethernet	10/100 MAC+PHY

3.2 lwIP porting onto Cros (ITRON)

We adopted lwIP version 1.3.2, because it was included in the evaluation kit of evaluation boards. However any OS was not attached and we had to port lwIP onto Cros. A technical document on lwIP web site describes how to port for a multi-task OS[18]. The primary porting issues are definition of data types, preemption protection, semaphores, mailboxes, timeout mechanism, and threads, which correspond the supported functions by ITRON ver. 4.0, as follows in table 3.

Table 3 lwIP Porting Functions

Index	lwIP Requirement	ITRON Function
1	Definition of data types	Depend on CPU Architecture.
2	Preemption Protection	Hardware Abstraction Layer supports.
3	Semaphores	Supported.
4	Mailboxes	Supported.
5	Timeout Mechanism	Supported as Timer Handler.
6	Threads	Supported as Tasks.

Moreover, we added some codes for mutual exclusion of memory buffers to execute application programs on multiple tasks. Finally, we were able to implement a sample application of TCP/IP communication by using 58KB RAM including a small margin.

3.3 Another trial for poring of KAME

KAME protocol stack is very famous for the implementation of BSD-based IPv6 and there are actual achievements in the development of various systems. Although BSD-based protocol stacks consume a lot of RAM, we tried to port KAME [21] onto the same hardware and OS before porting of lwIP.

We designed an original dynamic memory allocation mechanism for malloc and free functions by excluding virtual memory functions. In evaluation, such RAM size enabled the implementation only to initialize KAME protocol stack but not to allocate memory to handle the first received ARP packet. Therefore, we suspended evaluation of KAME on the board.

4. Evaluation and consideration

We investigated the details of RAM usage for basic TCP/IP programs and the influence of congested network. In addition, we considered the influence of stagnant data among an IP network interface and other interface processes.

4.1 Performance of CPU and network

Generally speaking, it is difficult to compare microprocessor throughput only with the standard of MIPS simply, because total performance for IP network can be decided by two primary factors as follows.

(1) CPU MIPS performance (including RAM access performance)
(2) Network interface hardware and driver performance

Current hardware technology of microprocessors and Ethernet interfaces provides enough high network throughput for practical communications, unless a high cost process such as voice codec is executed.

lwIP Wiki Site [19] reported that TCP on some implementations with device drivers saturates a 100 Mbit/s link, for example, on a Blackfin® BF536 processor @ 300 MHz (300 MIPS) and a TMS320 C6743 processor @ 200 MHz (1600 MIPS). Both of them are DSP.

A TI ARM® 32-bit Cortex™-M3 in case of 50MHz can attain processor speeds of over 60 MIPS [16]. The CPU frequency on our board is 16MHz. Therefore, the CPU performance of LM3S9U90 can afford 6.4 Mbit/s by simple calculating from MIPS.

On the other hand, our evaluation programs execute intermittent transmission of at most several tens of kilobytes considering RAM size. Therefore, we assumed that the data throughput of the board is high enough for our evaluation.

4.2 RAM usage on basic protocol sequence

4.2.1 A TCP client

We counted RAM usage in a simple TCP/IP client sample program (IPv4). RAM cells used in the TCP client process are shown in table 4 and fig.2 by classifying types defined in table 1. In this sequence, a client process transmits a request packet (several tens of bytes) and receives a response packet (383B) through a TCP connection to a web server process on a PC. Timings to count RAM usage are shown as follows.

Table 4 RAM usage in a TCP client process

Type \ Timing	(1)	(2)	(3)	(4)	(5)	(6)
(A)TCP_PCB(cells)	0	1	1	1	1	1
(B)TCP_SEG(cells)	0	0	0	1	0	1
(C)NETBUF(cells)	0	0	0	0	1	1
(D)NETCONN(cells)	0	1	1	1	1	0
(E)TCPIP_MSG_INPKT(cells)	1	1	1	1	2	2
(F)SYS_TIMEOUT(cells)	6	6	7	7	7	7
(G)PBUF_POOL(cells)	0	0	0	0	5	5
(H)MEM HEAP(bytes)	92	92	92	200	92	172

(1) Before TCP client process start
(2) After TCP socket creation
(3) After TCP connection creation
(4) After TCP request packet transmission
(5) After some TCP response packets reception
(6) After TCP connection close

The followings are the memory size and allocation behavior of each type((A)～(H)), which are observed in the evaluation programs. The number in parenthesis indicates byte size of each cell.

(A) TCP_PCB (156B): When TCP socket is created, a TCP_PCB cell is allocated for TCP connection and preserved continuously until closing timing.
(B) TCP_SEG (20B): A TCP_SEG cell is allocated in transmitting and a cell is allocated in receiving. This cell is preserved for a while after send and receive commands.
(C) NETBUF (20B): A NETBUF cell is allocated in receiving and released after closing TCP connection.
(D) NETCONN (52B): A NETCONN cell is allocated in creating TCP socket and released after closing of TCP connection.
(E) TCPIP_MSG_INPKT (20B): At first a TCPIP_MSG_ INPKT cell is allocated initially. A TCPIP_MSG_ INPKT cell is allocated in receiving and preserved for a while after close command. These cells increase or decrease a little according to the circumstances.
(F) SYS_TIMEOUT (16B): A SYS_TIMEOUT cell is allocated in connecting additionally and preserved until closing TCP connection perfectly, because the TCP connection needs management for timeout functions.
(G) PBUF_POOL (256B): When receiving packets, some PBUF_POOL cells are allocated to store received packets and will be released after TCP connection closed. PBUF_POOL cell plays a key role in RAM usage especially in case of large size packets receiving.
(H) MEM HEAP: MEM HEAP is used for miscellaneous purposes to manage data linkage, size information, and so on. Consumed memory heap increases 108 (= 200 – 92) bytes for PBUF_POOL management cells that link PBUF_POOL cells by just receiving.

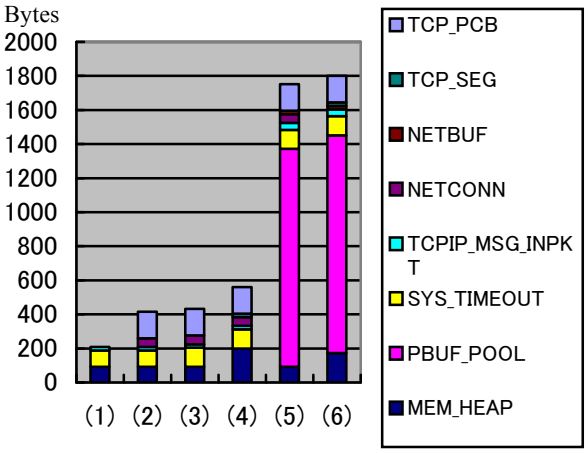

Fig. 2 Total RAM usage in a TCP client process

4.2.2 A TCP server

We also counted RAM usage in a simple TCP/IP server sample program (IPv4). RAM cells used in the TCP server process are classified by the previously mentioned types, as shown in table 5 and fig.3. In this sequence, a server process receives a request packet (several tens of bytes) and transmits a response packet (383B) through a TCP connection to a client process (a web browser) on a PC. Timings to count RAM usage are shown from (1) to (8).

The memory size and allocation behavior of each type((A)～(H)) in the case of a server evaluation program is similar to the case of a client. The large difference is that MEM_HEAP is consumed so much for transmission on the timing (8). Therefore we must consider that point for memory allocation estimation of a TCP server process on lwIP.

Table 5 RAM usage in a TCP server process

Type \ Timing	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
(A)TCP_PCB(cells)	0	1	1	0	1	1	1	1
(B)TCP_SEG(cells)	0	0	0	0	1	1	1	1
(C)NETBUF(cells)	0	0	0	0	0	0	1	0
(D)NETCONN(cells)	0	1	1	1	2	1	1	1
(E)TCPIP_MSG_INPKT(cells)	0	0	0	0	2	2	2	2
(F)SYS_TIMEOUT(cells)	6	6	6	6	7	7	7	7
(G)PBUF_POOL(cells)	0	0	0	0	2	2	4	4
(H)MEM HEAP(bytes)	92	92	92	92	92	92	92	556

(1) Before TCP server process start
(2) After TCP socket creation
(3) After TCP socket bind execution
(4) After TCP socket listen execution
(5) After TCP connection accept
(6) After source TCP connection close
(7) After TCP request packet reception
(8) After TCP response packet transmission

4.3 Network traffic

Basic protocol sequences of TCP includes retransmission processes unlike that of UDP.

Generally speaking, embedded systems with a simple application are supposed to be unused in crowded IP network, because congested IP network influences RAM usage for network buffers of network devices drastically [22]. In such cases an overflow phenomenon occurs very easily because of a small size RAM to store packet data. If an application program continues to create packets to be transmitted, a TCP/IP protocol stack needs to preserve quite a few packets for re-transmission under congested network. Then we tried to exemplify that in this evaluation.

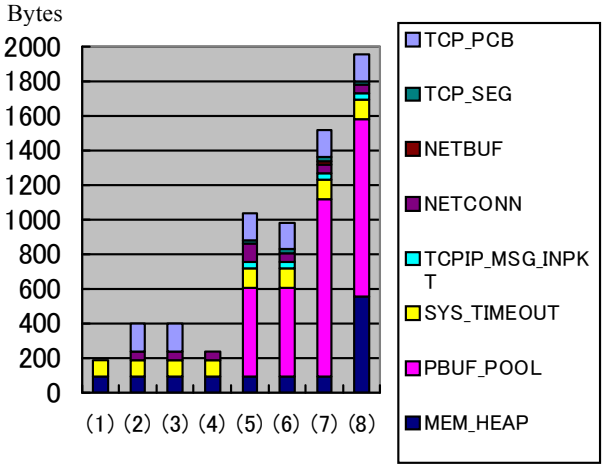

Fig. 3 Total RAM usage in a TCP server process

We measured RAM usage by the almost same TCP client program as previously evaluated. This downloads lots of data (5,954B) from a TCP server for continuous transmision, simulating a case of a congestion by delaying of TCP acknowledgements on a client. The both cases of without and with a congestion are shown in table 5 and 6.

The both of results show that TCPIP_MSG_INPKT and PBUF_POOL have more than doubled in numbers since a congestion occurs, because retransmitted packets from the TCP server increase by retransmission. Although actually some of those packets might not be received at the embedded system, enough buffer spaces should be reserved only for frequent-retransmission.

Table 6 RAM usage in a TCP client process (without congestion)

Type \ Timing	(1)	(2)	(3)	(4)	(5)
(A)TCP_PCB(cells)	0	1	1	1	1
(B)TCP_SEG(cells)	0	0	0	1	0
(C)NETBUF(cells)	0	0	0	0	1
(D)NETCONN(cells)	0	1	1	1	1
(E)TCPIP_MSG_INPKT(cells)	0	0	0	0	0
(F)SYS_TIMEOUT(cells)	6	6	7	7	7
(G)PBUF_POOL(cells)	0	0	0	0	0
(H)MEM HEAP(bytes)	92	92	92	204	92

(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
1	1	1	1	1	1	1	1
1	2	0	0	0	0	0	1
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0
2	2	3	4	5	5	5	5
7	7	7	7	7	7	7	7
9	9	9	12	15	15	15	15
92	92	92	92	92	92	92	172

(1) Before TCP client process start
(2) After TCP socket creation
(3) After TCP connection creation
(4) After TCP request packet transmission
(5) After TCP response packet [267B] reception
(6) After TCP response packet [536B] reception
(7) After TCP response packet [1072B] reception
(8) After TCP response packet [1072B] reception
(9) After TCP response packet [1072B] reception
(10) After TCP response packet [1072B] reception
(11) After TCP response packet [1072B] reception
(12) After TCP response packet [58B] reception
(13) After TCP connection close

Table 7 RAM usage in a TCP client process (with congestion)

Type \ Timing	(1)	(2)	(3)	(4)	(5)	(6)
(A)TCP_PCB(cells)	0	1	1	1	1	1
(B)TCP_SEG(cells)	0	0	0	1	0	1
(C)NETBUF(cells)	0	0	0	0	1	0
(D)NETCONN(cells)	0	1	1	1	1	0
(E)TCPIP_MSG_INPKT(cells)	0	0	0	0	0	2
(F)SYS_TIMEOUT(cells)	6	6	7	7	7	7
(G)PBUF_POOL(cells)	0	0	0	0	0	9
(H)MEM HEAP(bytes)	92	92	92	204	92	92

(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0
4	5	7	9	10	12	14	14	14	14
7	7	7	7	7	7	7	7	7	7
12	15	21	27	30	34	38	38	38	38
92	92	92	92	92	92	92	92	92	172

(1) Before TCP client process start
(2) After TCP socket creation
(3) After TCP connection creation
(4) After TCP request packet transmission
(5) After TCP response packet [267B] reception
(6) After TCP response packet [536B] reception
(7) After TCP response packet [1072B] reception
(8) After TCP response packet [1072B] reception
(9) After TCP response packet [536B] reception
(10) After TCP response packet [536B] reception
(11) After TCP response packet [536B] reception
(12) After TCP response packet [536B] reception
(13) After TCP response packet [536B] reception
(14) After TCP response packet [536B] reception
(15) After TCP response packet [58B] reception
(16) After TCP connection close

4.4 Consideration for application program implementation

There are various kinds of IP communication application for an embedded device whose basic structure is drawn in fig.4.

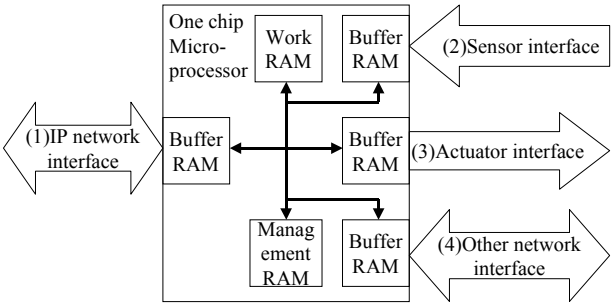

Fig. 4 Basic structure of a target embedded device with a one-chip microprocessor

(1) IP network interface: Ethernet with TCP/IP stack.
(2) Sensor interface: Measurement for temperature, humidity, illumination, magnetic field, motion, and so on.
(3) Actuator interface: DC motor, stepping motor, relay, etc.
(4) Other network interface: Serial communication such as RS-232C/422/485, USB, SATA, FireWire, GPIO, I^2C and so on.

Each interface processing program uses RAM for buffer, work, and management. As mentioned above, a TCP/IP protocol stack consumes RAM mostly for network buffers, which are designed as PBUF_POOL in lwIP.

We can estimate RAM usage for buffer handling of other interfaces similarly, but data conversion processes such as a voice codec and a video codec need a lot more work RAM than buffer RAM exceptionally. In such cases, a DSP is usually used.

This allows us to design a buffer sharing mechanism among those interfaces for the efficient use of RAM. That mechanism realizes reduction of memory copy between a TCP/IP protocol stack and other interface-processing programs. The buffer copy process needs a control mechanism to reduce stagnant data, because these communication interfaces have different speeds (fig. 5). The more different speeds they have, the larger buffer size is demanded.

One simplified computation model for buffer size is described as follows. The model assumes that data burst transmission occurs intermittently and massively computing is not executed among interfaces. Thus, necessary buffer size can be calculated by the size of concentrated data chunk.

MaxBPipout : Maximum bit rate performance of outgoing IP data
MaxBPotherout : Maximum bit rate performance of outgoing other interface data
MaxBSipout : Maximum buffer size for outgoing IP data
MaxBSipin : Maximum buffer size for incoming IP data
MaxBDipin : Maximum burst duration of incoming IP data
MaxBDotherin : Maximum burst duration of incoming other interface data

MaxBRipin : Maximum burst bit rate of incoming IP data
MaxBRotherin : Maximum burst bit rate of incoming other interface data

MaxBSipout = (MaxBRotherin - MaxBPipout) * MaxBDotherin
MaxBSipin = (MaxBRipin – MaxBPotherout) * MaxBDipin

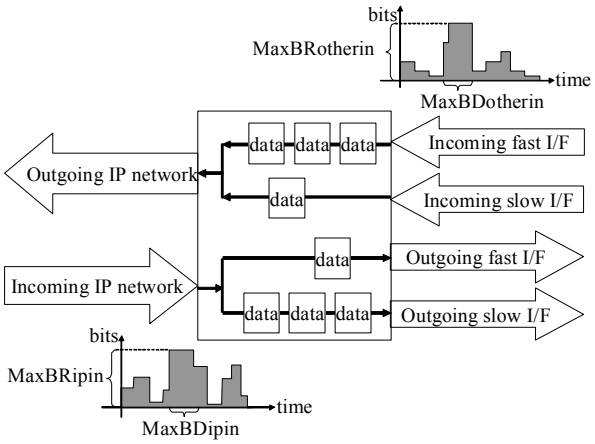

Fig. 5 Buffering control of stagnant data for communication

If MaxBSipout and MaxBSipin are lower than 0, we don't have to consider stagnant data among communication interfaces. From the results of the experiment by an IP network interface and a RS-232C serial interface, we could recognize the tendency that the stagnant data cause consumption of PBUF_POOL to increase by making the communication of RS-232C busy.

We plan to measure the above-mentioned data in detail by changing burst duration, burst bit rate, and interface throughput. In addition, the buffer size should be estimated actually by considering a congestion of IP network, which influences the stagnant data closely.

5. Conclusion

We implemented a typical embedded system with a microprocessor to investigate the details of RAM usage in a TCP/IP protocol stack for an embedded system.

We recognized the most important factor for dynamic memory allocation of TCP connection management is storing areas (PBUF_POOL, TCPIP_MSG_INPKT, MEM_HEAP) for packet data in lwIP. Those areas make up most of RAM usage, and RAM usage of other than the three types is relatively small. And a server process uses more MEM_HEAP than a client process, which cannot be ignored. MEM_HEAP is used mainly to manage buffer structures.

On the other hand, the PBUF_POOL usage increases more than a twofold by congested network. In addition it also increases by stagnant data among communication interfaces.

As a conclusion, it is important for application to avoid buffer copy among some interfaces including an IP network interface. For example, some API might be designed to construct packet data for the capability of sharing from an IP network interface and another one. To do so, we need to design unified formats of protocol data units to be shared among such interfaces.

We plan to investigate these situations in more detail, design a general buffer access mechanism among some interfaces, and establish the estimation method for RAM usage.

Reference

1) W. Richard Stevens: TCP/IP Illustrated, Volume 1: The Protocols, ISBN 0-201-63346-9, (1994)
2) W. Richard Stevens, Gary R. Wright: TCP/IP Illustrated, Volume 2: The Implementation, ISBN 0-201-63354-X, (1995)
3) W. Richard Stevens: TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols, ISBN 0-201-63495-3, (1996)
4) Kei Asai, Tsuyoshi Sato, Naoshi Sakamoto: Development Case of TCP/IP Protocol Stack (in Japanese), Design Wave Magazine 2004 September & October , CQ Publishing Co., Ltd., (Tech Village Web Site , http://www.kumikomi.net/archives/2004/12/21tcpip1.php?page=1), (2004)
5) Akihiro Shiozu, Koki Abe: Design and Implementation of Low Cost TCP/IP Protocol Stack and Its Performance Evaluation, TECHNICAL REPORT OF IEICE, IN, 106(358), pp.55-60, (Nov 9th, 2006)
6) Compact Internet Protocol Suite, http://cipsuite.sourceforge.net/
7) Tsukasa Abe, Hitoshi Yoshimura and Hiroshi Kubo: Implementation and Evaluation of TCP/IP protocol stack for embedded system, IPSJ Journal, Vol.44 No.6, pp.1583-1592 (June 2003)
8) TINET, http://www.toppers.jp/en/tinet.html
9) TOPPERS, http://www.toppers.jp/en/
10) Adam Dunkels: Full TCP/IP for 8 Bit Architectures, the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003), San Francisco, (May 2003)
11) lwIP, http://savannah.nongnu.org/projects/lwip/
12) lwIP & uIP, http://dunkels.com/adam/
13) Otsuka Yuzo, Namiki Mitaro: A development and evaluation of TCP/IP protocol stack for embedded systems, IPSJ SIG Technical Report, 2003-OS-93, pp.75-82 (2003)
14) Jian Xu: The Research and Implementation of Embedded TCP/IP Protocol Stack,The 2nd International Conference on Computer Application and System Modeling, Published by Atlantis Press, Paris, France, pp.584-587 (2012)
15) R.Manikandan: MINI TCP/IP FOR 8-BIT CONTROLLERS, Journal of Theoretical and Applied Information Technology, 30th September 2011. Vol. 31 No.2, JATIT & LLS., pp.109-112 (2011)
16) ARM Information Center, ARM Limited., http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.home/index.html
17) Yoshiaki Katayama, Hisayoshi Kurosawa, Kazuho Uemura: Basic software of W-CDMA cellular phone (in Japanese), Mitsubishi Electric Corporation Technical Report 77(2), pp. 154-157 (Feb 2003)
18) lwIP Wiki : Porting for an OS, http://lwip.wikia.com/wiki/Porting_for_an_OS
19) lwIP Wiki : Available device drivers, http://lwip.wikia.com/wiki/Available_device_drivers
20) Stellaris® ARM® Cortex™-M3 Microcontrollers, Texas Instruments Incorporated., http://www.ti.com/lsds/ti/arm/arm_cortex_m_microcontrollers/arm_cortex_m3/stellaris_arm_cortex_m3/overview.page
21) The KAME project, http://www.kame.net/
22) Christian E. Legare, Micrium: Achieving TCP/IP performance in embedded systems, http://www.iar.com/Global/Resources/Developers_Toolbox/RTOS_and_Middleware/Achieving%20TCPIP%20performance%20in%20embedded%20systems.pdf

Acknowledgments Many thanks for the members' effort to develop the embedded hardware, OS, application software, and network systems in Information Technology R & D Center of Mitsubishi Electric Corporation.