JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 10, pp. 125-133, October 2020
https://doi.org/10.9708/jksci.2020.25.10.125

TCP/IP Using Minimal Resources in IoT Systems

Seung-Chul Lee*, Dongha Shin**

*M.S Student, Dept. of Computer Science, Sangmyung University, Seoul, Korea
**Professor, Dept. of Electronics, Sangmyung University, Seoul, Korea

[Abstract]

In this paper, we design 4-layer TCP/IP that utilizes minimal memory and processor resources in Internet of Things(IoT) systems. The TCP/IP designed in this paper has the following characteristics. First, memory resource is minimized by using minimal memory allocation. Second, processor resource is minimized by using minimal memory copy. Third, the execution time of the TCP/IP can be completed in a deterministic time. Fourth, there is no memory leak problem. The standard in minimal resources for memory and processor derived in this paper can be used to check whether the network subsystems of the already implemented IoT systems are efficiently implemented. As the result of measuring the amount of memory allocation and copy of the network subsystem of Zephyr, an open source IoT kernel recently released by the Linux Foundation, we found that it was bigger than the standard in minimal resources derived in this paper. The network subsystem of Zephyr was improved according to the design proposed in this paper, confirming that the amount of memory allocation and copy were decreased by about 39% and 67%, respectively, and the execution time was also reduced by about 28%.

▸ **Key words**: TCP/IP, IoT, Minimal Resources, Zephyr, Network Subsystem

[요 약]

본 연구에서는 Internet of Things(IoT) 시스템에서 최소의 메모리 및 프로세서 자원을 사용하는 4계층의 TCP/IP에 관하여 연구하고 설계한다. 본 연구에서 설계한 TCP/IP는 다음과 같은 특징을 가지고 있다. 첫째, 메모리 할당량을 최소화하여 메모리 자원을 최소로 사용한다. 둘째, 메모리 복사량을 최소화하여 프로세서 자원을 최소로 사용한다. 셋째, TCP/IP의 수행 시간이 고정 시간에 완료될 수 있다. 넷째, 메모리 누수 문제가 발생하지 않는다. 본 연구에서 도출된 메모리 할당량 및 복사량에 대한 최소 자원 기준은 기 구현된 IoT 시스템의 통신 서브시스템이 효율적으로 구현되었는지를 점검하기 위해 유용하게 사용될 수 있다. 최근 리눅스 재단에서 발표한 공개 소스 커널인 Zephyr의 통신 서브시스템의 메모리 할당량 및 복사량을 측정한 결과, 본 연구에서 도출한 최소 자원 기준보다 더 크다는 것을 발견하였다. 본 연구에서 제안한 설계 방법에 따라 Zephyr 통신 서브시스템을 개선하여 메모리 할당량 및 복사량이 각각 약 39% 및 67% 감소함을 확인하였으며, 이에 따른 수행 시간도 약 28% 감소하였다.

▸ **주제어**: TCP/IP, 사물 인터넷, 최소 자원, Zephyr, 통신 서브시스템

• First Author: Seung-Chul Lee, Corresponding Author: Dongha Shin
*Seung-Chul Lee (chewin9@naver.com), Dept. of Computer Science, Sangmyung University
**Dongha Shin (dshin@smu.ac.kr), Dept. of Electronics, Sangmyung University
• Received: 2020. 08. 05, Revised: 2020. 09. 24, Accepted: 2020. 09. 24.

Copyright © 2020 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

I. Introduction

Internet of Things(IoT) 시스템의 통신 서브시스템은 데스크 탑 컴퓨터나 서버 컴퓨터의 통신 서브시스템과는 달리 매우 제한된 메모리 및 프로세서 자원을 사용한다. 본 연구에서는 IoT 시스템에서 최소의 메모리 및 프로세서 자원을 사용하는 통신 서브시스템에 관하여 연구하고 설계한다. 본 연구에서 설계한 통신 서브시스템은 기본적인 통신 기능만을 제공하며 구현이 편리한 TCP/IP 4계층 모델을 참조하여 데이터를 할당하는 L4(Application) 계층과 다수의 IoT 시스템에서 활용이 가능한 L3(Transport) 및 L2(Internet) 계층을 대상으로 설계하였다.

TCP/IP 4계층의 기본 동작은 각 계층에서 헤더와 데이터를 캡슐화하여 하위 계층으로 전달하는 동작으로 주로 동적 메모리 할당[1] 및 해제[2]와 메모리 복사[3]로 이뤄진다. 동적 메모리 할당 및 해제와 메모리 복사는 각각 메모리 자원과 프로세서 자원을 사용하며, 이를 최소화하여 메모리 및 프로세서 자원을 최소로 사용한다. 일반적인 동적 메모리 할당 및 해제는 수행 시간이 고정 시간에 완료되지 않는다. 본 연구에서는 안정적인 동작과 응답성을 보장하기 위해 일반적인 동적 메모리 할당 및 해제와 유사한 기능을 제공하면서도 수행 시간이 고정 시간에 완료될 수 있는 메모리 풀 방식의 동적 메모리 할당 및 해제를 사용한다. 동적 메모리 할당 및 해제를 주로 수행하는 TCP/IP 4계층은 메모리 누수 문제[4][5]가 발생할 가능성이 높다. 메모리 누수 문제는 주로 프로그래머가 동적 메모리 할당 및 해제를 수행하는 장소와 시간을 잘못 선정하여 발생한다. 본 연구에서는 동적 메모리 할당 및 해제를 수행하는 장소와 시간에 대한 규칙을 정하여 메모리 누수 문제가 발생하지 않도록 설계하였다.

본 논문의 구성은 다음과 같다. 본 논문의 2장에서는 본 연구에 필요한 사전 지식으로 현존하는 IoT 시스템의 통신 서브시스템 현황과 TCP/IP 4계층 모델을 살펴보고, 메모리 풀 방식의 메모리 할당과 메모리 누수 문제를 기술하며, 프로그램의 수행 시간을 사이클 단위로 측정하는 방법에 대하여 설명한다. 3장에서는 본 연구에서 설계한 TCP/IP 4계층의 L4, L3 및 L2 계층 설계와 본 연구에서 도출된 메모리 할당량 및 복사량에 대한 최소 자원 기준을 설명한다. 4장에서는 Zephyr 통신 서브시스템의 메모리 할당량 및 복사량을 측정하여 평가하고, 본 연구에서 제안한 설계 방법에 따라 Zephyr 통신 서브시스템을 수정 및 개선한다. 5장에서는 본 연구의 기여도와 앞으로의 연구 방향에 대하여 서술한다.

II. Preliminaries

1. Network Subsystem of Existing IoT Systems

IoT 시스템은 제한된 메모리 및 프로세서 자원을 사용하기 때문에 일반 컴퓨터에서 사용하는 풍부한 기능을 제공하는 통신 서브시스템이 아닌 기본적인 통신 기능만을 제공하는 통신 서브시스템을 사용한다. IoT 시스템의 통신 서브시스템은 복잡하고 이론적으로 치우친 OSI 7계층 모델[6]을 사용하지 않고, 구현이 편리하고 간단한 TCP/IP 4계층 모델[7]을 사용하여 설계 및 구현되어 있다. 아래 Table 1은 IoT 시스템에서 대표적으로 사용되는 커널들의 통신 서브시스템 및 프로토콜을 정리한 표[8][9]이다.

Table 1. IoT systems network subsystem

Kernel (Network Subsystem)	Application Protocol	Transport Protocol	Internet Protocol
Contiki-NG [10] v4.4 (uIP, Rime)	CoAP, MQTT	TCP, UDP	uIPv4, uIPv6, 6LoWAPN, RIME
RIOT[11] Release-2020.01 (TCP/IP)	CoAP, MQTT-SN	TCP, UDP	IPv4, IPv6, 6LoWPAN
FreeRTOS [12] v10.3.1 (FreeRTOS+TCP)	CoAP, MQTT	TCP, UDP	IPv4, IPv6, 6LoWPAN
Zephyr[13] [14] v1.14.1 (TCP/IP)	CoAP, MQTT, LWM2M	TCP, UDP	IPv4, IPv6, 6LoWPAN

네트워크 패킷은 TCP/IP 4계층 모델을 통해 송신되는 데이터의 기본 단위이며, 제어 정보가 담겨있는 헤더와 사용자 데이터가 있는 페이로드로 구성된다. 데이터 송신은 각 계층에서 헤더와 네트워크 패킷을 캡슐화하여 하위 계층으로 전달하는 과정을 반복한다. 캡슐화란 상위 계층으로부터 전달받은 네트워크 패킷을 전부 데이터로 취급하고, 해당 계층의 프로토콜 헤더를 추가하는 과정이다. 각 계층에서 캡슐화를 수행한 네트워크 패킷은 계층마다 서로 다른 명칭인 PDU(Packet Data Unit)을 사용하며, 각 계층에서 사용하는 PDU는 L4 계층은 메시지, L3 계층은 세그먼트 또는 데이터그램, L2 계층은 패킷, L1 계층은 프레임이다. 아래 Fig. 1은 캡슐화 수행 모습이다.

www.kci.go.kr

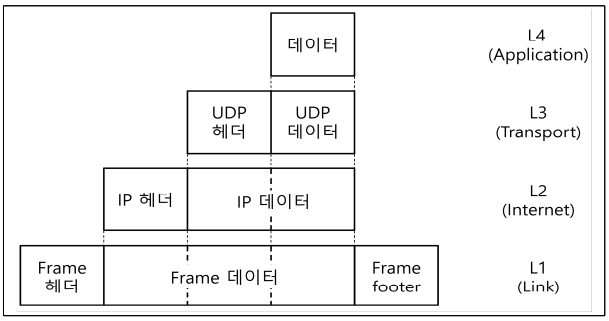

Fig. 1. Encapsulation

2. Memory Management of IoT Systems

IoT 시스템의 메모리 관리는 일반적인 내장형 실시간 커널의 메모리 관리 방법을 사용한다. 내장형 실시간 커널의 메모리 관리는 동적 메모리 할당 및 해제를 고정 시간에 수행해야 하며, 메모리 단편화(fragmentation)가 발생하지 않아야 한다. 일반적인 동적 메모리 할당 및 해제는 ANSI C에서 정의된 malloc[1] 및 free[2] 함수를 사용하는데, 기본적으로 두 함수는 고정 시간 내에 수행의 완료를 보장하지 않으며, 가변적인 크기의 메모리를 할당하여 메모리 단편화가 발생할 가능성이 있다. 이러한 문제를 해결하기 위해 IoT 시스템에서는 동적 메모리 할당 및 해제를 임베디드 시스템에서 사용하는 메모리 풀(memory pool) 방식[15]의 메모리 할당을 사용한다. 메모리 풀은 하나 이상의 메모리 파티션으로 구성되며, 메모리 파티션은 고정 크기의 메모리 블록을 가진다. 각 메모리 파티션은 구조체 MCB(Memory Control Block)로 표현된다. 구조체 MCB는 Table 2와 같은 정보를 가지고 있으며, 아래 Fig. 2는 메모리 풀의 모습이다.

Table 2. Fields of structure MCB

Fields	Describe
void *MCBAddr	Points to beginning address of memory partition
void *MCBFreeList	Points to beginning of free list of memory blocks
int MCBBlkSize	Size (in bytes) of each memory block
int MCBBlks	Total number of blocks in the partition
int MCBFree	Number of memory blocks free

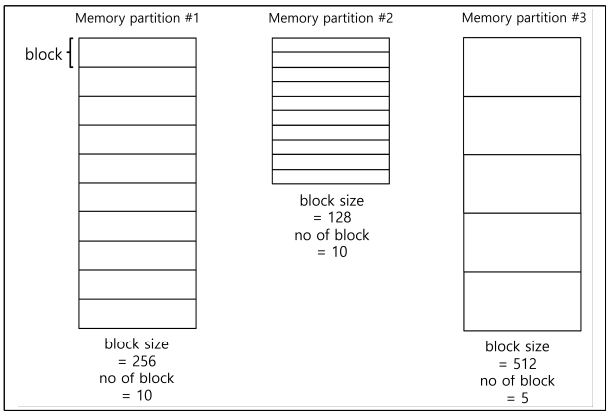

Fig. 2. Memory Pool

다음 Fig. 3, Fig. 4는 메모리 풀 방식의 메모리 할당 의사코드와 메모리 풀 방식의 메모리 해제 의사코드이다.

```
void *m_alloc(int size)
{
  mcb = suitable MCBBlkSize for size;
  if(mcb->MCBFree > 0)
  {
    mblk = mcb->MCBFreeList;
    mcb->MCBNFree--;
  }
  return mblk;
}
```

Fig. 3. Pseudocode for Memory Allocation of Memory Pool Method

```
void m_free(void *mblk)
{
  mcb = mblk's memory partition;
  mcb->MCBFreeList = mblk;
  mcb->MCBNFree++;
}
```

Fig. 4. Pseudocode for Memory Free of Memory Pool Method

메모리 풀 방식의 메모리 할당 및 해제는 미리 할당한 메모리 블록을 사용함으로써 고정 시간에 수행이 가능하며, 고정 크기의 메모리 블록을 할당하여 메모리 단편화 문제를 방지할 수 있다.

3. Memory Leak Problem

메모리 누수 문제는 동적 메모리 할당 및 해제를 주로 수행하는 프로그램에서 발생할 가능성이 높다. 메모리 누수 문제란 프로그램이 동적 메모리를 할당하여 사용한 후 할당한 동적 메모리를 해제하지 않고 계속 점유하여 메모리를 재활용할 수 없는 현상을 의미한다. 메모리 누수 문제는 다음과 같은 상황에서 발생한다.

www.kci.go.kr

- 메모리를 할당하고 사용이 끝난 후 해제하지 않는 경우
- 메모리 할당 시 사용한 인수(포인터, 크기)가 아닌 다른 인수를 사용하여 해제하는 경우
- 할당한 메모리에 대한 포인터를 잃어버린 경우

4. Measurement of Program Execution Time

본 연구에서 사용한 NXP 사의 FRDM-K64F 보드[16]는 ARM Cortex-M4 프로세서[17]가 탑재되었으며, ARM Cortex-M4 프로세서의 기본적인 instruction set[18]은 이론적으로 고정된 클럭 사이클로 수행된다. ARM Cortex-M4 프로세서의 산술 연산은 1 클럭 사이클에 수행되며, 메모리 load 및 store는 2 클럭 사이클에 수행된다. 본 연구에서는 FRDM-K64F 보드에서 TCP/IP 4계층의 수행 시간을 사이클 단위로 측정하기 위해 기 개발된 pcounter 프로그램을 사용하였다. Pcounter 프로그램은 측정하려는 프로그램의 수행 시간을 프로세서의 클럭 사이클 단위로 측정하여 출력하는 측정 프로그램으로, 사용자가 측정하려는 프로그램의 시작과 끝 지점을 지정하면 두 지점 사이의 프로그램을 수행하는 데 걸린 클럭 사이클을 측정한다.

III. TCP/IP Using Minimal Resources

1. Design Idea

본 연구에서 설계한 TCP/IP 4계층의 L1 계층은 하드웨어의 종속성으로 인해 설계하지 않고, 데이터를 할당하는 L4 계층과 다수의 IoT 시스템에서 활용이 가능한 L3 및 L2 계층을 대상으로 설계하였다.

TCP/IP 4계층의 기본 동작은 각 계층에서 헤더와 데이터를 캡슐화하여 하위 계층으로 전달하는 동작으로 주로 동적 메모리 할당 및 해제와 메모리 복사로 이뤄진다. 동적 메모리 할당과 메모리 복사는 각각 메모리 자원과 프로세서 자원을 사용한다. 각 계층에서 수행하는 동적 메모리 할당의 총합을 메모리 할당량이라 하고, 메모리 복사의 총합을 메모리 복사량이라 하며, 이를 최소화하여 메모리 및 프로세서 자원을 최소로 사용하도록 설계한다. TCP/IP 4계층에서 수행하는 동적 메모리 할당 및 해제는 ANSI C에서 정의된 malloc[1] 및 free[2] 함수를 사용하지 않고, 임베디드 시스템에서 사용하는 메모리 풀 방식의 메모리 할당 및 해제를 사용하여 TCP/IP 4계층의 수행 시간이 고정 시간에 완료될 수 있으며, 고정 크기의 메모리 블록을 할당하여 메모리 단편화 문제를 방지할 수 있다. 메모리 누수 문제가 발생하는 상황은 주로 프로그래머가 동적 메모리 할당 및 해제를 수행하는 장소와 시간을 잘못 선정하여 발생한다. 본 논문에서는 TCP/IP 4계층의 계층 구조를 활용하여 동적 메모리 할당 및 해제를 수행하는 장소와 시간에 대한 규칙을 다음과 같이 정하여 메모리 누수 문제를 방지하였다.

- 각 계층에서 할당한 메모리는 반드시 해당 계층이 끝나기 전에 해당 계층에서 해제한다.
- 메모리 할당 시 사용한 인수(포인터, 크기)를 사용하여 할당한 메모리를 해제한다.
- 할당한 메모리에 대한 포인터를 잃어버리지 않도록 관리한다.

본 연구에서 설계한 TCP/IP 4계층의 L4, L3 및 L2 계층의 설계 기준은 다음과 같다.

- 메모리 할당량의 최소화(메모리 자원 최소화)
- 메모리 복사량의 최소화(프로세서 자원 최소화)
- 메모리 풀 방식의 메모리 할당 사용(고정 시간 수행)
- 메모리 누수 문제를 방지하기 위한 규칙 사용(메모리 누수 문제 방지)

2. Implementation

본 연구에서는 5가지 기본 함수(primitive function)을 사용하여 L4, L3 및 L2 계층을 설계하였으며, 5가지 기본 함수는 다음과 같다.

- 메모리 할당: void *m_alloc(int size)
- 메모리 해제: void m_free(void *ptr)
- 메모리 복사: void m_copy(void *dst, void *src)
- 메모리 초기화: void m_init(void *dst, void *src)
- 메모리 캡슐화: void *m_encap(void *hdr, void *payload)

아래 Fig. 5, Fig. 6 및 Fig. 7은 본 연구에서 설계한 L4, L3 및 L2 계층의 의사코드이다.

```
l4_layer(user_data)
{
  size = sizeof(l4_header) + sizeof(user_data);
  message = m_alloc(size);
  m_init(message->header, l4_header);
  m_copy(message->data, user_data);
  l3_layer(message);
  m_free(message);
}
```

Fig. 5. L4 Layer Pseudocode

www.kci.go.kr

```
l3_layer(message)
{
  size_header = sizeof(l3_header);
  segment_header = m_alloc(size_header);
  m_init(segment_header->header, l3_header);
  segment = m_encap(segment_header, message);
  l2_layer(segment);
  m_free(segment_header);
}
```

Fig. 6. L3 Layer Pseudocode

```
l2_layer(segment)
{
  size_header = sizeof(l2_header);
  packet_header = m_alloc(size_header);
  m_init(packet_header->header, l2_header);
  packet = m_encap(packet_header, segment);
  l1_layer(packet);
  m_free(packet_header);
}
```

Fig. 7. L2 Layer Pseudocode

각 계층에선 해당 계층의 프로토콜 헤더와 상위 계층으로부터 전달받은 데이터를 캡슐화하여 하위 계층으로 전달하는 작업을 수행한다. 데이터 송신에 필요한 각 계층의 프로토콜 헤더와 데이터를 위한 메모리를 할당하고 데이터를 복사함으로써 설계 기준 첫 번째, 두 번째를 만족한다. 기본 함수 m_alloc은 메모리 풀 방식의 메모리 할당을 사용하여 설계 기준 세 번째를 만족한다. 또한, 각 계층에서 할당한 메모리는 해당 계층이 끝나기 전에 해제함으로써 설계 기준 네 번째를 만족한다.

3. Analysis

본 연구에서 설계한 TCP/IP 4계층의 메모리 할당량 및 복사량을 계산하면 다음과 같다. L4 계층에서 n 바이트의 message를 송신할 때, L4 계층에서 message를 위한 메모리 n 바이트와 L3 및 L2 계층 헤더를 위한 메모리 c 바이트를 할당함으로써 (n+c) 바이트의 메모리 할당량 및 L4 계층에서 message를 한번 복사함으로써 n 바이트의 메모리 복사량을 도출하였다. 여기서 c는 n과 독립적인 상수로 이론적인 최솟값은 UDP 프로토콜 사용 시 48 바이트, TCP 프로토콜 사용 시 68 바이트이다.

- 메모리 할당량: (n+c) 바이트
- 메모리 복사량: n 바이트

4. Minimal Resource Standard

앞 절에서 도출된 메모리 할당량 및 복사량을 최소 자원 기준이라 한다. 최소 자원 기준은 기 구현된 다수의 IoT 통신 서브시스템이 효율적으로 구현되었는지를 점검하기 위해 유용하게 사용될 수 있다. 기 구현된 IoT 통신 서브시스템의 메모리 할당량 및 복사량을 측정하고, 측정값이 최소 자원 기준보다 더 크다면 해당 IoT 통신 서브시스템은 개선할 여지가 남아 있다는 것을 의미한다.

IV. Improving Zephyr Network Subsystem

1. Procedure for Improvement

본 연구에서는 Zephyr 통신 서브시스템을 평가하기 위해 다음과 같은 절차를 따른다.

1. Echo, CoAP 및 HTTP 응용 프로그램을 수행하여 메모리 할당량 및 복사량을 측정한다.
2. 측정한 메모리 할당량 및 복사량이 최소 자원 기준을 만족하는지 확인한다.
3. 최소 자원 기준을 만족하지 않는다면, 이유를 찾기 위해 Zephyr 통신 서브시스템 소스 코드를 분석한다.
4. Zephyr 통신 서브시스템 소스 코드 분석을 통해 식별한 최소 자원 기준을 만족하지 않는 이유를 수정한다.
5. 소스 코드 수정을 통해 개선된 Zephyr 통신 서브시스템을 재측정한다.
6. 개선 전 Zephyr 통신 서브시스템의 메모리 할당량 및 복사량과 개선된 Zephyr 통신 서브시스템의 메모리 할당량 및 복사량을 비교한다.

2. Measurement and Verification of Zephyr Network Subsystem

본 연구에서는 최근 LTS 버전인 Zephyr v1.14.1의 통신 서브시스템을 평가하기 위해 ARM Cortex-M4 프로세서가 탑재된 NXP 사의 FRDM-K64F 보드에 MCR20A 트랜시버[19]가 탑재된 NXP 사의 FRDM-CR20A 라디오 모듈[20]을 연결하여 Echo, CoAP 및 HTTP 응용 프로그램을 수행하여 메모리 할당량 및 복사량을 측정하였다. 본 연구에서 사용한 IEEE 802.15.4 프로토콜의 MTU(Maximum Transmission Unit)는 127 바이트[21]이기 때문에 데이터의 크기를 128 바이트씩 증가시켰으며, Zephyr가 정상적으로 데이터를 송신하는지 확인하기 위해 IEEE 802.15.4 라디오 모듈이 연결된 Raspberry pi 2.0 기기에서 Linux를 수행하여 데이터를 수신한 결과 데이터가 정상적으로 송신됨을 확인하였다. 추가적으로 pcounter 프로그램을 사용하여 Zephyr 통신 서브시스템의 수행 시간을 사이클 단위로 측정하였다. 메모리 할당량 및 복사량의 측정 범위는 L4, L3 및 L2 계층이며, 수행 시간의 측정 범위는 L4 계층에서 데이터 전송을 시작할 때

www.kci.go.kr

부터 L2 계층의 수행이 마칠 때까지의 시간을 측정하였으며, 측정 결과는 다음과 같다.

Echo 및 CoAP 응용 프로그램은 UDP, IPv6, IEEE 802.15.4 프로토콜을 사용하였으며, L4 계층에서 n 바이트의 데이터를 송신할 때 (2.18n+c) 바이트의 메모리 할당량 및 (3n+d) 바이트의 메모리 복사량을 사용한다(c와 d는 n과 독립적인 상수).

HTTP 응용 프로그램은 TCP, IPv6, IEEE 802.15.4 프로토콜을 사용하였으며, L4 계층에서 n 바이트의 데이터를 송신할 때 (3.36n+c`) 바이트의 메모리 할당량 및 (4n+d`) 바이트의 메모리 복사량을 사용한다(c와 d는 n과 독립적인 상수).

최소 자원 기준은 (n+c) 바이트의 메모리 할당량 및 n 바이트의 메모리 복사량으로 Zephyr 통신 서브시스템은 최소 자원 기준을 만족하지 않는 것을 확인하였다.

3. Source Code Analysis and Modification of Zephyr Network Subsystem

Zephyr 통신 서브시스템은 아래와 같은 이유 때문에 최소 자원 기준을 만족하지 않는다. 첫째, Zephyr 통신 서브시스템은 L3 계층의 캡슐화 과정에서 물리적으로 분리되어 있는 L3 및 L2 계층 헤더와 데이터를 물리적으로 연속적인 메모리에 저장하기 위해 추가적으로 메모리를 할당하고 데이터를 복사한다. 이 과정에서 데이터의 크기만큼 메모리를 할당하고, 데이터를 복사함으로써 메모리 할당량 및 복사량이 증가하였다. 본 연구에서는 물리적으로 분리되어 있는 헤더와 데이터를 포인터를 사용하여 소프트웨어적으로 연결하여 추가적인 메모리 할당 및 데이터 복사를 제거하였다. 둘째, Zephyr 통신 서브시스템은 L2 계층에서 헤더를 압축[22]하고 압축된 크기만큼 헤더와 데이터 사이에 메모리 공간이 생긴다. 이로 인해 헤더와 데이터의 연속성이 없어지며, Zephyr 통신 서브시스템은 연속성을 유지하기 위해 메모리 공간에 데이터를 복사한다. 이 과정에서 데이터를 복사함으로써 메모리 복사량이 증가하였다. 본 연구에서는 압축된 헤더와 데이터를 포인터를 사용하여 소프트웨어적으로 연결하여 데이터 복사를 제거하였다. 셋째, Zephyr 통신 서브시스템은 TCP 프로토콜 사용 시, L2 계층에서 수행하는 헤더 압축으로 인해 헤더 정보가 변경되어 수신 호스트로부터 ack를 수신할 수 없게 된다. 이를 해결하기 위해 Zephyr 통신 서브시스템은 헤더와 데이터의 복사본을 만든다. 이 과정에서 헤더와 데이터 크기만큼 메모리를 할당하고, 헤더와 데이터를 복사함으로써 메모리 할당량 및 복사량이 증가한다. 본 연구에서는 헤더 복사본만 만들고 포인터를 사용하여 소프트웨어적으로 헤더 복사본과 데이터를 연결하여 데이터 복사본을 제거하였다. 넷째, Zephyr 통신 서브시스템은 메모리 풀 방식의 메모리 할당을 사용한다. 메모리 할당 시 메모리 블록(128 바이트)과 할당한 메모리 블록의 주소와 크기 등을 저장하는 구조체 net_buf(24 바이트)를 같이 할당하여 메모리 할당량이 증가하였다. Zephyr 소스 코드에서 구조체 net_buf는 다양한 곳에서 사용되어 구조체 net_buf 할당을 제거하는 수정은 하지 못하였다. 데이터 크기에 비례하여 구조체 net_buf를 할당하기 때문에 메모리 할당량에서 n의 기울기 값에 영향을 주었다.

4. Re-measurement Comparison of Zephyr Network Subsystem

코드 수정을 통해 개선된 Zephyr 통신 서브시스템에서 Echo, CoAP 및 HTTP 응용 프로그램을 수행하여 메모리 할당량 및 복사량과 수행 시간을 재측정하였으며 측정 결과는 아래와 같다.

UDP 프로토콜을 사용하는 Echo 및 CoAP 응용 프로그램은 L4 계층에서 n 바이트의 데이터를 송신할 때, (1.18n+c1) 바이트의 메모리 할당량 및 (n+d1) 바이트의 메모리 복사량을 사용한다. n과 독립적인 상수 c1, d1의 값은 약 300 바이트, 46 바이트이다.

TCP 프로토콜을 사용하는 HTTP 응용 프로그램은 L4 계층에서 n 바이트의 데이터를 송신할 때, (1.36n+c2) 바이트의 메모리 할당량 및 (n+d2) 바이트의 메모리 복사량을 사용한다. n과 독립적인 상수 c2, d2의 값은 약 500 바이트, 68 바이트이다.

4.1 Memory Allocation Comparison

아래 Fig. 8의 (a-1), (a-2) 및 (a-3)은 Zephyr 통신 서브시스템과 개선된 Zephyr 통신 서브시스템에서 Echo, CoAP 및 HTTP 응용 프로그램을 수행하여 측정한 메모리 할당량을 정리한 그림이다.

Fig. 8의 (a-1), (a-2) 및 (a-3)을 보면, 개선된 Zephyr 통신 서브시스템의 메모리 할당량은 최소 자원 기준의 메모리 할당량보다 n에 대한 더 큰 기울기 값을 갖는다. 이는 Zephyr 통신 서브시스템은 메모리를 관리하기 위해 구조체 net_buf를 n에 비례하여 할당하기 때문에 기울기 값이 증가하였다. 추가적으로 TCP 프로토콜을 사용하는 HTTP 응용 프로그램은 헤더 복사본과 데이터를 연결하기 위해 구조체 net_buf를 n에 비례하여 할당하여 기울기 값이 더 증가하였다. 최소 자원 기준의 메모리 할당량 (n+c) 바이트에서 c 값은 L3 및 L2 헤더의 크기이며, 이론적인 최솟값은 UDP 프로토콜 사용 시 48 바이트, TCP 프로토

www.kci.go.kr

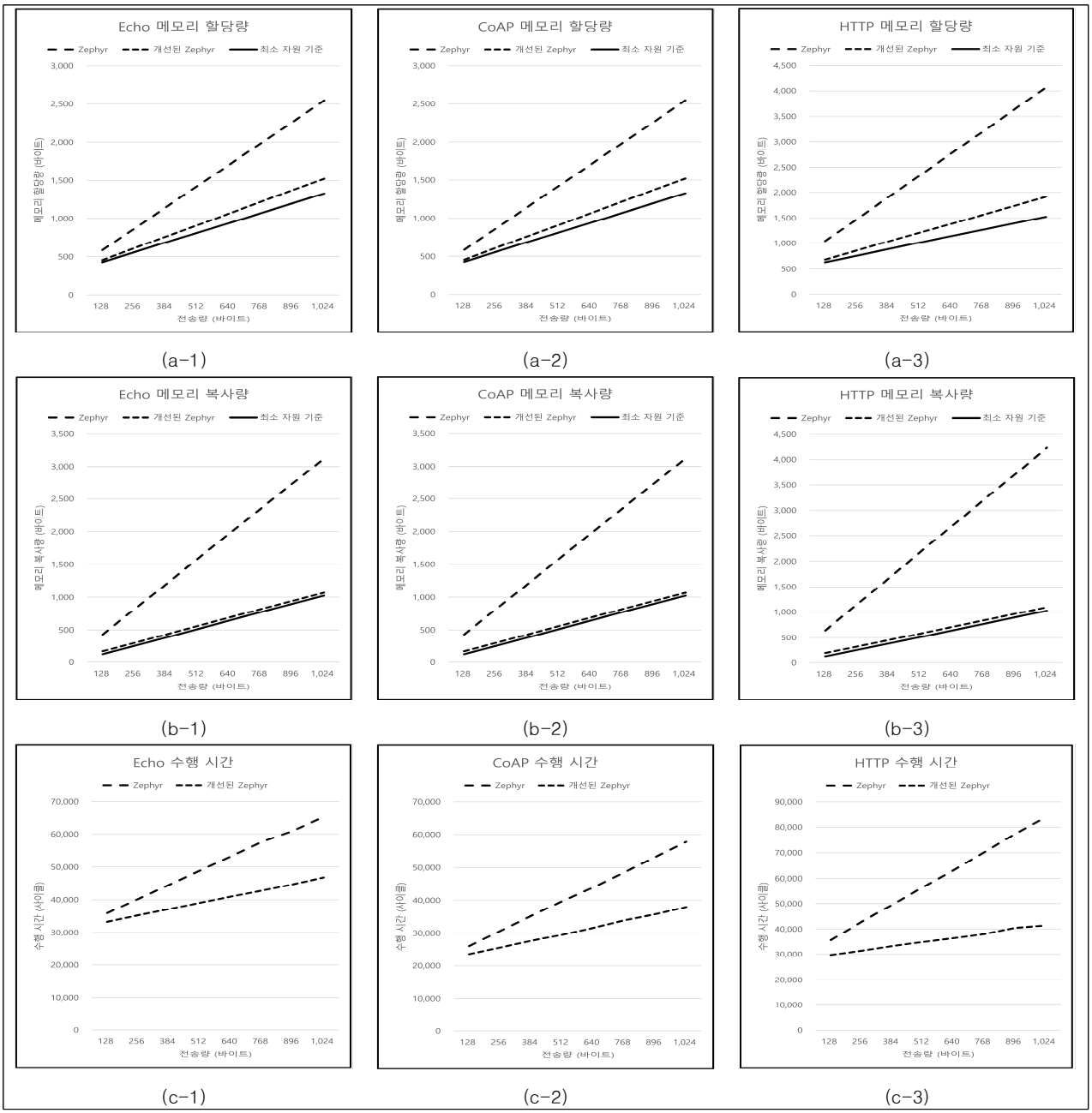

Fig. 8. Performance Comparison

콜 사용 시 68 바이트이다. Zephyr 통신 서브시스템에서는 고정 크기의 메모리 블록을 사용하여 L3 및 L2 헤더 크기보다 더 많은 메모리를 할당한다. UDP 프로토콜을 사용하는 Echo 및 CoAP 응용 프로그램의 메모리 할당량 (1.18n+c1) 바이트에서 c1 값은 약 300 바이트이며, TCP 프로토콜을 사용하는 HTTP 응용 프로그램의 메모리 할당량 (1.36n+c2) 바이트에서 c2 값은 약 500 바이트이다.

아래 Table 3은 Echo, CoAP 및 HTTP 응용 프로그램의 메모리 할당량 감소율을 비교 정리하였으며, 개선된 Zephyr 통신 서브시스템의 메모리 할당량은 개선 전 Zephyr 통신 서브시스템에 비해 약 39% 감소함을 확인하였다.

Table 3. Comparison of memory allocation reduction rate

Transmission size (bytes)	Echo (%)	CoAP (%)	HTTP (%)	Avg (%)
128	21.92%	21.92%	34.62%	26.15%
256	29.63%	29.63%	41.85%	33.70%
384	33.57%	33.57%	45.80%	37.64%
512	35.96%	35.96%	48.29%	40.07%
640	37.56%	37.56%	50.00%	41.71%
768	38.71%	38.71%	51.25%	42.89%
896	39.58%	39.58%	52.20%	43.78%
1,024	40.25%	40.25%	52.95%	44.49%
Avg	34.65%	34.65%	47.12%	38.80%

www.kci.go.kr

4.2 Memory Copy Comparison

위의 Fig. 8의 (b-1), (b-2) 및 (b-3)는 Zephyr 통신 서브시스템과 개선된 Zephyr 통신 서브시스템에서 Echo, CoAP 및 HTTP 응용 프로그램을 수행하여 측정한 메모리 복사량을 정리한 그림이다.

Fig. 8의 (b-1), (b-2) 및 (b-3)을 보면, 개선된 Zephyr 통신 서브시스템의 메모리 복사량은 최소 자원 기준의 메모리 복사량과 n에 대한 기울기 값이 동일하다. 다만 개선된 Zephyr 통신 서브시스템의 메모리 복사량은 L2 계층에서 수행하는 헤더 압축 과정에서 헤더 정보를 복사하기 때문에 메모리 복사량이 증가하였다. UDP 프로토콜을 사용하는 Echo, CoAP 응용 프로그램의 메모리 복사량 (n+d1) 바이트에서 d1 값은 약 46 바이트이며, TCP 프로토콜을 사용하는 HTTP 응용 프로그램의 메모리 복사량 (n+d2) 바이트에서 d2 값은 약 68 바이트이다.

아래 Table 4는 Echo, CoAP 및 HTTP 응용 프로그램의 메모리 복사량 감소율을 비교 정리하였으며, 개선된 Zephyr 통신 서브시스템의 메모리 복사량은 개선 전 Zephyr 통신 서브시스템에 비해 약 67% 감소함을 확인하였다.

Table 4. Comparison of memory copy reduction rate

Transmission size (bytes)	Echo (%)	CoAP (%)	HTTP (%)	Avg (%)
128	59.53%	59.53%	69.38%	62.81%
256	62.90%	62.90%	71.88%	65.89%
384	64.11%	64.11%	72.84%	67.02%
512	64.73%	64.73%	73.35%	67.60%
640	65.11%	65.11%	73.66%	67.96%
768	65.36%	65.36%	73.88%	68.20%
896	65.54%	65.54%	74.03%	68.37%
1024	65.68%	65.68%	74.15%	68.50%
Avg	64.12%	64.12%	72.89%	67.04%

4.3 Execution Time Comparison

Zephyr 통신 서브시스템은 메모리 할당량 및 복사량 이외에도 메모리 초기화, 캡슐화 동작 등을 수행하는 시간이 소요되기 때문에 추가적으로 수행 시간을 측정하였다. 위의 Fig. 8의 (c-1), (c-2) 및 (c-3)은 Zephyr 통신 서브시스템과 개선된 Zephyr 통신 서브시스템에서 Echo, CoAP 및 HTTP 응용 프로그램을 수행하여 측정한 수행 시간을 정리한 그림이다.

기존 Zephyr 통신 서브시스템의 수행 시간은 (32n+e) 사이클이며, 개선된 Zephyr 통신 서브시스템의 수행 시간은 (15n+e) 사이클이다(e의 값은 약 20,000-30,000 사이클). 이는 기존 Zephyr 통신 서브시스템은 1 바이트를 송신하기 위해 CPU 32 사이클을 사용한다는 것을 의미하며, 개선된 Zephyr 통신 서브시스템은 1 바이트를 송신하기 위해 CPU 15 사이클을 사용한다는 것을 알 수 있다(사용하는 CPU에 따라 다를 수 있음). Fig. 8의 (c-1), (c-2) 및 (c-3)을 보면, 메모리 할당량 및 복사량이 감소함에 따라 수행 시간도 감소하였고, 전송량이 커질수록 더 큰 폭으로 감소함을 볼 수 있다.

아래 Table 5는 Echo, CoAP 및 HTTP 응용 프로그램의 수행 시간 감소율을 비교 정리하였으며, 개선된 Zephyr 통신 서브시스템의 수행 시간은 개선 전 Zephyr 통신 서브시스템에 비해 약 28% 감소함을 확인하였다.

Table 5. Comparison of execution time reduction rate

Transmission size (bytes)	Echo (%)	CoAP (%)	HTTP (%)	Avg (%)
128	7.33%	9.52%	16.71%	11.19%
256	12.48%	16.72%	26.72%	18.64%
384	16.91%	21.45%	32.72%	23.69%
512	20.20%	25.73%	38.28%	28.07%
640	23.05%	28.16%	42.37%	31.19%
768	25.63%	30.22%	45.74%	33.86%
896	26.51%	32.86%	47.54%	35.64%
1024	28.26%	34.34%	50.63%	37.74%
Avg	20.04%	24.87%	37.59%	27.50%

V. Conclusion

본 연구에서는 TCP/IP 4계층 모델을 참조하여, 데이터를 할당하는 L4 계층과 다수의 IoT 시스템에서 활용이 가능한 L3 및 L2 계층을 설계하였다. 본 연구에서 설계한 TCP/IP 4계층은 다음과 같은 특징을 가지고 있다. 첫째, 메모리 할당량을 최소화하여 메모리 자원을 최소로 사용한다. 둘째, 메모리 복사량을 최소화하여 프로세서 자원을 최소로 사용한다. 셋째, 메모리 풀 방식의 메모리 할당을 사용하여 통신 서브시스템의 수행 시간이 고정 시간에 완료될 수 있다. 넷째, 동적 메모리 할당 및 해제의 장소 및 시간에 대한 규칙을 정하여 메모리 누수 문제가 발생하지 않는다. 본 연구에서 설계한 통신 서브시스템은 L4 계층에서 n 바이트의 데이터를 송신할 때, (n+c) 바이트의 메모리 할당량 및 n 바이트의 메모리 복사량이 필요하다(c는 n과 독립적인 상수).

본 연구에서 제안한 설계 방법에 따라 Zephyr 통신 서브시스템의 소스 코드를 개선 및 수정하여 메모리 할당량 및 복사량을 측정한 결과 개선 전 Zephyr 통신 서브시스템에 비해 각각 약 39% 및 67% 감소하였고, 이에 따른 수행 시간도 약 28% 감소하였다.

본 연구는 IoT 통신 서브시스템 설계에 아래와 같은 기여도를 가지고 있다. 첫째, IoT 시스템에서 최소 자원을 사용하는 통신 서브시스템 설계 방법을 제안하였다. 둘째,

www.kci.go.kr

본 연구에서 설계한 통신 서브시스템은 IoT 커널의 이미 개발된 통신 서브시스템을 평가하는데 활용할 수 있다. 셋째, 최근 개발이 활발한 Zephyr 통신 서브시스템을 개선하여 메모리 및 프로세서 자원 사용량을 줄일 수 있었다.

본 연구에서는 실험적으로 Zephyr 통신 서브시스템을 평가 및 개선하여 성능 향상을 확인하였지만, 보안과 관련된 TLS 및 DTLS 설계가 이뤄지지 않았다. 향후 연구에서는 TLS 및 DTLS 설계를 추가하고, Contiki-NG, FreeRTOS 커널의 통신 서브시스템을 분석 및 개선하여 성능을 향상시키는 것을 목표로 한다.

ACKNOWLEDGEMENT

This work was supported by Sangmyung University's School Research Fund in 2019.

REFERENCES

[1] Malloc manual, Free Software Foundation, *https://www.man7.org/linux/man-pages/man3/malloc.3.html*.

[2] Free manual, Free Software Foundation, *https://www.man7.org/linux/man-pages/man1/free.1.html*.

[3] Memcpy maual, Free Software Foundation, *http://man7.org/linux/man-pages/man3/memcpy.3.html*.

[4] J. Maebe, M. Ronsse and K.D. Bosschere, “Precise Detection of Memory Leaks”, In International Workshop on Dynamic Analysis(WODA), pp.25-31, May 2004. DOI: 10.1049/ic:20040295

[5] Y. Dong, W. Yin, S. Wang, L. Zhang and L. Sun, “Memory Leak Detection in IoT Program Based on an Abstract Memory Model SeqMM”, IEEE Access, 7, pp.158904-158916, November 2019. DOI: 10.1109/ACCESS.2019.2951168

[6] Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model, "International Organization for Standardization/International Electrotechnical Commission. ISO 7498", 1994.

[7] R. Braden, RFC 1122: Requirements for Internet Hosts-communication Layers, 1989. DOI: 10.17487/RFC1122

[8] M. H. Qutqut, A. Al-Sakran, F. Almasalha and H. S. Hassanein, “Comprehensive Survey of the IoT Open-source OSs,” IET Wireless Sensor Systems, 8(6), pp.323-339, December 2018. DOI: 10.1049/iet-wss.2018.5033

[9] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, “Operating Systems for Low-end Devices in the Internet of Things: a survey”, IEEE Internet of Things Journal, 3(5), pp.720-734, December 2015. DOI: 10.1109/JIOT.2015.2505901

[10] Contiki, *http://www.contiki-os.org*.

[11] RIOT, *https://www.riot-os.org*.

[12] FreeRTOS, *www.freertos.org*.

[13] Zephyr Project, *https://www.zephyrproject.org*.

[14] Zephyr Project Documentation, *https://docs.zephyrproject.org/1.14.1/*.

[15] Q. Li, C. Yao, “Real-time concepts for embedded systems”, CRC press, 2003. DOI: 10.1201/9781482280821

[16] NXP, "FRDM-K64F Freedom Module User's Guide Rev. 1", FRDMK64FUG, 2016.

[17] J. Yiu, "The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, 3rd Edition", Newnes, 2013. DOI: https://doi.org/10.1016/C2012-0-01372-5

[18] ARM, "ARM Cortex-M4 Processor Technical Reference Manual", Revision: r0p1, ARM 100166_0001_00_en, 2015. DOI: 10.1016/B978-0-12-382091-4.00025-6

[19] NXP, “NXP-MCR20A 2.4 GHz Low-Power Transceiver Reference Manual Rev. 3”, MCR20ARM, 2016.

[20] G. Montenegro, N. Kushalnagar, J. Hui and D. Culler, RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks, 2007. DOI: 10.17487/RFC4944

[21] NXP, “NXP-Freescale Freedom Development Board FRDM-CR20A User’s Guide Rev, 0”, FRDMCR20AUG, 2015.

[22] J. Hui, P. Thubert, RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks, 2014. DOI: 10.17487/RFC6282

Authors

Seung-Chul Lee received the B.E. degree and M.S degree in Computer Science from Sangmyung University in 2018 and 2020. His research interests include TCP/IP, IoT, Minimal Resources, Zephyr, Network Subsystem.

Dongha Shin received the B.S. degree in Computer Engineering from Kyungpook National University in 1980, the M.S. degree in Computer Engineering from Seoul National University in 1982 and the Ph.D. degree in Computer Science from University of South Carolina in 1994. During 1982-1996, he stayed in ETRI as a technical staff to study expert systems, word processing systems, file systems and language processing systems. During 1997-current, he is a professor of Dept. of Electronics in Sangmyung University. His research interests include real-time kernels, kernel protection, embedded computing and compiler implementation.

www.kci.go.kr