

Implementation of Packet Queue with Two Dimensional Array on Embedded System

Jae-Woo AHN*, Jong-Oh KIM**, Seong-Gon CHOI*

*Information & Communication Engineering, Chungbuk University, Cheongju-si Chungcheongbuk-do, Korea

**Fisys, 168, Gajeong-ro, Yuseong-gu, Daejeon, Korea

jwahni@cbnu.ac.kr, jokim@fisys.co.kr, choisg@cbnu.ac.kr

Abstract— Internet of things (IoT) and sensor network are basically aimed at wireless. Therefore, in order to use existing wired equipment in IoT system or sensor network, a system which makes wired equipment as wireless equipment is needed. So, we studied how Ethernet dedicated devices can be turned into Wi-Fi-enabled devices.

The Ethernet devices used in the research used TCP/IP, and a packet queue was needed to utilize them. We created the packet queue on different microcontroller units (MCUs). Since the heap size and the stack size are different from each other and the size of the space using the variables used is different, the available areas of the queue are also different. This paper describes the implementation of two-dimensional queue using a stack area for storing packets and confirms the effectiveness of two-dimensional queue by comparing the existing linked list and the environment without queue.

Keywords— Embedded system, Packet Queue, Circular Queue, linked list, WSN

I. INTRODUCTION

With the development of IoT and wireless sensor network (WSN), wireless communication is an important issue. Wireless communication is evolving in many areas such as healthcare, agriculture, environment monitoring, smart metering, and Vehicular ad hoc network (VANET). [1] The demand for small-scale embedded systems to drive IoT equipment and sensors is also increasing.

Research on the use of wireless embedded systems for processing temperature and humidity data in agricultural and manufacturing environments where temperature and humidity are important[2], Embedded System Scheduling and Utilization of Queues in a Wireless Local Area Network (WLAN) Environment[3], Queue management of high-speed network embedded devices[4], priority queue management on embedded system[5] and such research are being studied in the wireless environment using the embedded system.

Therefore, we studied how to convert existing wired devices into wireless devices and implemented them. In the implementation, two MCUs were used as one MCU for processing wired data and the other MCU for processing wireless data, and SPI was selected as a communication method between these MCUs. During the development process, it is necessary to process data in order to convert wired communication data to wireless. In order to do this, it is

not possible to use the existing queue, so a new queue is created.

Packet size was predictable because it was developed with the expectation of converting wired IP camera data to wireless. The heap size of the used MCU is smaller than the stack size, so we created the queue using two-dimensional array declared as a global variable.

II. RELATED WORK

The embedded system shown in Figure 1 is a system for the wired to wireless converting equipment running on the TCP/IP stack using Ethernet. Ethernet data is connected to MCU via PHY chip. PHY chip provides the analogue signal to physically access the link. It is connected to the MCU via Media Independent Interface (MII). The MCU processes the data through MAC handling, IP handling, and passes the processed data to the Wi-Fi module via SPI communication. The Wi-Fi module transmits the data which received through the SPI to the destination via the Wi-Fi access point (AP). In the MCU that communicates via Ethernet, there is a middleware that is responsible for input and output of Ethernet frames. In this middleware, a function of storing packets in a queue is performed. The data stored in the queue is packet data to be transmitted to the Wi-Fi module through the SPI. This means that when data is transmitted through the Wi-Fi module, the MAC header must be changed. The reason of this is that there is a MAC address of the PHY chip between the MCU and Ethernet, and there is a network processor between the Wi-Fi and the MCU. therefore the MAC address of the Ethernet and the MAC address of the Wi-Fi are different each other. In order to modify the MAC header, the Ethernet Middleware must store the Ethernet frame except for the

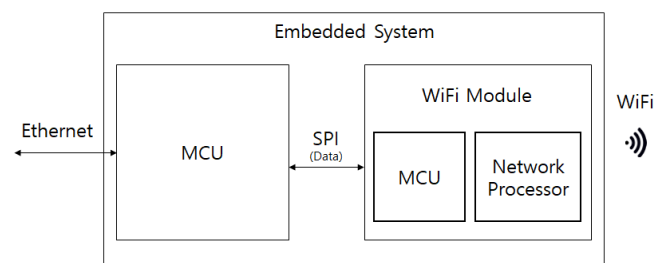


Figure 1. Wired to wireless converting system model

MAC header when storing it in the queue. MAC header is fixed to 14 bytes from the start of the Ethernet frame. For this reason, it can be easily implemented by storing the 14 bytes after the Ethernet frame in the queue. Then, the data stored in the queue is transmitted to the Wi-Fi module through SPI communication.

Ethernet processing is done in middleware, and packet transmission method in Wi-Fi module is using Raw Socket. The driver of the MCU that processes the Ethernet does not support the data transmission through Raw Socket, therefore it is implemented by modifying the middleware. However, because Wi-Fi module supports Raw Socket, we implemented Wi-Fi transmission/reception functions through Socket programming.

The raw sockets supported by MCU of Wi-Fi module support both L2 and L3. L2 Raw Socket is a socket that transmits data through selected Wi-Fi channel when data including MAC header is input. L3 Raw Socket uses data except MAC header. The MAC header is created by socket and added. And transmits the created packet. L3 Raw Socket is suitable for this system because it is aiming for one-to-one communication between specific equipment. In case of Raw Socket reception, all packets coming from connected AP are received.

III. CIRCULAR PACKET QUEUE

The size of a packet, except for a Mac, generally does not exceed 1500bytes. The device that transmits chunks of more than 1500bytes at a time is considered to be one of the devices to be used with IOT and sensor network and is designed considering only maximum transmission unit (MTU) size of 1500bytes.

The reason for excluding the MAC is to modify it because Ethernet (wired) and Wi-Fi (wireless) MCU have both MAC address and IP address when wired data goes wireless communication. That is, when the wired data is received, the MAC address and the IP address of the PHY chip are used in the MCU that processes wired data, and when the wired data is handed over to the MCU for wireless data processing through the SPI, the MAC header is completely removed. In the MCU for wireless data, the MAC header is attached to the MAC address information corresponding to the MAC address and the final destination. This function is performed by the L3 Raw Socket mentioned above. Conversely, when wireless data is transmitted by wire, the packet is transmitted to the SPI except for the MAC header of the packet received through the wireless link, and the wired-connected MCU creates MAC header to perform wired (Ethernet) transmission. The MCU can know the MAC address through the ARP that occurs when the first Ethernet connection is made. Using this information, MAC header is created and applied in middleware. Both methods allow Ethernet to Wi-Fi / Wi-Fi to Ethernet bidirectional communication. There is a problem in implementing such a system. That is, the heap memory area of the MCU responsible for wired communication is smaller than the remaining stack memory. It is impossible to expand the

area of Heap memory and design a new queue using stack memory[5].

A. Linked List Queue

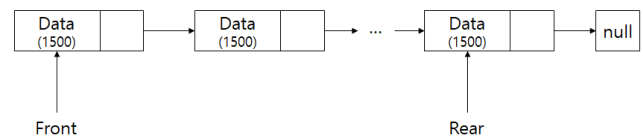


Figure 2. Linked lists structure

Linked list is a data structure that stores data in such a way that each node is connected in one line with data and pointers. As the name implies, the nodes that hold the data are connected, and the node's pointer is responsible for the connection to the next or previous node. Figure 2 shows the structure of a single linked list. In a single linked list, each node has a data space and one pointer space, and the pointer of each node points to the next node.

This structure is used in a typical embedded system. Linked list uses the heap area using Malloc. The advantage of this structure is that MCU can use all the remaining heap area. First-In-First-Out (FIFO) is easy to implement because it can manage start and end points.

B. Circular Queue

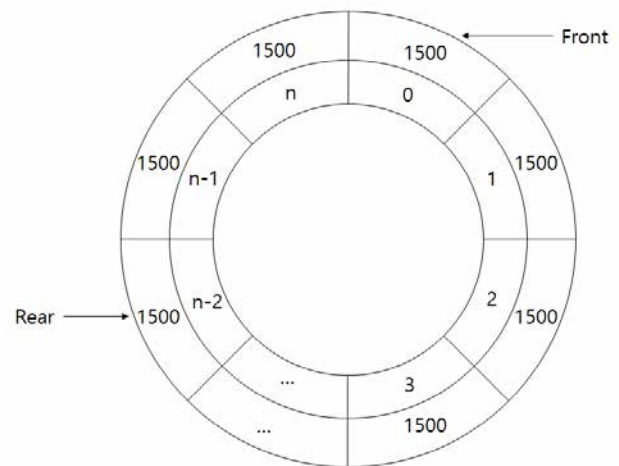


Figure 3. Circular queue structure using 2 dimensional array

When a queue is declared as an array and Enqueue and Dequeue continue to occur, there is a problem in the linear queue that an actual data space remains after reaching the last array, but overflow occurs. The circular queue is a complement to this linear queue problem. When the front reaches the end of the queue, it sends the data to the front of the queue and connects it to the circle. The problem with circular queues is that the length of the queue is fixed. However, as mentioned above, because the remaining stack memories are used, the queue length does not need to be variable and is used to store a larger number of packets.

The circular packet queue using a two-dimensional array is composed of a space where the first subscript is used as an index and a second subscript is used to store actual data. Rear and front are started with 0 index, and if Enqueue is executed, rear index is increased by 1. There is a way to know if the array is full. Before enqueueing data, if the next index of the current rear is pointing to the front, the queue is full. Conversely, if the front and rear indicate the same index, the MCU can recognize that the queue is empty. If the rear and front do not point to the same index, the MCU knows that there is data in the queue. And it can Dequeue. Dequeue transmits the data of the second subscript pointed by the front index and increases front by 1.

IV. IMPLEMENTATION

The MCU is responsible for the wired data where the proposed circular queue is needed. The MCU for wireless data processing used the existing linked list because the size of the heap area was sufficient[6].

A. Implementation Method

```
buffer[Index][Data size]

Packet arrival

if(Queue is not full)
    Copy data to buffer[Rear]
    Rear ++
    If(Rear > n(number of buffer))
        Set Rear to 0
else
    Drop packet
```

Figure 4. Pseudocode of enqueue process

Figure 4 shows the pseudocode used in the implementation. The declaration is a two-dimensional array. The first subscript is the size of the index. And the second component is 1500 because the packet size is 1500bytes or less. When a packet is received, it checks whether the queue is full and inserts the data into the rear index. After inserting the data, increase the rear index by 1 and check whether the rear index has reached the highest point of the index. If it has been reached, initialize it to zero.

Figure 5 shows the pseudocode of the Dequeue process. Dequeue starts when a packet received on Ethernet is enqueued.

```
buffer[Index][Data size]

if(Queue has data)
    Transmit data from buffer[Front]
    Front ++
    If(Front > n(number of buffer))
        Set Front to 0
else
    Waiting for data
```

Figure 5. Pseudocode of dequeue process

[ID]	Interval		Transfer	Bandwidth	
[4]	0.00-1.00	sec	110 KBytes	898 Kbits/sec	
[4]	1.00-2.00	sec	98.4 KBytes	806 Kbits/sec	
[4]	2.00-3.00	sec	111 KBytes	906 Kbits/sec	
[4]	3.00-4.00	sec	140 KBytes	1.15 Mbits/sec	
[4]	4.00-5.00	sec	102 KBytes	834 Kbits/sec	
[4]	5.00-6.00	sec	55.6 KBytes	455 Kbits/sec	
[4]	6.00-7.00	sec	108 KBytes	883 Kbits/sec	
[4]	7.00-8.00	sec	68.4 KBytes	561 Kbits/sec	
[4]	8.00-9.00	sec	55.6 KBytes	455 Kbits/sec	
[4]	9.00-10.00	sec	102 KBytes	834 Kbits/sec	
Test Complete. Summary Results:					
[ID]	Interval		Transfer	Bandwidth	
[4]	0.00-10.00	sec	0.00 MBytes	0.00 Kbits/sec	sender
[4]	0.00-10.00	sec	950 KBytes	778 Kbits/sec	receiver
CPU Utilization: local/receiver 0.5% <0.2%u/0.3%>, remote/sender 0.1% <0.0%u/0.1%>					

Figure 6. No queue

When a packet is enqueued, the front and rear indexes are not equal anymore, and then dequeue is processed. Dequeue transmits data to the Wi-Fi module through the SPI. When the dequeue is completed, increment the Front index by 1 and decide whether to initialize the index by looking at the current position of the index like Enqueue.

B. Implementation Results

We checked the performance of the proposed queue. In the case of using no queue, applying linked list, applying the proposed queue, we analyzed the performance of each case. The program used for performance analysis is iperf3. Because it was developed to target IP cameras using TCP, we measured the performance over TCP. To make the target and experiment environment similar, we proceeded by setting the Ethernet side to client and the Wi-Fi side to server. After the initial connection, the client sends data and the server sends only ack.

Figure 6 shows the performance test without queue. The speed was measured at 778 kbits/sec. Figure 7 shows the result of applying the linked list and Figure 8 shows the result using the circular queue. The linked list was measured at 2.15 Mbits/sec and the proposed circular queue at 2.46 Mbits/sec. Iperf3 operates in such a way that it transmits a number of packets at a time and waits for ack before transmitting the next data. And this is the same in IP camera environment. Thus, the larger the size of the received queue, the better the data transmission speed. As the heap area is smaller, the size of the queue is smaller than the proposed queue using the stack area.

[4]	4429.00-4430.00	sec	273 KBytes	2.23 Mbits/sec	
[4]	4430.00-4431.00	sec	270 KBytes	2.21 Mbits/sec	
[4]	4431.00-4432.00	sec	270 KBytes	2.21 Mbits/sec	
[4]	4432.00-4433.00	sec	269 KBytes	2.20 Mbits/sec	
[4]	4433.00-4434.00	sec	273 KBytes	2.24 Mbits/sec	
[4]	4434.00-4435.00	sec	273 KBytes	2.23 Mbits/sec	
[4]	4435.00-4436.00	sec	267 KBytes	2.19 Mbits/sec	
[4]	4436.00-4437.00	sec	279 KBytes	2.28 Mbits/sec	
[4]	4437.00-4438.00	sec	266 KBytes	2.18 Mbits/sec	
[4]	4438.00-4439.00	sec	269 KBytes	2.20 Mbits/sec	
[4]	4439.00-4440.00	sec	272 KBytes	2.23 Mbits/sec	
[4]	4440.00-4441.00	sec	274 KBytes	2.24 Mbits/sec	
[4]	4441.00-4441.53	sec	142 KBytes	2.19 Mbits/sec	

[ID]	Interval		Transfer	Bandwidth	
[4]	0.00-4441.53	sec	0.00 Bytes	0.00 bits/sec	sender
[4]	0.00-4441.53	sec	1.11 GBytes	2.15 Mbits/sec	receiver
iperf3: interrupt - the client has terminated					

Figure 7. Linked list

```

[ 41 5991.00-5992.00 sec 320 KBytes 2.62 Mbits/sec
[ 41 5992.00-5993.00 sec 321 KBytes 2.63 Mbits/sec
[ 41 5993.00-5994.00 sec 299 KBytes 2.45 Mbits/sec
[ 41 5994.00-5995.00 sec 286 KBytes 2.34 Mbits/sec
[ 41 5995.00-5996.00 sec 317 KBytes 2.60 Mbits/sec
[ 41 5996.00-5997.00 sec 321 KBytes 2.63 Mbits/sec
[ 41 5997.00-5998.00 sec 319 KBytes 2.61 Mbits/sec
[ 41 5998.00-5999.00 sec 321 KBytes 2.63 Mbits/sec
[ 41 5999.00-6000.00 sec 309 KBytes 2.53 Mbits/sec
[ 41 6000.00-6000.54 sec 166 KBytes 2.52 Mbits/sec
-----
Test Complete. Summary Results:
[ ID] Interval Transfer Bandwidth
[ 41 0.00-6000.54 sec 0.00 Bytes 0.00 bits/sec sender
[ 41 0.00-6000.54 sec 1.72 GBytes 2.46 Mbits/sec receiver
CPU Utilization: local/receiver 0.6% <0.2%/0.4%>, 0%>
remote/sender 0.0% <0.0%/0.0%>
iperf3: interrupt - the client has terminated

```

Figure 8. Circular queue

Therefore, it can be seen that the proposed scheme can store a larger number of packets and that the transmission speed is also faster.

V. CONCLUSIONS

In this paper, we have dealt with a solution to the queuing problem, which is one of the problems encountered in implementing an embedded system that converts wired device to wireless device for use in IoT / sensor networks. According to the characteristics of the MCU, the space size of the HEAP area and the STACK area was different. To solve this problem, the implementation of the queue using the STACK area was carried out in the form of a circular queue using a two-dimensional array.

As a result of the implementation, it was confirmed to be 778 kbits/sec for no queue, 2.15 Mbits/sec for linked list, and 2.46 Mbits/sec for proposed queue. Based on the experimental results in a situation where the heap area is narrow, utilizing the stack area as much as possible is a solution.

The problem to be solved in the future is that packets smaller than 1500 bytes occupy an array of 1500 bytes size. This problem was since the size of the transmission data of the SPI could not be made variable and remained for convenience of implementation. It should be improved in the future.

ACKNOWLEDGMENT

This work was supported by “Human Resources Program in Energy Technology” of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade, Industry & Energy, Republic of Korea. (No. 20164030201330).

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(2018-0-01396) supervised by the IITP(Institute for Information & communications Technology Promotion).

*corresponding author is S.G. Choi (choisg@cbnu.ac.kr)

REFERENCES

- [1] Evolution of Wireless Sensor Networks towards the Internet of Things: a Survey, Luca Mainetti, Luigi Patrono, Antonio Vilei, “SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks, 2011.
- [2] Implementation of Wireless Temperature and Humidity Monitoring on an Embedded Device, Muhamad F. A. Samusudin, Roslina Mohamad, Nuzli Mohamad Anas, Hafizal Mohamad, 2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), 2018.
- [3] Scheduling disciplines for multimedia WLANs: Embedded round robin and wireless dual queue, Ravindra S. Ranasinghe, Lachlan L.H. Andrew, David A. Hayes, David, Everitt, ICC 2001 IEEE International Conference on Communications, Conference Record, 2001.
- [4] Architecture of an Embedded Queue Management Engine for High-Speed Network Devices, M. Alisafae, S. M. Fakhraie, M. Tehranipoor, 48th Midwest Symposium on Circuits and Systems, 2005.
- [5] STM32F215XX, STM32F217XX Datasheet – production data, STMicroelectronics, 2016
- [6] CC3220 SimpleLink Wi-Fi and Internet of Things Network Processor Programmer’s Guide, Texas Instruments, 2018



Jae Woo Ahn received B.S. degree in the College of Electrical & Computer Engineering, Chungbuk National University, Korea in 2017. He is currently a M.S. candidate in School of Electrical & Computer Engineering, Chungbuk National University. His research interest is network programming.



Jong Oh Kim received B.S. and M.S. degree in Electronics Engineering from Kyeongbuk National University in 1990 and 1992 respectively. He is currently CEO in Fisis Inc. His research interests is SDN, NFV.



Seong Gon Choi received B.S. degree in Electronics Engineering from Kyeongbuk National University in 1990, and M.S. and Ph.D. degree from KAIST, Korea in 1999 and 2004, respectively. He is currently a professor in the College of Electrical & Computer Engineering, Chungbuk National University. His research interests include smart grid, IoT, mobile communication, high-speed network architecture and protocol