The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 19, No. 5, pp.63-71, Oct. 31, 2019. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2019.19.5.63
JIIBC 2019-5-9

IIoT용 우선순위 토픽 기반 MQTT에 관련한 연구

A Study on MQTT based on Priority Topic for IIoT

오세춘*, 김영곤**

Se-Chun Oh*, Young-Gon Kim**

요 약 4차 산업혁명시대를 맞이하여 스마트팩토리의 구축에 관한 많은 연구가 진행되고 있다. 이러한 스마트팩토리의 구축과 관련된 다양한 기술들 중에서 핵심기술 중의 하나는 데이터의 송수신을 처리하는 IoT용 프로토콜 부문이다. 이와 관련하여 일반적으로 MQTT 프로토콜이 가장 많이 사용되고 있으나 기존의 MQTT 기술은 메시지의 우선순위 개념이 없기 때문에 실시간성을 요구하는 산업용 현장에 적용하기에는 다소 부족한 면이 있다. 특히 특정 설비의 고장발생에 따른 관련 설비 전체의 비상정지 등과 같은 긴급 상황에서는 긴급한 메시지의 우선순위 처리가 매우 중요하다. 이를 개선하기 위해 우선순위 기반 MQTT에 관한 연구도 일부 진행되고 있으나 이러한 연구들은 MQTT 표준 규격을 변형한 방식이기 때문에 실제 현장에서 사용하기에는 문제점을 안고 있다. 따라서 본 연구에서는 MQTT 표준을 준수하면서도 메시지의 우선순위 처리가 가능한 MQTT에 관련한 연구를 실시하고 이를 검증한다.

Abstract Recently, there has been a lot of research on the construction of smart factory in the 4th Industrial Revolution era. Among the various technologies involved in the deployment of smart factory, one of the key technologies is the IoT protocol sector that handles the transmission and reception of data. In this regard, the MQTT protocol is generally used most commonly, but the existing MQTT technology lacks the concept of priorities of messages, so it is somewhat insufficient to be applied to an industrial field requiring real-time property. Priority handling of urgent messages is critical, especially in emergency situations, such as the emergency shutdown of the entire relevant facility following the failure of a particular facility. To improve this, research on priority-based MQTT is being conducted somewhat, but these studies have problems with actual field use because they are a variant of the MQTT standard. Therefore, this study conducts and verifies studies related to MQTT, which can prioritize messages while adhering to existing MQTT standards.

Key Words : MQTT, IIoT, Priority, mosquitto, SmartFactory

Ⅰ. 서 론

IoT(Internet of Things)는 지능화된 각종 사물들이 네트워크를 통해 서로 연결되어 사물들 간의 통신, 사물과 인간 간의 통신 등을 지원하여 다양한 부가가치를 창출하는 기술이다. IoT는 기존의 소셜 네트워크 서비스는

*정회원, 한국산업기술대학교 컴퓨터공학부
**정회원, 한국산업기술대학교 컴퓨터공학부(교신저자)
접수일자 2019년 7월 15일, 수정완료 2019년 9월 3일
게재확정일자 2019년 10월 4일

Received: 15 July, 2019 / Revised: 3 September, 2019 / Accepted: 4 October, 2019
**Corresponding Author: ykkim@kpu.ac.kr
Dept. of Computer Engineering, Korea Polytechnic University, Korea

물론이고 스마트팩토리(SmartFactory), 스마트팜(SmartFarm) 등의 다양한 분야에 새롭게 적용되고 있다. 이러한 대부분의 시스템들은 제한된 네트워크 및 컴퓨팅 능력의 환경에서 동작되기 때문에 경량의 메시지 전송 프로토콜의 사용이 필수적이다. 이러한 환경에서 사용될 수 있는 IoT용 경량 메시지 전송 프로토콜은 MQTT(Message Queuing Telemetry Transport), CoAP(Constrained Application Protocol), XMPP(eXtensible Messaging and Presence Protocol) 등이 있다. CoAP의 경우 end-to-end 프로토콜이면서 UDP(User Datagram Protocol) 기반의 프로토콜이란 점이 단점이며, XMPP의 경우 메시지의 전달여부를 확인하는 QoS(Quality of Service) 기능이 없다는 단점을 가지고 있다. 따라서 근래의 대부분의 IoT 시스템에서는 이러한 단점들을 극복할 수 있는 MQTT 프로토콜이 가장 많이 사용되어지고 있다[1].

4차 산업혁명 시대를 맞이하여 다양한 산업분야에서 새로운 변화가 일어나고 있으며 최근에는 그 중에서도 공장의 지능화를 추구하여 생산성의 혁신을 가져올 수 있는 스마트팩토리 분야가 많은 연구의 대상이 되고 있다[2][3][4]. 스마트팩토리 분야뿐만 아니라 IoT 기술을 적용하고 있는 많은 산업분야에서는 특히 정보의 실시간성, 가용성, 효율성 등이 매우 중요한 요소인데 이러한 특징을 갖는 산업분야에 적용되는 IoT 기술을 IIoT(Industrial IoT)라고 부른다. 그러나 현재 대부분의 산업용 시스템이 기존의 IoT용 MQTT를 그대로 사용하고 있어서 우선순위를 갖는 메시지의 긴급처리 기능이 존재하지 않는 문제점이 있다. 제조현장에서 특정 설비의 이상상태 발생 시 관련 현장설비들의 긴급정지가 필요한 비상상황, 의료용으로는 환자의 이상상태에 대한 정보전송 등이 매우 중요한 우선순위로 먼저 처리해야할 대표적인 예이다.

따라서 최근에는 MQTT에서 우선순위 메시지 처리를 지원하기 위한 다양한 연구[5][6][7]들이 진행되고 있으나 대부분의 연구는 MQTT의 고정 헤더 부분을 수정하거나 추가하여 우선순위 필드를 만들고 이를 이용해 우선순위가 높은 메시지 패킷을 우선적으로 처리하는 방식이라서 다른 상위 어플리케이션에서 MQTT 메시지를 활용할 수 없게 되는 치명적인 문제가 발생할 수 있다. 또한 대부분의 연구에서는 입력된 메시지의 우선순위 처리를 위해 복수개의 입력 우선순위 큐를 사용하고 있는데 이러한 경우 복수개의 큐 관리 등으로 인해 많은 CPU 리소스를 사용하는 문제가 있다.

따라서 본 연구에서는 MQTT 표준을 준수하면서 우선순위를 사용하는 기법과 단일 큐를 사용하여 복수 큐 방식보다 상대적으로 적은 CPU 리소스를 사용하면서도 메시지의 우선순위 처리를 할 수 있는 'multi-scanned priority sorting 알고리즘'을 제안한다.

II. 관련연구

1. MQTT 메시지 전달 방식

MQTT는 IoT와 같이 제한된 대역폭을 갖는 통신 환경에 적합한 메시지 전송 프로토콜로서 적은 코드로도 구현이 가능하며 클라이언트/서버의 구조가 아닌 메시징(messaging) 개념을 이용한 푸시 기술을 사용하고 있다. 즉 그림 1에서 처럼 MQTT 브로커(broker)를 중심으로 메시지의 생성 및 송신을 담당하는 publisher와 메시지를 수신만 하는 subscriber로 구성되며, publisher에서 생성된 모든 메시지는 브로커에 의해 취합되고 브로커는 특정 메시지의 수신을 희망하는 모든 subscriber들에게 해당 메시지를 전송해주는 구조를 갖는다. 따라서 브로커는 중간에서 중재기능을 담당하고 있기 때문에 publisher 기능과 subscriber의 기능을 모두 갖고 있다.

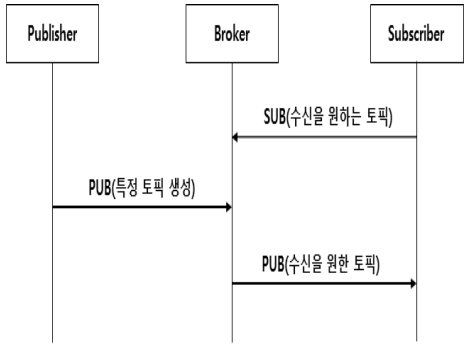

그림 1. MQTT 기본 구성
Fig. 1. Basic structure of MQTT

모든 메시지들은 토픽(topic)이라는 메시지의 이름과 필요할 경우 추가되는 데이터 값(payload)으로 구성된다. 토픽은 '/' 기호를 사용하여 계층구조로 형성할 수 있다. 그림 2는 MQTT의 동작 구조이며 좌측의 publisher들에서 생성된 '센서/온도' 메시지 및 '센서/습도' 메시지는 브로커로 모아지며, 브로커에서는 우측의 subscriber들 중에서 특정 메시지의 수신을 희망하는 모든 subscriber들에게 해당 메시지를 개별적으로 전달하게

된다. 그림 2 우측 상단의 subscriber는 센서와 관련된 모든 메시지들을 수신하기 위해 수신 희망 메시지를 '센서/*'의 형태로 브로커에 등록하여 모든 센서 관련 메시지들을 브로커로부터 수신할 수 있다. 우측 하단 클라이언트의 경우는 publisher와 subscriber의 기능을 하나의 클라이언트에서 모두 가지고 있는 경우이다.

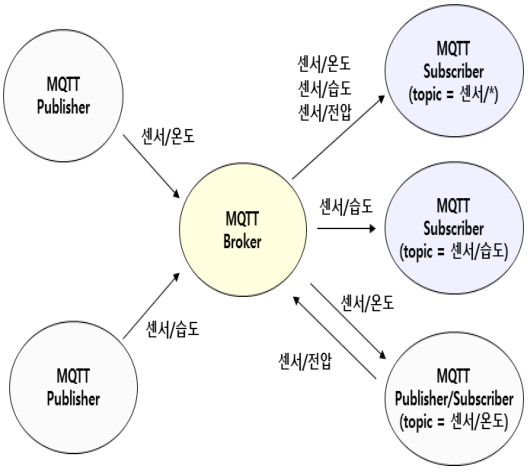

그림 2. MQTT의 동작 구조
Fig. 2. Example of MQTT operation

2. MQTT 메시지 규격

MQTT는 TCP/IP 프로토콜을 통해 패킷을 주고받는데 MQTT 프로토콜 규정[8]에 의하면 첫 2 바이트는 항상 고정 헤더로 사용하도록 규정하고 있다.

그림 3은 2 바이트 크기 고정 헤더의 형식이다. 첫 번째 바이트에 있는 MQTT Control Packet Type은 해당 MQTT 패킷의 종류를 정의하는 필드이며, 이 4 비트 필드의 구성에 따라서 현재 처리하고 있는 패킷이 MQTT에서 정의한 14 종류의 패킷(그림 4) 중에서 어떤 종류인지를 판단해 이에 따른 적절한 조치를 취하게 한다. 첫 번째 바이트의 나머지 하위 4 비트는 MQTT의 패킷 종류에 따라서 다르게 구성되는 부분이며, 두 번째 바이트는 고정 헤더 이후에 존재하는 정보의 총 바이트 수를 지정한다. 'PUBLISH' 또는 'SUBSCRIBE' 등의 메시지 송수신과 관련된 패킷일 경우 뒤에 따라 올 수 있는 정보는 토픽 또는 토픽과 페이로드로 구성되는 메시지 정보이다.

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type				개별 MQTT Control Packet type에 따른 Flag			
Byte 2	남은 길이 (Remaining length)							

그림 3. MQTT 고정 헤더 형태
Fig. 3. Fixed header of MQTT

MQTT Control Packet Type 필드				패킷 종류	MQTT Control Packet Type 필드				패킷 종류
0	0	0	0	Reserved	1	0	0	0	SUBSCRIBE
0	0	0	1	CONNECT	1	0	0	1	SUBACK
0	0	1	0	CONNACK	1	0	1	0	UNSUBSCRIBE
0	0	1	1	PUBLISH	1	0	1	1	UNSUBACK
0	1	0	0	PUBACK	1	1	0	0	PINGREQ
0	1	0	1	PUBREC	1	1	0	1	PINGRESP
0	1	1	0	PUBREL	1	1	1	0	DISCONNECT
0	1	1	1	PUBCOMP	1	1	1	1	Reserved

그림 4. MQTT 패킷 종류
Fig. 4. Control packet types of MQTT

그림 5는 본 연구에서 집중적으로 다루게 될 'PUBLISH' 패킷에 대한 고정 헤더를 보여주고 있다. RETAIN 필드는 메시지의 저장 여부를 선택하는 필드인데 이 필드가 '1'로 된 메시지들은 브로커 내부에 저장을 한 후 전송을 하게 된다. 메시지를 저장하는 이유는 해당 메시지가 전송된 이후에 해당 메시지의 수신을 원하는 새로운 subscriber가 생길 경우에도 브로커가 해당 메시지를 새로운 subscriber에게 보내주기 위한 용도로 사용하기 때문이다. QoS 필드는 'PUBLISH' 패킷을 이용해 메시지를 전송할 경우 수신측의 수신여부를 어떤 방법으로 확인할 것인 지를 결정하는 필드인데 QoS는 0~2 까지 총 3단계가 있다. DUP 필드는 현재 보내고 있는 메시지가 신규 전송인지 또는 재전송인지를 알려주는 필드이다. QoS가 1 이상인 경우에는 'PUBLISH' 패킷을 받은 수신측에서는 패킷을 잘 받았다는 의미로 송신측으로 'PUBACK' 또는 'PUBREC' 패킷을 특정시간 내에 전송해 주어야 하는데, 특정시간 내에 이 패킷이 도착하지 않을 경우 송신측은 전송 실패로 간주하고 다시 'PUBLISH' 패킷을 보내는데 이때 재전송임을 알리기 위해 DUP 필드를 '1'로 설정해서 보낸다.

Bit	7	6	5	4	3	2	1	0
Byte 1	0	0	1	1	DUP	QoS		RETAIN
Byte 2	남은 길이 (Remaining length)							

그림 5. 'PUBLISH' 패킷의 고정 헤더
Fig. 5. Fixed header of 'PUBLISH' packet

3. 우선순위 기반 MQTT에 대한 기존의 연구

메시지의 우선순위를 지원하는 MQTT를 구현하기 위해서는 크게 두 가지 분야에 대한 연구가 필요하다. 첫째로는 publisher에서 브로커로 전송되는 메시지에 대해서 어떠한 형태로 우선순위를 표시해 전송할 것인가에 대한 연구 분야며, 두 번째로는 브로커의 입력단에서 복수개의 publisher에서 전송되어 처리를 대기하고 있는 메시지들을 대상으로 이를 분석하여 우선순위가 높은 메시지를 우선적으로 처리하는 연구 분야다.

패킷의 내부에 해당 메시지의 우선순위를 기록하기 위해서 대부분의 연구들은 'PUBLISH' 패킷의 고정 헤더 부분을 수정하는 방식을 사용하고 있다. 즉 고정 헤더의 일부를 변형하여 우선순위 필드를 추가하거나[5], 고정 헤더의 크기를 기존의 2 바이트에서 3 바이트로 임의 확장하여 우선순위 필드를 적용하는 방식[6]도 있다. 이와는 다르게 그림 4에서 알 수 있듯이 MQTT에서 규격으로 지정한 14개의 패킷 종류 이외에 예약(Reserved)된 패킷 2 종류(패킷 종류 0번, 15번)를 우선순위 처리용 패킷으로 활용하는 방식[7]도 있다. 그러나 이러한 방식들로 메시지 우선순위 처리 방식을 구현할 경우 MQTT 표준 규격에서 벗어나기 때문에 Node-RED 또는 AWS 등의 수많은 어플리케이션 프로그램들과 호환되지 않을 가능성이 매우 높아 바람직하지 않은 접근 방법이다.

다음으로 복수개의 publisher에서 전송된 메시지들을 MQTT 브로커의 입력단에서 메시지 우선순위에 따라 순서를 바꾸어 처리할 필요가 있는데 이를 구현하기 위해서 대부분의 연구[5][6][7]에서는 우선순위 종류만큼 복수개의 우선순위 큐를 별도로 만들어서 메시지의 우선순위에 따라 해당 우선순위 큐로 메시지를 넣고 브로커가 메시지를 처리할 때 우선순위가 높은 큐의 메시지들부터 먼저 처리한 후 낮은 우선순위의 큐의 메시지들을 처리하는 방식이 대부분이다. 복수개의 우선순위 큐를 구성하여 처리하는 방식에 있어서 우선순위가 낮은 큐의 지나친 처리지연을 방지하기 위해 'Weight Round Robin(WRR)' 방식의 메시지 스케줄링 방식에 대한 연구[7]도 있으나 이러한 방법은 복수개의 큐를 생성, 관리하고 스케줄링을 해야 하기 때문에 단일 큐 방식보다 상대적으로 많은 CPU 리소스가 필요하다는 단점이 있다.

Ⅲ. 제안하는 우선순위 MQTT 알고리즘

3장에서는 기존의 연구들에서 갖는 문제점을 해결한 새로운 우선순위 기반 MQTT 방식을 제안한다. 첫 번째 제안은 기존의 MQTT 프로토콜에 충실한 우선순위의 설정 및 전달방법에 대한 내용이며, 두 번째 제안은 브로커의 입력단에서 순차적으로 입력된 복수개의 메시지들을 단일 큐를 사용하여 복수 큐 방식보다 상대적으로 적은 CPU 리소스를 이용하면서 우선순위에 따른 순서대로 메시지를 처리하는 'multi-scanned priority sorting 알고리즘' 대한 내용이다.

1. 우선순위의 설정 및 전달 방법

'PUBLSIH' 패킷에서 우선순위 정보를 실어서 전송하기 위해서 본 연구에서는 기존의 고정 헤더 필드를 변형하거나 추가하는 방식 대신에 토픽 문자열의 첫 글자에 우선순위를 부여하는 새로운 방식을 제안한다.

MQTT 규격에 의하면 토픽을 구성하는 문자열은 MQTT 버전 5에서 새롭게 제시된 'Topic Alias' 기능을 적용하는 토픽을 제외하고는 모두 UTF-8 코드의 문자열로 구성되어야 하며, UTF-8 코드 중에서도 non-ASCII 형식의 코드와 제어용 코드가 아닌 일반 문자 코드 부분만을 토픽의 문자열 구성에 사용할 것을 권고하고 있다. 따라서 토픽 문자열을 구성할 때는 '+', '/', '#', '$' 등의 몇 가지 MQTT 표준에서 예약(Reserved)된 문자와 제어용 코드 이외의 대부분의 ASCII 형식의 문자들을 이용해서 구성할 수 있는데, 본 연구에서는 이 중에서 잘 사용되지 않는 몇 개의 문자를 우선순위 결정용 문자로 지정하고 이 문자를 토픽 문자열의 첫 글자로 활용함으로써 해당 토픽의 우선순위를 부여하는 방법을 제안한다.

예를 들어 그림 6에서처럼 잘 사용하지 않는 '^' 문자가 토픽의 첫 문자일 경우 가장 높은 우선순위의 토픽으로 간주하고, '_'가 토픽의 첫 문자인 경우에는 다음 우선순위의 토픽이며 그 이외의 문자가 토픽의 첫 문자인 경우에는 일반 우선순위 토픽으로 정의하는 방식이다. 이러한 방식은 기존의 MQTT 표준을 준수하면서도 'PUBLISH' 패킷 내에 해당 토픽의 우선순위를 실어서 보낼 수 있는 간단하면서도 확실한 방식이다.

^topic1/sub-1/sub-2 → 우선순위 가장 높음
_topic2/sub-1/sub-2 → 우선순위 2번째 높음
topic3 → 우선순위 없음
topic4/sub-1 → 우선순위 없음
topic5/sub-1/sub-2/sub3 → 우선순위 없음

그림 6. 토픽의 첫 문자에 의한 우선순위 부여 방식의 예
Fig. 6. Example of prioritizing by first character in a topic

2. 브로커에서의 우선순위별 처리 방법

본 연구에서 제안하는 두 번째 부분은 브로커의 입력단에 적용되며 복수개의 publisher들로부터 순차적으로 입력받고 처리를 기다리고 있는 복수개의 메시지들을 우선순위에 맞게 처리해 주는 'multi-scanned priority sorting 알고리즘' 대한 내용이다.

MQTT 브로커용 소프트웨어는 mosquitto, RSMB, HiveMQ, RabbitMQ 등 매우 다양한 종류가 있으나 최근에는 오픈소스 프로젝트인 mosquitto를 가장 많이 사용하고 있다. 따라서 본 연구에서도 mosquitto를 기반으로 알고리즘을 검토 및 구현하였다. Mosquitto 프로그램의 컴파일 시에 EPOLL 옵션을 선택하여 제안된 알고리즘을 구현했는데 만약 SELECT 옵션을 선택할 경우라도 제시된 알고리즘의 기본 개념을 동일하게 적용하면 특별한 문제없이 쉽게 적용이 가능하다.

3. 알고리즘의 세부 동작 설명

그림 7은 본 연구에서 제안하는 우선순위 기반의 메시지 처리방식인 'multi-scanned priority sorting 알고리즘' 대한 흐름도이다.

그림 7의 좌측부분은 입력처리를 기다리는 패킷들에 대하여 우선순위 정보를 판단하고 정리하는 부분이다. Publisher들이 브로커로 패킷을 보내면 이 패킷들은 도착되는 순서대로 운영체제에서 관리하는 소켓(socket)들의 수신버퍼에 차례대로 저장된다. 따라서 입력되는 메시지들에 대해서 우선순위에 따른 순서로 처리하기 위해서는 mosquitto의 입력처리 부분에 본 연구에서 제안하는 알고리즘을 적용해야 한다.

첫 번째 단계는 epoll_event() 함수를 이용해 입력 처리를 기다리는 패킷의 총 개수인 n을 받고 이를 이용하여 해당 패킷의 우선순위를 저장하는 prio_flag[n-1]의 구조체를 생성하는 단계이다. 이때 만약 대기 중인 패킷의 개수 n이 1개 이하일 경우에는 우선순위와 관련된 처리를 할 필요가 없기 때문에 제안하는 알고리즘을 적용하지 않는다.

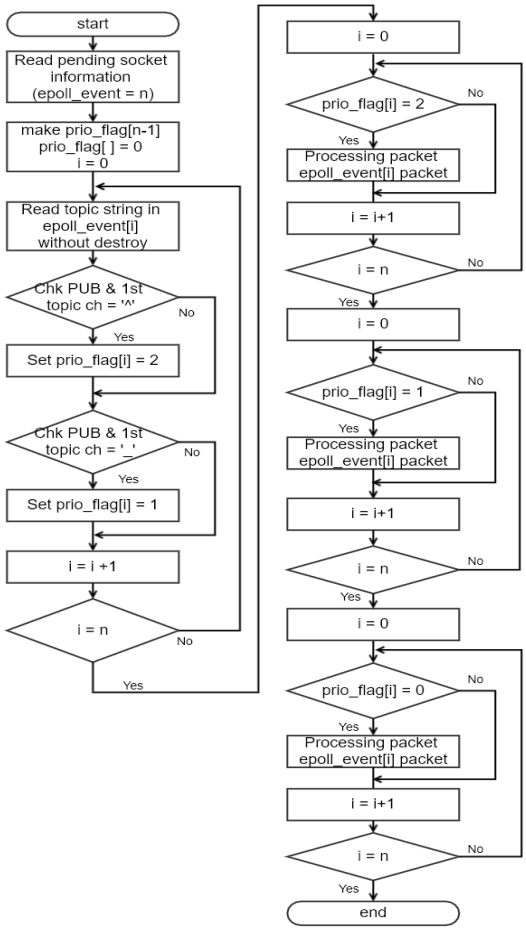

그림 7. 제안하는 신규 알고리즘
Fig. 7. Proposed new algorithm

두 번째 단계는 입력처리를 대기하고 있는 n개의 패킷들에 대해서 메시지들의 우선순위를 판별하는 단계이다. 본 알고리즘에서는 우선순위를 3단계로 구성하여 설명하고 있으나 필요에 따라 쉽게 확장이 가능한 구조이다. 개별 메시지의 우선순위를 판단하기 위해서는 변수 i를 0부터 n-1(입력 대기 중인 패킷의 수)까지 증가시키면서 다음의 작업을 반복하게 되는데 여기서 변수 i는 대기 중인 패킷들의 도착순서이다. 특정 i에 대해서 recv() 함수를 비파괴(non-destructive) 방식으로 호출하여 해당 패킷에 들어있는 MQTT 정보를 읽어서 'PUBLISH' 패킷이면서 동시에 토픽의 첫 문자가 가장 높은 우선순위를 의미하는 '^' 문자인 경우에는 숫자 2를, 'PUBLISH' 패킷이면서 동시에 토픽의 첫 문자가 두 번째 우선순위를 의미하는 '_' 문자인 경우는 숫자 1을, 그 이외의 경우에는

숫자 0을 해당 prio_flag[i]에 기록한다. 이렇게 n개의 패킷 수만큼 i를 이용해 반복 작업을 하면 i번째 입력된 패킷의 우선순위는 prio_flag[i]에 우선순위 값인 2, 1, 0 값 중의 하나로 저장되어진다.

세 번째 단계는 prio_flag[ ]에 정리된 우선순위 값을 기반으로 mosquitto의 입력 처리부로 우선순위에 의거해 개별 패킷을 넘겨주는 절차가 진행된다. 이는 그림 7 흐름도의 우측부분인데 총 3번의 반복(루프) 작업을 진행하게 된다. 첫 번째로는 우선순위가 가장 높은 토픽들을 우선적으로 mosquitto 입력처리부로 넘겨주는 절차인데, 이때는 i를 0부터 n-1까지 1씩 증가시키면서 prio_flag[i]에 2의 값으로 설정된 모든 토픽들을 차례로 mosquitto 입력처리부로 넘겨주어 최상위 우선순위를 갖는 모든 토픽을 먼저 처리하며, 그 이후 다시 i를 0부터 n-1까지 1씩 증가시키면서 prio_flag[i]의 값이 1인 모든 토픽들을 mosquitto 입력처리부로 넘겨주어 두 번째 우선순위를 갖는 모든 토픽들을 처리하게 하고, 마지막으로 다시 i를 0부터 n-1까지 1씩 증가시키면서 prio_flag[i]의 값이 0인 우선순위 지정이 없는 토픽이거나 'PUBLISH'가 아닌 패킷들을 mosquitto의 입력처리부로 넘겨주어 모든 토픽을 처리하게 된다.

Ⅳ. 실험 및 결과

1. 실험 환경

제안하는 알고리즘을 검증하기 위한 실험 환경은 i5 PC와 Windows 10 환경에서 가상머신 소프트웨어인 VMware를 이용하여 'Raspbian stretch with Desktop[9]을 설치하여 구성했다. 이는 향후 라즈베이 파이를 MQTT 브로커의 용도로 사용하기 위한 목적이며 Raspbian 이외에 Ubuntu나 Windows 등의 기타 운영체제를 사용해도 동일한 방법으로 실험환경을 구축할 수 있다. Mosquitto의 소스는 ELCIPSE Mosquitto 홈페이지[10]에서 version 1.6.2를 다운로드 받아서 Raspbian 위에 설치를 했으며 본 연구에서 제안하는 알고리즘을 적용하기 위해서 관련된 일부 소스 파일들을 수정했다. Publisher는 Paho의 Python MQTT 라이브러리[11]를 이용해서, subscriber는 mosquitto_sub 프로그램을 이용해서 구성했다.

그림 8은 제안된 알고리즘 검증을 위한 publisher, subscriber와 브로커의 구성도이다. Publisher는 'pub0'부터 'pub9' 까지 총 10개를 동시에 실행시켰으며 각 publisher는 5ms마다 자신만이 갖는 고유한 메시지를 브로커로 전송하는 것으로 설정하였다. 특히 이중에서 'pub8'은 우선순위를 가장 높게 설정한 '^' 문자로 시작하는 토픽을, 'pub9'는 두 번째 우선순위로 설정한 '_' 문자로 시작하는 토픽을 브로커로 전송하도록 하였다.

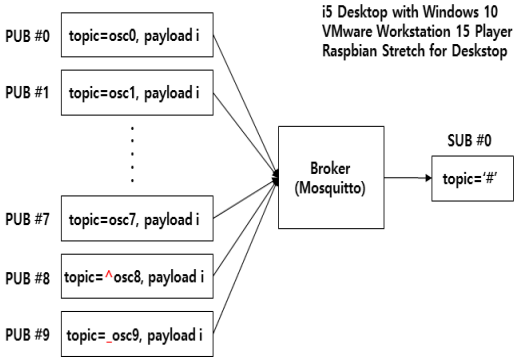

그림 8. 알고리즘의 검증을 위한 실험 환경
Fig. 8. Experimental Environment for Verification of Algorithms

2. 실험 결과

그림 9는 제안된 알고리즘을 검증하기 위해서 위의 조건을 실행했을 때 입력처리를 대기하고 있는 패킷들의 개수, 입력된 패킷들을 우선순위에 맞게 정렬하는 절차 그리고 정리된 우선순위에 따라 패킷을 mosquitto 내부에서 처리하는 모습을 mosquitto의 Log__Printf() 기능을 이용해 화면에 표시한 내용이다. 이 내용 중에 패킷의 처리와 관계없는 부분들은 임의로 삭제를 했으며, 앞의 라인번호 2 자리 숫자는 설명을 위해 임의로 삽입한 부분이다.

라인번호 01은 대기 중인 입력 패킷의 숫자를 epoll_event() 함수로 받아오는 단계이며 현재 9개의 패킷이 입력처리 대기 중인 것을 알 수 있다.

라인 번호 02~10는 모든 대기 중인 패킷에 대해 prio_flag[ ]에 우선순위를 할당하면서 표시되는 내용들인데 대기 중인 패킷이 존재하는 소켓의 파일 디스크립터 번호, 첫 번째 고정 헤더 중 상위 4 비트(30h 값이 PUBLISH 패킷을 의미)만을 남기고 나머지를 0으로 처리한 값 그리고 토픽 문자열을 보여주고 있는데 9개의 모든 패킷이 'PUBLISH' 패킷임을 알 수가 있다. 이 중에서 가장 우선순위가 높은 토픽인 '^osc/8'은 3 번째로, 두 번째 우선순위를 갖는 '_osc9' 토픽은 8번째로 입력되었다는 것을 알 수 있다.

```
01: 1562808599: OSC fdcount= 9
02: 1562808599:  - event on fd = 13 -> HDR=30h, Topic= 'osc/5'
03: 1562808599:  - event on fd = 8 -> HDR=30h, Topic= 'osc/0'
04: 1562808599:  - event on fd = 9 ->  HDR=30h, Topic= '^osc/8' -> 1st- Priority
05: 1562808599:  - event on fd = 11 -> HDR=30h, Topic= 'osc/3'
06: 1562808599:  - event on fd = 14 -> HDR=30h, Topic= 'osc/6'
07: 1562808599:  - event on fd = 10 -> HDR=30h, Topic= 'osc/2'
08: 1562808599:  - event on fd = 15 -> HDR=30h, Topic= 'osc/7'
09: 1562808599:  - event on fd = 16 -> HDR=30h, Topic= '_osc/9' -> 2nd- Priority
10: 1562808599:  - event on fd = 12 -> HDR=30h, Topic= 'osc/4'
11: 1562808599:  -- processing 1ST prioriry socket fd = 9
12: 1562808599:  -- processing 2ND priority socket fd = 16
13: 1562808599:  -- processing NO priority socket fd = 13
14: 1562808599:  -- processing NO priority socket fd = 8
15: 1562808599:  -- processing NO priority socket fd = 11
16: 1562808599:  -- processing NO priority socket fd = 14
17: 1562808599:  -- processing NO priority socket fd = 10
18: 1562808599:  -- processing NO priority socket fd = 15
19: 1562808599:  -- processing NO priority socket fd = 12
20: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, '^osc/8', ... (4 bytes))
21: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, '_osc/9', ... (4 bytes))
22: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/5', ... (4 bytes))
23: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/0', ... (4 bytes))
24: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/3', ... (4 bytes))
25: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/6', ... (4 bytes))
26: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/2', ... (4 bytes))
27: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/7', ... (4 bytes))
28: 1562808599: Sending PUBLISH to sub0 (d0, q0, r0, m0, 'osc/4', ... (4 bytes))
```

그림 9. Mosquitto의 실행 화면
Fig. 9. Running screen of Mosquitto

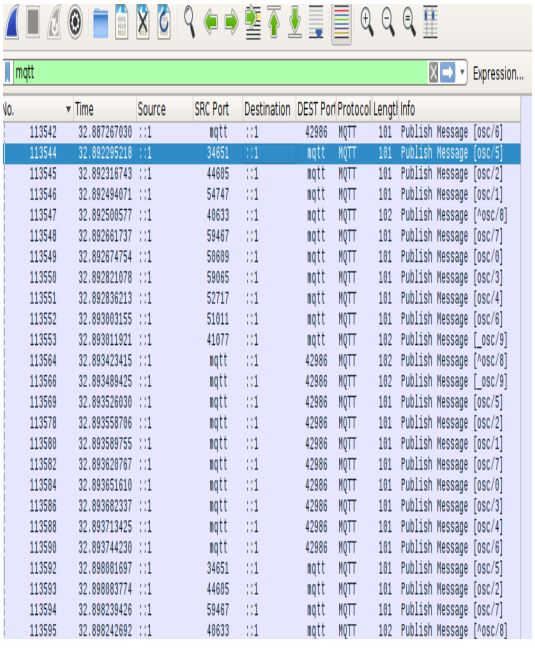

그림 10. Wireshark을 이용한 패킷처리 순서의 분석
Fig. 10. Analysis of Packet Processing Order Using Wireshark

라인번호 11~19는 우선순위가 기록되어 있는 prio_flag[ ]를 이용해 3번의 반복 루프가 돌면서 메시지가 처리되는 내용이다. 첫 번째 반복 루프에서 prio_flag[i]=2인 '^osc/8' 메시지가 먼저 처리되었고 두 번째 우선순위를 처리하는 반복 루프에서는 '_osc/9'가 처리되었고 세 번째 루프에서는 그 이외의 메시지들이 입력된 순서대로 처리가 되었음을 나타내고 있다.

그림 10은 패킷들의 흐름을 'Wireshark' 프로그램을 이용해 분석한 내용이다. 패킷번호 113544 ~ 113553까지는 10개의 publisher에서 송신한 'PUBLISH' 패킷이 mosquitto의 입력단으로 들어가는 것을 보여주고 있고, 제안된 알고리즘을 거친 후에 subscriber로 출력되는 패킷의 순서는 4번째로 입력된 패킷번호 113547의 '^osc/8' 패킷이 제일 먼저 처리가 되어 subscriber로 전송되었음을 패킷번호 113564로 확인할 수 있고, 10번째로 입력된 패킷번호 113553의 '_osc/9' 패킷이 두 번째로 처리되어 전송되었음을 패킷번호 113566에서 확인할 수 있다. 그 이후로는 우선순위가 없는 패킷들이 입력된 순서대로 출력되고 있음을 확인할 수 있다.

정량적인 분석을 위한 실험은 표 1과 같이 총 5종류의 형태로 구성하였고, 각 실험조건을 실행할 때 모든 publisher들은 병렬로 실행이 되며 각각 200ms 주기로 메시지를 반복 출력하고 추가적인 하나의 subscriber에서 수신하는 것으로 설정하였다. 각 실험에서 주의할 점은 시뮬레이션 조건에서 사용되는 리눅스의 경우 비동기적인 특성을 가지며 또한 개별 publisher 간의 실행 시점의 차이로 인해 실험을 할 때마다 실행 환경에 따라 결과가 다르게 나타나는 경우가 있다. 따라서 실험 시에 비정상 데이터를 제거하면서 각 실험 조건 별로 총 10번의 반복을 통해 실험 결과의 오차를 최소화 하였으므로 수집된 데이터들은 실제 처리된 정확한 시간을 파악하기에는 한계가 있더라도 추세 파악의 용도로는 활용이 가능할 것으로 판단된다.

표 1. 실험 조건별 publisher의 구성
Table 1. Composition of the Publisher by experimental conditions

실험조건	우선순위별 publisher의 수			합계
	최상위	차순위	일반	
1	1	1	8	10
2	1	1	23	25
3	1	1	48	50
4	1	1	73	75
5	1	1	98	100

표 2는 mosquitto의 입력 대기열에서 동시에 입력되어 처리를 기다리고 있는 메시지들의 평균 개수 및 동시에 입력된 메시지 개수들에 대한 점유율을 정리한 표이다. 동시에 실행되는 publisher의 수가 늘어날수록 특정

순간에 입력 대기열에 존재하는 메시지의 수가 평균 1.14개에서 1.80개로 증가함을 알 수 있으며, 실험 1에서는 최대 3개의 입력 메시지만이 동시에 대기열에 존재하지만 동시에 실행되는 publisher의 수가 늘어날수록 입력 대기열에서 동시에 대기하는 메시지의 수가 증가하게 되고 특히 100개의 publisher가 동시에 실행되는 실험 5에서는 동시에 최대 12개의 메시지(점유율 0.1%)가 mosquitto 입력 대기열에서 처리를 기다리고 있기 때문에 긴급 메시지에 대한 우선순위 처리의 중요성을 정량적으로 파악할 수 있다.

표 2. mosquitto 입력 대기열 내의 메시지 수 및 점유율
Table 2. Number and share of messages in the musquito input queue

실험	평균 (개)	MQTT 입력 대기열의 대기 메시지 수 및 점유율 (%)											
		1	2	3	4	5	6	7	8	9	10	11	12
1	1.14	87.5	11.3	1.2									
2	1.29	76.4	19.9	2.0	1.6	0.1							
3	1.44	71.1	21.8	3.4	2.2	0.4	0.7	0.1	0.3	0.1			
4	1.61	64.6	23.6	4.6	4.6	0.8	0.8	0.2	0.4	0.0	0.2		
5	1.80	60.7	22.0	5.5	6.9	1.4	2.4	0.5	0.2	0.1	0.1	0.1	0.1

그림 11은 5종류의 실험들에 대한 메시지 평균 처리 시간에 대한 분석 그래프이다. 실험 1에서는 일반 우선순위 메시지의 평균 처리시간이 170㎲인 반면에 1번째 우선순위 메시지는 128㎲, 2번째 우선순위 메시지는 135㎲로 빠르게 처리됨을 보여주고 있다. 메시지 수가 늘어날수록 메시지의 평균 처리시간은 증가하나 여전히 우선순위가 높은 메시지는 일반 우순순위의 메시지보다 빠르게 처리되는 경향을 확실히 파악할 수 있다.

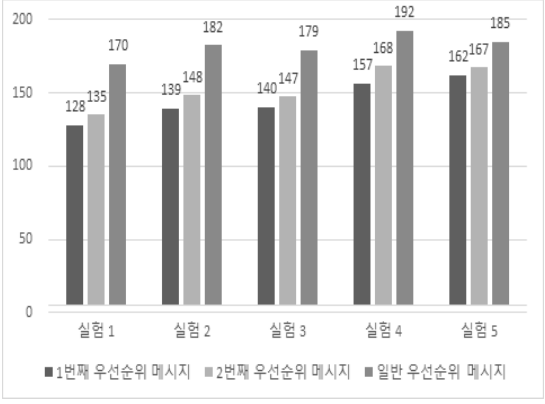

그림 11. Publisher 수에 따른 메시지 평균 처리 시간
Fig. 11. Latency time according to number of publishers

V. 결 론

IoT 시스템의 구축에 많이 사용되는 메시지 프로토콜인 MQTT(mosquitto)를 기반으로 IIoT의 환경에 반드시 필수적이지만 지원되지 않았던 기능인 우선순위 기반의 메시지 처리 기법을 적용하기 위해서 우선순위 설정 방안과 'multi-scanned priority sorting 알고리즘'을 제안하고 이에 대한 결과를 검증했다. 본 연구에서 제안된 방식과 알고리즘은 기존의 우선순위 기반 메시지 처리에 관련한 연구와는 다르게 MQTT 표준을 완벽하게 준수하면서도 매우 단순화된 알고리즘을 사용하여 상대적으로 적은 CPU 리소스만으로도 실행이 가능하다는 장점을 가지고 있음을 확인할 수 있었다.

References

[1] Hun Jung, "Study on the MQTT protocol design for the application of the real-time HVAC System", International Journal of Internet, Broadcasting and Communication Vol.8, No.1 19-26, 2016.
DOI: http://dx.doi.org/10.7236/IJIBC.2016.8.1.19

[2] Se-Chun Oh, Tae-Hyung Kim, Young-Gon Kim, "Implementation of factory monitoring system using MQTT and Node-RED", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 18, No. 4, pp.211-218, Aug 2018.
DOI: https://doi.org/10.7236/JIIBC.2018.18.4.211

[3] Jae-wook Ko, Hye-Jeong Kim, Bo-Kyung Lee, "Factory environmental management system based on MQTT using LoRa", The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), Vol. 18, No. 6, pp.83-90, Dec 2018.
DOI: https://doi.org/10.7236/JIIBC.2018.18.6.83

[4] Bo-Hun Kim, Hwang-Rae Kim, "Design of the Smart Control System for Industrial Automation Equipment", Journal of the Korea Academia-Industrial cooperation Society, Vol. 18, No. 4 pp. 677-684, 2017.
DOI: https://doi.org/10.5762/KAIS.2017.18.4.677

[5] Sung-jin Kim, Chang-heon Oh, "Method for Message Processing According to Priority in MQTT Broker", Journal of the Korea Institute of Information and Communication Engineering, Vol. 21, No. 7, pp.1320~1326 Jul 2017.
DOI: https://doi.org/10.6109/jkiice.2017.21.7.1320

[6] Geonwoo Kim, Jiwoo Park, Kwangsue Chung, "Priority-based Multi-level MQTT System to Provide Differentiated IoT Services", Journal of KIISE, Vol. 45, No. 9, pp.969-974, Sep 2018.
DOI: https://doi.org/10.5626/JOK.2018.45.9.969

[7] Yong-Seong Kim, Hwi-Ho Lee, "Message Queue

Telemetry Transport Broker with Priority Support for Emergency Events in Internet of Things", Sensors and Materials, Vol. 30, No. 8, pp.1715-1721, Aug 2018. DOI: http://dx.doi.org/10.18494/SAM.2018.1864

[8] http://mqtt.org/

[9] https://www.raspberrypi.org/

[10] https://mosquitto.org/

[11] https://pypi.org/project/paho-mqtt/

저 자 소 개

오 세 춘(정회원)

- 1984.2 고려대학교 전자공학과(공학사)
- 2017.2 한국산업기술대학교 컴퓨터공학과(공학석사)
- 1999.9 ~ 2010.1 삼성전자 LCD 총괄 상무이사
- 2017.3 ~ 한국산업기술대학교 컴퓨터공학부 교수

- 관심분야 : 임베디드시스템, IoT, 스마트팩토리

김 영 곤(정회원)

- 1983.2 경북대학교 전자공학과(공학사)
- 1985.2 연세대학교 본대학원 전자공학과(공학석사)
- 2000.2 한국과학기술원 전산학과(공학박사)
- 1985 ~ 2007 KT 수석연구원
- 2007 ~ 한국산업기술대학교 컴퓨터공학부 교수

- 관심분야 : 소프트웨어공학, 정보통신시스템, 객체지향 분석 및 설계