IEEJ TRANSACTIONS ON ELECTRICAL AND ELECTRONIC ENGINEERING
IEEJ Trans 2021
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI:10.1002/tee.23511

Paper

Hardware Implementation of MQTT Broker and Precise Time Synchronization Using IoT Devices

Koutarou Yamamoto*[a], Non-member
Akihiro Fukuhara*, Non-member
Hiroaki Nishi**, Member

In recent years, while internet of things (IoT) devices and cloud computing environments have been penetrated, the increase in communication latency and concentration of traffic caused by data centers has deteriorated the quality of some network services. Message Queuing Telemetry Transport (MQTT) is a well-known protocol for exchanging messages between dedicated applications. MQTT employs a publish–subscribe model in which a broker mediates communication between IoT devices and applications. To improve service quality, the processing latency of the broker should be reduced. Moreover, the broker's throughput must be increased. IoT devices, in particular, must maintain a highly accurate time to provide some IoT services. This study proposed hardware implementation of MQTT broker and evaluated the accuracy of an MQTT-based time synchronization method. The proposed MQTT architecture's hardware resource utilization is also provided. We confirmed that our implementation achieved high throughput, low latency, and low jitter MQTT broker, which satisfies the smart city services for automated drive and power control, which must be less than a few milliseconds. © 2021 Institute of Electrical Engineers of Japan. Published by Wiley Periodicals LLC.

Keywords: field-programmable gate array; MQTT; edge computing; internet of things; time synchronization

Received 19 July 2021; Revised 29 August 2021

1. Introduction

While internet of things (IoT) devices and cloud computing environments have grown in popularity in recent years, the increase in communication latency caused by using centralized cloud services has become a problem. Edge computing, which deploys a distributed processing environment between the cloud and devices, has garnered attention in response.

To reduce traffic and processing load, there is a movement to change the protocol used by IoT devices for communication from the conventional HTTP to something else. Message Queuing Telemetry Transport (MQTT) has been adopted as an IoT protocol by major cloud providers such as Amazon Web Services [1] and Google Cloud Platform [2], and is used in edge computing as the Akamai IoT Edge Cloud [3]. In this study, we focused on MQTT.

MQTT has a simple mechanism and data frame format. Its small memory footprint in implementation, in particular, is beneficial for general resource-limited IoT devices. The following characteristics of MQTT merit their use for IoT.

- MQTT can easily establish multiple connections.
  - A typical IoT device sleeps to conserve power, and then wakes up at regular intervals to transmit the sensed data. Because the power required for wireless communication is relatively high, lowering the cost of communication power is critical when power resources are scarce. When data is sent to multiple destinations, it must generally be transmitted individually, which increases power consumption. In other words, even if there are multiple destinations, data must be distributed in a single communication. Brokers are in charge of distribution in the case of MQTT. As a result, it is sufficient for IoT devices to communicate only once. Furthermore, the MQTT packet is simple, and its size is smaller than that of the other protocols. This also contributes to lower power consumption in the communication, and memory footprint in its implementation.
- MQTT has various functions in IoT messaging.
  - MQTT's retain function allows the broker to keep track of the last unsubscribed message. This retains function lowers IoT device communication costs for message retransmission. Furthermore, MQTT's Quality of Service (QoS) mechanism ensures that the messages reach. This QoS function offers a more robust mechanism for ensuring that messages are delivered. By using these functions, IoT devices can offload the function responsible for messaging. Retention and QoS functions are not mandatory in the MQTT OASIS standard.

[a] Correspondence to: Koutarou Yamamoto. E-mail: yamamoto@west.sd.keio.ac.jp

*Graduate School of Science and Technology, Keio University, 3-14-1, Hiyoshi, Kohoku-ku Yokohama 223-8522, Japan

**Department of System Design, Faculty of Science and Technology, Keio University, 3-14-1, Hiyoshi, Kohoku-ku Yokohama 223-8522, Japan

In 2019, the OASIS released MQTT v5, a new standard that extended a function of MQTT without compromising its simplicity, and the most significant advancement was the message property. It provides additional and separated messages that are filed as a property of the message. MQTT v5's new message property allows for general server–client style communication as well as publish–subscribe style communication. When a message requesting a message is published, it can immediately receive the target message as a response. Furthermore, this property can be used as a subchannel communication line separate from the main message, as well as a low-cost additional message channel.

2. Objectives of Hardware-Based MQTT Broker

MQTT is a publish–subscribe model, so communication is centered on the MQTT broker. A large number of devices are available in IoT environments. If the broker's performance is inadequate, it may result in an increase in the processing time and a decrease in availability. For applications where the allowable delay is limited, such as machine control and power system stability, this can be a serious issue. As a result, an MQTT broker with low latency and high throughput is required. Hardware-based MQTT brokers can increase throughput, decrease processing delay, and reduce power consumption. These features are especially beneficial in edge computing environments. Furthermore, because IoT services frequently require highly accurate time synchronization, time synchronization in MQTT is standardized in IEEE 1451 (mainly discussed in P1451.0 [4] and IEEE P21451-1-6 [5] standards). Without the use of any additional protocols, accurate time synchronization should be implemented.

This study uses a field-programmable gate arrays (FPGA) for hardware implementation. FPGAs consume less power, have lower latency, and have higher throughput than conventional processors. In network applications, FPGAs have become popular because their pipelined architecture is efficient for processing network streams. For example, FPGA implementations of TCP/IP stacks with line-rate support for 10G Ethernet, have been proposed for deployment in data centers [6] and implementations of anonymizing hardware for encapsulating personal information in edge computing environments [7].

Our main contributions can be summarized as follows:

- We proposed an FPGA-based MQTT broker architecture with low latency, high throughput, and low jitter.
- We proposed a time synchronization method in IoT devices that did not require any additional protocol implementation and evaluated its applicability to the proposed MQTT broker.

The proposed design's target performance has been defined based on the network infrastructure and application requirements in recent years.

- To accommodate the network bandwidth in modern network environments, the throughput should be at least 10 Gbps.
- The processing delay time in the broker should not exceed 300 μs. This value is calculated based on IEC61000-4, which states that the time accuracy of control in power system stabilization should be approximately 1 ms [8].
- According to IEEE C37.118.1–2011 [9], a time synchronization accuracy of $\pm 26\,\mu$s or better is required when assuming communication between phasor measurement units (PMUs) that measure the phase of voltage and current.

3. Related Work

3.1. Lightweight protocol acceleration for IoT To keep up with the growing number of IoT devices and traffic, researchers are working on improving server performance for speed and various functions in lightweight IoT protocols. An SoC FPGA implementation of a low-latency Constrained Application Protocol (CoAP) server has been proposed [10]. When compared to an interrupt-driven software server this implementation increased the processing speed by 308.83%. However, because this is an accelerator for CoAP, it cannot be adapted to the publish–subscribe model like MQTT. CoAP's functionality is simpler than MQTT's, but it is difficult for CoAP to meet the demands of today's IoT environment.

A lightweight and scalable MQTT broker (muMQ) [11] has been proposed as a software implementation of an MQTT broker that uses a highly scalable user-level TCP stack for multicore systems (mTCP) [12] and Data Plane Development Kit (DPDK) [13] for the broker to speed up the process. This research enables multicore operation at the TCP connection level to handle multiple connections at the same time and reduces overhead by utilizing the TCP/IP stack at the user level. When the broker is implemented on a 16-threaded Intel Xeon E5-2650 server machine and connected to clients via 10G Ethernet, the throughput of PUBLISH can reach 930 275 messages per second, which is 5.38 times higher than the throughput of Mosquitto [14]. However, a 10 Gbps line rate is achieved when the frame size is 1344 bytes or greater, and frames of this size are rarely used in an MQTT environment. Furthermore, processing jitter is not taken into account.

3.2. Time synchronization Global Navigation Satellite System (GNSS), as represented by Global Positioning System (GPS), Precision Time Protocol (PTP), and Network Time Protocol (NTP), exists as time synchronization between devices. GNSS is primarily used in time synchronization networks to synchronize absolute time with respect to UTC. The Primary Reference Time Clock (PRTC) [15] is the device that receives the time, and even PRTC-A, which has the most significant error among the standardized PRTCs, can be synchronized with an error of less than 100 ns relative to Universal Time Coordinated (UTC). However, GNSS necessitates the use of dedicated antennas, which are prohibitively expensive and difficult to install in each IoT device due to cost and space constraints. PTP is a time synchronization protocol that uses timestamp information imprinted on packets to distribute and synchronize time information in a synchronous network [16]. If high accuracy is required, the timestamp should be imprinted by hardware, and the segment should use a network switch that supports PTP. NTP, like PTP, is a network time synchronization protocol that is primarily used in a server–client model. Because timestamp imprinting is affected by jitter and other factors caused by the operating system, NTP, which is software-based, has a synchronization accuracy of the order of ms [17]. In IoT devices, synchronization between devices via a network, such as NTP and PTP, is effective. These protocols may be difficult to use due to the additional cost of dedicated hardware and the scarcity of computing resources in IoT devices. As a result, MQTT time synchronization is required.

4. Proposed Method

The MQTT broker must have a low latency, high throughput, and low jitter. The jitter is affected by the number of subscribers

and publishers. Furthermore, time synchronization is necessary because the IoT requires highly accurate time information to avoid data loss. In this study, we concentrate on PUBLISH and SUBSCRIBE, with the goal of meeting the following requirements:

- 10 Gbps line-rate processing
- 300 µs or less processing delay
- Time synchronization accuracy of 52 µs or better between 100 subscribers

Because these requirements cannot be achieved with existing software-implemented brokers, we propose a hardware implementation of the broker. We also propose a time synchronization method using a hardware-implemented broker. The proposed methods in this study assumes that One Way Delay (OWD) for all client is mostly the same, and that OWD will not vary in each message.

4.1. Destination search method MQTT is a publish–subscribe model that is topic-based. The broker must look for subscribers who have subscribed to a given topic and distribute messages to the subscribers' destinations. Because topics vary in length, have a hierarchical structure, and have a variable number of subscribers, a single associative memory cannot be used as a search mechanism. Existing work on software implementations searches for target subscribers using a dynamic tree structure with branches represented by pointers [11]. The memory management of such a tree structure with an undefined number of branches and depth, on the other hand, is unsuitable for hardware processing.

The most basic destination search method that can be implemented in hardware is to arrange subscribers in memory and then search for subscriptions to the corresponding topics linearly. This method is inefficient because the number of subscribers in memory affects the destination search processing time. As a result, in this study, we propose a destination search method based on two hash tables that can be executed in a specific amount of time.

Figure 1 shows the flow of the destination search using a hash table. For a destination search, two types of memory were used: Topic Tables and Subscriber Tables. H() and H′() are hash functions that find the hash value of the key in these hash tables. The Topic Table uses the topic name as a key to record the number of subscribers to a topic, and the Subscriber Table uses the 'topic named' as a key to record the subscriber's destination information. The ID is the number assigned to each subscriber at the time of subscription.

- When a message is published to the topic name, 'topic1/topic2/humid', the Topic Tables display the number of subscribers to this topic by using this topic name as the key. Figure 1 depicts an example in which three subscribers are obtained by referring to the Topic Table. The subscriber's destination information is then obtained by referring to the Subscriber Table. When referring to the Subscriber Table, the key is changed to 'topic name+ID' for ease of table management with no disadvantages. IDs, in particular, can have values ranging from one to the number of subscribers. The Subscriber Table is referred to in this example by three types of the keys: 'topic1/topic2/' + 'humid1', 'humid2', and 'humid3'. As a result, the subscriber's destination information is obtained.
- When subscribing to a message, the Topic Table displays the topic's subscribers count. Based on this number, the new subscriber's destination information is added to the Subscriber Table. For example, when a subscriber sends the SUBSCRIBE message with the subject 'topic1/topic2/humid', the Topic Table is referred to as the key, and the number of subscribers to this topic, 3, is obtained. As a result, the new subscriber ID is 4. The Subscriber Table is then accessed using the key, 'topic1/topic2/humid4' and the destination information for the new subscriber is written. The value of the Topic Table is changed from three to four because the number of subscribers to this topic increased.

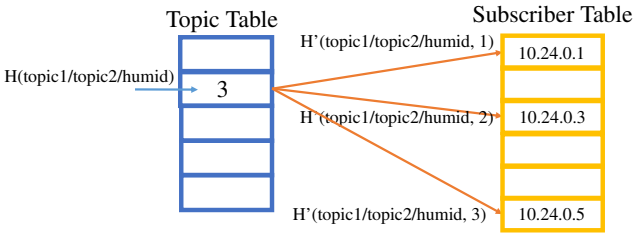


Fig 1. Searching for a destination using a hash table

This research method also accepts multilevel wildcards denoted by '#'. In the case of SUBSCRIBE, the topic name is added to the Topic Table, and the destination information is added to the Subscriber Table, as in the previous procedure. If the topic name contains '/', which indicates a hierarchical structure, the search is performed with the topic name replacing the part after the '/' with a '#'. For example, when a message is published to the topic name, 'topic1/topic2/humid', the topic name is extracted into three different names: 'topic1/topic2/humid', 'topic1/topic2/#', and 'topic1/#'.

If there are n subscribers connected to the broker and m topics, the average number of subscribers for each topic is n/m. For a single topic, the average computation time is (n) for linear search and (n/m) for our method. Because sending a packet to each subscriber takes (n/m) time, this method can achieve line-rate processing in PUBLISH.

4.2. Time synchronization method MQTT's time synchronization algorithm and message format adhere to IEEE P21451-1-6 [5] and IEEE P1451.0 [4], which are based on the NTP [17]. This MQTT-based time synchronization (MQTT-TS) method is the same as the regular PUBLISH/SUBSCRIBE style messaging. The time server publishes the time on a specific topic to the broker. Clients that synchronize with the time server subscribe to the topic ahead of time. For example, subscribers who want to get time information should subscribe by sending a regular SUBSCRIBE message with a topic such as 'topic1/time'. The publisher distributes time information by sending a regular PUBLISH message with this topic specified. This is how the time is distributed in the same way as normal message distribution. As a result, no new functions need to be added to the client to receive the message.

5. Implementation

5.1. Implementation environment M-KUBOS [18], an FPGA computing platform developed by PALTEK Corporation, is used as an FPGA implementation board. It is powered by a

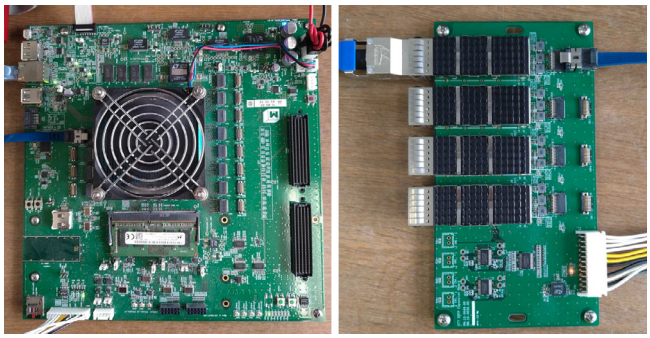


Fig 2. M-KUBOS and QSFP expansion board

Zynq UltraScale+ MPSoC, the XCZU19EG2FFVC1760. Figure 2 shows the M-KUBOS board. It supports a variety of peripheral device interfaces, including DDR4, CAN, Ethernet, and USB3.0. To support 10 Gbps networks a Quad Small Form-factor Pluggable (QSFP) GTY transceiver extension board connected to the mainboard via a FireFly cable was designed and used.

5.2. Hardware design As described in Section 4, we implemented the MQTT broker in hardware with time synchronization. This MQTT broker is not just for time synchronization; it also supports general MQTT communication.

MQTT Version 3.1.1 [19] was used in the implementation. Multilevel wildcards were supported using the method outlined in Section 3.1. For the options, QoS was set to 0, and the Will and Retain functions were not supported. In MQTT-v3.1.1 [19], these functions and single-level wildcard are an option, not required. Therefore, our MQTT broker is fully functional. In addition, the same is true for QoS. The maximum length of topics and messages was 256 bits; however, it is extendable and is not an essential object for this implementation.

Because the hardware implementation cost of using the TCP/IP stack is high, this implementation used the UDP/IP stack. A study that evaluated the performance of protocols such as TCP and UDP for video traffic on mobile networks [20] discovered that there was not much difference in latency between UDP and TCP. When the number of frames was less than 100; that is, when the amount of data transferred was small, the delay time is the same whether TCP or UDP is used. In addition, research on the performance of various protocols in mobile IP networks [21] has shown that the difference in latency between TCP and UDP decreases when the network bandwidth is increased. Given the small size of MQTT packets, particularly when using time synchronization, and the fact that the evaluation environment in this study is a local network environment with sufficient bandwidth, which is the target environment of the proposed time synchronization mechanism, the effect of protocol differences in the transport layer is small.

Figure 3 depicts the hardware design. The AXI4-Stream protocol was used to connect the modules. First, the packet is routed through the Ethernet subsystem IP provided by Xilinx [22]. An open-source UDP/IP stack [23] first removes the preambles, SFD, and FCS. Then, it confirms that the MAC and IP addresses point to itself and verifies the IP and UDP checksums. Subsequently, the MQTT parser extracts the message type and topic from the incoming packet, and the lookup engine searches for and adds destinations based on the topic. The message queue holds messages during processing in the lookup engine. The MQTT packet generator

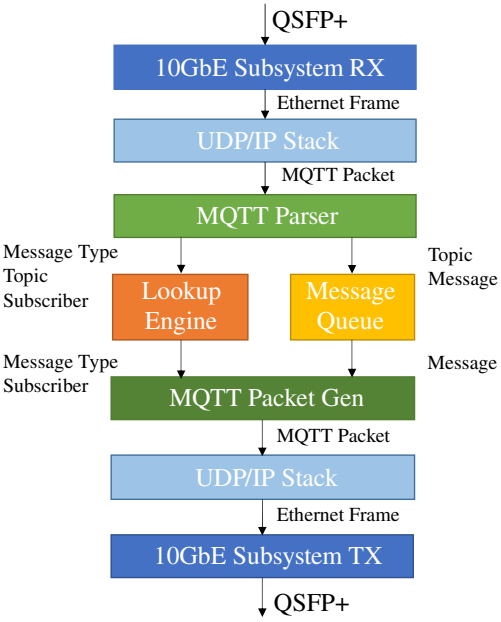


Fig 3. Hardware design

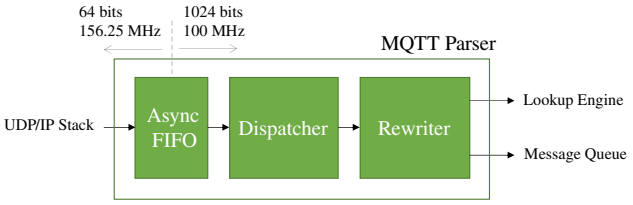


Fig 4. MQTT parser

generates MQTT packets based on the destination information output from the lookup engine and messages retrieved from the message queue. Finally, the generated MQTT packet is sent to the QSFP expansion board via the UDP/IP stack and the 10G Ethernet subsystem IP. This study implemented four modules: MQTT parser, lookup engine, message queue, and MQTT packet generator. The functions of the modules are described as follows.

5.3. MQTT parser As shown in Fig. 4, the MQTT parser consists of an asynchronous FIFO, a dispatcher, and a rewriter. The asynchronous FIFO [23] traverses the clock domain and converts the data width. For the purposes of this implementation, the payload sent out by the Ethernet Subsystem and UDP/IP stack at 64-bit, 156.25 MHz is converted to 1024-bit, 100 MHz. The dispatcher retrieves the topic name and message content based on the topic length and message length contained in the MQTT packet. The message queue is populated with the topic name and message. The Rewriter rewrites the topic name to support a multilevel wildcard, and it issues a PUBLISH request.

The MQTT parser module supports multilevel wildcards. The configuration of the Rewriter is shown in Fig. 5. The topic name entered into the topic register is saved, and the PUBLISH request is issued based on the topic name in the topic register. If the topic name in the topic register contains '/', the position of '/' is detected by Comparator and Priority Encoder, and the topic name is rewritten by replacing the part after '/' with '#'. For example,

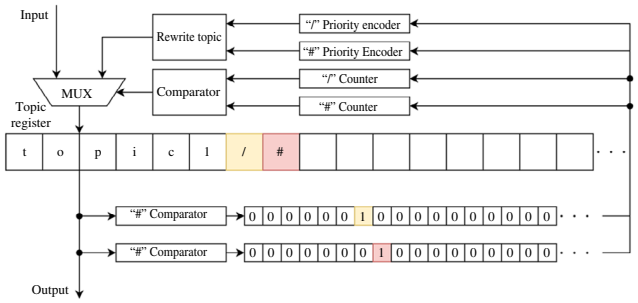


Fig 5. Rewriter

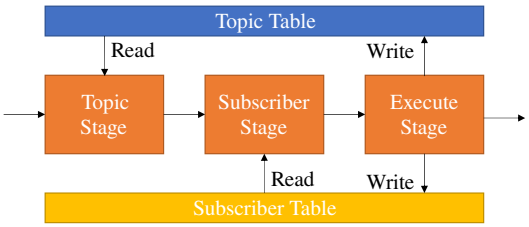


Fig 6. Lookup engine

when a PUBLISH message with a topic name, 'topic1/humid' is received, 'topic1/humid' is registered in the topic register. Then, the position of '/' in the topic is detected, the part after 'topic1/' is replaced with '#'. Therefore, the rewritten topic name, 'topic1/#' is registered in the topic register to adapt to the topic search mechanism using a hash function.

5.4. Lookup engine As shown in Fig. 6, the lookup engine consists of two hash tables, a Topic Table and Subscriber Table, and a pipeline with three stages: topic, subscriber, and execute. The Topic Table is a hash table where the key is 'topic name', and the value is 'number of subscribers to that topic'. The Subscriber Table is also a hash table whose keys are 'topic name+ID' and whose values are 'destination IP address and destination port'. The topic stage refers to the Topic Table using the topic name as a key to obtain the number of subscribers to that topic. If the message type is PUBLISH, the Subscriber Stage refers to the Subscriber Table using 'topic name+ID' as a key to obtain the subscriber's destination information. In the execution stage, if the message type is SUBSCRIBE in, an entry is added to the topic and subscriber tables. Section 3 describes the destination search method that employs two hash tables.

In this implementation, the Topic Table has a 12-bit width and 65 536 words depth, while the Subscriber Table has a 48-bit width and a depth of 65 536 words. Both tables use block-RAM (BRAM) to reduce memory access time and the utilization of flip-flop (FF) and lookup table (LUT). Because the number of concurrent sessions depends on the size of the port ID of the IP packet, these table sizes are not sufficient.

5.5. Message queue During the lookup engine's destination search, the message queue is the synchronous FIFO module that holds the message length, message content, topic length, and topic name. The FIFO depth was set to 1024 in this implementation. During the evaluation described in Section 6.1, the maximum number of subscribers is 1000. Therefore, the FIFO depth

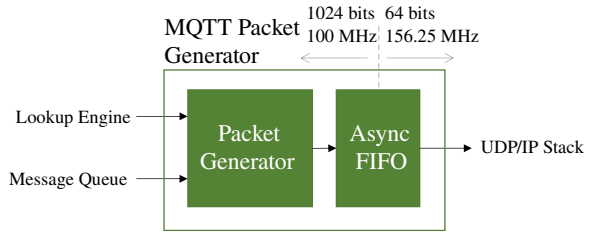


Fig 7. MQTT packet generator

is sufficient. This synchronous FIFO makes use of BRAM, which conserves FF resources.

5.6. MQTT packet generator The MQTT packet generator, as shown in Fig. 7, is made up of a packet generator and an asynchronous FIFO. The packet generator obtains the destination information from the lookup engine and the message from the message queue, for generating MQTT packets based on this information. Queuing, data width conversion, and clock domain traversal are all performed by async FIFO.

6. Evaluation

6.1. Evaluation of time synchronization accuracy

The time synchronization accuracy of the implemented FPGA-based MQTT broker was compared to Mosquitto version 1.4.15 [14], a commonly used software-implemented MQTT broker. Figure 8 shows the evaluation environment. The machine used for the measurement performed as follows:

- Software-based MQTT Broker
  - Intel Core i7-4790 3.60GHz 8 thread, Ubuntu18.04
- Publisher, Subscriber
  - Raspberry Pi3 model B (RPi) RAM 4GB model, Raspberry Pi OS Lite kernel version 5.4
- Network switch
  - MikroTik CRS305-1G-4S + IN

The following two points were prioritized for time synchronization accuracy.

1. Latency from publisher to subscriber
   a. After sending the PUBLISH message RPi0, the publisher, operates the General Purpose Input/Output (GPIO), and RPi1, the subscriber operates the GPIO. The time difference between the two GPIO operations was measured using an oscilloscope. This measured time indicates the delay between the publisher and subscriber.
   b. The mean of the measured time difference in the case of the software-implemented broker was 462.2 µs with a standard deviation of 36.95 µs. The average for the hardware-implemented broker, on the other hand, was 87.37 µs with a standard deviation of 14.34 µs. Our implemented broker's average was 18.9% of that of the software-implemented broker, achieving a target value of 300 µs.

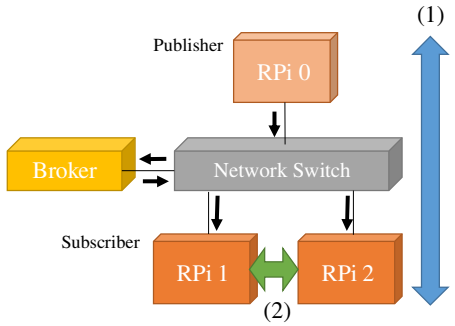


Fig 8. Evaluation environment

2. Packet arrival time difference between subscribers

   a. When subscribers RPi1 and RPi2 receive a PUBLISH message, they operate the GPIO. The time difference between the two subscribers' GPIO operations was measured using an oscilloscope. Experiments were also carried out in which the number of subscribers was varied to investigate the effect of increasing the number of subscribers on the time difference. We measured the time difference with RPi1 subscribing first and RPi2 subscribing last to analyze the case with the greatest time difference.
   b. The measurement results of the packet arrival time difference between subscribers in a software-implemented broker are shown in Fig. 9(a). The figure also plots the results of the hardware-implemented MQTT broker for comparison. Each result is the average of the 20 measurements. The average time difference between the two subscribers was 47.43 µs with a standard deviation of 51.37 µs. The average time difference between 100 subscribers was 340.0 µs with a standard deviation of 86.32 µs. It was discovered that as the number of subscribers increased, the time difference increased approximately linearly. It was also demonstrated that with the software implementation, a time synchronization with an accuracy of within 52 µs among 100 subscribers was impossible.
   c. The measurement results of the packet arrival time difference between subscribers in our hardware-implemented broker are shown in Fig. 9(b). Note that the values on the vertical axis of the graph are different because the hardware implementation of the broker is faster than the software implementation. As with the software implementation experiment, each result represents the average of 20 measurements. The average time difference between two subscribers was 13.78 µs with a standard deviation of 8.601 µs, and the average time difference between 100 subscribers was 19.92 µs with a standard deviation of 13.62 µs. As with software implementation, it was discovered that as the number of subscribers increased, the time difference increased approximately linearly. In addition, under the condition of 100 subscribers, a target accuracy of within 52 µs was achieved. It was also discovered that a time synchronization accuracy of up to 600 subscribers could be achieved.

The measured time was influenced by delays and jitter caused by the GPIO operation of the Raspberry Pi, the device drivers and network stack of the network interface, and the network switches. Ethernet switches are known for the jitter of up to several tens of microseconds and a delay of several milliseconds [24]. Using a network switch that supports PTP may thus improve performance.

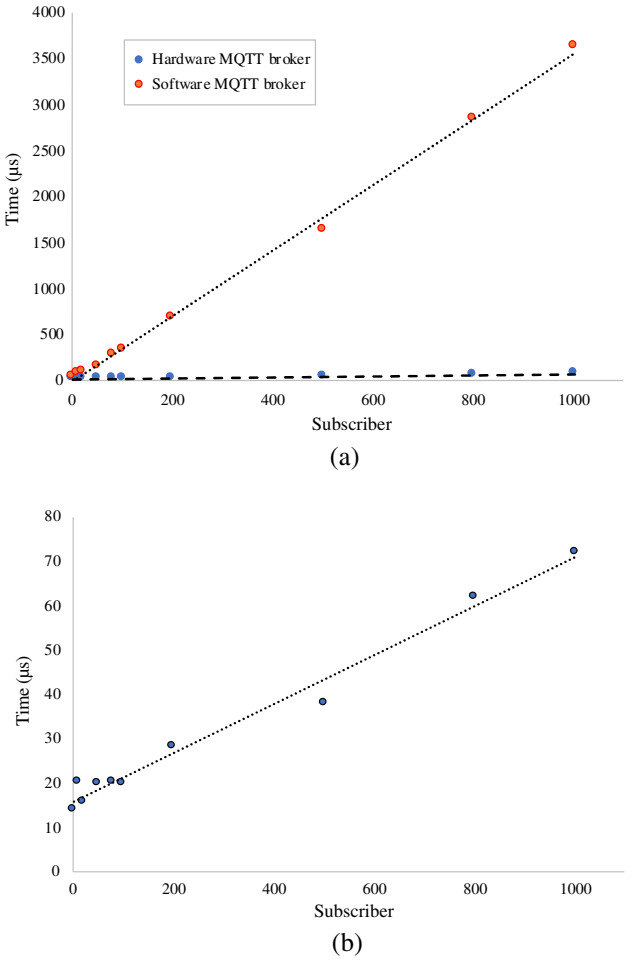


Fig 9. Time difference between subscribers. (a) software-implemented broker and (b) hardware-implemented broker

6.2. Throughput The MQTT parser, lookup engine, message queue, and MQTT packet generator all run at 100 MHz in this implementation. Because all these modules can process a PUBLISH or SUBSCRIBE every cycle, the pipeline has a one-cycle processing interval. As a result, this implementation has a throughput of 100 Mmps (messages per second). The software-implemented MQTT broker 'muMQ', described in Section 3.1, has a maximum PUBLISH throughput of 930 275 mps [11]. As a result, this study achieved a high throughput of 107 times.

The UDP stack operates at a frequency of 156.25 MHz and has data width of 64-bit, whereas the MQTT broker operates at 156.25 MHz and has data width of 1024-bit. As a result, the effect of clock reduction on the transfer rate must be considered. The total size of three Ethernet, IP, and UDP headers is 42 bytes, and the minimum MQTT PUBLISH packet size is 48 bytes. In the MQTT broker, a packet of this size is processed in one cycle, whereas it takes six cycles in the UDP stack. As a result, lowering the clock rate does not reduce the transfer rate, and an SFP+ line rate of 10 Gbps was achieved. There is an FPGA implementation of the TCP stack that supports a 10 Gbps line rate [6], and even if

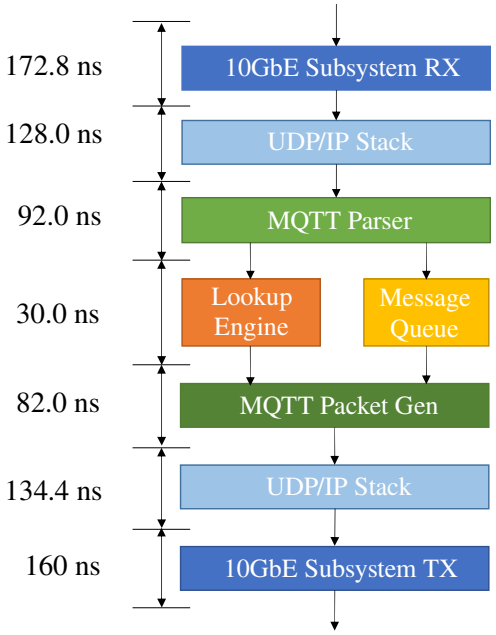


Fig 10. Latency on each model

the UDP stack in this implementation is changed to a TCP stack, a line rate of 10 Gbps can be supported.

6.3. Latency Figure 10 shows the processing time for each module. Message queue requires a minimum of two cycles from input to output, while lookup engine has three stages and requires a minimum of three cycles. Therefore, the latency of 30 ns in Fig. 10 is the latency of the lookup engine.

According to Fig. 10, the MQTT broker's processing time between receiving and sending a packet was 799.2 ns. In the case of SUBSCRIBE, this delay time is used to add new subscriber destination information. In the case of PUBLISH, the message is sent to the subscriber with this delay time. Therefore, a delay time of less than 300 µs was obtained.

The processing latency for sending and receiving in a TCP stack [6] was 5 µs for an FPGA implementation supporting 10 Gbps. As a result, even when the TCP stack is used, a delay time of less than 300 µs is achievable.

6.4. Resource utilization The resource allocation in the FPGA is shown in Fig. 11, and the resource utilization is shown in Table I. The total usage rate for FF and LUT, was approximately 2% and 3%, respectively. The lookup engine's Topic Table and Subscriber Table increased BRAM consumption, but it was only about 10% of the total resources. As a result, it was discovered that there was room on the M-KUBOS board for additional applications to be implemented.

We used the UDP stack in this implementation, but there is also an FPGA implementation of the TCP stack that supports a 10 Gbps line rate [6]. The TCP/IP stack resource usage was 78 248 for FF, 62 545 for LUT, and 315 for BRAM in the Virtex-7 series [5]. Assuming the same number of resources, this implementation's resource utilization is 7.48% for FF, 11.97% for LUT, and 16.01% for BRAM. Therefore, it is possible to implement it using the TCP/IP.

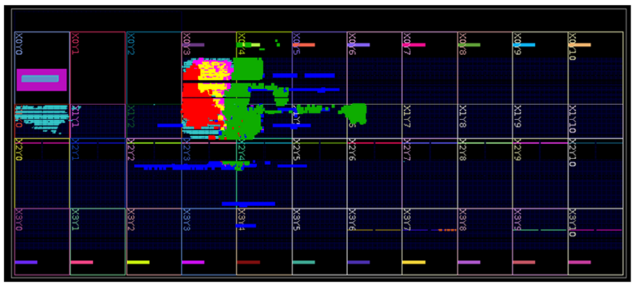


Fig 11. Implementation design

Table I. Resource utilization

Module	FF	LUT	BRAM
MQTT parser	1015	3343	0
Lookup engine	265	501	108
Message queue	24	1652	18.5
MQTT packet generator	1938	296	0
UDP/IP Stack and 10GbE subsystem	7550	6479	6
Total	21 901	18 532	151
Available	1 045 440	522 720	984

7. Discussion

In this paper, we implemented an FPGA-based MQTT broker with the goal of meeting the following requirements. The target performance has been defined based on the network infrastructure and application requirements in recent years.

- To accommodate the network bandwidth in modern network environments, the throughput should be at least 10 Gbps.
- The processing delay time in the broker should not exceed 300 µs. This value is calculated based on IEC61000-4, which states that the time accuracy of control in grid stability should be approximately 1 ms [8].
- According to IEEE C37.118.1-2011 [9], a time synchronization accuracy of ±26 µs or better is required when assuming communication between PMUs that measure the phase of voltage and current.

In Section 6.2, the hardware-implemented MQTT broker was shown to support 10 Gbps line rate processing, achieving a throughput 107 times higher than the existing software-implemented MQTT broker.

In Section 6.3, we measured the processing time of the hardware-implemented MQTT broker itself. The processing time was 799.2 ns, which was within the target of 300 µs. The actual time taken from the publisher to the subscriber is the processing time of the MQTT broker plus the delay time of the network lines and network switches. In Section 6.1, we actually built an evaluation environment and measured the time from publisher to subscriber. With the hardware-implemented MQTT broker in the evaluation environment, the time from publisher to subscriber was 87.37 µs, achieving the target of 300 µs.

In Section 6.1, we also measured the packet arrival time difference between subscribers to investigate the accuracy of the time synchronization of the hardware-implemented MQTT broker. This indicates that the proposed MQTT broker is capable

of time synchronization using regular PUBLISH/SUBSCRIBE messages as proposed in Section 4.2. In the case of the hardware-implemented MQTT broker, the packet arrival time difference between 100 subscribers was 19.92 μs, achieving the target of 52 μs.

In Section 6.4, it was discovered that there was room on the M-KUBOS board for additional functions to be implemented. The implementation of this study was based on MQTT-v3.1.1. In MQTT-v3.1.1, QoS, Will, Retain functions, and single-level wildcards are optional and not required. Therefore, our MQTT broker is fully functional. However, in terms of resources, these additional functions can be implemented. In particular, the Retain function can be implemented by storing the latest PUBLISH message in the Topic Table and distributing it during SUBSCRIBE. In addition, ICMP module has been released in the research of FPGA implementation of TCP stack [6], and the Will function can be realized by using this module for dead/active management.

The above evaluation shows that the proposed MQTT broker and time synchronization method meet the target requirements. This shows that our proposed destination search in lookup engine and time synchronization methods are effective. In the real environment such as smart buildings, smart factories, and smart cities, high throughput, low latency, and low jitter are required. In a smart grid, the time accuracy of power system stabilization control must be about 1 ms [8]. Also, a time synchronization accuracy of ±26 μs or better is required when assuming communication between PMUs that measure the phase of voltage and current [9]. According to 3GPP TR 22.804 [25], the acceptable delay for machine control in a smart factory is between 1 and 100 ms, and the number of devices to be controlled is less than 100 at most. Therefore, low latency is required in these cases. This study shows that our hardware-implemented MQTT broker is capable of 10Gbps line-rate processing and has lower latency than the software-implemented MQTT broker. The low latency of our MQTT broker allows it to be used in environments such as the above.

8. Conclusion

We proposed an FPGA-based MQTT broker as well as a method for time synchronization. Among 100 subscribers, the proposed MQTT broker supports 10 Gbps line-rate processing and achieves a processing delay time of less than 300 μs and a time synchronization accuracy of 52 μs. As a result, our implementation has low latency, high throughput, and low jitter. In addition, it can be used for grid stability control in smart grids and machine control in smart factories.

Acknowledgments

This work was supported by JST CREST Grant Number JPMJCR19K1, MEXT/JSPS KAKENHI Grant (B) Number JP20H02301, and the commissioned research by National Institute of Information and Communications Technology, Japan (NICT, Grant Number 22004).

References

(1) AWS IoT-Amazon Web Services. *Amazon Web Services*. https://aws.amazon.com/iot/. Accessed January 06, 2021.
(2) Cloud IoT Core. *Google Cloud*. https://cloud.google.com/iot-core. Accessed January 06, 2021.
(3) IoT Edge Connect—Akamai. https://www.akamai.com/us/en/products/performance/iot-edge-connect.jsp. Accessed January 06, 2021.
(4) IEEE draft standard for a smart transducer interface for sensors and actuators common functions, communication protocols, and transducer electronic data sheet (TEDS) formats. *IEEE* Std P1451.0/D6.04, 2006.
(5) P21451-1-6 - Standard for a smart transducer interface for sensors, actuators, and devices - Message queue telemetry transport (MQTT) for networked device communication.
(6) Sidler D, Alonso G, Blott M, Karras K, Vissers K, and Carley R. Scalable 10Gbps TCP/IP stack architecture for reconfigurable hardware. *Proc. of 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015; 36–43. https://doi.org/10.1109/FCCM.2015.12
(7) Fukuhara A, Iwai T, Sakuma Y, Nishi H. Implementation of content-based anonymization edge router on NetFPGA. *Proc. of2019 IEEE 13th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC)*, 2019; 123–128. https://doi.org/10.1109/MCSoC.2019.00025
(8) Nishi H. Infrastructure and services for smart community. *The Journal of the Institute of Electronics, Information and Communication Engineers* 2015; **98**(2):112–117.
(9) IEEE standard for synchrophasor measurements for power systems. *IEEE Std C37.118.1–2011 (Revision of IEEE Std C37.118–2005)*, 2011. https://doi.org/10.1109/IEEESTD.2011.6111219
(10) Brasilino LRB, Swany M. Low-latency CoAP processing in FPGA for the internet of things. *Proc. of 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and SocialComputing (CPSCom) and IEEE Smart Data (SmartData)*, 2019; 1057–1064. https://doi.org/10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00182
(11) Pipatsakulroj W, Visoottiviseth V, Takano R. muMQ: a lightweight and scalable MQTT broker. *Proc. of 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2017; 1–6. https://doi.org/10.1109/LANMAN.2017.7972165
(12) mtcp-stack/mtcp. *GitHub*. https://github.com/mtcp-stack/mtcp. Accessed January 06, 2021.
(13) Data Plane Development Kit. *DPDK*. https://www.dpdk.org/. Accessed January 06, 2021.
(14) Light RA. Mosquitto: Server and client implementation of the MQTT protocol. *Journal of Open Source Software* 2017; **2**(13):265. https://doi.org/10.21105/joss.00265.
(15) Arai K, Murakami M. Trends in standardization of high precision time and frequency synchronization technologies for 5G Mobile networks. *Global Standard Frontline NTT Network Service Systems Laboratories*, 2018; 5.
(16) Horita K, Shiobara S, Okamawari T, Teraoka F, Kaneko K. PTP accuracy measurement comparison between boundary clock and VLAN priority. *Proc. of 2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2017; 1–6. https://doi.org/10.1109/ISPCS.2017.8056748
(17) IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588–2008 (Revision of IEEE Std 1588–2002)*, 2008; 1–300. https://doi.org/10.1109/IEEESTD.2008.4579760
(18) FPGA computing platform M-KUBOS PALTEK corporation. *PALTEK corporation*. https://www.paltek.co.jp/design/original/m-kubos/index.html. Accessed January 05, 2021.
(19) MQTT Version 3.1.1 OASIS Srandard 29 October 2014. *OASIS*. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Accessed August 21, 2021.
(20) Wang H, Jin Y, Wang W, Ma J, Zhang D. The performance comparison of PRSCTP, TCP and UDP for MPEG-4 multimedia traffic in mobile network. *International Conference on Communication*

Technology Proceedings, 2003. ICCT 2003, 2003; 403–406: **1**. https://doi.org/10.1109/ICCT.2003.1209108

(21) Janevski T, Pelivanoska K. Performance evaluation during handover in WLAN network with different transport protocol variants. *2012 20th Telecommunications Forum (TELFOR)*, 2012; 87–90. https://doi.org/10.1109/TELFOR.2012.6419154

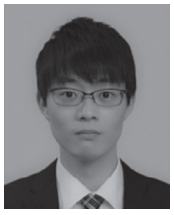
(22) 10G/25G ethernet subsystem. https://www.xilinx.com/products/intellectual-property/ef-di-25gemac.html. Accessed December 23, 2020.

(23) alexforencich/verilog-axis: Verilog AXI stream components for FPGA implementation. https://github.com/alexforencich/verilog-axis. Accessed December 25, 2020.

(24) Waqar M, Kim A, Cho PK. A transport scheme for reducing delays and jitter in ethernet-based 5G fronthaul networks. *IEEE Access* 2018; **6**:46110–46121. https://doi.org/10.1109/ACCESS.2018.2864248.

(25) 3GPP TR 22.804. *Study on communication for automation in vertical domains*. http://www.3gpp.org/DynaReport/22804.html. Accessed August 21, 2021.

Koutarou Yamamoto (Non-member) received his B.E. degree from Keio University, Japan, in 2021. Currently, he is working toward a master's degree in Keio University. His research interest includes FPGA-based MQTT broker and service migration.



Akihiro Fukuhara (Non-member) received his M.S. degree from Keio University, Japan, in 2021. His research interest includes FPGA-based MQTT broker and data anonymization.

Hiroaki Nishi (Member) has been a researcher with Real World Computing Partnership from 1999 and has been at the Central Research Laboratory, Hitachi Ltd. from 2002. He has been a Professor at Keio University since 2014. He is the chair of IEEE P21451-1-6 and IEEE P2992, and member of IEEE 1451 families, IEEE P2668, and IEEE P2805. He was also a member of the ITU-T Focus Group on Smart Sustainable Cities WG2. He has been a member of several committees established by the Ministry of Internal Affairs and Communications. The main theme of his current research is a total network system that includes the development of hardware and software architecture.