Distributed MQTT Brokers Infrastructure with Network Transparent Hardware Broker

Tokimasa Toyohara
Graduate School of Science and Technology,
Keio University
3-14-1 Hiyoshi, Kohoku, Yokohama,
Kanagawa 223-8522, Japan
toyohara@west.sd.keio.ac.jp

Hiroaki Nishi
Department of System Design,
Faculty of Science and Technology,
Keio University
3-14-1 Hiyoshi, Kohoku, Yokohama,
Kanagawa 223-8522, Japan
west@sd.keio.ac.jp

***Abstract*—In recent years, the number of IoT devices has been rapidly increasing. Therefore, MQ Telemetry Transport (MQTT) has attracted attention as a lightweight protocol for IoT devices. MQTT employs a publish/subscribe model of communication via a broker, and since the performance of the MQTT broker directly affects the quality of service, various studies have been conducted to improve its performance. Among them, the hardware implementation of MQTT broker has higher performance than software implementations in terms of publish throughput and processing delay. However, there are some problems such as inflexibility and resource limitations. Therefore, we propose a distributed MQTT brokers infrastructure using the hardware broker. In this architecture, the software broker on the cloud and the hardware broker at the edge region work in cooperation to achieve low latency processing at the edge region and software flexibility. The hardware broker operates network transparently, so that end devices communicate with the software broker in the cloud and do not need to be aware of the presence of the hardware broker at the edge region. This enables resource and processing load distribution without affecting end devices. We implemented the network transparent hardware broker in a field-programmable gate array (FPGA) and evaluated it.**

Keywords—MQTT, FPGA, edge computing, network transparency

I. Introduction

In recent years, the number of Internet of Things (IoT) devices has increased rapidly. In IoT devices, which generally require limited resources and low power consumption, the lightweight MQ Telemetry Transport (MQTT) [1] is widely used as a communication protocol. This study focuses on MQTT. MQTT uses a publish-subscribe communication method via a broker, which offloads functions such as setting the destination of data and sending data to multiple destinations to reduce the number of communications. In addition, the simple packet structure reduces power consumption during communication.

In MQTT, all communication is concentrated in the broker. Inadequate performance of the MQTT broker in terms of delivery capacity or communication throughput can lead to increased processing times and reduced availability. This is especially a serious problem in applications with limited tolerable latency, such as machine control, and power system stability. Therefore, MQTT broker acceleration has been studied. Software brokers have been studied, including muMQ [2] accelerated by multi-core processing, load balancing with multiple brokers [3]-[11], and PAMS [12] to deploy brokers on network switches,. For hardware brokers, the hardware implementation of MQTT broker using the field programmable gate array (FPGA) has been studied [13].

In particular, the hardware broker has a publish throughput of 100 Mmps (messages per second) [13]. The maximum PUBLISH throughput of muMQ, a software broker, is 930,275 mps [2]. Therefore, the hardware implementation is shown to provide 107 times the publish throughput. In addition, the processing delay of the hardware implementation is 799.2 ns. If a mechanism can be built to enable larger networks to take advantage of these benefits, services that have been difficult to provide due to latency and scale will be possible.

The disadvantages of the hardware broker are that the number of clients that can be registered is limited by on-chip memory resources, and it is less flexible than a software-based broker as they do not support common MQTT options such as Will and Retain. One possible solution to the resource limitation is to use external memory instead of Block RAM (BRAM), which is on-chip memory. However, the cycles required for memory access become a bottleneck, and the benefits of hardware implementation cannot be fully realized due to lower overall throughput and increased processing delays. Another solution is to install multiple hardware brokers, but this would require management of IP addresses for the number of brokers on the end device side, and the benefits of MQTT could not be fully utilized. It is also difficult from a resource perspective to implement options such as Will and Retain in a hardware broker because the messages corresponding to the topics must be stored in the broker.

To solve these issues and enable larger networks to benefit from the hardware broker, this paper proposes a new distributed MQTT broker infrastructure using a network transparent hardware broker. The contributions of this paper can be summarized as follows:

- This paper proposes a new distributed MQTT broker infrastructure in which the Host Broker, a software broker running in the cloud, and the Transparent Broker, a hardware broker running on the communication path at the edge region, cooperates to perform processing. Therefore, the infrastructure achieves both low latency processing at the edge region and software flexibility.
- The Transparent Broker operates transparently over the network, so that end devices communicate with the Host Broker in the cloud and do not need to be aware of the hardware broker on the edge. In other words, in the distributed MQTT broker infrastructure, end devices can benefit from high-performance processing without prior knowledge of the Transparent Broker. It also enables resource and processing load balancing

2379-1896/23/$31.00 ©2023 IEEE
DOI 10.1109/CANDAR60563.2023.00032

among Transparent Brokers and between Transparent Broker and Host Broker without affecting end devices.

- To realize the proposed infrastructure, we implement the Transparent Broker, a network-transparent hardware broker, which has network-transparent processing, and resource and processing load-balancing functions through memory operations. Although the proposed mechanism can be adapted to conventional TCP-based MQTT, we assume MQTT For Sensor Networks (MQTT-SN) [14] and implement MQTT broker using UDP. The implementation is an extension of the hardware implementation of the MQTT broker [13], and is evaluated in terms of throughput, latency, time synchronization accuracy, and resource utilization based on similar target performance. In addition, to verify whether low latency and low jitter applications can be implemented using MQTT, time synchronization is performed over MQTT as one of the evaluation applications, and its accuracy is evaluated.

The rest of this paper is organized as follows. Section II introduces related studies on MQTT broker acceleration and network transparency. Section III explains distributed MQTT infrastructure, and Section IV describes network transparent hardware broker in detail. We present evaluation in Section V, and finally conclude this paper in Section VI.

II. Related Work

A. Software MQTT broker acceleration

There are existing studies to improve the performance of software MQTT brokers.

muMQ [2], a lightweight and scalable MQTT broker, was proposed. It used a highly scalable user-level TCP stack for multicore systems (mTCP) [15] and Data Plane Development Kit (DPDK) [16] to accelerate processing. Publish throughput is up to 930,275 mps, which is 5.38 times of Mosquitto [17]. However, the publish throughput is inferior to that of the hardware broker.

Cooperation of multiple MQTT brokers for load balancing has also been studied: ILDM [3], MQTT-ST [4], D-MQTT [5], a scalable MQTT system based on the Chord mechanism [6], DMLT [7], and SoD-MQTT [8] all involve multi-hop transfers between brokers and can cause high latency. Although Detti et al. [9] reduced inter-broker communication, additional functionality is needed for end devices to manage sessions with multiple brokers. EdgePub [10] reduced latency by deploying a broker on the end devices. However, the end-device needs to be modified. TD-MQTT [11] is a transparent distributed MQTT broker, and subscribers can access the broker transparently through the redirection process, but it is not transparent to publishers. Our proposal does not introduce latency due to multi-hop forwarding because the broker is distributed over the communication path. Moreover, since it is network transparent, no modification to the end device is required.

PAMS [12] was proposed to accelerate MQTT-SN networks by implementing some of the broker functions in network switches. It uses in-network acceleration to reduce communication latency. Although similar to the proposed method in that part of the broker function is performed in the network, its implementation in software switches is inferior to the hardware broker in terms of performance.

Several of the methods presented in this section can be adapted to the software broker in the infrastructure proposed in this paper.

B. FPGA application

Field-programmable gate arrays (FPGAs) feature lower power consumption, lower latency, and higher throughput than conventional processors in certain applications. FPGAs have become popular in network applications because their pipelined architecture is efficient for processing network streams. Therefore, FPGA implementations for data centers and edge computing environments have been proposed [18][19]. In addition, FPGA migration methods [20] have been proposed to provide more flexibility for FPGA applications. This is accomplished by replicating the circuit through dynamic reconfiguration and reading and writing the internal state of the Flip-Flops (FFs) by scan-chain [21].

Our proposed method writes Block RAM (BRAM) values, which is an on-chip memory.

C. Hardware implementation of MQTT broker

The hardware implementation of MQTT broker has been proposed [13]. In subscriber lookup, two hash tables implemented in BRAM on FPGA are used to enable destination lookup that can be performed in a certain time. It is implemented on FPGA, with QoS set to 0, and supports only UDP. Targets were set based on recent network infrastructure and application requirements, achieving line-rate processing of 10 Gbps and processing latency of less than 300μs by a latency of 799.2 ns. Publish throughput was 100 Mmps which is 107 times higher than muMQ [2]. In addition, to verify whether low latency and low jitter applications can be implemented using MQTT, time synchronization was performed over MQTT as one of the evaluation applications, and its accuracy was evaluated. Assuming time synchronization between phasor measurement units (PMUs), time synchronization accuracy within 52μs was achieved among 100 PMUs. These evaluations showed that this hardware broker has high throughput, low latency, and low jitter.

Although the implementation was UDP, TCP support was also mentioned. From a resource perspective, it was found that a TCP/IP stack [18] supporting a line rate of 10 Gbps could be implemented. In addition, the processing delay of this TCP/IP stack was found to be 5 μs [18]. Therefore, when added to 536.8 ns, which excludes the processing delay of the UDP/IP stack in the implementation, the processing delay time target is found to be reached.

We extend this hardware implementation to implement a network transparent hardware broker. It thereby solves the issues of flexibility and resource constraints while maintaining the performance described above.

D. Network transparency

Network transparency is a style of service delivery that provides edge services cooperating with the cloud at intermediate locations on the Internet without affecting the end devices [22]. End devices do not need to be aware of the existence of services deployed on the edge; they can benefit from the services simply by communicating toward the cloud. This characteristic allows for flexible service provisioning, such as moving the service locations or load balancing, without changing the end-device network settings.

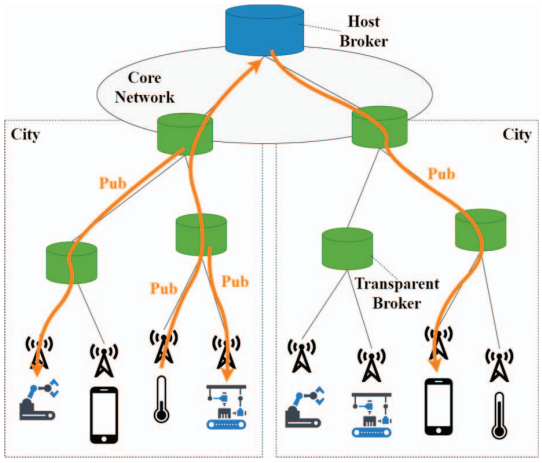

Fig. 1. Proposed infrastructure

Authorized stream content analysis (ASCA) [23][24] is a technology for analyzing packet streams at intermediate points on the Internet to provide network transparent services, while considering privacy. It enables flexible service provision by analyzing layer seven information. To ensure privacy, only streams authorized by the Opt-In method are analyzed as pre-authentication. Even encrypted streams can be analyzed by receiving a common key from the cloud [25].

In our proposal, the hardware broker has network transparency and performs the processing without any modification to the end devices.

III. Proposed Infrastructure

In this paper, we propose a distributed MQTT broker infrastructure. It consists of a software broker running in the cloud and a hardware broker running on the communication path at the edge region, which cooperates to perform processing. Fig. 1 shows an image of the proposed infrastructure. The Host Broker, a software broker, is located at the top, and the Transparent Broker, a hardware broker, is distributed in the middle region. End devices send packets to the Host Broker, and the Transparent Broker captures and processes the packets along the communication path. The proposed mechanism enables the following.

First, a hardware broker installed at the edge can distribute messages with high throughput, low latency, and low jitter. As mentioned in Section 2, hardware brokers have high performance and can be installed at the edge region to reduce communication latency. This allows MQTT to be adapted to applications with limited tolerable latency, such as machine control and power system stability.

Second, the use of a software broker provides flexibility that cannot be obtained with a hardware broker The MQTT options Will and Retain require the broker to store the topic and corresponding message, which is difficult to achieve with a hardware broker from a memory resource perspective. Therefore, these options are handled by the software broker. Other extensions can be handled in the same way.

Finally, the network-transparent distributed infrastructure eliminates the need for any modifications to the end device. The hardware broker in this architecture is network transparent, processing packets destined for the software broker along the communication path. End devices

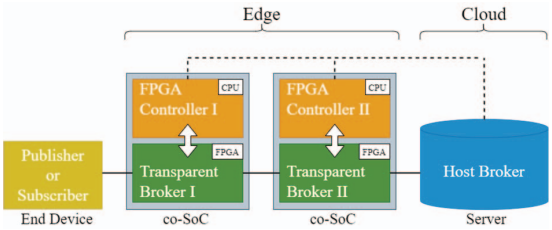

Fig. 2. Assumed environment

communicate with the software broker in the cloud and do not need to be aware of the hardware broker at the edge region. Therefore, end devices can benefit from high-performance processing without prior knowledge of the hardware broker. Similarly, it is possible to transparently load-balance resources and processing. In particular, since the hardware broker has limited memory resources, it is possible to save resources by distributing processing to other hardware brokers or software broker when memory is tight. In addition, processing can be executed by a specific broker depending on the requested delay for each application.

A. *Two types of MQTT brokers*

Two types of MQTT brokers are used in this infrastructure: the Host Broker and the Transparent Broker. There is also an FPGA Controller as a controller that controls the Transparent Broker.

The Host Broker is a software broker. It is the only one located upstream of the network, such as in the cloud, and has its own IP address. Although the software broker has lower processing performance than the hardware broker, it can use ample resources such as memory and storage. Since the Host Broker receives all MQTT packets, it knows the correspondence between all topics and messages. It also keeps track of the availability of all Transparent Brokers.

The Transparent Broker is a hardware broker. It is distributed over midstream communication paths in the network, such as in the edge region, and performs processing with network transparency. Network transparency is a style of service delivery that provides edge services cooperating with the cloud at intermediate locations on the Internet without affecting the end devices. The Transparent Broker analyzes packets that pass through it and captures MQTT packets destined for the Host Broker. For non-target packets, it works completely as a wire. Although the hardware broker has limited resources compared to the software broker, it can provide high throughput, low latency, and low jitter. Deploying a hardware broker in the edge region enables message distribution with minimal network latency and processing delay. The Transparent Broker is assumed to be deployed on FPGAs on CPU-FPGA co-system on chips (SoCs) such as Xilinx's ZYNQ series.

The CPU on the SoC executes the FPGA Controller. It controls the writing of BRAM contents and the associated packet buffering for the Transparent Broker on the same SoC. The FPGA Controller has its own IP address and communicates with the Host Broker, notifying the Host Broker of the operating status of the Transparent Broker and controlling the Transparent Broker in response to commands from the Host Broker.

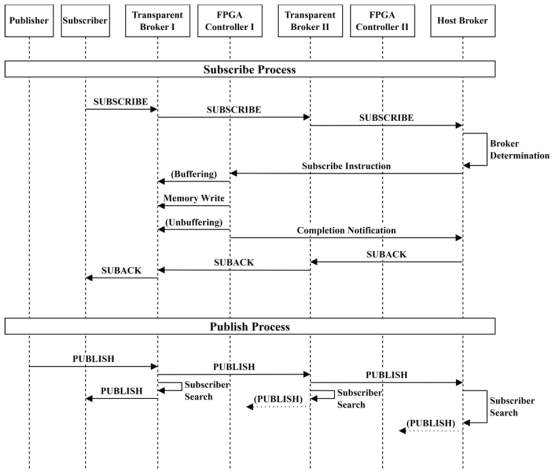

Fig. 3. The flow of publish and subscribe process

B. *Publish and subscribe processing*

The following describes the processing procedures in publish and subscribe. Assume the environment shown in Fig. 2. One Host Broker is installed in the cloud, which is upstream of the network, and Transparent Brokers are distributed over the communication path in the edge region, which is midstream of the network. In Addition, end devices such as sensor nodes and mobile terminals are connected to the network. The flow of each process is shown in Fig. 3.

First is the subscribe process. When an end device sends a SUBSCRIBE packet to a Host Broker, all Transparent Brokers on the communication path forward the packet without any processing. When the Host Broker receives a Subscribe packet, the Host Broker decides which Broker to process it. At this time, either one of the Transparent Broker on the communication path or the Host Broker can be selected. If the Transparent Broker is selected, the Host Broker issues a subscribe instruction to the FPGA Controller of the Transparent Broker. The instruction is communicated via an HTTP or similar and contains information about the subscriber. The FPGA Controller writes the Transparent Broker's BRAM based on the received information in order to add subscriber entries. At this time, if writing to an address affects existing processing, the MQTT packets is temporarily buffered to maintain processing consistency. When the writing is completed, the FPGA Controller returns a completion notification to the Host Broker. If packet buffering has been performed, the buffer is released. Upon receiving the completion notification, the Host Broker returns a SUBACK packet to the end device. If the Host Broker is selected, it adds a subscriber entry for itself and returns a SUBACK packet.

The selection of brokers is judged comprehensively based on the operating status and memory usage of each Transparent Broker, the positional relationship between publisher and subscriber, the acceptable delay of applications using MQTT, and other factors. The acceptable delay of the application can be specified from the end device by using User Properties in MQTT version 5.

Next is the publish process. When an end device sends a PUBLISH packet to the Host Broker, the Transparent Broker on the communication path searches for the subscriber corresponding to the topic in its own entry. If they are found, the Transparent Broker sends a PUBLISH packet to the subscriber. At this time, the Transparent Broker behaves as if a PUBLISH packet has been sent from the Host Broker. This means that the end device does not need to be aware of the existence of the Transparent Broker. At the same time as the subscribers are searched, the received PUBLISH packet is forwarded to the Host Broker. All Transparent Brokers on the communication path perform the same process. The Host Broker similarly checks its own entry and performs the publish process.

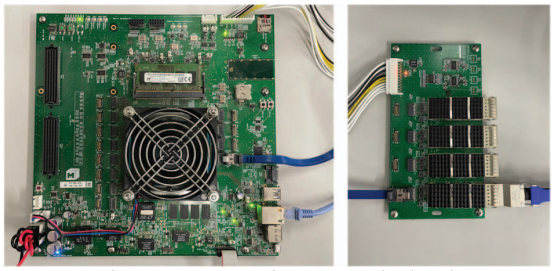

Fig. 4. M-KUBOS and QSFP extension board

This architecture enables the distribution of messages with minimal network and processing delays, as long as the Transparent Broker's resources permit. When the Transparent Broker is not activated, the end devices communicate with the Host Broker as in conventional MQTT communication. The Host Broker receives all PUBLISH and SUBSCRIBE packets and knows the overall relationship among topics, messages, publishers, and subscribers. It also knows the entry information of the Transparent Broker and is therefore responsible for overall control. The MQTT options Will and Retain are also handled by the Host Broker.

C. *Resource and processing load balancing*

It is sometimes desirable to change the broker that performs the processing in response to changes in resource utilization, changes in the acceptable delay of the application, or the concentration of processing on a particular broker. For this purpose, we propose a load-balancing technique by manipulating the memory contents of the hardware broker. In this load-balancing method, the Host Broker also assumes the role of orchestrator for overall control. The specific procedures are the same as those described for subscribe process in Section 3-B. When the Host Broker issues a memory write instruction, the FPGA Controller executes the operation on the Transparent Broker. The Transparent Broker has network transparency and can load-balance without any modification to the end devices.

IV. Implementation

In this paper, we implemented the Transparent Broker and FPGA Controller, which are hardware brokers. Although the proposed mechanism can be adapted to conventional TCP-based MQTT, we assume MQTT-SN and implement MQTT broker using UDP. The Host Broker, which is a software broker, can be implemented by extending the software implementation of the existing MQTT Broker.

A. *Implementation Environment*

M-KUBOS [26], an FPGA computing platform developed by PALTEK Corporation, was selected as the hardware-based computing node. It is equipped with a Zynq UltraScale+ MPSoC, XCZU19EG2FFVC1760, and has an ARM Cortex-A53 64-bit quad-core processor and Cortex-R5 quad-core

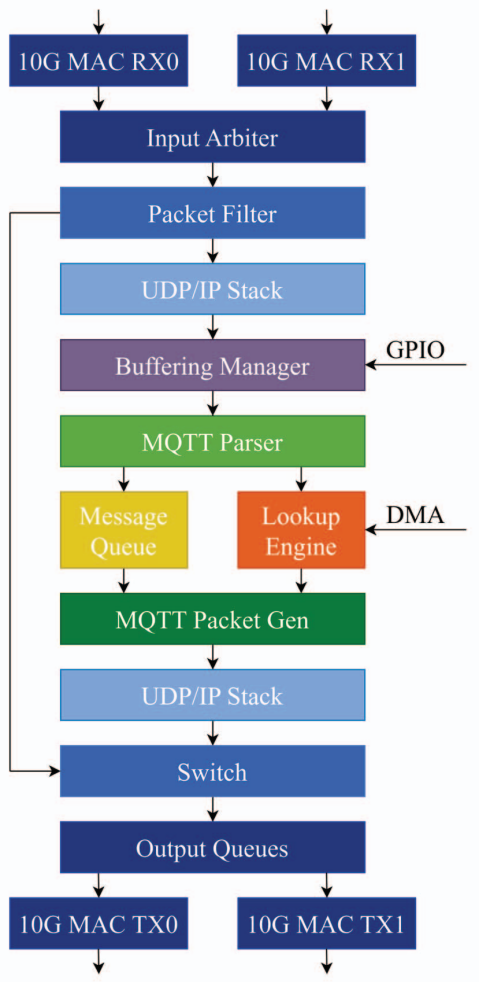

Fig. 5. Hardware design

processor as core processors. The architecture uses the PYNQ [27] Python framework to manipulate programmable logic (PL) components from processing system (PS) components. In this study, the Transparent Broker was implemented in the PL component and the FPGA Controller in the PS component, and PYNQ was used to control GPIO and DMA transfer for buffering control and memory operations. Fig. 4 shows the M-KUBOS board. In this implementation, a QSFP GTY transceiver extension board that to the mainboard using a FireFly cable was designed and used to support 10 Gbps networks. The development environment was Vivado 2019.1, and the development language was Verilog HDL.

B. *Hardware Design*

We implemented the Transparent Broker as an extension of the hardware implementation of the MQTT broker proposed in the previous study [13]. For options, as in the previous study, QoS was set to 0, and the Will and Retain mechanisms were not supported by the Transparent Broker, since they are handled by the Host Broker. In addition, UDP packets were targeted.

Fig. 5 shows the hardware design. The AXI4-Stream protocol was used to connect the modules. We developed an original network interface card (NIC) model based on the reference NIC model [28] of NetFPGA [29]. Two 10Gbps

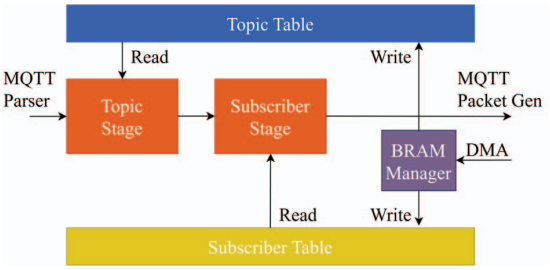

Fig. 6. Lookup Engine

ports are provided for uplink and downlink to support network transparent processing. The input arbiter module places bits on TUSER bus of the AXI stream according to the input port, and the output queue module determines the output port based on the bits on the TUSER bus. This allows packets to flow from uplink to downlink and downlink to uplink, acting as wires. The broker architecture is implemented between the input arbiter and the output queue module.

The packet filter module determines whether the input packet is an MQTT packet addressed to the Host Broker by IP address and port number. In addition, it also checks if the packet is a PUBLISH packet. If not, the packet is bypassed to the switch module. If so, the packet is duplicated and forwarded to both the broker logic and the switch module. Then, the UDP/IP stack removes the preamble, SFD, and FCS, and performs IP and UDP checksum verification. Since this design is not a stand-alone server and processing is performed network transparently, only the necessary functions were carved out from the open-source UDP/IP stack [30]. If a buffering instruction is issued by the FPGA Controller, packets are buffered into the buffer inside the buffering manager. The MQTT parser extracts the message type and topic from the incoming packet, and the lookup engine searches for the destination based on the topic. The message queue holds messages during processing by the lookup engine. The MQTT packet generator generates MQTT packets based on the destination information output from the lookup engine and messages retrieved from the message queue. The MQTT packet generator generates MQTT packets based on the destination information output from the lookup engine and messages retrieved from the message queue. The generated MQTT packets are sent through the UDP/IP stack and the switch module. The switch module sends packets sequentially in a round-robin manner. No major modifications have been made to the MQTT parser, the message queue, and the MQTT packet generator from previous study [13]. The functions of the other modules are described below.

C. *Buffering manager*

The buffering manager and buffer are controlled by GPIOs from the FPGA Controller. When a buffering instruction is issued, input packets are stored in the buffer, and when an un-buffering instruction is issued, packets are sequentially removed from the buffer.

D. *Lookup engine*

The lookup engine consists of two hash tables, the Topic Table and the Subscriber Table, as shown in Fig. 6. The Topic Table takes a hash value of a topic name as a key and returns the number of subscribers to that topic as a value. The Subscriber Table takes a topic name and id as keys and returns the subscriber's destination information as a value. The id is a

number in the range from 1 to the number of subscribers. By repeatedly accessing the subscriber table while increasing the number of ids, the destination information of all subscribers can be retrieved. In this implementation, the Topic Table is 12-bits in width and 65,536 words in depth, while the Subscriber Table is 48-bits in width and 65,536 words in depth.

In the previous study [13], the lookup engine consisted of two dual-port BRAMs and a three-stage pipeline to support both publish and subscribe processes. However, in this implementation, the pipeline was changed to a two-stage pipeline in order to use only the publish process. Instead, the free BRAM port was connected to the BRAM Manager. The BRAM Manager writes to each BRAM from the address and value received from the FPGA Controller via DMA transfer. The contents of the two hash tables are written directly from FPGA Controller, replacing the subscribe process.

E. FPGA Controller

Since M-KUBOS is equipped with a Zynq UltraScale+ MPSoC, the PL components can be controlled from the PS components. The PS components operates as an FPGA Controller and receives commands from the Host Controller to operate the PL components, the PS components operates as the FPGA Controller and receives commands from the Host Controller to operate the PL components.

V. Evaluation

A. Throughput

This implementation added functionality without modifying the pipeline of the prior study's architecture. Therefore, the delivery performance was maintained and a publish throughput of 100 Mmps was achieved. Similarly, an SFP+ line rate of 10 Gbps was achieved. Although this mechanism has two links, uplink and downlink, MQTT packet processing operates only on the input from the downlink, so 10 Gbps performance is not a problem.

B. Latency

Latency was measured by using Integrated Logic Analyzer [31]. It captures actual internal waveforms of packet processing. The overall latency of this implementation was 806.4 ns with the MQTT broker, and 377.6 ns without the MQTT broker. The latency of the MQTT broker logic is not identical to previous study [13] due to the insertion of the Buffering Manager and modification of the Lookup Engine pipeline. The other latency is the latency of the NIC-related logic. This resulted in a processing latency of less than 300 µs.

C. Time synchronization accuracy

In terms of time synchronization accuracy, previous study [13] has shown that the measurement time was affected 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. Therefore, the effect of jitter caused by the MQTT broker is not considered in this implementation. Due to the round-robin output of the switch module, time synchronization accuracy takes twice as long in this implementation. However, previous study has achieved time synchronization accuracy of up to 600 subscribers, we believe that a time synchronization system of up to 300 subscribers can be achieved in this implementation. Therefore, a time synchronization system with 100 subscribers was achieved.

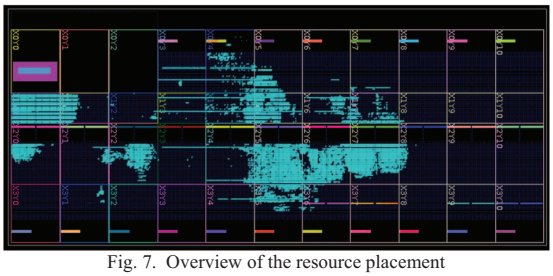

Fig. 7. Overview of the resource placement

Table I. Resource utilization

Resource	Utilization	Available	Utilization (%)
LUT	40,360	522,720	7.72
FF	51,695	1,045,440	4.94
BRAM	261	984	26.52

D. Resource utilization

The FPGA resource allocation is shown in Fig. 7, and the resource utilization is shown in Table I. The total usage rate was approximately 2-3 times higher for FF, LUT, and BRAM, respectively, compared to the previous study [13]. This was due to the addition of CPU connection logic and NIC-related logic, as well as the addition of FIFOs with these logics. Nevertheless, there was still room for additional applications to be implemented on the M-KUBOS board.

In this implementation, the topic table and the subscriber table were implemented with the same size with previous study in order to compare resources. The maximum depth of these tables that could be implemented on the M-KUBOS board was investigated. The maximum depth was 524,288 words, at which BRAM utilization was 60.06%.

E. Memory writing time

Writing to two BRAMs by the BRAM Manager completes in one clock cycle. The MQTT logic runs at 100 MHz, so a write to one address completes in 10 ns. The MQTT logic runs at 100 MHz, so a write to one address completes in 10 ns, and a write to n addresses completes in $10n$ ns.

Service downtime occurs when packet buffering is performed during BRAM writing. Service downtime during migration must be 1 ms or less, excluding network delays for supporting latency-sensitive services, such as ancillary services [32]. Therefore, this architecture allows up to 100,000 addresses to be written sequentially at the same time.

VI. Conclusion

In this study, we proposed a distributed MQTT broker infrastructure with a network transparent hardware broker. The Host Broker, a software broker in the cloud, and the Transparent Broker, a hardware broker at the edge region, work in concert to achieve low-latency processing at the edge region and software flexibility. The Transparent Broker operates network transparently, so end devices communicate with the Host Broker and do not need to be aware of the Transparent Broker's presence at the edge region. This enables resource and processing load distribution without affecting the end devices. We implemented and evaluated the network-transparent hardware Broker on M-KUBOS. We assume MQTT-SN and implement MQTT broker using UDP. As a result, 10Gbps line-rate processing, processing delay of less than 300µs, and time synchronization accuracy within 52 µs was achieved among subscribers were achieved while

acquiring network transparent processing and memory write function by on-chip CPU.

In the future, we plan to implement the Host Broker and verify processing in cooperation with the Transparent Broker. In addition, although this implementation was done using UDP for MQTT-SN, we are planning to extend it to TCP. For TCP support, the Transparent Broker needs to detect the session between the end device and the Host Broker and send packets from the Transparent Broker so that the TCP session matches up from the end device's perspective. These functions have already been studied in software [22][23]. By extending these functions to hardware, the proposed method can be extended to TCP and support MQTT.

ACKNOWLEDGMENT

This paper is based on the results obtained from the JST CREST Grant Number JPMJCR19K1.

REFERENCES

[1] MQTT, https://mqtt.org/ (accessed Aug. 8, 2023).

[2] W. Pipatsakulroj, V. Visoottiviseth and R. Takano, "muMQ: A lightweight and scalable MQTT broker," 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Osaka, Japan, 2017, pp. 1-6, doi: 10.1109/LANMAN.2017.7972165.

[3] Ryohei BANNO, Jingyu SUN, Susumu TAKEUCHI, Kazuyuki SHUDO, "Interworking Layer of Distributed MQTT Brokers, IEICE Transactions on Information and Systems, " 2019, Volume E102.D, Issue 12, Pages 2281-2294, Released on J-STAGE December 01, 2019, Online ISSN 1745-1361, Print ISSN 0916-8532, https://doi.org/10.1587/transinf.2019PAK0001.

[4] R. Banno and K. Osawa, "Acceleration of MQTT-SN protocol using P4," 2022 IEEE 11th International Conference on Cloud Networking (CloudNet), Paris, France, 2022, pp. 16-21, doi: 10.1109/CloudNet55617.2022.9978851.

[5] L. Staglianò, E. Longo and A. E. C. Redondi, "D-MQTT: design and implementation of a pub/sub broker for distributed environments," 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Barcelona, Spain, 2021, pp. 1-6, doi: 10.1109/COINS51742.2021.9524110.

[6] C. Ruenpitak, A. Phonphoem, A. Jansang, W. Tangtrongpairoj and C. Jaikaeo, "Scalable Distributed Broker System for Very Large MQTT Networks," 2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE), Bangkok, Thailand, 2022, pp. 1-6, doi: 10.1109/JCSSE54890.2022.9836246.

[7] Y. Noda, K. Ishibashi and T. Yokotani, "A proposal on the control mechanism among distributed MQTT brokers over wide area networks," 2022 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS), BALI, Indonesia, 2022, pp. 70-75, doi: 10.1109/IoTaIS56727.2022.9976024.

[8] T. Sylla, R. Singh, L. Mendiboure, M. S. Berger, M. Berbineau and L. Dittmann, "SoD-MQTT: A SDN-Based Real-Time Distributed MQTT Broker," 2023 19th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Montreal, QC, Canada, 2023, pp. 92-97, doi: 10.1109/WiMob58348.2023.10187779.

[9] A. Detti, L. Funari and N. Blefari-Melazzi, "Sub-Linear Scalability of MQTT Clusters in Topic-Based Publish-Subscribe Applications," in IEEE Transactions on Network and Service Management, vol. 17, no. 3, pp. 1954-1968, Sept. 2020, doi: 10.1109/TNSM.2020.3003535.

[10] C. Bouallegue and J. Gascon-Samson, "EdgePub: A Self-Adaptable Distributed MQTT Broker Overlay for the Far-Edge," 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), Paris, France, 2022, pp. 1-8, doi: 10.1109/FMEC57183.2022.10062858.

[11] F. Hmissi and S. Ouni, "TD-MQTT: Transparent Distributed MQTT Brokers for Horizontal IoT Applications," 2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), Hammamet, Tunisia, 2022, pp. 479-486, doi: 10.1109/SETIT54465.2022.9875881.

[12] R. Banno and K. Osawa, "Acceleration of MQTT-SN protocol using P4," 2022 IEEE 11th International Conference on Cloud Networking (CloudNet), Paris, France, 2022, pp. 16-21, doi: 10.1109/CloudNet55617.2022.9978851.

[13] Yamamoto, K., Fukuhara, A. and Nishi, H. (2022), Hardware Implementation of MQTT Broker and Precise Time Synchronization Using IoT Devices. IEEJ Trans Elec Electron Eng, 17: 209-217. https://doi.org/10.1002/tee.23511

[14] A. Stanford-Clark and H. L. Truong, MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2, IBM Corporation, 2013.

[15] mtcp-stack/mtcp, GitHub, https://github.com/mtcp-stack/mtcp (accessed Aug. 8, 2023).

[16] Data Plane Development Kit, DPDK, https://www.dpdk.org/ (accessed Aug. 8, 2023).

[17] 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.

[18] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers and R. Carley, "Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, Canada, 2015, pp. 36-43, doi: 10.1109/FCCM.2015.12.

[19] A. Fukuhara, T. Iwai, Y. Sakuma and H. Nishi, "Implementation of Content-Based Anonymization Edge Router on NetFPGA," 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 2019, pp. 123-128, doi: 10.1109/MCSoC.2019.00025.

[20] A. Fukuhara, S. Shohata and H. Nishi, "FPGA Context-based Live Migration Maintaining Network Consistency," 2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW), Naha, Japan, 2020, pp. 81-86, doi: 10.1109/CANDARW51189.2020.00027.

[21] I. Kim and H. B. Min, "Operation about multiple scan chains based on system-on-chip," 2008 International SoC Design Conference, Busan, Korea (South), 2008, pp. II-191-II-194, doi: 10.1109/SOCDC.2008.4815716.

[22] R. Morishima and H. Nishi, "Network Transparent Fog-based IoT Platform for Industrial IoT," 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 2019, pp. 920-925, doi: 10.1109/INDIN41052.2019.8972178.

[23] W. A. S. P. Abeysiriwardhana, J. Wijekoon, R. L. Tennekoon, and H. Nishi, "Software-accelerated Service-oriented Router for Edge and Fog Service Enhancement Using Advanced Stream Content Analysis," IEEJ Transactions on Electronics, Information and Systems, vol. 139, no. 8, pp. 891–899, 2019.

[24] T. Miura, J. L. Wijekoon, S. Prageeth and H. Nishi, "Novel infrastructure with common API using docker for scaling the degree of platforms for smart community services," 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), 2017, pp. 474-479 .

[25] H. Hiraga and H. Nishi, "Network Transparent Decrypting of Cryptographic Stream Considering Service Provision at the Edge," 2021 IEEE 19th International Conference on Industrial Informatics (INDIN), Palma de Mallorca, Spain, 2021, pp. 1-6, doi: 10.1109/INDIN45523.2021.9557366.

[26] FPGA computing platform M-KUBOS PALTEK corporation, PALTEK corporation. https://www.paltek.co.jp/design/original/m-kubos/index.html (accessed Aug. 8, 2023).

[27] PYNQ: PYTHON PRODUCTIVITY, http://www.pynq.io/ (accessed Aug. 8, 2023).

[28] The NetFPGA Project. http://netfpga.org/ (accessed Aug. 8, 2023).

[29] NetFPGA SUME Reference NIC, https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/NetFPGA-SUME-Reference-NIC (accessed Aug. 8, 2023).

[30] alexforencich/verilog-axis: Verilog AXI stream components for FPGA implementation, https://github.com/alexforencich/verilog-axis (accessed Aug. 8, 2023).

[31] Xilinx. Integrated Logic Analyzer v6.2 Product Guide (PG172). 2016.

[32] Nishi H. Infrastructure and services for smart community. The Journal of the Institute of Electronics, Information and Communication Engineers 2015; 98(2):112–117.