

Lightweight Transfer Mechanisms and Their Performance Evaluation in IoT DEP

Haruto Toyama¹, Atsuko Yokotani², Takaharu Suzuki³, and Tetsuya Yokotani^{4,*}

¹ Graduate School of Electrical Engineering and Electronics, Kanazawa Institute of Technology, Nonoichi, Ishikawa, Japan

² Center for Electric, Optic and Energy (EOE) Applications, Kanazawa Institute of Technology, Nonoichi, Ishikawa, Japan

³ Office of Industry-University Collaboration Kanazawa Institute of Technology, Nonoichi, Ishikawa, Japan

⁴ Department of Electronics and Information Systems Engineering, College of Engineering, Kanazawa Institute of Technology, Nonoichi, Ishikawa, Japan

Email: c6601357@st.kanazawa-it.ac.jp (H.T.); yokotani@s.ttc.or.jp (A.Y.); suzuki.takaharu2357@gmail.com (T.S.); yokotani@neptune.kanazawa-it.ac.jp (T.Y.)

*Corresponding author

Manuscript received May 8, 2026; accepted August 8, 2026; published September 22, 2026

Abstract—Different Internet of Things (IoT) platforms have been considered to efficiently offer multiple IoT services. One such platform is the IoT Data Exchange Platform (IoT DEP), which is currently being standardized by ISO/IEC JTC1/SC41. A key challenge for the application of IoT DEP is the implementation of an information transfer method that minimizes processing load and ensures low-latency communication. This paper proposes a lightweight transfer mechanism applicable to IoT DEP, specifically involving the application of IoTA, which is a lightweight communication method. Further, it demonstrates a method to implement IoTA at the data link layer using raw socket communication to achieve low-latency transfer. Furthermore, performance evaluation was performed using a prototype to verify the effectiveness of the proposed method.

Keywords—Internet of Things (IoT) Platform, IoT Service, Raw Socket Communication, Low Latency Transfer, International Standardization

I. INTRODUCTION

Internet of Things (IoT) services have been introduced in various fields [1]. Recently, platforms with common functions have been considered to reduce the initial and operational costs of IoT services. Diverse architectures exist for IoT platforms, as reported previously [2–4]. This study focused on the IoT Data Exchange Platform (IoT DEP), an IoT platform centered on the information access function defined in Ref. [4]. The IoT DEP has been standardized by ISO/IEC JTC1/SC41 [5], which was promoted by the authors. The current standard documents on IoT DEP mainly specify the requirements, architecture, and functional definitions of the components. One challenge for IoT DEP application is the information transfer method between end systems. In industrial networks, authentication and confidentiality are often required to mitigate unauthorized access and data tampering. Accordingly, this study develops a low-processing-load information transfer method based on the IoT DEP architecture by applying IoTA [6], which provides lightweight delivery confirmation and integrated security functions. To clarify the performance characteristics of the proposed method, a comparative evaluation with the standardized and widely adopted QUIC is conducted. Furthermore, focusing on industrial applications, this study aimed to reduce the processing load at the Internet layer via raw socket communication to achieve low-latency communication. In addition, the validity of the method

developed herein was verified by performance evaluation using a prototype.

II. OVERVIEW OF IoT DEP

This section provides an overview of the current state of IoT DEP and identifies challenges for future specification enhancements.

A. Overview of the Current IoT DEP

The current architecture of the IoT-DEP is shown in Fig. 1. Although IoT DEP is designed to operate over the Internet, they do not involve complex processing like IP dynamic routing. For IoT services, fixed communication paths are established between nodal points that act as termination and/or connecting points, providing a virtual separation from existing Internet services. Furthermore, simplified connection processing is achieved by applying name-based communication protocols, such as MQTT, DDS, or ICN, between end point connecting devices and between servers and nodal points.

The IoT DEP has been under discussion since 2017 within ISO/IEC JTC1/SC41/WG4 and its predecessor, ISO/IEC JTC1/WG10. In 2020, its requirements, architecture, and functional definitions were published as the international standard ISO/IEC 30161 Part 1. Furthermore, in 2022, the requirements for coordination control between nodal points were published as ISO/IEC 30161 Part 2. These details are reported by Yokotani *et al.* [7, 8].

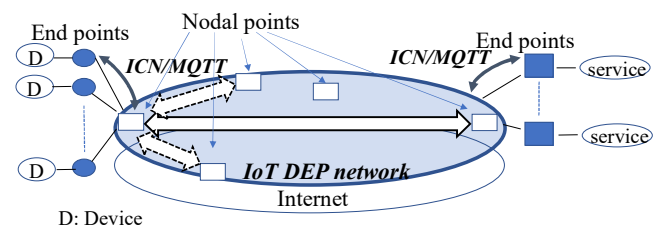


Fig. 1. Architecture of the current IoT DEP.

B. Reinforced Functions in IoT DEP

Discussions have begun on future functional enhancements for the IoT DEP aiming toward the next step of standardization, such as ISO/IEC 30161 Part 3. Specifically, considering the monitoring and control services for industrial applications, the following functional enhancements were

achieved:

- Specifications of a transfer method designed to reduce communication processing load between nodal points and ensure low latency.
- Integration with information processing platforms, such as robotics platforms
- Enhancement of security functions, including detection of unauthorized traffic

This paper focuses on Point (1) and presents solutions in the following sections.

III. COMPARISON OF TRANSFER MECHANISMS

This section describes the transfer mechanisms designed to achieve a low processing load in IoT-DEP. First, it provides an overview of QUIC, currently considered applicable to the IoT and IoTA, and was devised for embedded systems by the authors. It then details the typical operation of IoTA.

A. Typical Mechanisms

To design a transfer mechanism for IoT-DEP, the authors presented protocol selection criteria for achieving lightweight and low-latency operations. The selection criteria were as follows:

- Ability to ensure security, including authentication and encryption
- Based on standard protocol stacks under consideration by ITU-T, assuming interoperability

QUIC [9] and IoTA are considered candidates that satisfy these conditions. The standard protocol stacks are shown in Fig. 2.

QUIC	IoTA
UDP	UDP
IP	IP
Data link	Data link
Physical media	Physical media

Fig. 2. Protocol stacks on QUIC and IoTA

The QUIC operates over UDP/IP [10]. Although traditionally focused on web communications, its application to the IoT has been explored in recent years [11–13]. QUIC integrates transport functions like packet loss detection and congestion control within the protocol itself to achieve communication stabilization and latency reduction. Furthermore, it provides secure communication by performing certificate-based authentication, data encryption, and tamper detection as integral parts of the connection establishment procedure [14]. In addition, its design minimizes the number of round trips required to establish a connection. While TCP requires a three-way handshake, QUIC typically establishes a connection in one round-trip time (RTT) and can transition to data transfer, thereby reducing the establishment delay.

Specifically, the client sends ClientHello as the QUIC initial packet at connection initiation, and the server responds

with ServerHello in QUIC Initial. The server then sends information, such as certificates needed for cryptographic parameter establishment, and the handshake is completed when the client returns a finished packet. This completes connection establishment in a single round trip, allowing for subsequent data transmission and delivery confirmation via ACKs.

Similarly, IoTA is a protocol that typically operates over UDP/IP. IoTA is a communication method that simplifies the information transfer function while addressing the security requirements of the IoT environment. Details of this process are provided in the following subsections.

B. Summary of IoTA

This subsection details the functional blocks of IoTA, a lightweight transmission method, along with the sequence from connection to data transfer. The overall architecture of IoTA is shown in Fig. 3. The IoTA incorporates control functions like flow control and retransmission control during data transfer, enabling reliability assurance and adjustment of delay characteristics via congestion mitigation. The IoTA also handles security functions like encryption and decryption. Generally, security mechanisms like TLS pose challenges for IoT devices owing to the implementation complexity and certificate exchange overhead [15]. Given this, IoTA was designed for low-resource environments.

The subsequent datagram represents the actual transfer units that are transmitted and received. As mentioned in Section A, IoTA provides an interface compatible with UDP/IP and can operate on existing network stacks.

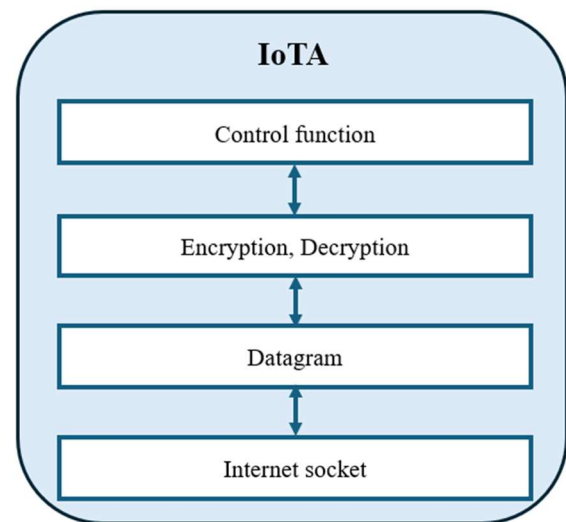


Fig. 3. IoTA functional blocks.

Fig. 4 shows a sequence diagram for IoTA. Similar to QUIC, IoTA completes connection establishment in one round-trip time (RTT) and then proceeds to data transmission and ACK reception. As shown in (i) in Fig. 4, the session establishment process and the client's public key were sent in the same datagram. After verifying the client's data, the server sends its public key to complete the establishment of the session. Subsequently, encrypted data transfer occurs, as shown in (ii) in Fig. 4, and confirmation is performed via ACK reception, as shown in (iii) in Fig. 4.

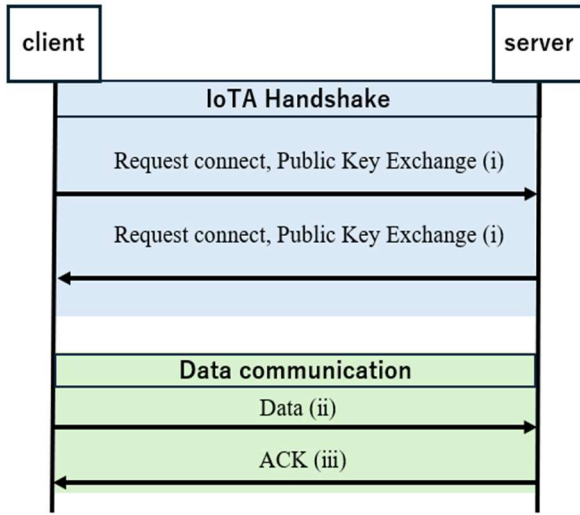


Fig. 4. Communication sequences in IoTA.

IV. RESULT AND DISCUSSION

This section proposes the use of raw socket communication as a strategy to achieve low latency in the IoT-DEP. Specifically, it outlines the implementation methods and other aspects based on the use of raw socket communication for IoTA.

A. Characteristics of Raw Socket Communication

Traditional UDP/IP-based communication is relatively lightweight and offers broad compatibility [16]. However, because the packet passes through the kernel network stack, processes like generation and parsing of UDP/IP headers, routing based on destination ports, and calculation and verification of UDP checksums occur for each packet. While these processes are effective for ensuring data integrity and detecting misdeliveries, they can potentially increase the processing time and cause latency variations in IoT environments.

Raw socket communication differs from standard UDP/IP communication paths in that it enables data transmission without passing through a kernel's UDP/IP stack layer. This reduces the overhead associated with the header processing, checksum calculation/verification, and port-based demultiplexing. Consequently, it suppresses the software-induced overhead during transmission, offering lower latency and reduced latency variations.

B. Implementation for Raw Socket Communication

This subsection describes the components and key points for implementing an application that sends and receives Ethernet frames directly at the data link layer. In standard socket communication, specifying the destination IP address prompts the kernel to generate an IP/UDP header and resolve the destination MAC address via the address resolution protocol (ARP), and ultimately transmit the data as an Ethernet frame. However, raw socket communication does not rely on the IP or UDP layer-processing paths. Therefore, an application must explicitly prepare the header information necessary to form a valid Ethernet frame. Fig. 5 shows a schematic of the Ethernet header.

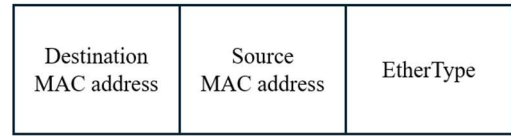


Fig. 5. Ethernet frame format.

First, a socket is created to access the data link layer. Specifying `SOCK_RAW` for `PF_PACKET` enables frame-level transmission and reception, including an Ethernet header. As part of constructing the Ethernet header, Fig. 6 illustrates the transmit header configuration. Here, the source and destination MAC addresses are written to the header, and the `EtherType` field is set to `ETH_P_ALL`. This formats the transmit buffer into a frame consisting of 'Ethernet header + payload,' making it ready for transmission to the data link layer.

```

eth_header = (struct ethhdr *)sock->send_buffer;
memcpy(eth_header->h_source,
        ifr.ifr_hwaddr.sa_data,
        sizeof(eth_header->h_source));
memcpy(eth_header->h_dest,
        sock->to_address.sll_addr,
        sizeof(eth_header->h_dest));
eth_header->h_proto = htons(ETH_P_ALL);
  
```

Fig. 6. Ethernet header configuration.

In addition, Fig. 7 shows the MAC address settings required for the Ethernet frames. The 'arg_address' in the figure is an input that provides the destination MAC address as a string, which is converted to a byte sequence and stored as destination information. The 'arg_interface' sets the interface name used for transmission and reception. Based on this information, the socket is bound to determine its communication destination. A raw socket assumes execution under administrative privileges.

```

for (i = 0; i < strlen(arg_address); i += 3)
{
    unsigned int byte;
    sscanf(arg_address + i, "%2x", &byte);
    dst_mac[i / 3] = (unsigned char)byte;
}

address.sll_family = AF_PACKET;
address.sll_protocol = htons(ETH_P_ALL);
address.sll_ifindex = if_nametoindex(arg_interface);
memcpy(address.sll_addr, dst_mac, sizeof(dst_mac));
address.sll_halen = htons(sizeof(dst_mac));
  
```

Fig. 7. MAC address configuration.

C. Characteristics of Raw Socket Communication

A schematic of the implementation of Raw socket communication process in IoTA is shown in Fig. 8.

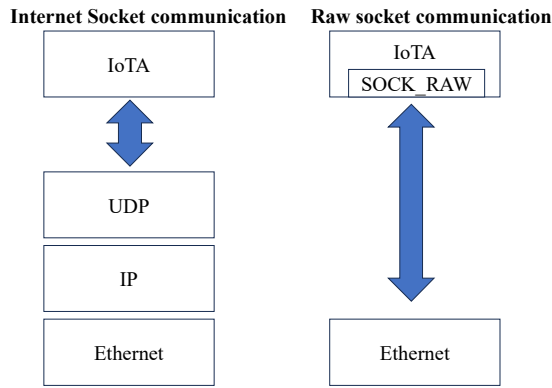


Fig. 8. Implementing raw socket communication in IoTA.

By specifying `SOCK_RAW` when creating a socket and executing it under administrator privileges, transmission and reception via raw sockets become feasible. When raw socket communication is used, link layer information, such as the destination MAC address, is created and configured on the application side.

V. PERFORMANCE EVALUATION

In this section, the authors describe the performance evaluation for each protocol based on four metrics: footprint, overhead required for connection establishment, reception efficiency within a specified time under bandwidth constraints, and latency. The QUIC implementation used in this evaluation was `quic-go v0.57.1`. Except for settings specified as experimental conditions for each evaluation item, such as disabling Path MTU Discovery, the default `quic-go` configuration was used.

A. Footprint

This subsection describes the evaluation of Flash usage as an indicator of footprint, representing the implementation scale. Flash usage is defined as the sum of the executable code and the constants contained in the text section of the executable binary, along with the static areas containing initial values in the data section. The main text focuses on and discusses Flash usage, which directly impacts the device deployment. Furthermore, the IoT device class specified in IETF RFC 7228 was used as a benchmark for the implementation scale of low-resource IoT devices.

Table 1. Flash memory consumption	
Category	Flash (B)
QUIC	6,169,332
IoTA	102,677

Table 1 presents the footprints of QUIC and IoTA. IoTA integrates security features within the protocol, resulting in relatively compact implementation. In contrast, QUIC consumes significant Flash memory, confirming that its implementation imposes a high load in low-resource environments like IoT.

B. Connection Establishment Overhead

The connection establishment overhead was evaluated for all three communication protocols, as shown in Fig. 9.

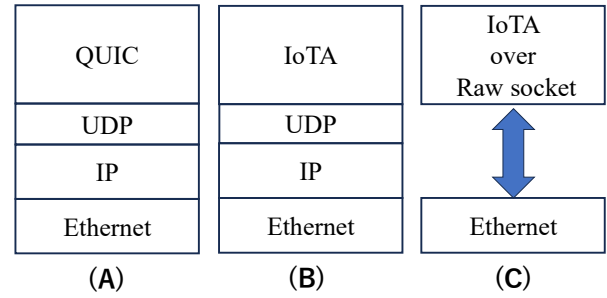


Fig. 9. Evaluation cases.

In addition, IETF RFC 9000 defines PADDING frames for QUIC, which are used to adjust packet lengths as needed [10]. To clearly compare the connection overheads, the increments caused by PADDING were excluded from the evaluation in this study. The results obtained under these conditions are presented in Fig. 10.

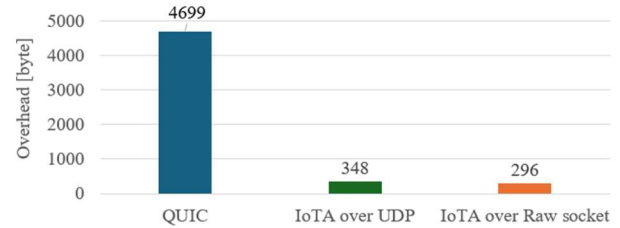


Fig. 10. Initial connection overhead.

The results show that IoTA over raw sockets has the lowest connection overhead. In contrast, QUIC incorporates security features, such as certificate-based authentication, into its connection establishment process, requiring additional control information during the connection setup, which is likely to increase its connection overhead.

C. Bandwidth Restriction

The experimental configuration for evaluation of reception efficiency under bandwidth constraints is illustrated in Fig. 11.

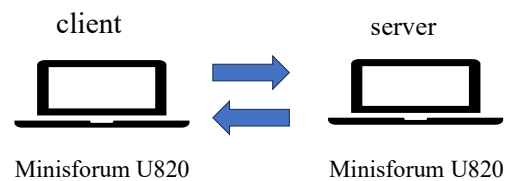


Fig. 11. Experimental configuration.

Two MinisForum U820 terminals running Ubuntu 22.04.5 were directly connected via a 1-Gbps wired link in a local environment. The evaluation method involves the client sending 1024 bytes of data every 0.1 s for 10 s after the connection establishment. The server-side application measures the received data volume as an indicator of reception efficiency. The transmission bandwidth was progressively limited from 256 to 16 kbit/s. Linux traffic control was employed to limit the bandwidth by controlling the transmission rate using a Token Bucket Filter (TBF). In the QUIC evaluation, Path MTU Discovery was disabled to avoid the influence of automatic packet size adjustment. The results are presented in Fig. 12.

The QUIC reception efficiency tended to deteriorate as the transmission rate decreased. In particular, at 16 kbit/s, the

server-side application received 4096 bytes with QUIC, whereas it received 19456 bytes with IoTA over raw sockets. This is believed to occur because, in low-bandwidth environments, the connection-establishment overhead becomes relatively dominant, reducing the time and bandwidth available for data transfer. In contrast, IoTA over raw sockets, with its smaller overhead, can transition to data transfer quickly after connection establishment, making it less susceptible to congestion.

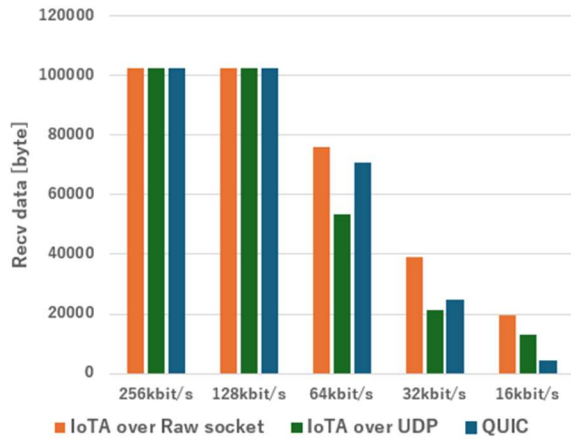


Fig. 12. Application-level throughput.

D. Latency

For the evaluation of latency during data transfer, the experimental setup based on the configuration in Fig. 11 is used. The latency measurement points are shown in Fig. 13.

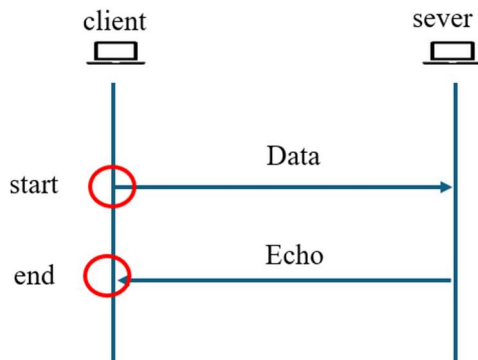


Fig. 13. Measurement points for latency in communication sequence.

During the measurement, data with size ranging from 128 to 8192 bytes were transmitted. The delay time, measured in the RTT from the start to the end indicated in Fig. 13 (from data transmission to the return of the ECHO response), was calculated using timestamps inserted during both the transmission and reception processes. For each data size, 500 trials were performed, and the mean and variance were calculated. As in the bandwidth restriction evaluation, Path MTU Discovery was disabled in the QUIC evaluation for the latency measurement. The results are presented in Figs. 14 and 15.

QUIC exhibited relatively small changes in latency even when the data size was increased from 128 to 8192 bytes, confirming its low dependence of latency on data size. However, its variance was the largest among the three methods, resulting in greater variability in the measured values. IoTA over UDP exhibits lower latency than QUIC for

data sizes up to 1024 bytes, but the latency increases for data sizes of 2048 bytes and above, reaching the highest for 8192 bytes. The IoTA over raw sockets demonstrated the lowest latency across all data sizes from 128 to 8192 bytes and the smallest variance. Therefore, under the conditions of this experiment, raw socket communication method is effective in achieving low latency and stable communication for relatively small data sizes typical of IoT.

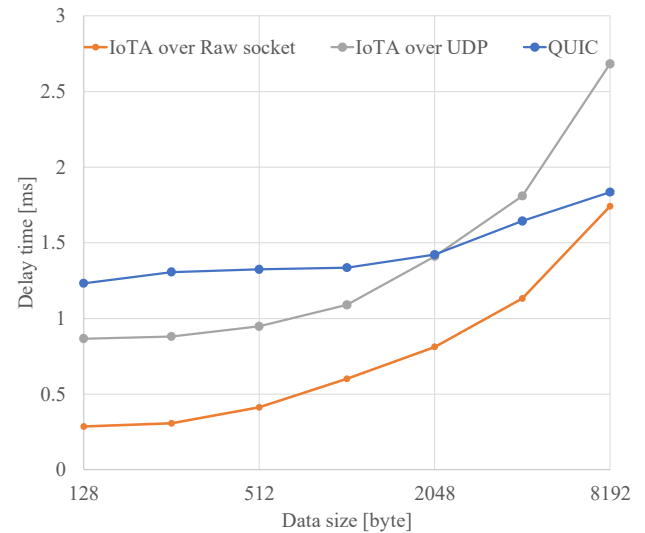


Fig. 14. Average delay.

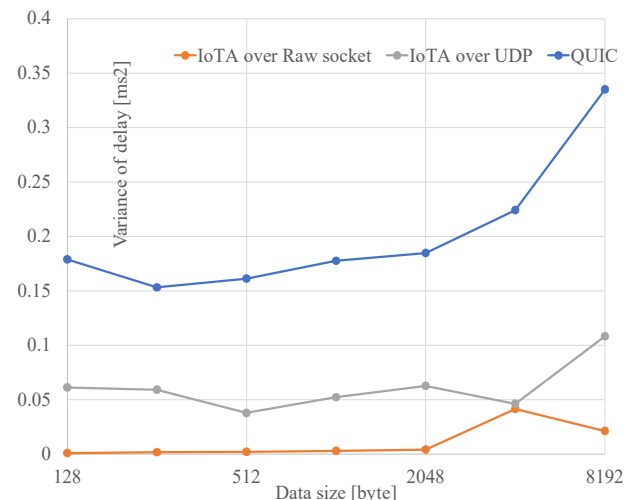


Fig. 15. Sample variance.

VI. CONCLUSIONS

In this study, authors developed a specific method and evaluated its performance by prototyping for low processing loads and low-latency transfer, representing a future research direction for IoT DEP. Specifically, the validity of IoTA over raw sockets was demonstrated. Future research should focus on implementing the method using low-cost CPUs and verifying specific services. In addition, this method should contribute to the standardization of next-generation IoT DEP.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Haruto Toyama conceived and initiated this study and drafted the manuscript. Atsuko Yokotani reviewed the

manuscript and contributed to improving the quality of the study. Takaharu Suzuki assisted with the experimental validation of this work. Tetsuya Yokotani, as the supervising professor, provided overall guidance throughout the study. All authors approved the final version of the manuscript.

REFERENCES

- [1] T. Yokotani, "IoT use cases analysis and possibility of adopting ICN technologies for these IoT use cases," in *Proc. IEEE World Symposium on Communication Engineering (WSCE) 2018*, pp. 7–12, 2018.
- [2] L. Babun *et al.*, "A survey on IoT platforms: Communication, security, and privacy perspectives," *Journal of Computer Network*, vol. 192, 108040, 2021.
- [3] M. Fahmideh and D. Zowghi, "An exploration of IoT platform development," *Information Systems*, vol. 87, 101409, 2020.
- [4] A. Yokotani, H. Mineno, and T. Yokotani, "A proposal on "Sandwich" architecture in the communication platform for IoT services," in *Proc. 15th International Conference on Future Computer and Communication (ICFCC 2023) collocated in the 13th International Workshop on Computer Science and Engineering (WCSE 2023)*, 2023, C12001.
- [5] ISO/IEC JTC1. Business Plan for JTC 1/SC 41 Internet of Things and Digital Twin. [Online]. Available: <https://www.iso.org/committee/6483279.html>
- [6] A. Yoshioka, M. Murakami, and T. Suzuki, "Study of resource-saving secure communication protocols for IoT devices," presented at Work-03: IoT Platform: Technology, Use Cases, and Standardization, 2022 IEEE World Forum on Internet of Things (WF-IoT), Yokohama, Japan, 2022.
- [7] T. Yokotani and K. Kawai, "Concepts and requirements of IoT networks using IoT Data Exchange Platform toward International standards," *IEEE Conference on Standards for Communications and Networking (IEEE CSCN 2019)*, 2019.
- [8] T. Yokotani *et al.*, "Survey and comparison of Interworking point routing mechanisms for IoT services in wide area ICNs," in *Proc. 2020 International Conference on Emerging Technologies for Communications (ICETC2020)*, 2020.
- [9] Y. Cui *et al.*, "Innovating Transport with QUIC: Design Approaches and Research Challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.
- [10] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. [Online]. Available: <https://datatracker.ietf.org/doc/rfc9000/>
- [11] D. Saif and A. Matrawy, "An experimental investigation of tuning QUIC-based publish–subscribe architectures in IoT," *IEEE Internet of Things Journal*, vol. 11, no. 3, pp. 4924–4933, 2024.
- [12] F. Fernández *et al.*, "And QUIC meets IoT: Performance assessment of MQTT over QUIC," in *Proc. 2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2020, pp. 1–6.
- [13] J. Dizdarević and A. Jukan, "Experimental benchmarking of HTTP/QUIC protocol in IoT cloud/edge continuum," in *Proc. IEEE International Conference on Communications (ICC)*, 2021, pp. 1–6.
- [14] M. Thomson and S. Turner. RFC 9001: Using TLS to Secure QUIC. [Online]. Available: <https://www.rfc-editor.org/info/rfc9001/>
- [15] R. T. Tiburski *et al.*, "Evaluating the use of TLS and DTLS protocols in IoT middleware systems applied to E-health," in *Proc. 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2017.
- [16] J. Postel. User Datagram Protocol. [Online]. Available: <https://www.rfc-editor.org/info/rfc768/>

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).