Implementation and Evaluation of CYPHONIC client focusing on Sequencing mechanisms and Concurrency for packet processing

Ren Goto
Aichi Institute of Technology,
Toyota, Aichi 470-0392,Japan
r0719en@pluslab.org

Kazushige Matama
Aichi Institute of Technology,
Toyota, Aichi 470-0392,Japan
matama@pluslab.org

Ryouta Aihata
Aichi Institute of Technology,
Toyota, Aichi 470-0392,Japan
sonarait@pluslab.org

Shota Horisaki
Meijo University,
Nagoya, Aichi 468-0073,Japan
shota.horisaki@ucl.meijo-u.ac.jp

Hidekazu Suzuki
Meijo University,
Nagoya, Aichi 468-0073,Japan
hsuzuki@meijo-u.ac.jp

Katsuhiro Naito
Aichi Institute of Technology,
Toyota, Aichi 470-0392,Japan
naito@pluslab.org

***Abstract*—Peer-to-Peer (P2P) based direct communication among devices can enhance communication speed and reduce the burden on cloud services in service models involving distributed and cooperative processing. The authors have proposed and developed CYber PHysical Overlay Network over Internet Communication (CYPHONIC), which offers a secure communication mechanism based on a P2P model. However, the performance of conventional CYPHONIC clients is limited because they process packets in a single-threaded manner, considering the simplicity of sequential packet processing. This paper presents a redesigned implementation of the CYPHONIC client that supports multi-threaded packet processing. The new client design incorporates a novel processing scheme that emphasizes sequencing mechanisms and concurrency in packet processing, as the order of packet processing is not guaranteed in multi-threaded processing. These improvements have significantly increased end-node throughput and enabled communication delay times to remain constant even as the number of connections increases.**

***Index Terms*—Internet of Things, Peer-to-Peer communication, Overlay network protocol, CYPHONIC**

I. Introduction

The rapid proliferation of mobile nodes, including Internet of Things (IoT) devices and smartphones, and the widespread adoption of various Internet services, have resulted in a continuous increase in network traffic [1]. IoT device processing capabilities have been improving year by year. New value and services are expected to be created by leveraging distributed computing and integrating with cloud services [2]. In services involving mutual communication and cooperation, direct communication between devices provides the shortest path, improving communication speed and reducing latency. Therefore, Peer-to-Peer (P2P)-based communication is ideal for such service models.

However, most practical services use a Client-to-Server (C2S) model where communication goes through a central server. In the C2S model, traffic passes through a central server, leading to the issue of route redundancy and higher latency. Additionally, managing the increased load and ensuring scalability due to device proliferation becomes complex in the operation of cloud services, potentially resulting in increased personnel costs [3], [4]. Nevertheless, many services still use the C2S model because of the Internet Protocol (IP) used in the Internet.

The current exhaustion of IPv4 addresses, the most widely used ones, has resulted in the introduction of Network Address Port Translation (NAPT) in each network, where multiple nodes are assigned private IP addresses and share a single global IP address. However, this mechanism hides the devices behind it from the global network. As a result, nodes on the global network cannot initiate communication with nodes behind NAPT, making direct communication between devices extremely difficult. This widely-known issue of connectivity has led to the proposal of various NAPT traversal techniques to achieve interconnection between applications beyond NAPT [5].

Another solution is the global spread to IPv6, which expanded the address space to 128 bits. Unfortunately, the incompatibility between conventional IPv4 and IPv6 has created interoperability issues [6]. In addition to these issues, various threats exist on the Internet, making it essential to realize secure communication by introducing encryption and access control technologies [7]. While individual solutions have been provided for these challenges, discussions on solutions that can address all of them simultaneously are still insufficient.

Against this background, the authors have proposed and developed CYber PHysical Overlay Network over Internet Communication (CYPHONIC), which provides a comprehensive solution and secure communication mechanism based on the P2P model [8]. CYPHONIC establishes optimal and secure communication paths between nodes before communication, providing end-to-end encrypted communication over the overlay network. The users can efficiently utilize our overlay network by installing the client program of CYPHONIC on their end devices.

The conventional prototyping of the client program was implemented with basic packet processing functionality to validate communication over our overlay network. However, a single-threaded approach resulted in degraded communication performance and inadequate performance due to the impact of the processing load of one module on other modules operating independently. Additionally, since an end device is expected

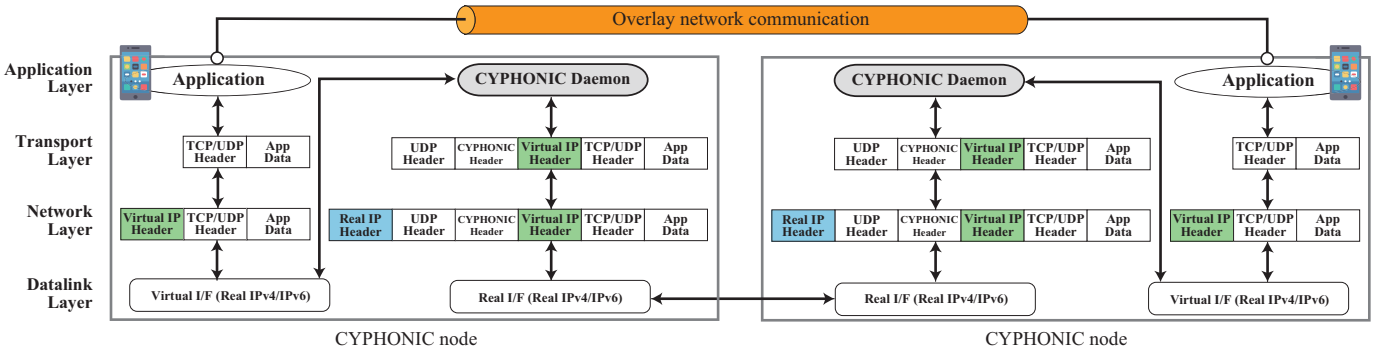

Fig. 1. Packet encapsulation and decapsulation flow

to communicate with multiple other devices simultaneously, faster and more efficient processing is necessary to provide stress-free services.

In this paper, we redesign the client program and implement a packet ordering mechanism and concurrency of functional modules to achieve fast packet processing. The proposed system eliminates dependencies by storing state information in caches and enables concurrent packet processing by asynchronously executing modules. We also introduce an ordering mechanism to align the order of incoming and outgoing packets, taking into account the different processing speeds of each thread.

In this verification experiment, we assessed the performance of standard protocols such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP) based on the number of connections with peer nodes. Additionally, we measured the resource usage and trends of the enhanced client program. The evaluation results show significant enhancements in processing performance, ensuring stable communication delay time even with an increased number of connections.

II. CYPHONIC

CYPHONIC extends the ability to establish direct communication in communication environments without relying on cloud service relays by constructing a virtual IP-based overlay network. We have added a proprietary overlay network layer to the TCP/IP protocol stack. CYPHONIC allows peer nodes to communicate via tunneling based on virtual IP addresses, achieving NAPT traversal and IPv4 to IPv6 interconnection. All devices are authenticated in advance using root certificates, and all communication traffic is encrypted. This mechanism provides comprehensive communication connectivity and zero-trust security.

CYPHONIC consists of CYPHONIC nodes, which are devices equipped with the CYPHONIC client program, and three types of cloud services: Authentication Service (AS), Node Management Service (NMS), and Tunnel Relay Service (TRS). AS handles the authentication processes to verify the legitimacy of CYPHONIC nodes and manages virtual IP addresses and Fully Qualified Domain Names (FQDNs) to identify the nodes on the overlay network uniquely. NMS manages the network information of both nodes, selects appropriate routes, instructs CYPHONIC nodes to establish tunnel communication, and manages the signaling process. TRS provides communication packet relaying services between devices when they are connected to different IP version networks or when they are behind Symmetric NAPT routers.

The CYPHONIC node is equipped with the CYPHONIC daemon, the client program for CYPHONIC. The CYPHONIC daemon operates as a background process, handling virtual interfaces and packet processing for application communication. By querying the NMS for the FQDN of the desired node, the CYPHONIC node obtains the peer's virtual IP address and communication path. Then, an end-to-end tunnel is established with the peer node, and all traffic is encrypted by directly exchanging a shared encryption key.

Fig. 1 provides an overview of packet encapsulation and communication in the overlay network. The CYPHONIC daemon intercepts application data through the virtual interface and encapsulates it into a virtual IP packet with the original CYPHONIC header. The packet is then encrypted using the shared encryption key. The encapsulated packet is sent to the peer node after adding a UDP header to the virtual IP packet and encapsulating it with the CYPHONIC node's real IP address. In reverse communication, the real IP and UDP headers are decapsulated, and the virtual IP packet is decrypted. The peer node extracts the data from the virtual IP packet and transfers it to the application layer. As a result, CYPHONIC establishes overlay network communication based on virtual IP.

III. Proposed System

In this paper, we propose a new processing scheme for the CYPHONIC daemon by revising its system model and incorporating concurrent execution of each module and a packet ordering mechanism. The conventional CYPHONIC daemon, executed as a single thread, encountered the problem of degraded communication performance due to the processing load of one module affecting other processes. Additionally, some processes depended on the state of other processes, making it challenging to achieve concurrency. It is assumed that a single device communicates with multiple devices si-

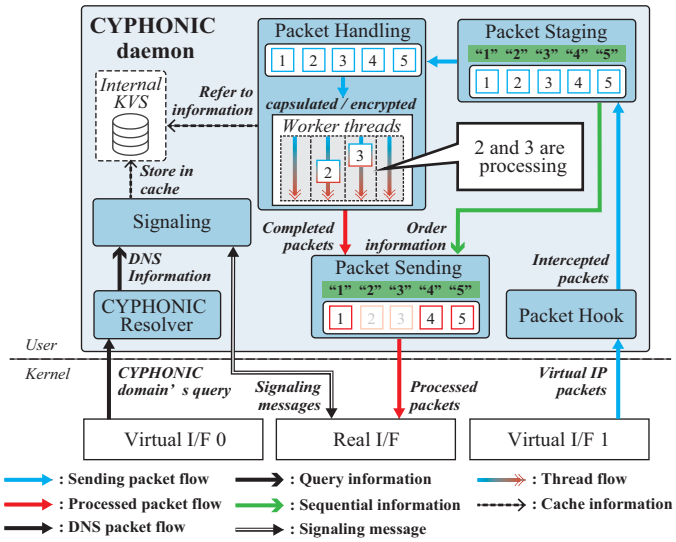

Fig. 2. System model of redesigned CYPHONIC daemon

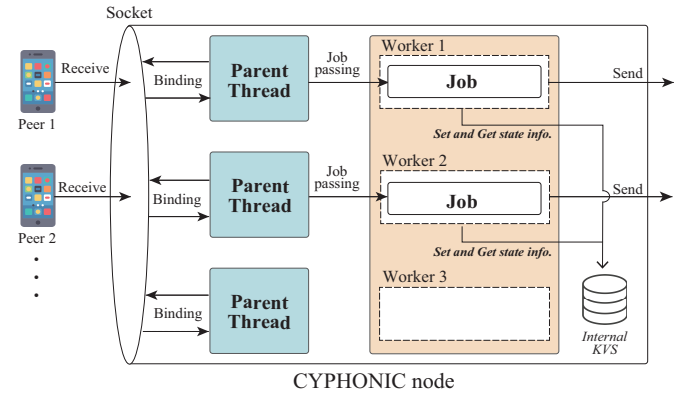

Fig. 3. Transactions and Threads allocation per peer connections

multaneously. Therefore, each module must be multi-threaded to speed up packet processing.

Fig. 2 illustrates the system model of the CYPHONIC daemon implemented in the proposed scheme. The CYPHONIC daemon provides four main functions: signaling processing with cloud services, Domain Name System (DNS) packet processing and overlay network establishment, encapsulation/decapsulation and encryption/decryption of virtual IP packets, and sending/receiving data on the actual network. These functions are implemented as the Signaling Module, CYPHONIC Resolver Module, Packet Handling Module, and Sending/Receiving Module.

A. Threads handle and Transactions

In the proposed system, we address the need for multi-threading by introducing communication states within the communication data and implementing a caching mechanism. This allows each module to run independently in its own thread, enabling faster internal processing.

Fig. 3 provides an overview of the transactional process for sending and receiving encapsulated messages in a multi-threaded CYPHONIC node. When communicating with a peer node, the parent thread that receives the encapsulated packet assigns jobs to worker threads for each transaction, separating the main thread that receives packets from the worker threads that handle actual processing.

After sending a packet, the worker thread is released, allowing another worker thread to handle response packets. The proposed system introduces a Key-Value Store (KVS)-based in-memory cache to temporarily store state information to address the possibility of different worker threads processing response packets. This cache is referenced and updated by all processing modules. By utilizing the in-memory cache, state information from the previous worker thread becomes accessible to the new worker thread, enabling continuous processing of transactions and facilitating multi-threading. Additionally, state information stored in the in-memory cache is promptly discarded if it remains unreferenced for a certain period due to communication disruption with the peer node.

The proposed system adopts an event-driven architecture with pre-generated worker threads to minimize the overhead of releasing and recreating multiple threads. Upon starting the CYPHONIC daemon, worker threads for encryption/decryption are generated based on the logical threads of the CPU. Furthermore, separated threads for sending, receiving, and KeepAlive are created for each peer connection. Worker threads continue to operate until the CYPHONIC daemon is stopped. However, threads for individual peer connections are promptly discarded if communication with the peer node is interrupted for a certain period.

B. Concurrency and Sequential processing schemes

Once the CYPHONIC node completes establishing the overlay network, it engages in data communication processing to exchange capsule messages with its peer nodes. The CYPHONIC daemon utilizes the Packet Hook Module to retrieve application data from the virtual interface. Intercepted virtual IP packets are sequentially stored in the receive buffer of the Packet Staging Module for processing. The encryption/decryption process in the Packet Handling Module is particularly resource-intensive. Consequently, worker threads are generated and allocated specifically for this purpose.

The packets are then sequentially forwarded to each worker thread for encryption, with the addition of CYPHONIC and UDP headers. Due to the OS or runtime state dependency, the processing order may differ from the order of received packets. To address this issue, the Packet Sending Module refers to the reception order stored in the Packet Staging Module and reorders the processed packets according to their arrival sequence. Finally, the packets are sent sequentially through the actual interface, ensuring concurrent processing while maintaining the order of incoming and outgoing packets.

IV. Performance Evaluation

The proposed system extends its functionality by leveraging the Go language, which has been conventionally used for

TABLE I
SPECIFICATIONS OF THE MEASURING DEVICES

Virtual Machine (CYPHONIC cloud: AS, NMS, TRS)	
OS	22.04 (Jammy Jellyfish)
CPU	Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz 2 cores, 2 threads
Memory	2GB RAM

Virtual Machine (CYPHONIC node)	
OS	22.04 (Jammy Jellyfish)
CPU	Intel(R) Core(TM) i9-13900 CPU @ 5.60GHz 2 cores, 2 threads
Memory	1GB RAM

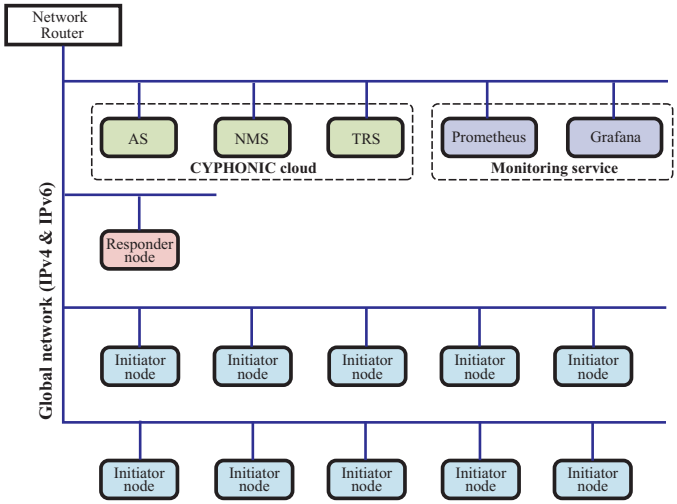

Fig. 4. Evaluation model

implementing the CYPHONIC daemon. To achieve multi-threading, lightweight threads called Goroutines generate worker threads.

We measured communication throughput and latency in the verification experiment using standard protocols such as TCP, UDP, and ICMP. Additionally, we evaluated machine resource utilization and Application Performance Management (APM) metrics. The former involved capturing signals of machine resources from the OS, such as CPU and memory usage. The latter focused on collecting metrics from the CYPHONIC daemon to comprehend its operational status on the application side.

Table I presents the specifications of the validation device, where virtual machines with 1GB of memory are used for CYPHONIC nodes. Fig. 4 illustrates the evaluation model where all instances are deployed within a closed network. One instance functions as a responder node, awaiting communication, while the other ten instances act as initiator nodes to initiate communication. Each node can establish tunnel communication with up to 10 peers.

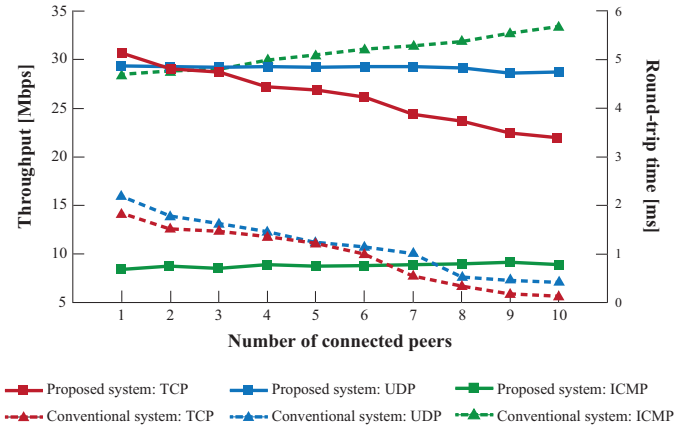

Fig. 5. Comparison of Throughput and Latency

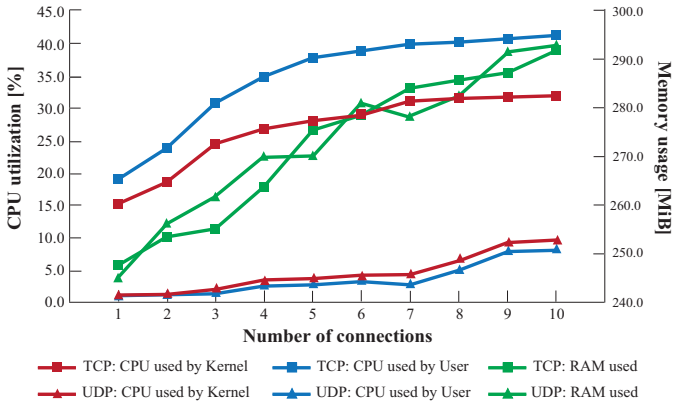

Fig. 6. CPU and Memory utilization per peer connections

A. Communication throughput and latency

The CYPHONIC node conducts measurements of the communication speed for capsule message processing, facilitating data exchange and internal processing time. We utilized network measurement tools such as iperf3 to measure throughput, and for round-trip time, we employed ping. In this validation process, we conducted measurements three times, each lasting 15 minutes, for different numbers of tunnel connections with peer nodes and repeated this process three times.

Fig. 5 compares communication throughput and latency between the conventional and proposed systems. The performance evaluation confirmed that adopting a multi-threaded processing approach significantly enhances the performance and throughput of CYPHONIC nodes. Moreover, while the conventional scheme tends to experience an increase in round-trip time with an increase in connections, the proposed method can maintain a constant round-trip time.

B. Machine resources utilization and APM metrics

The CYPHONIC daemon measures the utilization of machine resources during capsule message processing. We introduce NodeExporter to CYPHONIC nodes for metric output and use Prometheus for data collection to achieve observabil-

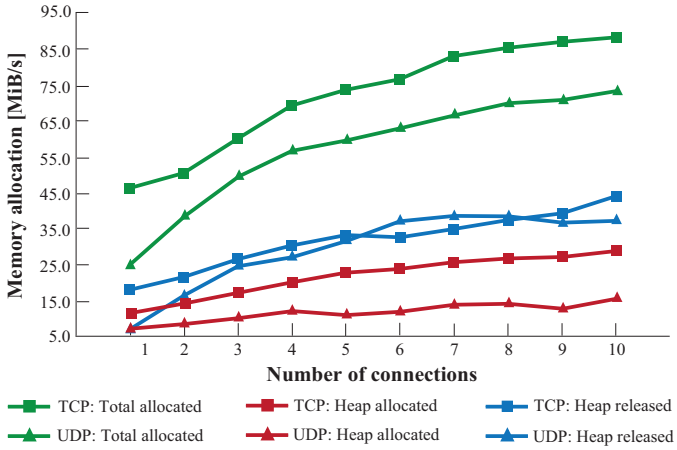

Fig. 7. Memory allocation per peer connections

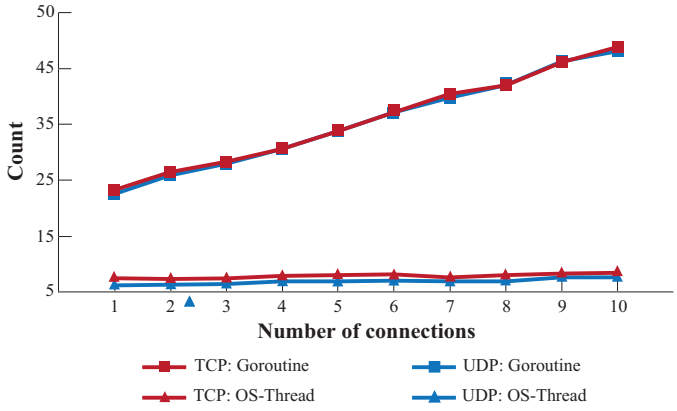

Fig. 8. Comparison of Goroutines and OS-Threads

ity. GoMetrics, provided by Grafana Labs, is utilized for APM metrics collection [9].

Fig. 6 illustrates the CPU utilization and memory usage of CYPHONIC nodes while processing TCP and UDP traffic. Overall, machine memory usage remains nearly equal for both protocols. However, TCP tends to exhibit higher CPU utilization than UDP due to the additional overhead associated with implementing reliability features.

We perform memory monitoring focused solely on the CYPHONIC daemon to mitigate observer effects. Fig. 7 displays the memory allocation trends in the CYPHONIC daemon. The proposed system efficiently manages state information using an internal cache, gradually increasing memory allocation per connection. Conversely, the heap area, responsible for dynamic data, is appropriately released upon connection termination, resulting in more memory being released than allocated.

Additionally, we analyze the increasing trend in threads due to parallel processing. Fig. 8 compares the trends of Goroutines and machine threads during the measurement. The CYPHONIC daemon generates processing routines per peer, increasing thread count with the number of connections. However, the number of threads observed from the OS remains constant. Notably, the Go language can theoretically execute hundreds of thousands of goroutines simultaneously, and by decoupling from the OS scheduler, the overhead of context-switch during thread switching is reduced [10], [11].

The evaluation demonstrates that the proposed multi-threading scheme enables advanced concurrency and efficient resource utilization. Given the recent improvements in device processing performance, further enhancements in communication throughput can be anticipated.

V. Conclusions

In this paper, we have re-engineered the client program to incorporate support for multithreading. The proposed approach achieves concurrent execution of each function by leveraging an event-driven architecture. Furthermore, we have introduced an ordering mechanism to synchronize the order of received and sent packets, considering variations in processing speeds among threads. During the evaluation phase, we validated that the communication performance of CYPHONIC nodes improved significantly without a significant impact on processing performance. The concurrent execution capability in the proposed system enables rapid and efficient processing, presenting substantial potential as a practical client service.

Acknowledgment

This work is supported in part by Grant-in-Aid for Scientific Research (C)(21K11877), the Japan Society for the Promotion of Science (JSPS), and the Cooperative Research Project of the Research Institute of Electrical Communication, Tohoku University.

References

[1] Cisco Systems, Inc., "Cisco Annual Internet Report (2018–2023)," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf, March 2020.

[2] M. Beevi, "A fair survey on Internet of Things (IoT)," 10.1109/ICETETS.2016.7603005, February 2016.

[3] S. Duangsuwan, A. Chusongsang, C. Teekanakvisit, and S. Promwong, "BER Detection of A2G Wireless Communication in Rician K-Factor Fading Channel for Massive IoT Connectivity Network," 10.1109/ISPACS48206.2019.8986353, December 2019.

[4] Z. Guo, K. Zhang, H. Xin, M. Bi, H. He, and W. Hu, "An optical access network framework for smart factory in the industry 4.0 era supporting massive machine connections," 10.1109/ICOCN.2017.8121481, August 2017.

[5] A. Keränen, C. Holmberg, and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," 10.17487/RFC8445, July 2018.

[6] P. Wu, Y. Cui, J. Wu, J. Liu, and C. Metz, "Transition from IPv4 to IPv6: A state-of-the-art survey," 10.1109/SURV.2012.110112.00200, January 2013.

[7] Y. Qigui, W. Qi, Z. Xiaojian, and F. Jiaxuan, "Dynamic Access Control and Authorization System based on Zero-trust architecture," 10.1145/3437802.3437824, October 2020.

[8] T. Yoshikawa, H. Komura, C. Nishiwaki, R. Goto, K. Matama, and K. Naito, "Evaluation of new CYPHONIC: Overlay network protocol based on Go language," 10.1109/ICCE53296.2022.9730323, January 2022.

[9] aaukhatov/grafana-dashboards, https://github.com/aaukhatov/grafana-dashboards, May 2023.

[10] T. Andersson and C. Brenden, "Parallelism in Go and Java: A Comparison of Performance Using Matrix Multiplication," nbn:se:bth-16548, June 2018.

[11] Alan A. A. Donovan and Brian W. Kernighan, "The Go Programming Language," isbn:978-0134190440, October 2015.