# CFcoQUIC: CPU/FPGA Co-design Accelerated QUIC for Low-Power IoT Communication

Xin Dong[1,2], Lizhuang Tan[1,2], Huiling Shi[1,2], Wei Zhang[1,2], Peiying Zhang[1,3]
[1]*Key Laboratory of Computing Power Network and Information Security, Ministry of Education*
*Shandong Computer Science Center (National Supercomputer Center in Jinan)*
*Qilu University of Technology (Shandong Academy of Sciences), Ji'nan, China*
[2]*Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing*
*Shandong Fundamental Research Center for Computer Science, Jinan, China*
[3]*Qingdao Institute of Software, College of Computer Science and Technology*
*China University of Petroleum (East China), Qingdao, China*
Email: 10431240209@stu.qlu.edu.cn, lzhtan@qlu.edu.cn, shihl@sdas.org, wzhang@sdas.org, zhangpeiying@upc.edu.cn

***Abstract*—In IoT environments, devices are often constrained by low power consumption and limited resources, making efficient connection establishment with minimal overhead crucial for real-time communication between edge devices and cloud servers. The QUIC protocol shows significant potential, but the encryption and decryption overhead is considerable. Reducing this overhead and improving connection establishment efficiency are critical to enhancing QUIC performance, especially for IoT edge devices that need to handle high-concurrency communication while maintaining low power consumption. This paper proposes CFcoQUIC, a CPU/FPGA co-design architecture that accelerates the handshake process and reduces the initial connection latency by parallelizing multiple encryption/decryption flows in high-concurrency environments. Experimental results demonstrate that the time for RSA encryption and decryption on FPGA is at least 19.1 times faster than on CPU, and the time for AES encryption and decryption is at least 5 times faster on FPGA. These results highlight the effectiveness of the proposed architecture in reducing QUIC handshake overhead in IoT environments with low-power edge devices.**

***Index Terms*—Internet of Things, Quick UDP Internet Connection, Encryption, Decryption, Field Programmable Gate Array**

## I. Introduction

In Internet of Things (IOT) environments, where devices are often constrained by low power consumption and limited resources, efficient communication is crucial for real-time interactions between edge devices and cloud servers [1]. However, IoT edge devices often need to handle a large number of simultaneous connections, making the reduction of connection establishment overhead a critical issue.

Quick UDP Internet Connections(QUIC) is a modern transport layer protocol based on UDP [2]. By integrating transport and security layer functions, it significantly reduces the connection establishment delay and solves the TCP head-of-line blocking problem [3]. It is particularly suitable for mobile network environments with high latency or high packet loss rates. Its mandatory use of TLS 1.3 encryption further enhances security, but also introduces significant CPU overhead. In addition, QUIC frequent data migration between user mode and kernel mode causes a surge in context switching overhead, further exacerbating the problem of resource contention. The QUIC protocol, while suitable for improving network performance, presents a challenge for low-power edge devices that need to maintain high throughput with minimal energy consumption.

Although the traditional TCP Offload Engine (TOE) [4] effectively reduces TCP processing overhead through hardware acceleration, its fixed-function architecture lacks the flexibility required to accommodate the tightly integrated transport and encryption mechanisms of the QUIC protocol [5].Therefore, we propose CFcoQUIC, a CPU/FPGA Co-design architecture, which offloads encryption and decryption parts of QUIC connection process to FPGA. Compared with traditional optimization solutions, it provides efficient, flexible, and cost-effective solutions that is particularly well-suited for modern network applications, especially in IoT environments, where both performance and power efficiency are critical.

This paper is organized as follows: Section 1 quantitatively analyzes the performance bottlenecks of the QUIC protocol and reviews existing research on hardware-accelerated network protocols, Section 2 details the design and implementation of the CFcoQUIC architecture, Section 3 compares the performance differences of the RSA/AES [6] [7] algorithms on CPU and FPGA, Section 4 verifies the feasibility of the architecture through experiments.

## II. Background and Related Work

This section investigates and analyzes the development of QUIC and the related research on hardware acceleration of QUIC, which provides a reference for the following research on the connection process of QUIC.

### A. Motivation

As illustrated in Fig.1, the encryption key exchange and TLS handshake within the QUIC protocol stack contribute to approximately 20%-30% of the CPU overhead, emerging as a primary performance bottleneck. This issue arises due

This work was supported in part by the Shandong Provincial Natural Science Foundation under Grant No. ZR2022LZH015.

to the design characteristics of QUIC [8]. Firstly, its integration of TLS 1.3 facilitates 0-RTT fast connection establishment and end-to-end encryption, significantly increasing the computational intensity of key derivation as well as data packet encryption and decryption [9]. Secondly, QUIC support for multiplexing and connection migration introduces additional state machine complexity, further intensifying the CPU scheduling burden [10]. Although traditional TCP Offload Engine (TOE) can mitigate TCP protocol processing overhead through hardware acceleration, their architectural design is not well-suited to accommodate QUIC customized transport layer logic and its deep integration with encryption mechanisms.

In IoT environments, where low-power devices and high-concurrency communication are essential, managing computationally intensive tasks such as encryption and connection management becomes increasingly challenging [11]. These edge devices often require real-time, secure communication with poor CPU performance, as the number of devices grows. The QUIC protocol, while offering significant benefits such as faster connection establishment and reduced latency, poses significant challenges in IoT applications due to its heavy reliance on TLS encryption and frequent transitions between user-mode and kernel-mode, resulting in higher CPU overhead.

To address these challenges, FPGA (Field-Programmable Gate Array) technology, with its reconfigurable hardware capabilities, has emerged as an effective tool for fine-grained acceleration of computational hotspots within the QUIC protocol stack [12]. In the data plane, it processes multi-stream encryption tasks in parallel using a customized pipeline design [13]. In the control plane, it employs on-chip BRAM to construct a QUIC connection state cache table, facilitating hardware-level multiplexing and connection migration state synchronization. This reduces CPU interrupt frequency and allows encrypted data frames to be directly injected into the network interface via the PCIe Gen4/5 interface and DMA engine, thereby mitigating the overhead associated with user-mode to kernel-mode context switching [14].

The FPGA-based partial offloading approach accelerates the encryption and decryption tasks of the QUIC protocol through hardware while preserving its core advantages, such as low latency and efficient multiplexing. This strategy effectively alleviates CPU resource overload. Furthermore, CFcoQUIC not only inherits the hardware acceleration principles of TOE but also transcends the protocol limitations of traditional TCP offloading through a reconfigurable architecture, providing a scalable implementation pathway for the next generation of high-performance QUIC network devices.

### *B. Hardware Offload*

Hardware-accelerated QUIC can be broadly categorized into two types. One approach involves directly deploying the QUIC protocol onto intelligent network interface cards, thereby offloading either the entire or a portion of the QUIC protocol stack to the hardware. This is analogous to the concept of TCP Offload Engine (TOE) [15], where certain functions or even the entire protocol stack are transferred to the network interface card for implementation. And the CPU only executes the part related to data processing. Yang et al. [16] proposed a QUIC acceleration scheme for hardware-software co-design on SmartNIC based on general purpose FPGA to provide background and reference. Given the limited resources of existing NICs, offloading the whole or part of the protocol is not universal.

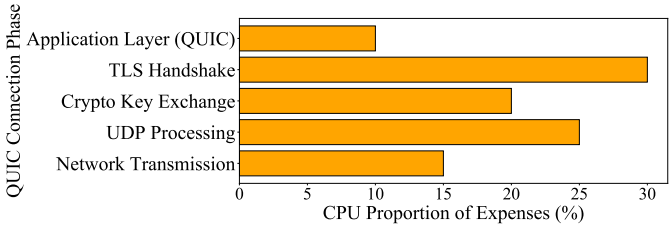



Fig. 1: Proportion of Expenses

The alternative approach involves offloading general functions of the QUIC protocol to FPGA. For instance, tasks such as the encryption and decryption processes in QUIC can be offloaded to FPGA, significantly reducing CPU usage. However, this approach presents certain challenges. As a relatively new protocol, QUIC frequently requires updates and debugging, which typically occurs in user mode. During the transmission and reception of messages, frequent data migrations between user and kernel modes occur, triggering frequent interrupts and consuming CPU resources [17]. This implies that real-time data exchange between the CPU and the intelligent network card is necessary, and such frequent exchanges can lead to performance degradation.

To mitigate the impact of data exchange, Nick et al. [18] proposed NanoPU, a novel network-optimized Data Processing Unit (DPU) prototype that achieves a line-to-line delay of only 65ns, a 13-fold improvement over the state-of-the-art. Additionally, Toke et al. [19] introduced a new framework called XDP for general-purpose programmable packet processing, which enables the offloading of critical functional components of the protocol stack while allowing the conventional network stack to handle the remainder. Based on this, we conclude that partial offloading to smart network cards requires higher-performance or more advanced solutions to counteract the performance losses caused by data transfer.

## III. Solution

The QUIC connection process demands significant CPU resources for data processing, which can be addressed through software acceleration, hardware offloading, or software-hardware collaboration [20]. This section introduces CFcoQUIC, the CPU/FPGA Co-design architecture, providing essential background for the subsequent experimental framework aimed at optimizing QUIC performance.

### *A. CPU/FPGA Co-design Architecture*

In the QUIC protocol connection process, three primary modes of operation are typically observed. The first mode is the existing architecture used in current PCs or servers,

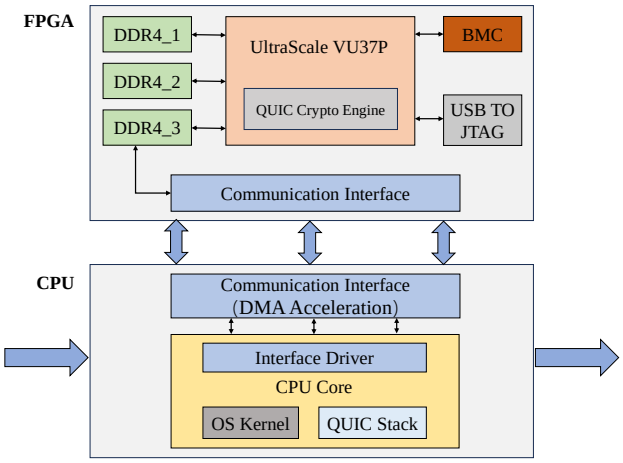



Fig. 2: CPU/FPGA Co-design Architecture

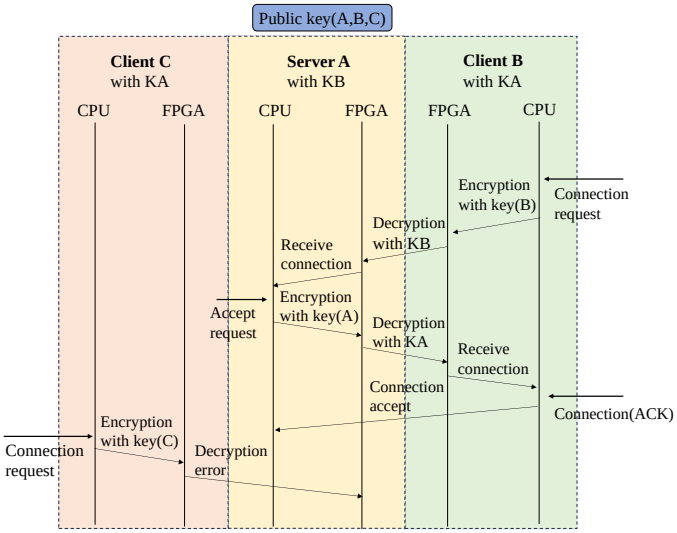



Fig. 3: Offloading Process

consisting of only a CPU and a network card. In this configuration, the CPU handles all QUIC processing tasks, while the network card is responsible for sending and receiving data. This approach significantly increases CPU utilization, which can lead to resource contention, preventing the CPU from handling other business-critical tasks, and increasing connection latency.

To address these limitations, the second and third modes are introduced. In the second mode, an FPGA programmable accelerator card is added to the first architecture, allowing for partial offloading of the QUIC protocol's functions. For instance, the connection establishment process in QUIC can be executed directly on the FPGA, which offers faster execution speed and helps mitigate the communication overhead. This approach improves processing speed and efficiency compared to the first mode and reduces CPU load, although it may introduce communication overhead due to data migration between kernel mode and user mode. The third mode, known as the intelligent network card mode, integrates the FPGA and network card into a unified system to efficiently offload data processing and communication tasks. This paper proposes a CPU/FPGA Co-design architecture based on the third mode, as illustrated in Fig. 2. The proposed architecture leverages a multi-channel PCIe interface and a DMA acceleration mechanism to facilitate low-latency, high-throughput data exchange between the CPU and FPGA. Additionally, it employs a load-aware strategy to enable elastic allocation of computing resources. To enhance encryption performance, a configurable QUIC encryption acceleration engine is integrated into the hardware, supporting the pipeline execution of cryptographic algorithms such as AES-256-GCM and RSA-512. Furthermore, a multi-level storage subsystem is incorporated to optimize data access efficiency, thereby improving overall system performance.

**Algorithm 1** AES Encryption Process

**Input:** 128-bit plaintext $P$, 128-bit key $K$, Number of rounds $Nr$
**Output:** 128-bit ciphertext $C$
**Key Expansion:** Generate round keys $K_0, K_1, ..., K_{Nr}$
**AES Encryption:**
$S \leftarrow P$
AddRoundKey$(S, K_0)$
**for** $i = 1$ to $Nr - 1$ **do**
    $S \leftarrow$ SubBytes$(S)$
    $S \leftarrow$ ShiftRows$(S)$
    $S \leftarrow$ MixColumns$(S)$
    $S \leftarrow$ AddRoundKey$(S, K_i)$
**end for**
**Final Round:**
$S \leftarrow$ SubBytes$(S)$
$S \leftarrow$ ShiftRows$(S)$
$S \leftarrow$ AddRoundKey$(S, K_{Nr})$
**return** Ciphertext $C = S$

### *B. Offloading Process*

In the QUIC protocol, cryptographic operations impose significant computational overhead, particularly in high-concurrency environments where multiple encryption/decryption tasks must be executed in real time. The FPGA-based acceleration engine in CFcoQUIC offloads the most computationally intensive components of QUIC. AES [21] based on Algorithm 1 , Used for data packet encryption and decryption. The FPGA implements a pipelined AES-GCM engine that parallelizes multiple encryption tasks, reducing processing latency. RSA [22] based on Algorithm 2 , Used for TLS 1.3 handshake authentication. The FPGA performs modular exponentiation in hardware to speed up signature verification. This section provides the detailed steps of these

cryptographic algorithms and their offloading process.

The QUIC connection establishment involves a client-server handshake with offloaded cryptographic operations, as illustrated in Fig.3. Upon receiving an encrypted connection request from the client, the FPGA acceleration card decrypts the payload using pre-provisioned keys and forwards the plaintext to the CPU via PCIe for TLS 1.3 handshake completion. The CPU validates the handshake parameters, updates the session state, and returns the response to the FPGA for session key-based encryption. The FPGA then transmits the encrypted ACK to the client, finalizing the connection. This partitioning leverages FPGA parallelism for cryptography while reserving protocol logic for the CPU, achieving a balance between latency and flexibility.

**Algorithm 2** RSA Key Generation

**Input:** Two large prime numbers $p, q$
**Output:** Public key $(n, e)$ and private key $(n, d)$

**Key Generation:**
$n \leftarrow p \times q$
$\phi(n) \leftarrow (p-1) \times (q-1)$
**for** $e = 2$ to $\phi(n) - 1$ **do**
  **if** $\gcd(e, \phi(n)) = 1$ **then**
    Select $e$
    **break**
  **end if**
**end for**
Find $d$ such that $d \times e \equiv 1 \pmod{\phi(n)}$
**return** $(n, e, d)$

## IV. Experiment Results

By comparing the performance of different encryption algorithm executed by FPGA and CPU respectively, we illustrate and analyze the feasibility of CFcoQUIC, a CPU/FPGA Co-design architecture, to selectively offload some functions of QUIC protocol to FPGA for processing.

### A. Setup

We use the F37X FPGA accelerator card equipped with a Xilinx Virtex VU37P chip. This card supports PCIe Gen 3.0 x16, providing a bandwidth of up to 128 Gbps for interconnection with the server. The memory system consists of three 72-bit, 8GB DDR4 onboard memory modules. During testing, a debugging interface was established to connect the FPGA accelerator card to a computer. The encryption and decryption algorithm was deployed onto the FPGA accelerator using Xilinx Vivado for performance evaluation.

The testing environment is illustrated in Fig.4. For benchmarking the encryption and decryption speeds of the CPU, the host system was configured with Intel(R) Xeon(R) E5-2690 v2 CPUs (3.00GHz, 10 physical cores), 256G RAM, and Ubuntu 22.04 operating system. The same encryption and decryption algorithms are used to evaluate the performance of CPU.

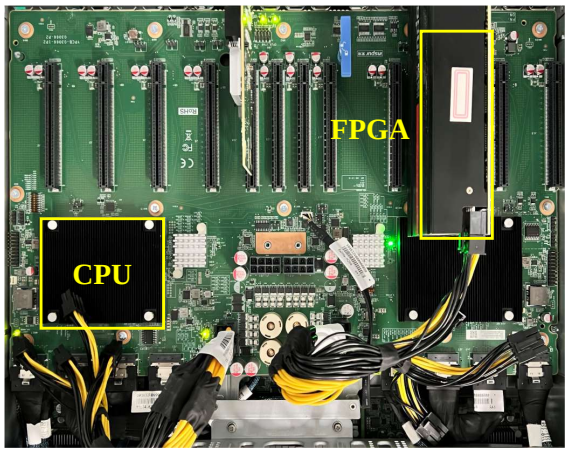
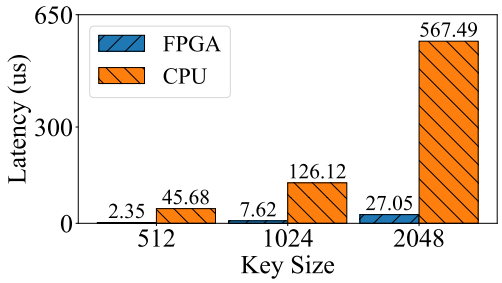
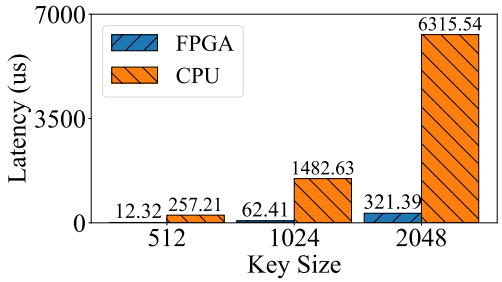



Fig. 4: Prototype System




(a) Encryption

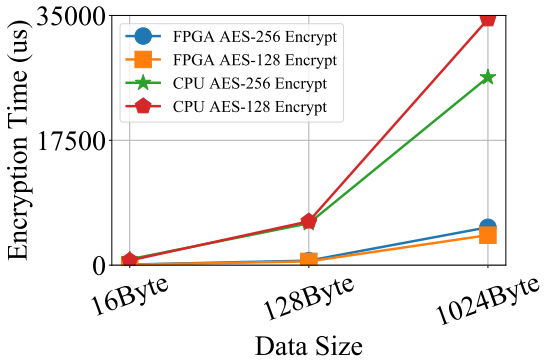



(b) Decryption

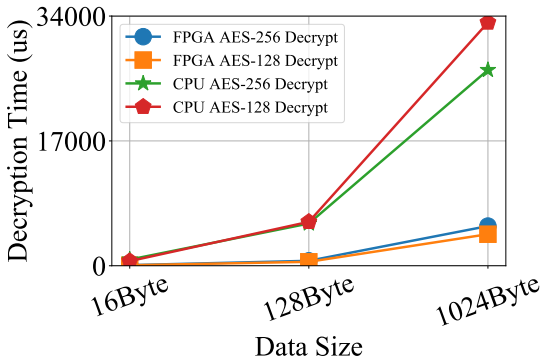
Fig. 5: RSA Computational Performance

### B. Results & Analysis

From the performance comparison depicted in Fig.5, it is evident that the FPGA outperforms the CPU, achieving at least a 19.1x improvement in the time taken to perform one RSA encryption and decryption cycle. Furthermore, as illustrated in Fig.6, the AES encryption and decryption delays for both the CPU and FPGA exhibit an approximately linear increase with the growth in data volume. However, due to its custom pipeline design and parallel computing units, the FPGA is capable of reducing the encryption and decryption delay by about 5X to 10X compared to the CPU. This performance boost is largely attributed to the utilization of FPGA embedded




(a) Encryption




(b) Decryption

Fig. 6: AES Computational Performance

bit multipliers and 18k-bit block RAM. Moreover, as the key length increases, the performance gap between the FPGA and CPU becomes more pronounced, further highlighting the superior efficiency of FPGA for handling the computational demands of encryption and decryption. This demonstrates that FPGA offers a clear performance advantage over CPU in executing encryption and decryption tasks.

## V. Conclusion

The QUIC protocol offers advantages over TCP, including reduced latency and integrated security, making it particularly suitable for high-latency and high-packet-loss environments such as mobile networks and IoT applications. However, it faces performance challenges in resource-constrained environments due to mandatory encryption and the high overhead of transferring data between kernel and user space. To address this,we propose CFcoQUIC, a CPU/FPGA Co-design architecture that offloads computationally intensive tasks to the FPGA, thereby reducing CPU load and improving QUIC performance in low-overhead environments, especially in IoT edge devices where low power and high-concurrency communication are critical. Experimental results demonstrate that the FPGA achieves least 5-19× acceleration for cryptographic operations compared to CPU-based implementations.However, this study highlights two critical directions for future research. While the current architecture primarily focuses on cryptographic acceleration, extending its capabilities to support QUIC-specific transport-layer functions (e.g., congestion control, packet reordering) remains a challenge. Implementing asymmetric cryptography (e.g., ECDHE key exchange) on FPGA necessitates more careful resource allocation to balance logic utilization and throughput. This requires further investigation into dynamic partial reconfiguration techniques to optimize efficiency. These advancements would enable CFcoQUIC to fully exploit the protocol capabilities while ensuring compatibility with evolving QUIC standards and expanding its potential in IoT environments.

## References


[1] Heble S, Kumar A, et al. A low power IoT network for smart agriculture. WF-IoT'18. IEEE, 2018: 609-614.

[2] Iyengar J, Thomson M. RFC 9000: QUIC: A UDP-based multiplexed and secure transport. Internet Emgomeeromg Task Force, 2021.

[3] Zhang X, Jin S, He Y, et al. QUIC is not Quick Enough over Fast Internet. WWW'24. 2024: 2713-2722.

[4] Dowling B, Fischlin M, Günther F, et al. A cryptographic analysis of the TLS 1.3 handshake protocol. Journal of Cryptology, 2021, 34(4): 37.

[5] Yang X, Eggert L, Ott J, et al. Making QUIC Quicker with NIC Offload. Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC. 2020: 21-27.

[6] Abualigah L, Abd Elaziz M, Sumari P, et al. Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer. Expert Systems with Applications, 2022, 191: 116158.

[7] Lychev R, Jero S, Boldyreva A, et al. How secure and quick is QUIC? Provable security and performance analyses. S&P'15. IEEE, 2015: 214-231.

[8] Kuhn N, Michel F, Thomas L, et al. QUIC: Opportunities and threats in SATCOM. International Journal of Satellite Communications and Networking, 2022, 40(6): 379-391.

[9] Langley A, Riddoch A, Wilk A, et al. The quic transport protocol: Design and internet-scale deployment. SIGCOMM'17. 2017: 183-196.

[10] Kumar P, Dezfouli B. Implementation and analysis of QUIC for MQTT. Computer Networks, 2019, 150: 28-45.

[11] Bembe M, Abu-Mahfouz A, Masonta M, et al. A survey on low-power wide area networks for IoT applications. Telecommunication Systems, 2019, 71: 249-274.

[12] Vipin K, Fahmy S A. FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications. ACM CSUR, 2018, 51(4): 1-39.

[13] Sasongko A, Kumara I M N, Wicaksana A, et al. Hardware context switch-based cryptographic accelerator for handling multiple streams. ACM Transactions on Reconfigurable Technology and Systems, 2021, 14(3): 1-25.

[14] Zazo J F, Lopez-Buedo S, Audzevich Y, et al. A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances.ReConFig'15. IEEE, 2015: 1-6.

[15] Chang E H, Wang C C, Liu C T, et al. Virtualization technology for TCP/IP offload engine. IEEE Transactions on Cloud computing, 2014, 2(2): 117-129.

[16] Wang J, Lv G, Liu Z, et al. QUIC cryption offloading based on NanoBPF. APNOMS'22. IEEE, 2022: 1-4.

[17] Puliafito C, Conforti L, Virdis A, et al. Server-side QUIC connection migration to support microservice deployment at the edge. Pervasive and mobile computing, 2022, 83: 101580.

[18] Ibanez S, Mallery A, Arslan S, et al. The nanopu: A nanosecond network stack for datacenters. OSDI'21. 2021: 239-256.

[19] Vieira M A M, Castanho M S, Pacífico R D G, et al. Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications. ACM Computing Surveys, 2020, 53(1): 1-36.

[20] Jaeger B, Zirngibl J, Kempf M, et al. QUIC on the highway: evaluating performance on high-rate links. IFIP Networking'23. IEEE, 2023: 1-9.

[21] Li Z Q, Cai B B, Sun H W, et al. Novel quantum circuit implementation of advanced encryption standard with low costs. Science China Physics, Mechanics & Astronomy, 2022, 65(9): 290311.

[22] Goshwe N Y. Data Encryption and Decryption using RSA Algorithm in A Network Environment. International Journal of Computer Science and Network Security, 2013, 13(7): 9-13.