# Poster: Accelerate and Secure Serverless Networks with QUIC

Kaiyu Hou†, Sen Lin†, Yan Chen†, Vinod Yegneswaran‡
† Northwestern University, ‡ SRI International, USA

## CCS CONCEPTS

• **Networks** → **Cloud computing**; *Transport protocols.*

## 1 INTRODUCTION

In serverless computing [3], cloud providers manage responsibility for all server-related tasks, including both hardware resource allocation and software runtime preparation. Cloud tenants are thus free to simply focus on designing discrete stateless functions and orchestrate them together for their high-level business logic.

Agile auto-scaling is among the major allures of serverless computing. Cloud providers can quickly launch new function instances in response to end-user requests, while saving operational costs. Since auto-scaled instances can be quickly destroyed by cloud providers, tenants only pay for the actual function execution time and do not need to reserve resources for burst requests. Because of both efficiency and economic advantages, serverless computing garners extensive attention from industry and is expected to become the dominant cloud computing paradigm [3] with a market share that is projected to surpass $21 Billion by 2025.

Fully encrypting all internal connections is now the best practice for major cloud providers. Although initiating reliable transportation and encryption introduces extra delays, it is not the dominant performance bottleneck in prior cloud computing: (*i*) transmission delay within the data center is negligible compared to execution times; (*ii*) connection setup latency of TCP and TLS can be simply mitigated by using persistent connections.

Nevertheless, many leading commercial serverless providers still use unencrypted TCP connections between internal serverless functions, sacrificing security for performance. This is due to new challenges bred in the serverless networking scheme. *(i)* A function instance can be initialized in milliseconds [1] and only processes a small sliver of the computational task. The latency introduced by TCP and TLS handshakes, even in the sub-millisecond-scale, can no longer be ignored. *(ii)* With the scale-zero-to-infinity feature, function instances are quickly scaled up and down by cloud providers. It is thus tough to maintain persistent connections between ephemeral functions. *(iii)* As serverless functions are commonly chained together to form task-specific workflows, cumulative handshakes aggravate the end-to-end latency.

We present a novel solution based on the emerging QUIC protocol, called QFaaS. It can simultaneously improve performance and provide security to existing serverless platforms without the requirements of any tenant code modification. QUIC [2] is a new transport protocol running over TLS 1.3 and UDP that has gained popularity in the wide-area Internet. It combines the advantages of both TLS 1.3 and UDP to provide a secure and reliable transport layer under 0-RTT connection setup cost, *i.e.*, the first data packet can be sent before any handshake round-trips occur. Due to the inherent advantages of reduced handshake costs while providing a secure network, it is appealing to consider the extension of this new protocol to solve this new challenge in serverless computing.

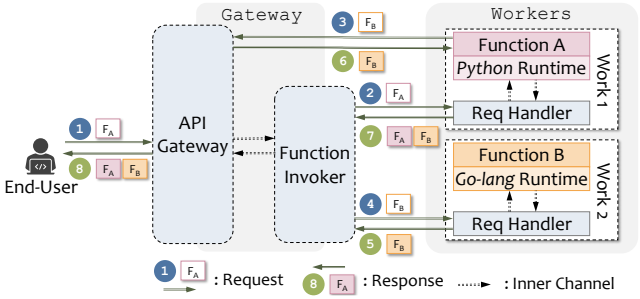


**Figure 1: The network-centric model of serverless computing.**

## 2 NETWORKING MODEL OF SERVERLESS

We first provide a new abstraction of the serverless architecture through the network-centric view (Figure 1). It will guide our QFaaS design. Serverless architecture is divided into two parts:

- Components in the gateway subsystem expose static function interfaces to end-users, manage running workers, and dispatch requests to corresponding functions. These services are all *stateful* and run on *permanent* machines. In existing serverless platforms, corresponding modules may have variant names. For example, in AWS, they are called frontend and worker manager [1]. Regardless of the names, they provide the same functionalities.
- Workers are *ephemeral* containers that comprise the request handler, the function runtime, and tenant functions. The request handler provides the internal communication ability for workers. It receives trigger requests from the gateway and sends function results back to the gateway. The function runtime provides isolated software stacks and programming language libraries to execute tenant functions. Therefore, tenant functions are decoupled from the management of ingress network connections.

Figure 1 shows an example where an end-user requests function $F_A$, while $F_A$ chained together with $F_B$ provides the service:

- (❶|❽) the end-user sends $F_A$ a request (❶) and receives responses of $F_A$ and $F_B$ (❽) from the connection with the API gateway. In the process, the API gateway acts as a transport layer *server*. Message flow details behind it are transparent to the end-user.
- (❷|❼) The API gateway forwards the $F_A$ trigger event to the function invoker. After the $F_A$ worker container is initialized, the function invoker sets a connection to the request handler in $F_A$ worker, sends request data (❷), and receives responses (❼)


*CoNEXT '21, December 7–10, 2021, Virtual Event, Germany*

ACM ISBN 978-1-4503-9098-9/21/12…$15.00
https://doi.org/10.1145/3485983.3493350

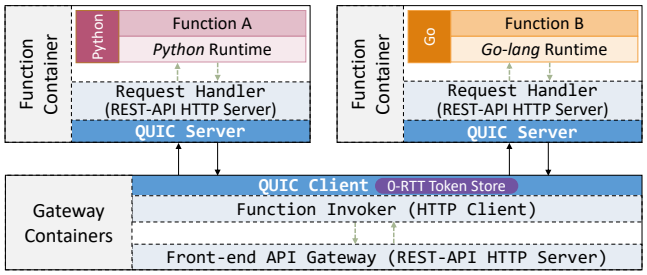


**Figure 2: The system design of QFaaS.** Modifications on the serverless platform are totally transparent to the tenants' applications.

by this connection. In this process, the function invoker plays the role of the transport layer *client* to initiate the connection. The request handler plays the role of the transport layer *server*.

- (❸|❻) Following the business logic, $F_A$ needs the response of $F_B$. Nevertheless, instead of sending a request to a worker of $F_B$, $F_A$ will send the request (❸) and receive responses of $F_B$ (❻) from the API gateway. $F_A$ does not need to care about any scheduling details of $F_B$. In this process, the API gateway acts as a transport layer *server* again, even though the connection is internal.
- (❹|❺) The function invoker initializes a worker for $F_B$, sets up a connection to its request handler, sends request data (❹), and receives responses (❺) from this connection.

# 3 SYSTEM DESIGN AND EXPERIMENTS

## 3.1 QFaaS System Architecture

In our network-centric model, the communication channels from the API gateway to the function invoker and request handlers to language runtime could be persistent connections or internal IPCs. The connection between end-users and the gateway (❶|❽) is initialized by end-users and could also be persistent connections. The connection from the function to the gateway (❸|❻) is initialized by functions. Therefore, the connections from the gateway to workers (❷|❼ and ❹|❺), which are fully controlled by the provider, really matter in the serverless network latency. First, we cannot simply use persistent connections to mitigate the connection setup latency. Second, this overhead will be multiplied when functions are chained together or the number of running instances is quickly scaled up.

Figure 2 shows the system design of QFaaS. We integrate the QUIC client in the gateway and the QUIC server in the worker request handlers (to replace the TCP and TLS clients and servers, respectively). All function requests that go through the gateway to workers would now benefit from the efficiency and security of QUIC. In serverless computing, all workers, as well as the code of the gateway and request handlers, are provided and controlled by cloud providers. Therefore, this modification is transparent to cloud tenants and does not request any change to tenants' function code.

The QUIC protocol 0-RTT mode uses the *QUIC connection token* on the client side to resume fast connections. Since the gateway, which is a stateful machine, plays the role of the QUIC client, it can maintain and manage the QUIC connection token cache all the time. Therefore, serverless applications under this design can further benefit from the 0-RTT feature of QUIC.

**Preliminary Experiments.** We implemented the QFaaS prototype on the OpenFaaS platform. The network delay within a typical data center is around 0.5ms; the AWS intra-regional delay varies from 1ms to 3ms. In both scenarios, our experiments (Figure 3) show that QUIC can reduce the single function end-user response latency by up to 28% compared with OpenFaaS using TLS and TCP. It even performs better than OpenFaaS using only insecure TCP.

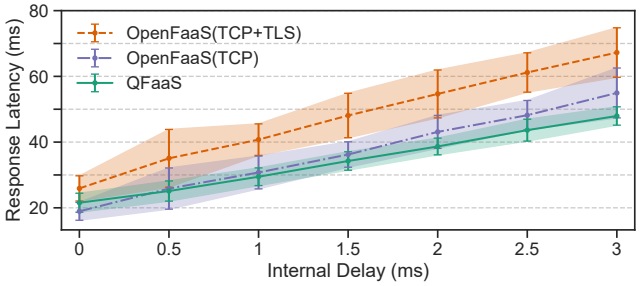


**Figure 3: Latency under variant intra-cloud delays.** QFaaS reduces 28% single function latency. It even performs better than using only TCP.

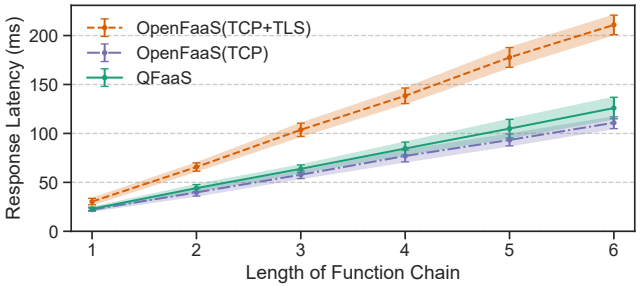


**Figure 4: Benefits of QFaaS with the function chain library.** QFaaS is 40% faster than OpenFaaS (TCP+TLS) when the chain length is 6.

## 3.2 Function Chain Library

We cannot simply replace the connection initiated from the function to the gateway (❸|❻) with QUIC. Because this connection is function code related and is also programming-language specific. For example, AWS and OpenFaaS suggest Python developers form function chains by using the Python `Requests` library.

We provide the QUIC-based function chain library to enable QUIC at ❸|❻ with slight tenant code modification. This chain library has QUIC as its underline transport layer protocol. We also integrate the QUIC server into the gateway. Thus, all function chain traffic invoked by the library will benefit from QUIC. The code modification to adapt to it is minimal. For instance, the Python developers only need to import the library and switch their `Requests` call to the QFaaS chain library call, which are only 2 lines of code modification. This design has a side benefit. End-users now can also initiate requests by QUIC and further accelerate the ❶|❽ connection.

**Preliminary Experiments.** As shown in Figure 4, when the intra-cloud delay is 0.5ms, the end-user response latency difference between QFaaS and OpenFaaS (TCP+TLS) increases as the chain length increases and reaches 40% when the length is 6.

# REFERENCES

[1] Alexandru Agache et al. 2020. Firecracker: Lightweight virtualization for serverless applications. In *the 17th USENIX NSDI*.
[2] IETF. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport.* RFC 9000.
[3] Eric Jonas et al. 2019. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint, arXiv:1902.03383* (2019).