QDSR: Accelerating Layer-7 Load Balancing by Direct Server Return with QUIC

Ziqi Wei, *Tsinghua Shenzhen International Graduate School and Peng Cheng Laboratory;* Zhiqiang Wang, *Tencent and Peng Cheng Laboratory;* Qing Li, *Peng Cheng Laboratory;* Yuan Yang, *Tsinghua University;* Cheng Luo and Fuyu Wang, *Tencent;* Yong Jiang, *Tsinghua Shenzhen International Graduate School and Peng Cheng Laboratory;* Sijie Yang, *Tencent;* Zhenhui Yuan, *Northumbria University*

https://www.usenix.org/conference/atc24/presentation/wei

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

QDSR: Accelerating Layer-7 Load Balancing by Direct Server Return with QUIC

Ziqi Wei*
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory

Zhiqiang Wang*
Tencent
Peng Cheng Laboratory

Qing Li
Peng Cheng Laboratory

Yuan Yang
Tsinghua University

Cheng Luo
Tencent

Fuyu Wang
Tencent

Yong Jiang
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory

Sijie Yang
Tencent

Zhenhui Yuan
Northumbria University

Abstract

Layer-7(L7) load balancing is a crucial capability for cloud service providers to maintain stable and reliable services. However, high flexibility of the L7 load balancers(LBs) and increasing downlink relaying service result in a heavy workload, which significantly increases the cost of cloud service providers and reduces end-to-end service quality. We proposes QDSR, a new L7 load balancing scheme that uses QUIC and Direct Server Return(DSR) technology. QDSR divides the QUIC connection into independent streams and distributes them to multiple real servers(RSs), enabling real servers to send data directly to the client simultaneously. Due to the lack of redundant relaying, QDSR enables high performance, low latency, and nearly eliminates additional downlink relaying overhead.

To evaluate the performance of QDSR, we implemented all its components using Nginx and Apache Traffic Server, deployed them in a real environment testbed, and conducted large-scale simulation experiments using mahimahi. The experimental results show that QDSR can process an additional 4.8%-18.5% of client requests compared to traditional L7 proxy-based load balancing schemes. It can achieve a maximum throughput that is 12.2 times higher in high-load scenarios and significantly reduce end-to-end latency and first packet latency.

1 Introduction

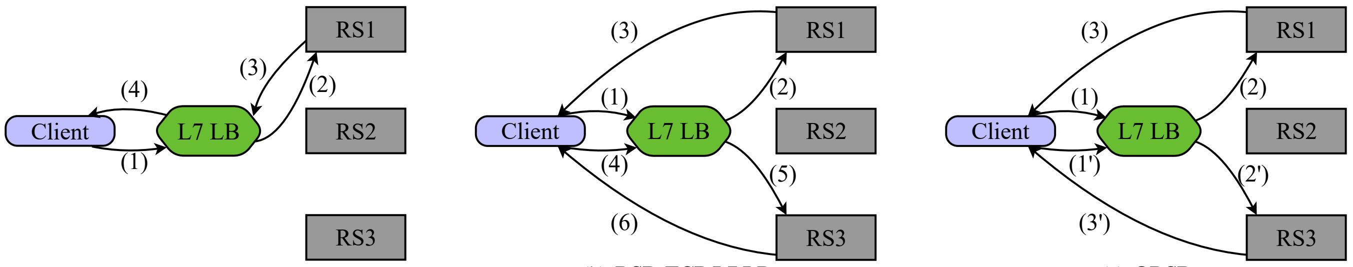
Load balancing plays an essential role in today's Internet [7, 42]. Service providers deploy load balancers (LB) to improve network performance and prevent interruptions caused by single-node failures [12, 27]. While load balancing can be realized by different technologies such as Anycast and Domain Name System (DNS) [8], a widely used LB leverages reverse proxy that works in the application layer(L7) [13]. L7 LB performs fine-grained control at the connection level or even at the request level, and the content-awareness enables L7 LB to achieve advanced functions, such as incorporating security protections and supporting sticky redirection [7]. As such, L7 load balancing becomes irreplaceable as HTTP and Transport Layer Security (TLS) have been widely adopted by various applications.

It is important to improve the performance for end users, e.g. latency, throughput, web page load time, etc. A huge effort has been spent on developing protocols such as HTTP/2 [37], HTTP/3 [5], and QUIC [20]. In particular, HTTP/2 abstracts stream multiplexing, and HTTP/3 uses QUIC instead of TCP to achieve better parallelism. Thus, a user can generate multiple requests simultaneously to accelerate the web page loading. Unfortunately, we find that for a connection that uses an L7 LB and multiple real servers (RS), the LB load capacity and parallelism always cannot be balanced.

A typical technique is the Proxy-based L7 LB. The L7 LB maintains the connection state with the client and splits the connection into request granularity, then integrates the required resources requested from different RSs and relays them to the client, as shown in Figure 1(a). However, from a functional perspective, it makes sense to filter and relay uplink traffic through the L7 LB due to the large amount of control information and potential attacks, but it is completely unnecessary to relay downlink traffic through the L7 LB again. The large amount of meaningless downlink relaying quickly makes the L7 LB a performance bottleneck, which reduces throughput and end-to-end latency. This problem is particularly evident for requests such as video streaming and large file downloads.

The typical solution to the above problem is the Direct Server Return(DSR) technology [11, 30, 46], which allows the RS to bypass the L7 LB and establish a data transmission channel directly with the client. The mature practice of DSR technology in the current network is the DSR-TCP schemes [6, 15], as shown in Figure 1(b). After receiving a request from the client, the L7 LB packages the connection context and hands it over to an RS, which can bypass the L7 LB and directly communicate with the client using the con-

*: Equal Contribution
Corresponding Author: Qing Li, liq@pcl.ac.cn



(a) **Proxy-based L7 LB** (b) **DSR-TCP L7 LB** (c) **QDSR**

Figure 1: L7 load balancing schemes.

text information. However, DSR-TCP technology couples the communication process between the client-LB and LB-RS, and there is no finer-grained context than the connection in TCP(compared to the stream in QUIC), which means that only one RS can send resources to the client at the same time for a connection, making it almost impossible for the multiplexing of multiple HTTP requests in HTTP2 and HTTP3 to work effectively in one connection. We call this the serial request dilemma of DSR-TCP. Furthermore, such DSR-TCP schemes hand off the entire transport state from the L7 LB to the RS, including TCP connection hand-off [9] and TLS state hand-off [15], leaving the RS being directly exposed to the WAN, which makes the RS vulnerable to attacks.

To solve the conflict between the load capacity and parallelism of L7 LB, we propose QDSR, a high-performance and cost-effective L7 load balancing scheme with QUIC and Direct Server Return. By combining QUIC and DSR technology, QDSR's finer-grained request processing method allows the L7 LB to balance both load and parallelism simultaneously, as shown in Figure 1(c). However, The development of QDSR involves addressing the following challenges:

Parallel transmission and security: Existing connection hand-off can only achieve serial redirection, which also exposes the RS directly to the WAN, severely limiting the request processing speed and introducing additional security risks. QDSR use stream hand-off to substitute connection hand-off, which enables parallel processing multiple request streams belonging to the same connection. To avoid direct attacks on the RS from the WAN, we design a heterogeneous up/downlink structure. The uplink traffic of each stream is processed by the L7 LB, while the downlink traffic is sent directly from the RS to the client, making it impossible for attacks from the WAN to directly reach the RS.

Maintaining connection consistency: We move data transmission from the L7 LB to the RS, while reserving the L7 LB's uplink control capability, which leads to the control-data separation and poses a challenge on maintaining connection consistency. To address the issue, QDSR establishes an auxiliary long connection between the L7 LB and each RS, through which the L7 LB and RSes exchange control information in a special form. QDSR ensures that the separated control and data capabilities do not violate connection consistency and transparent transmission is guaranteed for clients.

Packet number space isolation: Because an RS is not aware of other RSes that share the same connection, packet number conflicts between data packets from different RSes may be incurred. Simply pre-allocating packet numbers may lead to packet disorder and unnecessary retransmission due to path heterogeneity. To address the issue, we propose stream packet number space isolation, which is inspired by the packet number space management of multipath QUIC [25]. Our approach allows each RS to independently allocate packet numbers and exchange allocation information with the L7 LB through the auxiliary long connection.

The contributions of this paper is summarized as follows.

- We propose a new L7 load balancing solution, QDSR, which combines the characteristics of DSR technology and QUIC streams. QDSR achieves both enhanced parallelism and direct server return in the application layer, greatly enlarges the capacity of L7 LB to process more requests simultaneously, and improves user experience by greater throughput and less end-to-end latency.

- We overcome a number of challenges to realize QDSR. We implement DSR while protecting RSes against attacks from WAN. We synchronize states between the L7 LB and RSes for maintaining connection consistency and guarantee transparent transmission for clients. We also ensure that there are no packet number conflicts between packets from different RSes.

- We implement QDSR with open source projects commonly used in the industry. In particular, the L7 LB is realized with Nginx [29] and the RS is realized with Apache Traffic Server [2]. Our source code has been publicly released at: https://github.com/zhqiangwang/QDSR.

- We evaluated QDSR in both a real testbed and a large-scale simulation environment. The results show that our QDSR LB can process an additional 4.8%-18.5% of client requests, achieve a throughput that is over 10x in high-load scenarios, and significantly reduce end-to-end latency and first packet latency.

This work does not raise any ethical issues.

Table 1: Feature comparision between different load balancing schemes.

LB	Granularity	Load balancing identifier	Load balancing flexibility	Relaying overhead	Processing overhead	Latency overhead
L4 w/ DSR	Connection	5-tuple-based	Low	Uplink packets	Low	Low
Proxy-based L7	Parallel request	Content-based	High	Every packet	High	High
L7 w/ DSR-TCP	Serial request	Content-based	High	Uplink packets	Low	Low
L7 w/ QDSR	Parallel request	Content-based	High	Uplink packets	Low	Low

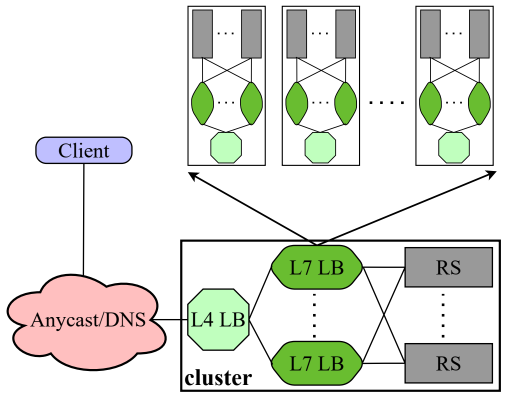


Figure 2: Typical traffic scheduling topology and load balancing chain.

2 Background and Motivation

2.1 Load Balancing Chain

The overall load balancing effect relies on the cooperation of each layer of the load balancing chain. Figure 2 illustrates the typical load balancing topology deployed by cloud service providers. A cloud network usually consists of many geographically distributed clusters with several or dozens of servers, usually acting as Layer-4(L4) LBs, L7 LBs and RSs. Some scenarios may integrate L7 load balancing instances and real service instances on the same physical server, or even in the same program [43]. This does not affect our analysis because for access traffics that require sticky redirections, the access requests distributed across different servers by L4 LB must be aggregated on one server eventually, which uses logical content-based L7 relaying. In other scenarios [1,36], there are some specialized clusters which only have LBs and all RSs are accessed over the WAN. They acts as an access gateway to perform load balancing strategies and attack prevention.

After determining which cluster acts as the access gateway to serve the client by Anycast/DNS scheduling, the client connects with one of the L7 LB assigned by the L4 LB and sends requests to it. At the designated L7 LB, the requests are assigned to different RSs according to the load balancing strategy, such as (weighted) round-robin, random, minimum load, and consistent hashing. In case that some requests are not served by any RS in this cluster, such as cache misses for static content, the L7 LB relays these requests to other clusters, and then relays the responses back to the client.

2.2 Load Balancing characteristics

Anycast and DNS are responsible for large-scale traffic scheduling. A schedule strategy usually involves trade-offs between cluster utilization, access latency, bandwidth cost, etc. Table 1 compares the characteristics of different load balancing schemes.

The L4 LB is responsible for connection-granularity redirection, with the load balancing identifier typically being a 5-tuple. To achieve content-based L4 redirection, some unpublished schemes propose that the L4 LB performs the TCP handshake with the client and then provisionally caches and peeks at the first few packets of the new TCP connection until obtaining the HTTP header. After obtaining the URL of the first request, the L4 LB selects an RS and replays the cached packets to it. However, these schemes blur the line between L4 LB and L7 LB and are difficult to apply to encryption protocols such as HTTPS, HTTP/2, and HTTP/3.

Proxy-based L7 load balancing is the most flexible scheduling scheme because the redirection connection and the client-facing connection are completely decoupled at the cost of the processing, relaying, and latency overheads.

DSR-TCP can be applied to encryption protocols based on the entire connection hand-off. In practice, it is not possible for multiple servers to send packets simultaneously over one TCP connection. Therefore, DSR-TCP can only achieve serial request-granularity redirection by multiple handing off the whole connection state.

2.3 QUIC and HTTP/3

Like TCP, QUIC is a reliable transport protocol that requires establishing a connection before sending messages, and acknowledges received packets. A QUIC connection is identified by a set of connection IDs (CIDs) instead of the 5-tuple (protocol, source IP, destination IP, source port, destination port). To achieve good resilience and network adaptability, it allows the client to change IP address and UDP port during transmission, called QUIC connection migration. To avoid privacy leakage [20] and achieve load balancing [12], the CIDs are dynamic and QUIC provides NEW_CONNECTION_ID and RETIRE_CONNECTION_ID frames to manage the life cycle of CIDs.

In a QUIC connection, endpoints can independently create lightweight, ordered byte-streams without mutual interference, called stream multiplexing. A QUIC endpoint avoids excessive data sending by combining two main mechanisms: the flow control advertised by the receiver, and the congestion control [19].

QUIC is a secure transport protocol with TLS/1.3 integration. Unlike TLS over TCP [39], there is no strict layering in the integration [38]. QUIC uses the TLS handshake and packet protection provided by TLS. TLS relies on the reliability and ordered delivery of QUIC.

HTTP/3, the next major version of HTTP, has been officially published by IETF [5]. Since HTTP/2 has a major problem with "head-of-line blocking", which is primarily caused by TCP's lack of concurrency, HTTP/3 boldly uses QUIC [20] implemented in userspace as its transport layer, instead of traditional TCP. In HTTP/3, the client must send only one request on a QUIC stream, and the server must reply on this stream. This means that independence and parallelism of request granularity are equivalent.

2.4 Motivation and Challenges

The proxy-based 7-layer load balancing method achieves fine-grained load balancing scheduling at the cost of a large amount of redundant downlink forwarding, while traditional DSR technology solves the problem of redundant forwarding but brings challenges in parallel processing and security. Based on this, we overcome the challenges of security, transparent transmission, maintaining connection consistency, and handling packet space isolation in the design and implementation process by combining the finer-grained context of QUIC and DSR technology, ultimately achieving the QDSR solution.

In the remaining sections of this paper, we will introduce the design of QDSR($ 3), implementation($ 4), test bed testing and large-scale simulation experiments($ 5). We then analyze the robustness of QDSR($ 6) and discuss some limitations and implementation issues($ 7). Finally, we provide related work($ 8) and conclusions($ 9).

3 Design of QDSR

Based on the aforementioned problems and challenges, the design principles of QDSR are as follows: 1) Real servers should establish unidirectional data transmission tunnels directly with the client, eliminating the need for downlink traffic to be relayed through the L7 LB. 2) Multiple RSs should be able to respond to multiple requests from the client simultaneously while ensuring that the client remains unaware of any changes in the server, just as the client always communicates with the L7 LB. 3) Connection consistency should be ensured throughout the communication process, and the L7 LB and RSs should flexibly exchange connection and flow states to avoid communication abnormalities. 4) All uplink traffic should be processed and relayed by the L7 LB first, ensuring that the RSs are not exposed to the WAN and vulnerable to malicious attacks.

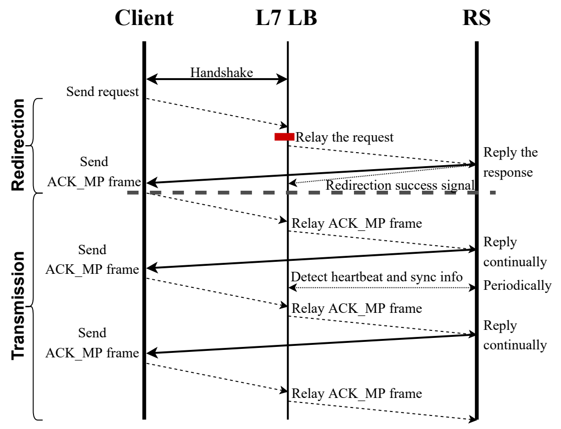


Figure 3: The sequence diagram of QDSR. The dash lines represent control information and the solid lines data streams.

Whenever the connection is alive, the L7 LB is the control center which is responsible for the routine protocol interaction such as connection handshake, TLS handshake, version negotiation, address validation, and control streams, as specified by RFC 9000 [20] and RFC 9114 [5]. RSs are only in charge of the QUIC connection's downlink response transmission and associated transmission states including congestion control state, probe MTU state, and so on.

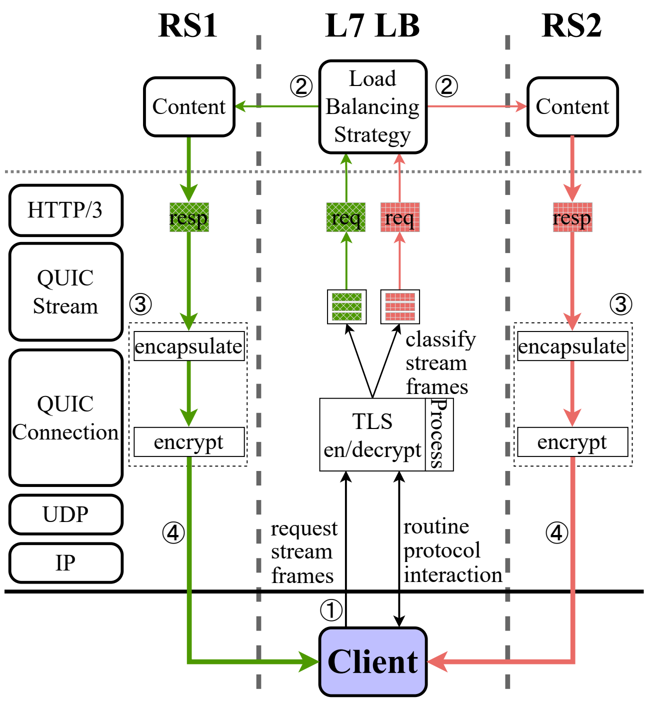
QDSR has two phases, the redirection phase ($ 3.1) and the transmission phase ($ 3.2), as illustrated in Figure 3.

3.1 Redirection Phase of QDSR

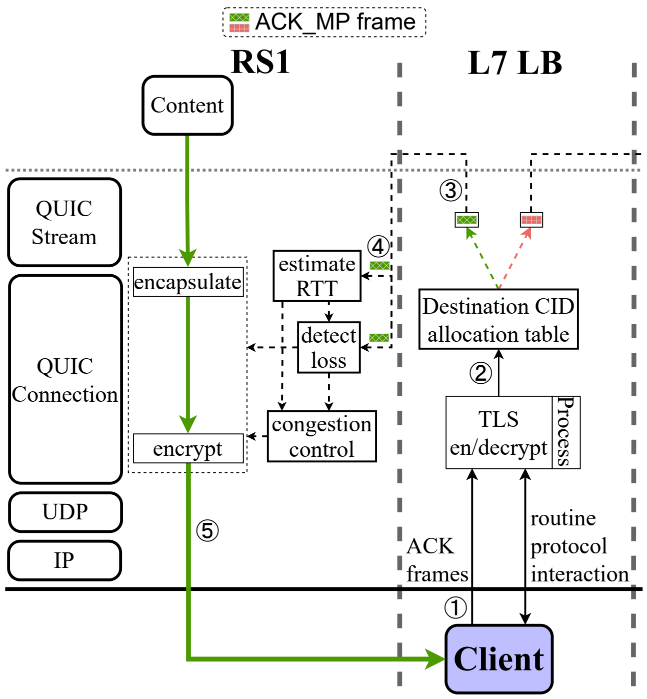
The overview of the redirection phase (also called stream hand-off) is illustrated in Figure 4(a). The L7 LB will establish the QUIC connection, negotiate and determine the cryptographic and transport parameters once the handshake is confirmed. Then, it can initiate QDSR. To begin HTTP transactions, the client initiates some bidirectional streams at any moment and sends requests over their corresponding streams.

When client-initiated bidirectional stream frames arrive at the L7 LB, it classifies them by stream ID, and assembles stream frames with the same ID into HTTP requests by stream offset. After an HTTP request is entirely assembled, the L7 LB selects one RS to handle this request according to the load balancing strategy.

Every time a request is received, the L7 LB serializes QUIC connections and streams into special HTTP headers, and then relays them to the RS via redirection connections or existing



(a) QDSR redirection phase (also called stream hand-off). ① The client initiates new streams and send requests. ② The L7 LB relays the request to the RS with QUIC connection and stream information. ③ The RS constructs auxiliary QUIC connection and stream, then encapsulates and encrypts packets of the response. ④ The RS rewrites UDP port and IP address of packets, then sends them to the client.



(b) QDSR transmission phase. ① The client sends ACK_MP frames to the L7 LB. ② The L7 LB classifies the ACK_MP frames by the destination CID allocation record table. ③ The L7 LB relays the ACK_MP information and other control signal to each hand-off stream. ④ The RS estimates RTT and detects loss for congestion control. ⑤ The RS continuously encapsulates and encrypts packets of the response, then sends them to the client, regulated by the congestion control and flow control.

Figure 4: Two phases of QDSR.

keep-alive redirection connections.

RS receives a request with a special HTTP header, decodes the header to get connection states, and constructs an unidirectional data transmission tunnel without handshakes or any other interaction, which is a partial simulation of the QUIC connection between L7 LB and the client. Then the RS sends the response directly to the client by the data transmission tunnel.

A QUIC connection in the L7 LB is shared by multiple RSs at one time, and there is an unidirectional data transmission tunnel in each of the multiple RSs. Each tunnel only simulates one stream that maps a request, unless one RS needs to handle multiple requests from the connection simultaneously. Metaphorically speaking, the entire process can be likened to the L7 LB delegating the task of sending streams to different RSs while concurrently processing control information, hence we called **Stream hand-off**.

To ensure connection consistency and transparent transmission for the client, the information that the RS needs to construct the unidirectional data transmission tunnel is detailed as follows and also summarized in Table 2.

CID and packet number: Every packet in QUIC belongs to a unique packet number space. There are three packet

Table 2: The main information required by the RS to encapsulate and encrypt the QUIC packets.

Field	**Description**
CID	Identifier of the connection
Stream ID	Identifier of the hand-off stream
Packet number	An available initial packet number
TLS materials	Used to encrypt packets
5-tuple	Used to spoof packets
Flow control	The initial max stream offset window

number spaces defined by IETF standards [20, 38], namely the initial packet number space, the handshake packet number space, and the application data packet number space. Each packet number space is associated with different contexts of sending, acknowledging, encrypting, congestion control, and loss recovery. To achieve path isolation, multipath extension [25] specified that each CID is associated with a unique packet number space for application data.

Due to QDSR allowing different RSs to deliver request data to the client simultaneously, it inevitably brings about challenges in maintaining connection consistency, particularly in how packet number spaces are managed. Inspired by the multi-path extension of QUIC, which allocates, records, and manages data packets from different CIDs but belonging to the same connection based on different packet number spaces, we assign different CIDs in the connection pool to different RSs so that all RSs currently responding to requests can use their respective CIDs for data transmission. In this way, each real server can independently maintain the context of its packet number space, such as loss detection, retransmission, and congestion control. In short, each real server achieves the purpose of simplifying state sharing among servers by exclusively occupying a packet number space.

At the start of stream hand-off, the L7 LB allocates an unused destination CID with its smallest unused packet number to the RS by writing them into the HTTP header. To avoid excessive consumption of destination CIDs, the RS returns the allocated destination CID and the maximum used packet number to the L7 LB upon completion of the HTTP transaction. The returned destination CID can then be re-allocated to subsequent RSs, with the unused destination CID with the smallest packet number being able to be serially reused by multiple RSs. The L7 LB is responsible for maintaining the destination CID allocation table throughout the entire stream switching process.

The standard specified that destination CIDs are issued by the receiver and may be retired by NEW_CONNECTION_ID frame or RETIRE_CONNECTION_ID frame [20] sent by the receiver. In QDSR, we are also compatible with this mechanism. When retiring the destination CID allocated to one RS, the L7 LB relays the retiring signal to the RS with a new unoccupied destination CID and the CID allocation record table is updated and the RS immediately replaces the retired CID with the new CID.

However, the above solution requires a certain level of capability from the client. For clients that support the multi-path extension of QUIC, QDSR can be supported without modification. For clients that without multi-path QUIC capability, minimal modifications are required to fully support QDSR's functionality. To avoid client modifications and further compatibility with IETF QUIC, we have designed a backward version of QDSR, the design and discussion of which can be found in Appendix A.

Stream ID: Each stream is assigned a unique stream ID, which the real server requires to encapsulate the response stream frames.

TLS materials: With QDSR, all uplink packets go to the L7 LB for decryption, while the RSs handle encryption. We do not modify the handshake of TLS and just share the negotiated 1-RTT security parameters to each RS because all packets of HTTP responses are encrypted by 1-RTT key. Packet number space isolation will not cause encryption isolation. When stream hand-off, the security materials, including current server write secret, cipher suite and header protection key [38], are determined by the handshake and then relayed to the RS to produce 1-RTT key.

In QUIC multipath [25], all application data packet number space share the 1-RTT key with the original application data packet number space. And the nonce for header protection is calculated by combining Initialization Vector (IV) with the packet number and the packet number space ID (PNSID) , which guarantees the uniqueness of the nonce to avoid attacks that modify the destination CID, as analysed in $ 6.1. In QDSR, we inherit this method and it will not introduce any additional latency.

QDSR also supports 0-RTT because the 1-RTT key is available when the L7 LB initiates stream hand-off. After handshake, the only opportunity to update encryption parameters is Key Update defined in TLS 1.3 [39]. The signal of Key Update is the toggle of Key Phase Bit in QUIC packet header [20]. When the signal arrives at the L7 LB, the L7 LB immediately updates the 1-RTT key and then relays them to all RSs. Every RS immediately updates the key and sends at least one packet by the new key. To avoid confusion by multiple key updates, the client must not initiate a subsequent Key Update until every packet number space receives a current key encrypted packet, like multipath extension [25].

5-tuple: The RSs require the use of the 5-tuple of the connection between the L7 LB and the client to deliver request data in the unidirectional data tunnel. Therefore, during the stream hand-off process, the L7 LB relays the 5-tuple to the RSs. In the event of connection migration, the 5-tuple will be modified. Following the completion of new path validation by the L7 load balancer, the modified 5-tuple will be relayed to each RS to facilitate the sending of packets.

Flow control: In order to prevent the RSs from exceeding the receive window, the L7 LB allocates and relays the receive window to the RSs. More details regarding this process can be found in $ 3.2.

For each RS, based on the information provided above, the encapsulation and encryption procedures are summarized in Figure 5. Firstly, the RS parses the request and prepares the response. Secondly, it compresses the HTTP response header using QPACK [22]. Next, the compressed headers and available content are encapsulated into HTTP/3 header frames and HTTP/3 data frames, respectively. Following the QUIC protocol specifications mentioned earlier, the RS encapsulates the HTTP/3 frames into QUIC stream frames. Then, it uses the TLS key to encrypt the packets [38] as QUIC packets. Finally, the RS sends the QUIC packets to the client using the latest 5-tuple.

If the RS successfully sends the response, it replies with a success signal to the LB, which then enters the transmission phase. However, if the response is not sent successfully, the RS replies to the LB with the response.

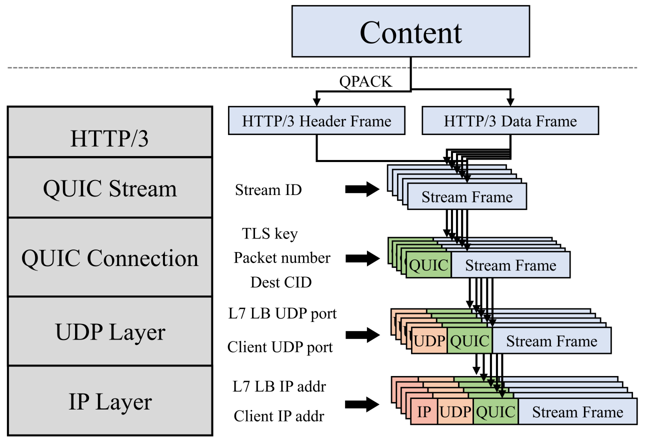


Figure 5: The procedures of encapsulation and encryption in the auxiliary connection instance executed by the RS.

3.2 Transmission Phase of QDSR

Once the client receives the packets sent by the RS, the redirection phase is completed and the transmission phase begins. The overview of the transmission phase is illustrated in Figure 4(b). The primary objectives of this phase are to maintain transmission reliability, perform congestion control, and implement flow control.

To acknowledge the received packets, the client sends ACK_MP frames defined in QUIC multipath [25] to the L7 LB. When the L7 LB receives an ACK_MP frame, it looks up the RS in the destination CID allocation table according to the PNSID in the frame, then relays the decrypted frame to the RS. Therefore, for each hand-off stream, the transmission loop of packets sent by an RS consists of the L7 LB, the RS and the client.

According to the method specified in RFC 9002 [19], the RS generates RTT samples and detects lost packets by the ACK_MP frame for its transmissions. Each RS and the L7 LB will perform its own congestion control algorithm independently, so that the transmission processes will not interfere with each other. By isolating packet number space, RSs are able to distinguish out-of-order packet arrivals and determine effective ACK delay through ACK_MP, which is critical to determining RTT and detecting lost packets [19].

In the classic congestion control algorithm, the inputs are loss signal and/or RTT, and the output is the current congestion window or the pacing rate, which is used to limit the sending rate of the unidirectional data tunnel between the RS and the client.

Flow control includes data flow control and concurrency control. The L7 LB is fully responsible for the concurrency control as regular QUIC connection. Data flow control involves simply relaying the window to the RSs for stream receive window advertised by the MAX_STREAM_DATA frame. But the MAX_DATA frame is a global window in the connection. Due to the heterogeneity of the network, evenly distributing the global windows to each RSs is not a good method. We recommend using the stream window to limit the size of occupied buffer and the global window is calculate by the number of streams and the stream window.

4 Implementation

To better replicate the real production environment, we have implemented QDSR on the open-source L7 LB Nginx and the open-source storage real server Apache Traffic Server, which are commonly used in the industry [4, 16, 34].

4.1 LB: Nginx

Our implementation is based on the QUIC+HTTP/3 branch of Nginx [29]. It consists of 2650 Lines-of-Code (LOC, C code). As the L7 LB, Nginx reserves the possibility of using the traditional proxy relaying for HTTP/3 transactions, and it is up to the RS to decide whether to use QDSR.

The proxy module of Nginx communicates with the RS using TCP and HTTP/1.1, as Nginx http proxy module only supports TCP between the L7 LB and the RSs. In our implementation, the L7 LB and the RS reuse this TCP connection for communication of stream hand-off after the request has been sent, and it will be kept alive even if the transaction is completed. Nginx attaches the information shown in Table 2 to HTTP headers when it relays the request to the RS. If the backend caches the content, it notifies the L7 LB and starts the process of the stream hand-off. The QUIC stack implemented in Nginx routinely interacts with the client and records the destination CID allocated to each RS. The hand-off streams may coexist with the regular streams, that is, they do not interfere with each other.

4.2 RS: Apache Traffic Server

We developed the prototype of the RS on Apache Traffic Server (release 9.1.1) [2], which consists of 4128 LOC (C++ code). The stream hand-off stack runs as some threads, called DSR threads, which are homogeneous. Each DSR thread is bound to a dedicated CPU core. The NET threads process HTTP transactions and interactions with the L7 LB by the interface of the stream hand-off stack. The DSR threads poll the processing hand-off stream and send packets with the guidance of the congestion control algorithm.

To behave as the L7 LB without breaking the connection, the RS uses the LB's 5-tuple message to transmit QUIC packets. If the *IP_HDRINCL* socket option is enabled on the raw socket by *setsockopt()*, the kernel does not generate an IP header for the packets and the application must specify an IP header, then the RSs writes client IP and LB IP to IP header.

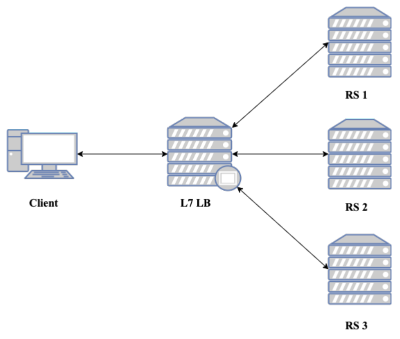


Figure 6: Testbed topology.

Table 3: Hardware configurations of machines.

Type	CPU	Memory	Link
A	Intel Xeon Silver 4114 2.2GHz, 20 cores	16 GB	40 Gbps
B	Intel Xeon Silver 4216 2.1GHz, 64 cores	256 GB	100 Gbps
C	Intel Xeon Silver 4208 2.1GHz, 32 cores	128 GB	100 Gbps

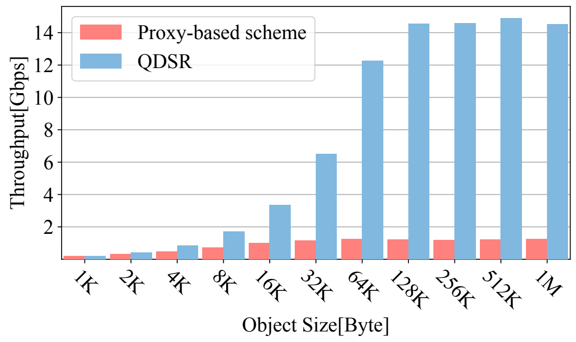


Figure 7: The bottleneck throughput of the proxy-based scheme and QDSR under 8 QUIC connections and 8 stream multiplexing in each connection.

5 Evaluation

To comprehensively evaluate the capabilities of QDSR, we conducted multiple experiments in both a real testbed and a large-scale simulation environment. Our testbed consisted of five real servers (3A, 1B, 1C) and a switch to connect them, with the hardware configurations listed in Table 3. The large-scale simulation environment was built using Mahimahi, a general-purpose network simulation emulator.

The topology of the testbed experiment is shown in figure 6. We used an A-type server as the L7 LB, two A-type servers and one B-type server as the RS, and a C-type server as the client. The reason for using an A-type server as the L7 LB is that its hardware configuration is lower, making it easier to become a performance bottleneck in subsequent experiments.

All servers used Ubuntu 18.04 as their operating system. For the client, we modified lsquic [35] to add the necessary packet number space isolation functionality. To avoid the client becoming a bottleneck, we extended the client to be multi-process, which means that during the experiment, the client established multiple QUIC connections with the L7 LB, and each connection contained multiple parallel streams. The client continuously sent requests to the L7 LB to maintain the desired level of concurrency.

Our baseline is the proxy-based L7 LB scheme, which is currently widely used in our global DCs. We do not consider DSR-TCP as a baseline for two reasons:1) DSR-TCP cannot achieve parallel processing of requests, which puts it in an unfair position compared to QDSR. 2) DSR-TCP will bring serious security issues in real network, making it impossible to use DSR-TCP in real network. We believe that using a solution that cannot be deployed as a baseline is inappropriate and may mislead future work.

5.1 Throughput

We first evaluated the throughput improvement brought by QDSR to L7 LB compared to the proxy-based scheme. In this experiment, the client continuously initiated bidirectional streams to send requests to the L7 LB and made the L7 LB the performance bottleneck until CPU utilization reached 100%. Specifically, we limited Nginx to run a single worker process and provided sufficient resources for the RSs and the client. Each RS was composed of multiple Apache Traffic Server instances, and each instance ran multiple DSR threads on its bound CPU core, with each hand-off stream assigned to a DSR thread. To avoid the influence of IO overhead on the experimental results, all content was cached in memory with extremely low overhead. As described in $ 4.1, we established sufficient active TCP connections between the L7 LB and the RSs for exchanging control messages to avoid the overhead of establishing TCP connections again.

The experimental results are shown in Figure 7. For both schemes, the overall throughput increased with the size of the requested objects, but the bottleneck for the L7 LB varied for different object sizes. In both QDSR and the traditional proxy-based scheme, when the requested object size was small, the bottleneck was processing requests, including maintaining QUIC connections, receiving and decrypting packets, and relaying them to the RSs. In this case, the cost difference between the relayed downlink traffic and the relayed uplink traffic was not significant, as the fixed cost of each call (context switch overhead) dominated [15]. Compared to the proxy-based scheme, the cost of relaying ACK_MP frames and relayed downlink traffic was almost equivalent.

As the size of requested objects increases, the throughput of proxy-based LB quickly reaches a bottleneck. When the

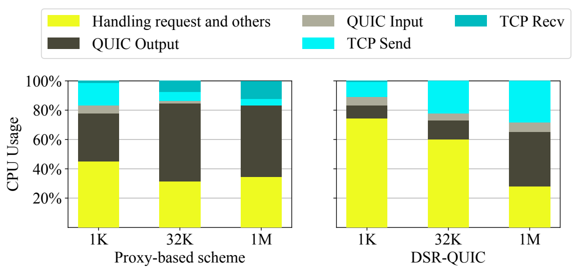


Figure 8: CPU utilization under different object size.

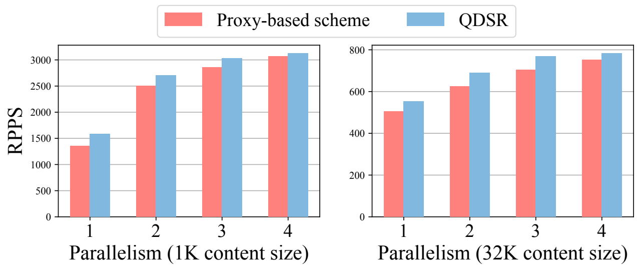


Figure 9: RPPS with different parallelism.

size of requested objects exceeds 16K, the number of requests processed per second decreases, and the throughput hardly increases. Therefore, the downlink traffic relaying becomes the bottleneck. In contrast, QDSR LB shows an exponential growth in throughput before the size of requested objects is smaller than or equal to 64K, indicating that the number of queries per second (QPS) remains almost unchanged, and the overhead of relaying ACK_MP frames to RSs can be negligible. When the size of objects is greater than or equal to 128KB, the bottleneck throughput reaches the maximum value. Compared with proxy-based scheme, QDSR significantly improves the bottleneck request threshold of L7 LB and has 12.2 times the maximum throughput.

To further analyze the reason for the throughput improvement brought by QDSR, we used perf and Flame Graph [14] to sample the CPU usage of L7 LB under different object sizes. The experimental results are shown in Figure 8. Compared with the traditional proxy-based approach, QDSR has almost no overhead caused by TCP Recv. This is because the downlink traffic is directly relayed to the client by RSs without being relayed to L7 LB through TCP connections first. The saved CPU resources are used to maintain and process more requests and synchronize and exchange connection states by TCP connections between L7 LB and RSs.

In addition, we also evaluated the benefits of QDSR from the client's perspective using requests processed per second(RPPS). In this experiment, we maintained a certain degree of parallelism flows at the client. For example, if the degree of parallelism is 3, the client maintains only three request flows at the same time and hands them over to the L7 LB for processing. The experimental results are shown in Figure 9 under different client parallelism and request sizes. Compared with the proxy-based scheme, QDSR achieved an additional request gain of 4.8%-18.5%. Compared with the experiment in Figure 7, since L7 LB did not reach its performance bottleneck, the benefits of QDSR mainly come from the reduction of RTT caused by the downlink traffic not requiring additional relaying, rather than the elongation of request completion time caused by the performance bottleneck of L7 LB.

5.2 Latency

Latency is an important factor affecting the quality of service for latency-sensitive applications such as live streaming and video calls. Figure 10 shows a comparison of end-to-end latency between QDSR and the proxy-based approach under different levels of background traffic. In this experiment, we evaluated RTT, one-way delay(OWD) for uplink and downlink traffic, and counted the number of context switches. In terms of implementation details, to evaluate one-way delay, we performed clock synchronization between the client and RSs, and the measured RTT did not include the processing delay at the application layer on the RSs. Both background and measurement traffic requested 1KB objects.

As shown in Figure 10, the latency does not increase with the increase of background traffic. When the LB is close to idle, the measurement traffic triggers continuous context switches and CPU migrations, which increase the latency. As the background traffic increases, the number of context switches and CPU migrations decreases sharply. Therefore, even though the CPU load is higher at this time, the latency does not increase significantly, but even decreases, causing non-smooth experimental behavior. For the proxy-based scheme, the downlink one-way delay is greater than the uplink one-way delay. QDSR and the proxy-based scheme do not have a significant difference in uplink one-way delay. However, for downlink one-way delay, QDSR reduces it by about 50% compared with the proxy-based scheme, and the RTT gain is mainly contributed by the downlink one-way delay. This is because RSs directly send responses to clients without involving the network stack (input and output) of L7 LB, which reduces latency.

5.3 End-to-End Performance over WAN

To evaluate the performance of QDSR in WAN, we used mahimahi [26] to build three different topologies as shown in Figure 11, which simulate the three possible relative positions of L7 LB, RSs, and clients in the wide area network. We evaluated the first packet delay of QDSR and the proxy-based scheme in different topologies and the stability of QDSR in different packet loss environments.

First packet delay determines the response speed of the service, which is directly related to the quality of service and

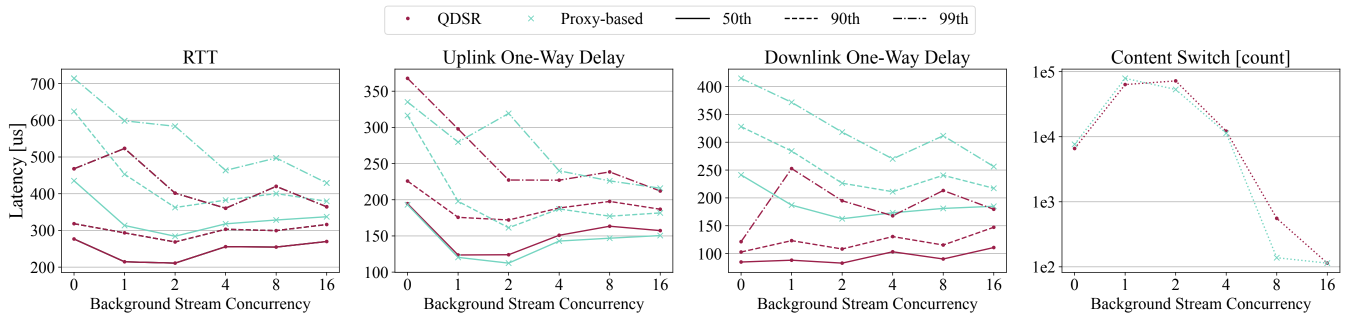


Figure 10: Latency comparison between QDSR and the proxy-based scheme.

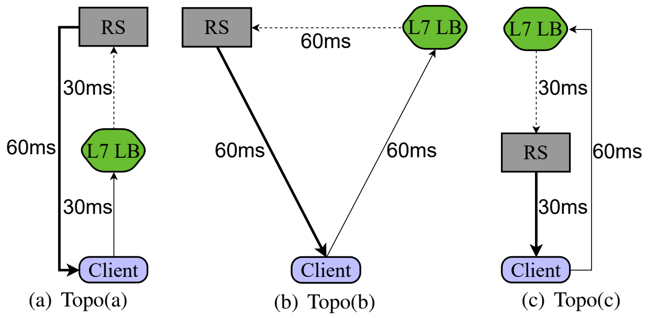


Figure 11: Three topologies for evaluating end-to-end performance. The thin solid line is the uplink, the dotted line is the redirect link and the thick solid line is the downlink.

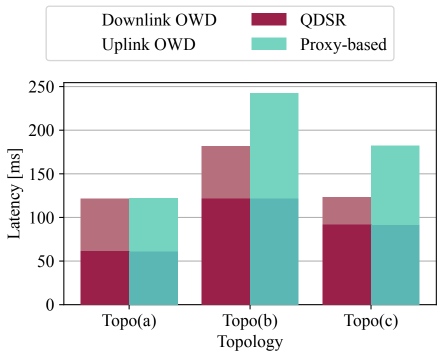


Figure 12: Average first packet latency in three topologies. OWD means one-way delay.

revenue, especially for live streaming and video-on-demand businesses. For the proxy-based scheme, the first packet delay is determined by the client-LB RTT and the LB-RS RTT. However, for QDSR, the first packet delay only includes the uplink one-way delay of client-LB, the redirection delay, and the downlink one-way delay of RS-client. When the router selects the nearest network path, the latter is usually not greater than the former. Figure 12 shows a comparison of the first packet delay between the two approaches in three typical network topologies. In topology A, the L7 LB is located between the client and RSs, and the RTT of RS-client is the sum of the RTT of RS-LB and LB-client paths. In this scenario, QDSR and the proxy-based approach obtained almost the same first packet delay, because the redirection delay overhead of QDSR almost offset the downlink relaying delay overhead of the proxy-based scheme, resulting in similar performance. However, in topologies B and C, L7 LB and RSs belong to different clusters, which leads to significant differences in delay between the RS-client path and the RS-LB-client path. QDSR obtained significant benefits from the downlink traffic relaying compared with the proxy-based scheme, significantly reducing the first packet delay.

Finally, we evaluated whether the path heterogeneity of QDSR would affect the overall download process in different packet loss environments. We chose topology A above and limited the link bandwidth to 12Mbps, used the CUBIC algorithm for congestion control, set the queue type to droptail, and set the queue depth to 480 packets. The experimental results shown in Figure 13 indicate that QDSR's actual bandwidth fluctuates around the theoretical bandwidth of 12Mbps in environments with packet loss rates of 0.01, 0.03, and 0.1. Higher packet loss rates lead to actual download speeds deviating from the theoretical value. The overall experimental effect is similar to the proxy-based scheme, and the path heterogeneity of QDSR does not affect the download process.

6 Robustness Analysis

6.1 Security

Spoofing IP: To establish a unidirectional data tunnel between the RS and the client, it is necessary to use the RS to spoof the IP address of the L7 LB in order to complete the data transmission process. Even if the L7 LB and RSs communicate over WAN, spoofing IP address is feasible. Anycast is a typical technology that multiple servers in different locations share the same IP address by BGP announcement. In QDSR, the uplink packets rely on the relaying of the L7 LB, rather than the network routing. Therefore, we do not need BGP announcement for the uplink packets, or even configure the same IP address for the L7 LB and RSs. We just spoof and send it out. From the scheduling perspective, the skew scheduling

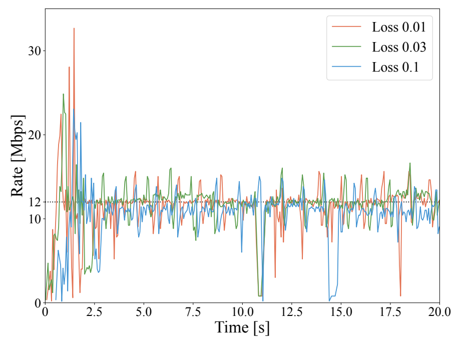


Figure 13: Download rates under different loss links.

of Anycast can be compensated by the precise scheduling of QDSR. To avoid information leakage, the communication between the L7 LB and the RSs must be encrypted and secure in real production environment. In the absence of the TLS secret, the attacker cannot exploit the spoof attack.

Multiple packet number spaces: A serious problem is how to resist attacks that modify the destination CID of QUIC packets to another valid destination CID of this connection, which will confuse the client with the packet number space. To solve this problem, the multipath draft [25] calculates the nonce of header protection by initialization vector, packet number, and PNSID ($ 3.1). If the destination CID is tampered with another valid CID of this connection, the nonce calculated by the client is not correct and the decryption will fail. This tampered packet will then be dropped by the client.

RS Protection: QDSR does not make any changes to the handshake, and all designs are made after the handshake is confirmed. In QDSR, the sender of packets cannot be distinguished by the client or attackers. Since we did not disclose the real IP address of the RSs and all uplink packets are relayed by the L7 LB, abnormal traffic and abnormal frames cannot be relayed to the RSs, and the RSs are still protected by the L7 LB. Therefore, we did not introduce any additional security issues for uplink packets.

6.2 Fault Tolerant

Network failures are typical exceptions, including failures on any link between the client, L7 LB, and RS, as well as crashes of the L7 LB or RS. To detect the occurrence of failures, heartbeats should be sent between them. If the L7 LB crashes or the client disconnects, the connection should be immediately terminated. However, if an RS crashes or disconnects from the L7 LB, the entire connection will not be interrupted, and the L7 LB will immediately send a RESET_STREAM frame to the client. The excess window announced by the MAX_DATA frame will be occupied by the RESET_STREAM frame. After receiving the RESET_STREAM frame, the client will close the stream. By recording a sufficiently large unused packet number, the destination CID assigned to the RS can be reclaimed, avoiding the use of redundant packet numbers. In summary, resetting a hand-off stream will not interrupt the connection or affect other streams.

Through stream isolation and packet number space isolation, QDSR inherits the flexibility of QUIC. Although QDSR is a many-to-one transmission model in physical terms, it is a one-to-one transmission model logically. If any RS is disconnected from the client, there are two solutions: one is to reset the hand-off stream, and the other is to use a proxy rollback scheme. The prerequisite for rollback is that there is a tunnel between the RS and the L7 LB, and the messages sent by the RS are relayed to the client through the tunnel.

7 Discussion

7.1 Limitation

Since the Nginx HTTP proxy module only supports TCP communication between the L7 LB and RSs, in our implementation, the LB and RSs communicate via TCP instead of QUIC. As part of future work, we will design a QUIC interface between LBs and RSs and integrate QDSR into an open-source QUIC library. This will help decouple LBs and RSs, further improving the deployability of QDSR.

Another issue of concern is the compatibility of QDSR with different congestion control tendencies. Currently, QDSR refers to well-known multipath transmission schemes [40, 47] and chooses to let each stream perform congestion control independently, which poses a challenge to the fairness of transmission. For scenarios that strictly require transmission fairness, a backward version of QDSR can be referred to in Appendix A. However, as we analyzed in the backward version, it may cause transmission disorder. A potential perfect solution is for the L7 LB to obtain real-time RTT information from the RS and globally allocate packet numbers to all streams in the order they arrive at the client based on the congestion control algorithm. We also consider this feature as part of future work and further evaluate the additional overhead and potential benefits it brings.

7.2 Flexible Scalability

Our work extends QUIC from point-to-point communication to many-to-one communication. It is not only compatible with point-to-point communication without changing the client, but more importantly, it obeys the server-client model with many-to-one communication, which allows application program architecture to become more flexible.

HTTP/2 provides the PUSH method, which allows the server to push additional responses without requesting them from the client, and HTTP/3 inherits this method. The PUSH method is beyond the server-client model but it is effective to reduce the latency for those very predictable scenarios. Some works [33, 41] achieve low latency by PUSH and they can introduce QDSR to improve their works.

7.3 Implication on QUIC Standard

QUIC is an emerging transport protocol, and some related proposals are being standardized. QDSR brings new transmission scenarios to the design of the protocol and is closely related to the multipath extension and connection migration of QUIC. Combining multipath and QDSR can easily design elegant server-side connection migration. Server-side connection migration is based on path management and stream hand-off, which can overcome the problem of packet relaying, achieve direct communication after migration, and contributes to overload prevention.

8 Related Work

Connection migration. The closest related work is Prism [15] which achieves TCP connection hand-off (DSR) via a programmable switch. Due to the limitation of TCP, it does not support multiple RSs sharing connections to send packets simultaneously. Also, the programmable switche in Prism raise the costs of equipment and deployment. Prism does not perform well for small file scenarios because of the communication between the programmable switch and the LB. A easier way is to replace the programmable switch with the LB like QDSR, which reduces the aforementioned costs.

QUIC connection migration is a promising mechanism defined by RFC 9000. However, currently, it only supports client-side connection migration. [31] proposes server-side connection migration, which can be used to achieve DSR. Unfortunately, it can only achieve serial request-granularity redirection and incur additional overheads for path validation. In contrast to connection migration, stream hand-off is more lightweight and flexible to the L7 load balancing.

L4 load balancing. L4 load balancing is a well-studied topic [3,12,21,27]. In the cluster, the L4 LB is responsible for the uniform distribution of the incoming connections to multiple L7 LBs and maintaining per-connection consistency. CID is the load balancing identifier in QUIC because of connection migration [12].

To ensure performance, the L4 LB should not interact with the client or maintain the connection state. Peeking at the application layer data to achieve the request-granularity redirection for the L4 LB is also unpractical because of the TLS encryption. Therefore, it only achieves the connection-granularity redirection and the request-granularity redirection must be accomplished by L7 LBs. QDSR and L4 load balancer perform their duties in request-granularity redirection and connection-granularity redirection, respectively.

L7 load balancing. Partition and replication are the commonly used methods that improve the availability and relieve the load imbalance of the distributed storage servers. For the popular contents, selective partition [45] and selective replication [23] cause some overheads but achieve considerable benefits. With partition and erasure coding, C2DN [43] reduces the costs of Midgress during server unavailability and relieves the write load imbalance of the SSDs. AccelTCP [28] proposes a hardware-assisted TCP stack architecture to accelerate proxy relaying, but it doesn't reduce link bandwidth. Yoda [13] operates as a packet-level translator and it can not support parallel request-granularity redirection.

Effective network stacks and optimization methods. Effective network stacks and optimization methods are able to greatly improve the performance of QDSR. i10 [17], a remote storage stack based on TCP in the kernel, achieves high performance through efficient batching resources and delayed doorbell. The design of the delayed relaying was inspired by the i10, which greatly reduces CPU overheads of the frontend to achieve higher throughput (Section 5).

AF_XDP [10] provides zero-copy semantics between kernel space and userspace, and is optimized for high performance packet processing. The user-level packet I/O engines, such as netmap [32] and DPDK [18], bypass the kernel and avoid kernel overheads. It can speed up QDSR because QUIC is based on UDP and is implemented in userspace. In [44], Yang proposes a hardware/software co-design of QUIC, which offloads part of the QUIC stack and TLS crypto operation into NIC. This brings some heuristic implications to high-performance QUIC stack implementations which will improve the performance of the backends in QDSR.

9 Conclusion

This paper presents design, implementation, and evaluation of QDSR, an effective L7 load balancing scheme by QUIC and DSR. QDSR achieves high performance and comparable flexibility with the conventional proxy-based scheme by eliminating unnecessary relaying loads without additional latency. By sharing the QUIC connection with multiple RSs, the L7 LB allows responses to be sent directly to the clients. In order to maintaining connection consistency and reduce overhead, the L7 LB orchestrates QUIC connections between multiple RSs. We believe that the implementation of QDSR can significantly reduce the overhead of L7 LB for cloud service providers and improve end-to-end service quality. It also provides guiding inspiration for the construction of server clusters and further work.

Acknowledgements

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06-4, the National Key Research and Development Program of China under grant No. 2022YFB3105000, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

References

[1] Alibaba. Server load balancer, 2022.

[2] Apache. Apache traffic server, 2021.

[3] Tom Barbette, Chen Tang, Haoran Yao, Dejan Kostić, Gerald Q. Maguire Jr., Panagiotis Papadimitratos, and Marco Chiesa. A High-Speed Load-Balancer design with guaranteed Per-Connection-Consistency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 667–683, Santa Clara, CA, February 2020. USENIX Association.

[4] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. The CacheLib caching engine: Design and experiences at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 753–768. USENIX Association, November 2020.

[5] Mike Bishop. Hypertext transfer protocol version 3 (http/3). Technical report, Internet Engineering Task Force (IETF), 2022. RFC 9114.

[6] Tomáš Boros and Ivan Kotuliak. Sdn-based transparent redirection for content delivery networks. In *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pages 373–377, 2019.

[7] David Chou, Tianyin Xu, Kaushik Veeraraghavan, Andrew Newell, Sonia Margulis, Lin Xiao, Pol Mauri Ruiz, Justin Meza, Kiryong Ha, Shruti Padmanabha, Kevin Cole, and Dmitri Perelman. Taiji: Managing global user traffic for large-scale internet services at the edge. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 430–446, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Cloudflare. What is anycast? | how does anycast work?, 2021.

[9] Jonathan Corbet. Tcp connection repair, 2012.

[10] Jonathan Corbet. Accelerating networking with af_xdp, 2018.

[11] Sr. Director. Ip transparency and direct server return with nginx and nginx plus as transparent proxy, 2016.

[12] Martin Duke, Nick Banks, and Christian Huitema. QUIC-LB: Generating Routable QUIC Connection IDs. Internet-Draft draft-ietf-quic-load-balancers-13, Internet Engineering Task Force, March 2022. Work in Progress.

[13] Rohan Gandhi, Y. Charlie Hu, and Ming Zhang. Yoda: A highly available layer-7 load balancer. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Brendan Gregg and other contributors. Frame graph, 2015.

[15] Yutaro Hayakawa, Michio Honda, Douglas Santry, and Lars Eggert. Prism: Proxies without the pain. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 535–549. USENIX Association, April 2021.

[16] Leif Hedstrom. Apache traffic server: More than just a proxy. Boston, MA, December 2011. USENIX Association.

[17] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. TCP $\approx$ RDMA: CPU-efficient remote storage access with i10. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 127–140, Santa Clara, CA, February 2020. USENIX Association.

[18] Intel. Dpdk, 2018.

[19] Jana Iyengar and Ian Swett. Quic loss detection and congestion control. Technical report, Internet Engineering Task Force (IETF), 2021. RFC 9002.

[20] Jana Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. Technical report, Internet Engineering Task Force (IETF), 2021. RFC 9000.

[21] Marios Kogias, Rishabh Iyer, and Edouard Bugnion. Bypassing the load balancer without regrets. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 193–207, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Charles 'Buck' Krasic, Mike Bishop, and Alan Frindell. Qpack: Header compression for http/3. Technical report, Internet Engineering Task Force (IETF), 2022. RFC 9204.

[23] Jialin Li, Jacob Nelson, Ellis Michael, Xin Jin, and Dan R. K. Ports. Pegasus: Tolerating skewed workloads in distributed storage with In-Network coherence directories. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 387–406. USENIX Association, November 2020.

[24] Tong Li, Kai Zheng, Rahul Jadhav, and Jiao Kang. Optimizing ACK mechanism for QUIC. Internet-Draft draft-li-quic-optimizing-ack-in-wlan-04, Internet Engineering Task Force, May 2022. Work in Progress.

[25] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-02, Internet Engineering Task Force, July 2022. Work in Progress.

[26] mahimahi contributors. mahimahi, 2016.

[27] Gal Mendelson, Shay Vargaftik, Dean H. Lorenz, Kathy Barabash, Isaac Keslassy, and Ariel Orda. Load balancing with jet: Just enough tracking for connection consistency. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, page 191–204, New York, NY, USA, 2021. Association for Computing Machinery.

[28] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. AccelTCP: Accelerating network applications with stateful TCP offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 77–92, Santa Clara, CA, February 2020. USENIX Association.

[29] Nginx. Nginx-quic, 2021.

[30] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. Ananta: Cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 207–218, New York, NY, USA, 2013. Association for Computing Machinery.

[31] Carlo Puliafito, Luca Conforti, Antonio Virdis, and Enzo Mingozzi. Server-side quic connection migration to support microservice deployment at the edge. *Pervasive and Mobile Computing*, 83:101580, 2022.

[32] Luigi Rizzo. netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, 2012.

[33] Sanae Rosen, Bo Han, Shuai Hao, Z. Morley Mao, and Feng Qian. Push or request: An investigation of http/2 server push for improving mobile performance. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 459–468, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[34] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 529–544, Santa Clara, CA, February 2020. USENIX Association.

[35] LiteSpeed Tech and lsquic contributors. lsquic, 2023.

[36] Tencent. Cloud load balancer, 2022.

[37] Martin Thomson and Cory Benfield. HTTP/2. Technical report, Internet Engineering Task Force (IETF), 2022. RFC 9113.

[38] Martin Thomson and Sean Turner. Using tls to secure quic. Technical report, Internet Engineering Task Force (IETF), 2021. RFC 9001.

[39] Filippo Valsorda, Thyla van der Merwe, Victor Vasiliev, Hoeteck Wee, Tom Weinstein, David Wong, Christopher A. Wood, Tim Wright, Peter Wu, and Kazu Yamamoto. The transport layer security (tls) protocol version 1.3. Technical report, Internet Engineering Task Force (IETF), 2018. RFC 8446.

[40] Bingyang Wu, Kun Qian, Bo Li, Yunfei Ma, Qi Zhang, Zhigang Jiang, Jiayu Zhao, Dennis Cai, Ennan Zhai, Xuanzhe Liu, et al. Xron: A hybrid elastic cloud overlay network for video conferencing at planetary scale. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 696–709, 2023.

[41] Mengbai Xiao, Viswanathan Swaminathan, Sheng Wei, and Songqing Chen. Evaluating and improving push based video streaming with http/2. In *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '16, New York, NY, USA, 2016. Association for Computing Machinery.

[42] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. From cloud to edge: A first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 37–53, New York, NY, USA, 2021. Association for Computing Machinery.

[43] Juncheng Yang, Anirudh Sabnis, Daniel S. Berger, K. V. Rashmi, and Ramesh K. Sitaraman. C2DN: How to harness erasure codes at the edge for efficient content delivery. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1159–1177, Renton, WA, April 2022. USENIX Association.

[44] Xiangrui Yang, Lars Eggert, Jörg Ott, Steve Uhlig, Zhigang Sun, and Gianni Antichi. Making quic quicker with nic offload. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, EPIQ '20, page 21–27, New York, NY, USA, 2020. Association for Computing Machinery.

[45] Yinghao Yu, Renfei Huang, Wei Wang, Jun Zhang, and Khaled Ben Letaief. Sp-cache: Load-balanced, redundancy-free cluster caching with selective partition. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2018.

[46] Wensong Zhang et al. Linux virtual server for scalable network services. In *Ottawa Linux Symposium*, volume 2000, 2000.

[47] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 418–432, 2021.

A QDSR without packet number space isolation

To achieve better compatibility with existing IETF QUIC standard and avoid modifying clients, we implemented a backward compatible design with a single packet number space by allocating enough packet numbers to each RS when relaying the ACK of the handoff stream, as shown in Figure 14. This design can also solve the problem of transmission fairness caused by each RS independently performing congestion control.However, it brings unwanted out-of-order packet arrivals, which violates the design philosophy of QUIC and may lead to the following potential risks:

- The out-of-order packets confuse the client instead of servers that the network becomes congested, which dramatically increases the frequency of ACKs sent by the client [24].
- Out-of-order packets make it impossible to determine the most recent packet in ACK frames, which invalidates the delay_ack and makes RTT measurements inaccurate.
- The anti-replay window will increase tremendous retransmission, as illustrated in Figure 15. The RS1 and RS2 are allocated a range of packet numbers respectively and sends packets. Due to the heterogeneity of RTTs between them and the client, some packets with larger numbers may arrive earlier than those with smaller numbers. To achieve anti-replay, when the client receives an ACK frame (sent by servers) that acknowledges the packet containing an ACK frame (sent by the client), the vast majority of QUIC implementations will erase those received histories that are below the largest packet number recorded in the ACK frame (sent by the client). After that, the client will reject the old packets, which will induce dramatic retransmissions.

This effect shows in Figure 15 will be more serious when the RTT gap between multiple senders is large, and it also happen in multipath without packet number space isolation.

Packet number space isolation will not bring additional latency to the client. In our implementation, each packet number space states maintained by the client only include received packet history. The less packets are lost or out of order, the lower the memory usage. Multipath and QDSR both have heterogeneous links, which is why both have multiple packet number space isolation.

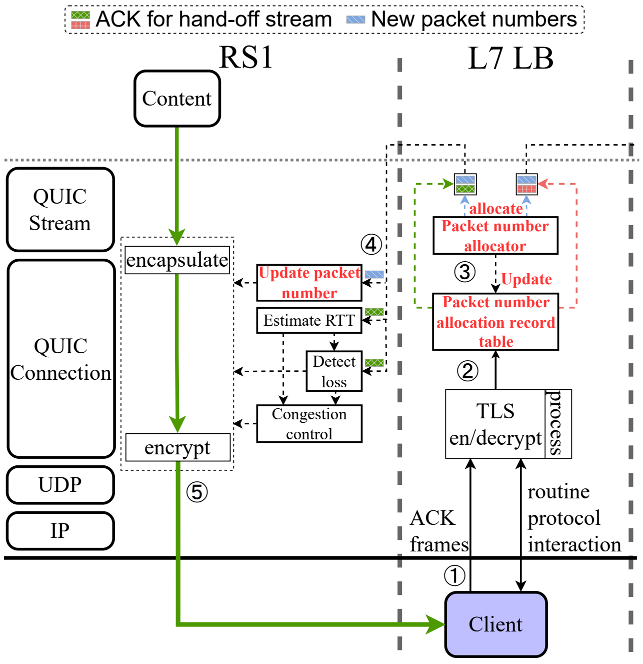


Figure 14: Backwards compatible transmission phase design of QDSR. The bold red font is the difference from the packet number space isolation design.

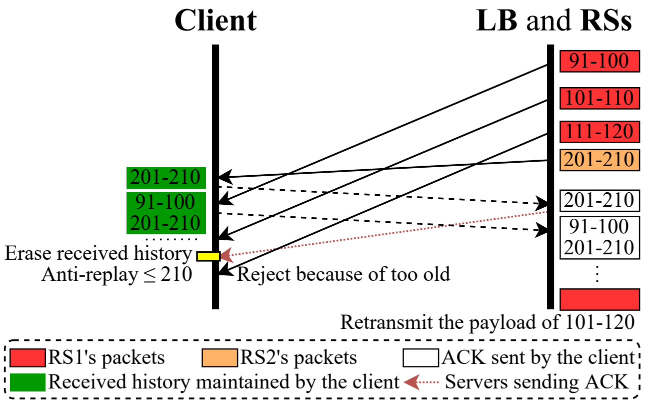


Figure 15: The impact of anti-replay mechanism on shared packet number space for multiple senders