IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-0443-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOMWKSHPS51825.2021.9484596

DCQUIC: Flexible and Reliable Software-defined Data Center Transport

Lizhuang Tan, Wei Su, Yanwen Liu
NGIID
Beijing Jiaotong University
lzhtan; wsu; 19120081@bjtu.edu.cn

Xiaochuan Gao
China Unicom
gaoxc50@chinaunicom.cn

Wei Zhang
Shandong Computer Science Center
National Supercomputer Center in Ji'nan
wzhang@sdas.org

***Abstract*—Numerous innovations based on the data center TCP support the rapid development of data center. However, with changes in topology, scale and traffic patterns, the new requirements of data center on transport protocol are more agile and more reliable. Improving transport performance by patching TCP seems to be facing a bottleneck. We explored the possibility of applying QUIC inside the datacenter networking. This paper proposes a new data center transport scheme based on QUIC, called data center QUIC (DCQUIC). We especially proposed an proactive connection migration mechanism suitable for datacenter networking. Like the efficiency of UDP and the reliability of TCP, DCQUIC exhibits exciting performance and scalability, and may become a potential transport technology to support the development and innovation of data centers in the future.**

1. Introduction

In the past decade, TCP has supported the rapid development of data centers [1]. Numerous network innovations based on TCP continue to emerge, covering transmission architecture [2], congestion control [3], [4], [5], streaming task scheduling [6], [7], TCP acceleration [8], [9], [10], and load balancing [11], [12]. These efforts have significantly improved the efficiency of datacenter networking.

However, with the evolution of datacenter networking topology, scale, and applications [13], there has been a bottleneck in further improving the efficiency [14]. The innovation of datacenter networking faces many challenges, including but not limited to:

1) Have we really discovered the characteristics and changing patterns of datacenter networking traffic? Are these phenomena appearance or substance [15]?
2) Do our innovative schemes follow the principle of "*Great Truths Are Always Simple*"? Are they really easy to deploy?
3) Can our solution support the long-term evolution of datacenter networking in actual deployment, instead of adding burden to it?

In this paper, We analyzed the feasibility of applying the Quick UDP Internet Connections (QUIC) protocol [16] inside the datacenter networking, which we call data center QUIC (DCQUIC). We test the performance improvement of DCQUIC in real datacenter networking, and these results are gratifying. There are three test items. The first is the performance comparison of DCTCP and DCQUIC. The second is the differential packet size transport for long and short flows. The third is connection migration technology to improve transport reliability. We open sourced the first version of DCQUIC [17], which is based on C language and can be directly used by data center applications.

The experimental results show that DCQUIC has advantages in establishing connection speed, flexibility and reliability. The experimental results are as follows:

1) Compared with DCTCP, the transport efficiency of DCQUIC has increased by 49.59%-73.02%.
2) By using a large packet size for short flows and a small packet size for long flows, differential packet size transport of DCQUIC reduces the completion time dropped by 9.17%-63.2%.
3) Connection migration technology can effectively improve transport reliability, even in the case of network failures or virtual machine migration. Compared with the existing application layer switching mechanism, connection migration reduces the task completion time by 67.92%.

2. Existing Work

Numerous TCP innovations have emerged for datacenter networking, such as flow classification, flow-based load balancing, flow priority-based scheduling, etc. Table 1 classifies and summarizes some typical works. It should be noted that in the UDP-based DCQUIC protocol, some [18], [19] of these innovations can be reserved, and some [1], [20], [21], [22], [23] need to be redesigned.

At present, most application-level service frameworks rely on TCP, such as Spring Boot/Spring Cloud, Google gRPC, Thrift, Finagle and Dubbo/Dubbox. These frameworks are responsible for service discovery, load balancing, fault tolerance, network transmission, serialization and other functions to support numerous data center services. Application developers can ignore specific network communication processes when calling services, which are taken over

978-1-6654-0443-3/21/$31.00 ©2021 IEEE

TABLE 1. TYPICAL TCP-BASED INNOVATIONS OF DATA CENTER TRANSMISSION.

Transport Protocol	Typical work
Congestion avoidance	DCTCP [1], D2TCP [20]...
Flow control	PIAS [21], HyFabric [23], ...
Bandwidth allocation	D^3 [18], FCTcon [19], ...
Flow classification	ElasticSketch [22], Memento [24], ...
Flow priority-based scheduling	PIAS [21], HyFabric [23], ...
Flow-based load balancing	Clove [25], IntFlow [26], ...

by remote communication protocols such as RMI, Socket, SOAP (HTTP XML) and REST (HTTP JSON). Most of these remote communication protocols are based on TCP.

The benefits of TCP as data center transport layer protocol are:

1) **Friendly to development**. Some businesses can directly use existing mature development models and communication frameworks, thereby reducing the workload of developers.
2) **Mature congestion control and other mechanisms**. Some congestion control, flow control, and reliability mechanisms that have performed well in other areas can continue to be used in data centers, and they still seem to perform well.

Although these TCP-based frameworks and mechanisms are so popular, and they have tried their best to simplify network transmission costs through long connection and multiplexing, etc., tens of thousands of TCP connections will cause huge overheads in computing, storage and networking. Moreover, the number of TCP connections per second processed by OS is also limited, which becomes a TCP performance bottleneck. Issues that still have room for improvement include:

1) **The overhead is still not small enough**. In datacenter networking, is it really necessary to sacrifice the efficiency of connection establishment and termination in exchange for reliability like WAN?TCP fast open (TFO) [27] can reduce one RTT by exchanging data during TCP handshake, but it still cannot achieve 0RTT. Long connection reduces response time and network congestion, but may harm the overall performance (computing resources and concurrency) of server.
2) **Congestion control is too conservative**. In datacenter networking, do we really need to detect link bottlenecks step by step? When a micro-burst [28] occurs, do we really know the limit of rapid recovery? The TCP-based transport layer is not convenient for us to verify them.
3) **The development and deployment of new features is slow**. Pull one hair and the whole body is affected. The intertwined protocol dependencies make it so difficult to modify a very small module, which is why data center owners would rather tolerate inefficiencies than mistakes.
4) **Patching causes confusion in operation and maintenance**. In order to achieve excellent congestion control, the protocols run on data center hosts and switches have been patched beyond recognition.

3. DCQUIC

In this section, firstly, we give the design goals of DCQUIC. Then, the design of DCQUIC datacenter networking transport system is given. Finally, we respectively discuss some key technologies and characteristics of DCQUIC that can be directly applied to datacenter networking, and improve the connection migration technology in standard QUIC to make it more suitable for datacenter networking.

3.1. Design Goals

The design goal of DCQUIC is to try to replace TCP with UDP in datacenter networking, and to separate and open the original TCP congestion control, flow control and connection mechanisms to support QUIC-based innovation, including:

1) **More efficient than DCTCP**. There are many DCTCP congestion control solutions, but the industry is still using a few of the most primitive and mature solutions. The reason for restricting the deployment of these congestion control schemes is that changes to DCTCP involve OS on both ends. By supporting pluggable development of congestion control, DCQUIC provides the most suitable congestion control algorithm for data center applications.
2) **More reliable than DCUDP**. Although UDP is already very efficient, including connectionless communication. But DCQUIC hopes to further improve transmission reliability, while UDP only considers sending and not receiving. Through mechanisms such as multiplexing and fast retransmission, even if packets are accidentally lost on the network, DCQUIC still will not experience performance collapse.

These innovations will support incremental deployment, from the TCP era, to the coexistence era of TCP and UDP, to the final UDP era. These innovations will make the data plane pay more attention to forwarding efficiency, and overturn some of the existing flow control, congestion control and development models, etc., which will be independent from the Kernel and support the software definition from application developers and administrators.

3.2. Standard QUIC

QUIC is a UDP-based transport protocol designed by Google. HTTP/3 has chosen to use QUIC instead of TCP as its transport layer protocol. QUIC has new features such

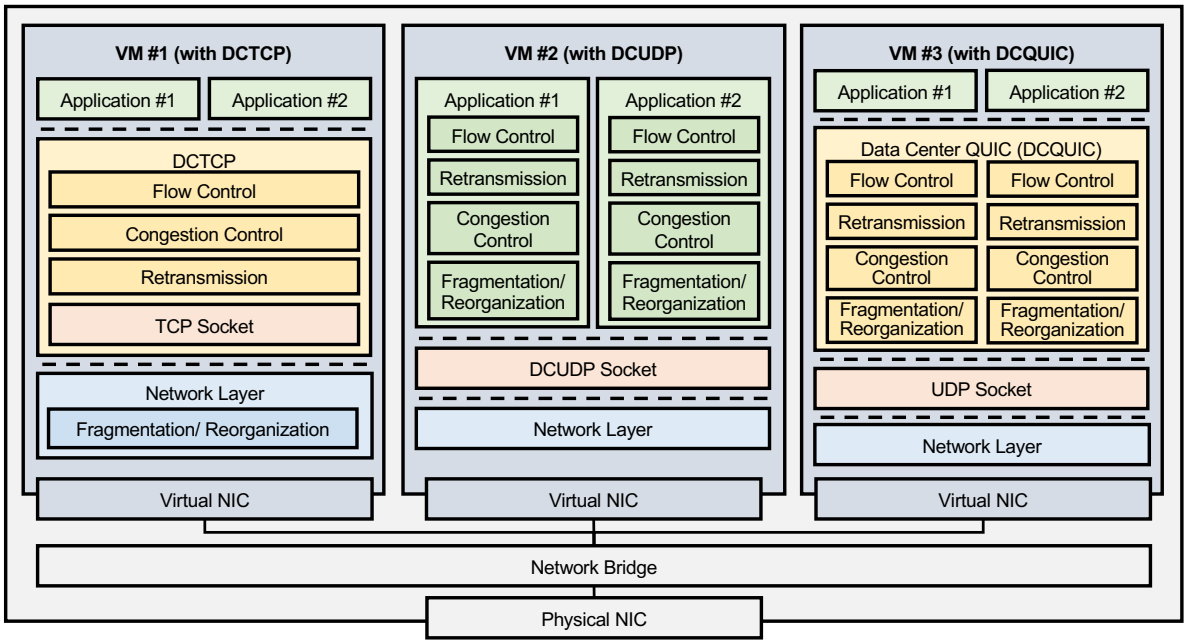

Figure 1. Protocol stack and functional modules of DCTCP, DCUDP and DCQUIC.

as low-latency connections, improved congestion control, multiplexing without head-of-line blocking, forward error correction, and connection migration, which can significantly improve transport efficiency and reliability [29]. In 2016, IETF started standardizing QUIC. In 2018, 35% north-south traffic of Google is QUIC.

3.3. Architecture of DCQUIC

Standard QUIC is suitable for Long Fat Network (LFN), but the datacenter networking is a Short Fat Network (SFN). In DCQUIC, the more important features include flow control, congestion control, retransmission, and connection migration. Since the Kernel is not involved, DCQUIC inherently supports data center owners to develop and deploy self-developed protocols, such as forward error correction and redundant transmission. The packet type and protocol format of DCQUIC are basically similar to QUIC. Compared with the existing QUIC, the research contents of DCQUIC that can be improved are listed in Section 5. As is shown in Figure 1, compared with DCTCP and DCUDP [30], DCQUIC has the following two advantages:

1) The UDP-based DCQUIC protocol stack is simple, with low coupling and high cohesion. It inherits the efficiency of UDP and the reliability of TCP. DCQUIC is implemented directly based on User Mode, rather than Kernel, which can be quickly iteratively updated without changing operating system.
2) DCQUIC supports the modular development of flow control, congestion control, multiplexing and retransmission mechanisms, etc., which will promote a new round of agile deployment of datacenter networking innovations.

We developed a prototype of DCQUIC [17], which is specifically optimized for the datacenter networking on the basis of standard QUIC.

3.4. Connection Migration in DCQUIC

DCQUIC continues to use the connection migration mechanism in QUIC. DCQUIC uses connection ID to identify a connection from the client to the server. The connection migration technology can help the data center realize the dynamic migration of virtual machines at L3/L4. In addition, connection migration also allows servers, VMs or containers to achieve congestion avoidance and redundant transmission, improving the reliability of transmission.

In DCQUIC, connection migration is divided into proactive connection migration and passive connection migration. The former is suitable for multi-NIC servers, VMs or containers to keep connection, and the latter is suitable for client after migration to keep connection.

3.4.1. Proactive Connection Migration. When a drastic increase in network delay or response interruption is detected, the proactive connection migration will promptly switch to other available source IP to continue to maintain existing connection with communication server to avoid service interruption.

Stream State Management maintains stream information, including stream establishment time, end time, and total transmitted bytes. It is expressed as $< StreamID, EstTime, EndTime, TotalBytes >$.

StreamID is a variable unsigned integer number. *EstTime* records the time of stream creation, which is initialized by *Create Stream*. *EndTime* means the end time of the stream, which is the moment when a frame with FIN=1 is received. *TotalBytes* represents the total number of bytes transmitted on one stream, and its function is to normalize and compare streams of different lengths. *TotalBytes* is calculated based on the frame with FIN=1 of each stream. Its calculation equation is:

$$TotalBytes = Offset + DataLength. \quad (1)$$

we can use the stream complete time per byte to evaluate efficiency. If all streams with fixed bytes, we can directly use response time.

$$StreamComplTimePerByte = \frac{(EndTime - EstTime)}{TotalBytes}. \quad (2)$$

For streams with different numbers of bytes, by Eq.2, we can evaluate the completion time of different streams. In addition to the completion time, proactive connection migration also needs to record the mapping relationship between stream and UDP, which can be expressed as $< StreamID, UDPSourceIP >$.

Proactive connection migration evaluates fluctuations in network quality by tracking the completion time of different flows of the same connection. When network fluctuations exceed a certain threshold, proactive connection migration will actively switch NIC to avoid more serious long-term network congestion. $StreamComplTimePerByte(m, n)$ means the per byte completion time from the m-th stream to the n-th stream:

$$StreamComplTimePerByte(m, n) = \frac{\sum_{i=m}^{n} StreamComplTimePerByte(i)}{n - m} \quad (3)$$

Assuming that the long-term observation window is W, the short-term observation window is $w(w < W)$, the latest stream id is n, and the network fluctuation threshold is α, then we believe that when Equation 4 is satisfied, the network performance deteriorates drastically, and the server needs to start proactive connection migration:

$$\frac{StreamComplTimePerByte(n - w, n)}{StreamComplTimePerByte(n - W, n - w)} > \alpha \quad (4)$$

We analyzed the performance of proactive connection migration when the switch fails in Section 4.3.

3.4.2. Passive Connection Migration. After VM is migrated, in addition to L2 technologies such as VXLAN, passive connection migration can continue to use the new source IP address to maintain the existing connection with communication server.

4. Experiment and Practice

In this section, we have completed three experiments to verify the performance of DCQUIC. Firstly, we compared the performance of DCQUIC and DCTCP. Secondly, we verified the possibility of improving transport performance by negotiating the DCQUIC maximum packet size. Thirdly, we verified the effect of maintaining connection through proactive connection migration when switch fails.

4.1. Performance Comparison of DCTCP and DCQUIC

We deployed DCTCP and DCQUIC in private data center, AWS and Tencent Cloud to verify their performance differences in the same business scenario. We measured the difference between the completion time of handshake, 10KB JSON data request, and 100KB JSON data request between DCTCP and DCQUIC during the RPC call. In order to ensure the fairness of result, the handshake time of DCTCP includes the TLS 1.3 handshake time.

Setting Private data center: Server and client are configured with 8-core Intel Xeon CPU E3-1245 V2 @ 3.40GHz, 4G memory and Intel Corporation 82583V Gigabit NIC, running 64-bit Ubuntu 16.04 (4.4.0-187 Kernel). The link bandwidth B is 1Gbps.

AWS: Server and client are two EC2s, with Intel Xeon CPU E5-2676 v3 @ 2.40GHz, 1G memory and virtual NIC, running 64-bit Ubuntu 16.04 (4.4.0-189-generic). The link bandwidth B is 1Gbps.

Tencent Cloud: Server and client are two CVMs, with Intel Xeon CPU E5-26xx v4 @ 2.40GHz, 2G memory and virtual NIC, running 64-bit Ubuntu 16.04 (4.4.0-157-generic). The link bandwidth B is 1Gbps.

All completion times are recorded by clients. For the three measurement items, we have carried out 10000 experiments. The congestion control algorithm is Cubic, and the ECN marking threshold is 20.

Experimental phenomena and analysis As shown in Figure 2, the average single request completion time of DCQUIC is ahead of DCTCP in three measurement items. Compared with DCTCP + TLS 1.3, DCQUIC's handshake completion time is reduced by 12.59ms/13.09ms/14.79ms (˜73.02%/68.96%/68.82%), which means that DCQUIC can establish a connection faster. When the request object size is 10KB, the completion time of DCQUIC (including connection establishment time) is 12.56ms/13.06ms/20.34ms (˜63.70%/60.85%/67.88%) ahead of DCTCP. This shows that the performance difference between DCQUIC and DCTCP in small task transport is mainly the time consumed by the handshake during the connection establishment phase. When the RPC object is 100KB in size, the server needs to split it into about 80 data packets and send them sequentially. DCQUIC is also ahead of DCTCP by 13.59ms/14.11ms/30.60ms (˜50.92%/49.59%/66.23%). The difference between the DCTCP and DCQUIC is no longer as huge as in the first two test projects. This is because as

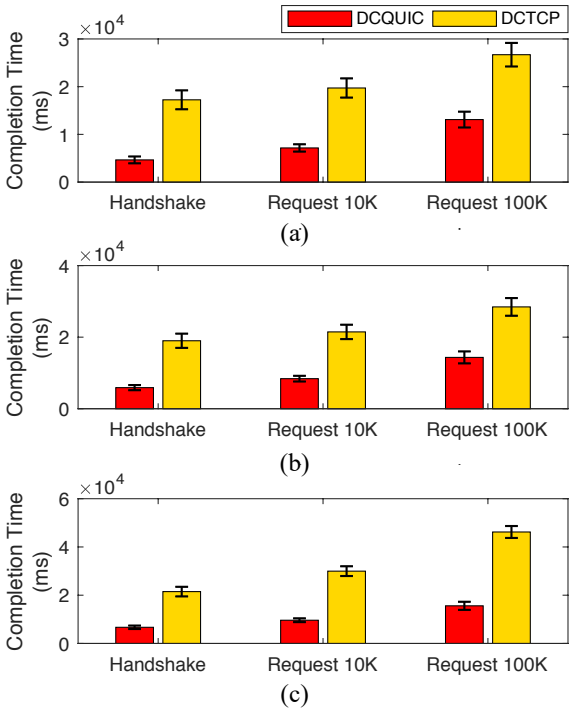

Figure 2. Performance comparison of DCTCP and DCQUIC. (a) Private data center. (b) AWS. (c) Tencent Cloud.

the size of the request object increases, compared with connection establishment, the data transport latency becomes the most important part of the task completion time.

4.2. Improve Transport Efficiency by Negotiating DCQUIC Packet Size

The total length field in UDP header is 2 bytes, so the total length of UDP packet is limited to 65535 bytes, which can fit into an IP packet, making the implementation of the UDP/IP protocol stack very simple and efficient. The maximum payload length in UDP is 65527 bytes. This results in the UDP load ratio up to 99.98%. Although the DCQUIC protocol header and control field need to occupy a part of the UDP data field, the DCQUIC information is necessary and can be defined by data center owner.

Under the TCP/IP architecture of data center, fragmentation is completed by IP protocol in Kernel, which leads to many different types of applications sharing same transport parameters, such as fragment length and send/receive buffer. This fact results in TCP packet size often being set to 1452 bytes. However, existing data centers have already realized the transport support for jumbo frames. For example, many Amazon EC2 instances support 9001 MTU or jumbo frames. We counted 50 of 52 instances support 9001 MTU in Amazon Ningxia region. Based on this fact, we have implemented a mechanism to negotiate packet size.

Firstly, we measured the transport performance of tasks with different scales in different *DCQUIC Maximum Packet Sizes (MPS)* and *UDP Socket buffers*.

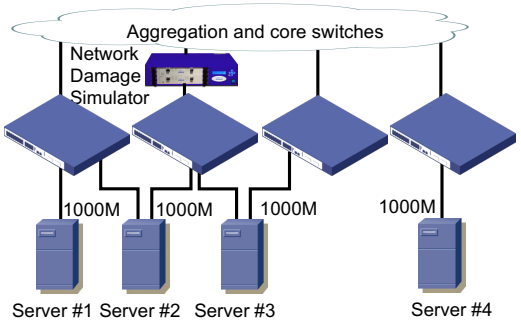

Figure 3. Experimental topology.

Setting Figure 3 shows the small-scale testbed, which consists of 4 servers and 4 switches. Switches are BMv2 Switches, which implement strict FIFO forwarding. Each server is an ADLINK aTCA-8214 blade server, with 8-core Intel Xeon CPU E3-1245 V2 @ 3.40GHz, 4G memory and Intel Corporation 82583V Gigabit NIC, running 64-bit Ubuntu 16.04 (4.4.0-187 Kernel). The link bandwidth B is 1Gbps and background traffic is 800Mbps. Two-way delay D is about 50us. The byte number L of transport tasks ranges from 5K to 10G. The congestion control is Cubic.

Experimental phenomena and analysis We ignore the buffers of the intermediate switches because they can be regarded as *UDP Socket buffer* on the receiving side. The measurement result is shown in Figure 4. In the case of commonly instance configuration (*UDP Socket buffer* = 200K, *QUIC Maximum Packet Size* = 1252B), the actual transport completion time is 1.8X - 9.6X the ideal completion time $T_{ideal} = L/B$. There are many factors that affect transport efficiency, such as MTU, receiver buffer, switch buffer, available bandwidth, congestion control, flow control, etc. We discuss the impact of the first two on transport performance.

For small transfer tasks, the transport bottleneck is fixed connection establishment delay. Especially when transport task $<$ 100KB, the task completion time is significantly reduced by increasing *MPS*.

For large transfer tasks, only increasing the receiving *UDP Socket buffer* or increasing *MPS* often can not achieve the ideal performance. When *MPS* = 9KB and *UDP Socket buffer* = 1MB, the actual transport time is very close to ideal transport time.

As shown in Figure 5, the throughput performance is consistent with the task completion time. We also found that transport performance does not increase linearly with the increase of receiving *UDP Socket buffer* and *MPS*. It is very difficult to find this turning point, but we can give an empirical value for reference. For small task ($L <$ 1MB), *MPS* can be 9KB. For large task ($L \geq$ 1MB), *MPS* can be 5KB, and *UDP Socket buffer* is 1MB. This empirical value is the result of balancing transport efficiency, computational overhead of sending and receiving, and packet loss rate. The setting of *Maximum Packet Size* in DCQUIC can be achieved with only 2 lines of code.

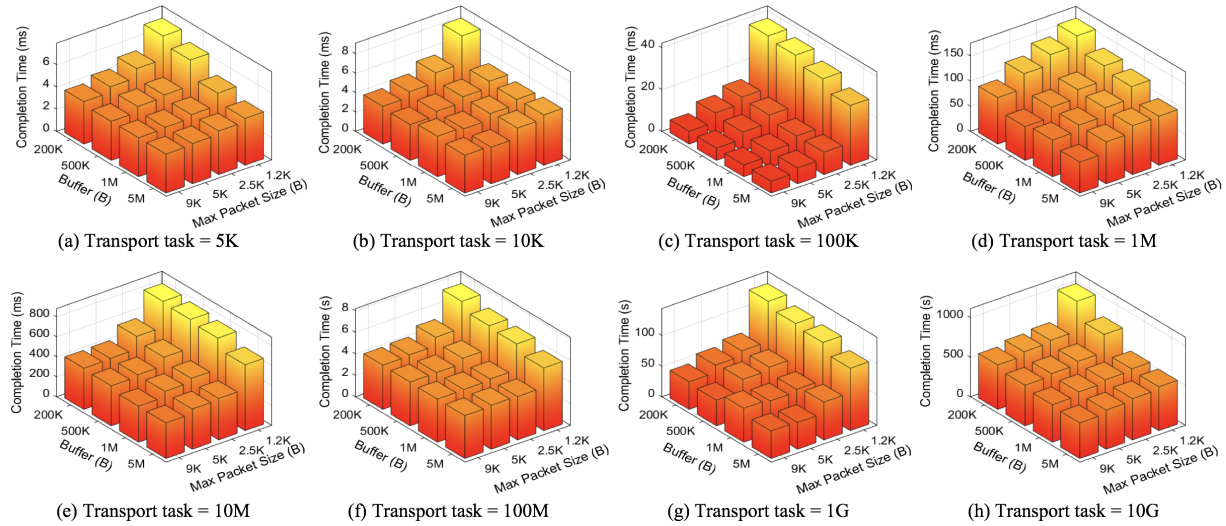

Figure 4. The impact of the maximum packet size and receiver buffer on completion time.

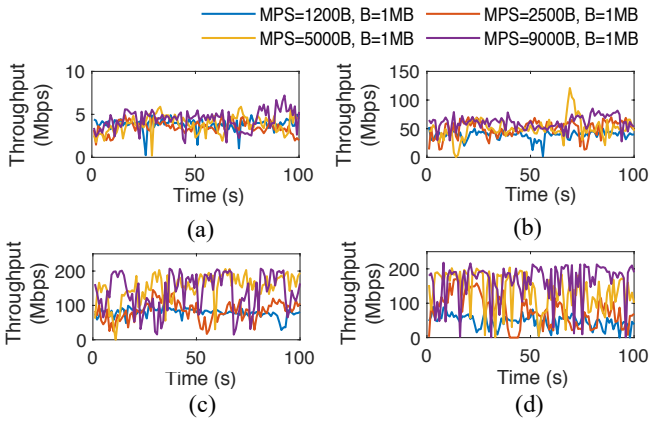

Figure 5. Throughput under different conditions: (a) Task is 10KB. (b) Task is 1MB. (c) Task is 100MB. (d) Task is 1GB.

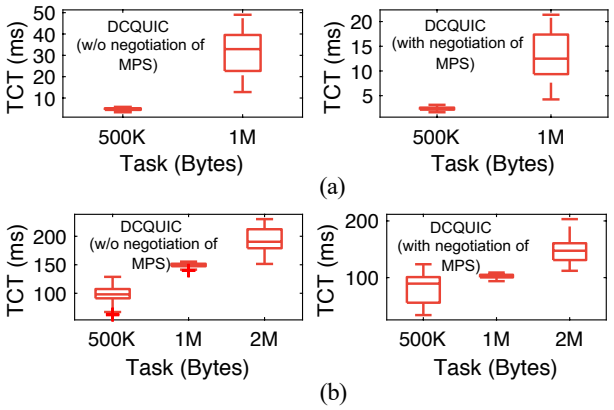

Figure 6. Comparison of task completion time: (a) Web search workload. (b) Data mining workload.

Secondly, we deployed DCQUIC Maximum Packet Size negotiation mechanism on the testbed, which can select the appropriate *MPS* according to the size of task.

Setting We use two common workload traffic: web search workload and data mining workload. Web search traffic is composed of two sizes of file blocks (100B and 10KB), each of which accounts for 50%. The data mining traffic is composed of three sizes of file blocks (0.5MB, 1MB and 2MB), and their proportions are 33.3%. The two workload services are randomly distributed on Server #1 and server #3, and 200 connections coexist during operation. We counted the task completion time (TCT) under the two transport schemes. The first is *UDP Socket buffer* = 200KB and *MPS* = 1252B. The second is *UDP Socket buffer* = 1MB and *MPS* = 5KB (if $L \geq$ 1MB) or 9KB (if $L <$ 1MB).

Experimental phenomena and analysis Figure 6 indicates the TCT results of different scenarios. We can get three conclusions:

1) In web search workload and data mining workload scenarios, negotiating the *DCQUIC Maximum Packet Size* is an effective way to reduce task completion time. For transport tasks with different sizes, Average TCT dropped by 9.17%-63.2%.
2) In a datacenter networking dominated by small task, e.g. web search services, the TCT is reduced more significantly by increasing *DCQUIC Maximum Packet Size*.
3) Due to the larger *DCQUIC Maximum Packet Size* used in the transport of small task, it has not fallen behind in the process of competing with large task. This indicates that the performance degradation of small task caused by increasing *DCQUIC Maximum Packet Size* of large task can be solved by increasing *DCQUIC Maximum Packet Size* of small task more radically. Therefore, DCQUIC is fair and friendly to both large and small task.

4.3. The Performance of Proactive Connection Migration in DCQUIC

In datacenter networking, network equipment failure is a normal phenomenon. Among all network device failures, switch failures are the dominant type in terms of both downtime. The data center can effectively avoid network service interruption caused by switch failure by configuring multi-NIC servers and using multi-path topologies. However, even though these methods have tried to reduce the impact of switch failures, the traditional datacenter networking with DCTCP still inevitably experiences short interruptions. These interruption times are mainly composed of the application server/client detecting the interruption and re-establishing the connection, etc. In this section, we try to use DCQUIC to solve the problem of service unavailability caused by the intermediate switch failure from client-side, so that client can smoothly use multiple NICs to maintain the connection with the server [31].

Setting In the topology shown in Figure 3, Server #1, Server #2 and Server #3 are three RPC clients, respectively call an RPC service on Server #4. Server #1 and Server #2 use TCP to transmit RPC object. When Server #2 detects that the network is unavailable, it will continue to request services by switching IP at application layer. Server #3 uses the RPC service framework based on DCQUIC, and uses proactive connection migration to continue to maintain this connection by changing the source IP/Port. We use a network damage simulator to increase forward delay to simulate switch congestion and failure. Some experimental parameters are set to $W = 50, w = 10, \alpha = 2$. The request object size is 100KB.

Experimental phenomena and analysis As shown in Figure 7, in 200th to 400th epochs, the network damage simulator increases the one-way delay by 100ms, which means that congestion causes delay to increase; in 500th to 700th epochs, it increases the packet loss rate to 100%, which means that the switch #2 fails. Server #2 has been requesting server #4 through switch #2, and even if the delay increases, server #2 still does not trigger link switch at the application layer. After the server #3 experiences a short delay increase, it switches to the path represented by the switch #3, thereby avoiding a more serious continuous delay increase. Server #3 smoothly migrated connections with poor network quality through proactive connection migration. In about the 500th epoch, server #2 re-establishes a new connection with server #4 through switch #1 after continuous packet loss and retransmission. In all 1000 epochs of request, DCQUIC reduced the task completion time by 67.92%.

5. Some Open Issues

DCQUIC may be a future data center transport solution, it has many key technologies that need to be overcome.

DCQUIC Offload The CPU and bus resources that DCQUIC/UDP/IP needs to consume during data transport need to be measured and evaluated, including crypto, connection establishment & teardown, packet reordering, and packet header formatting. It seems a promising direction to accelerate DCQUIC through DCQUIC offloading of hardware/software co-design [32].

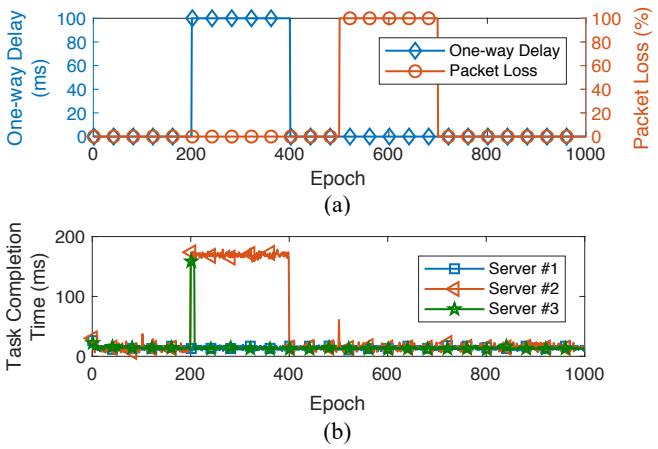

Figure 7. Experimental results of maintaining connections through connection migration. (a) Delay and packet loss rate of network damage simulator. (b)Task completion time of three RPC clients.

Modular DCQUIC DCQUIC inherently supports arbitrary expansion at the beginning of design, including function development and protocol expansion. In fact, our first prototype [17] also follows this principle.

Congestion Control Hybrid Deployment In order to ensure the fairness of all flows, the same congestion control mechanism is often used inside data center [33]. Since DCQUIC supports modular congestion control, in datacenter networking with mixed deployment of TCP and UDP and mixed deployment of different congestion control [34], it is necessary to study the efficiency [35] and fairness among different tenants and applications.

6. Conclusion

In this paper, we proposed a new data center transport scheme DCQUIC. DCQUIC retains some features of QUIC, and supports extensive scalability and deployability. DCQUIC not only brings intuitive performance improvements, but also supports numerous future innovative solutions.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant No.2018YFB1800305, the National Natural Science Foundation of China under Grant No. 61802233 and the Natural Science Foundation of Shandong Provincial under Grant No. ZR2019LZH013. (Corresponding Author: Wei Zhang)

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *SIGCOMM '10*. ACM, Aug. 2010, pp. 63–74, https://doi.org/10.1145/1851182.1851192.

[2] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance," in *SIGCOMM '17*. ACM, Aug. 2017, pp. 29–42, https://doi.org/10.1145/3098822.3098825.

[3] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *NSDI '16*. USENIX Association, Mar. 2016, pp. 537–549, https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/bai.

[4] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC vivace: Online-learning congestion control," in *NSDI '18*. Renton, WA: USENIX Association, Apr. 2018, pp. 343–356, https://www.usenix.org/conference/nsdi18/presentation/dong.

[5] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High Precision Congestion Control," in *SIGCOMM '19*. ACM, Aug 2019, pp. 44–58, https://doi.org/10.1145/3341302.3342085.

[6] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *SIGCOMM '18*. ACM, 2018, pp. 191–205, https://doi.org/10.1145/3230543.3230551.

[7] K. Kaffes, T. Chong, J. T. Humphries, A. Belay, D. Mazières, and C. Kozyrakis, "Shinjuku: Preemptive Scheduling for μsecond-scale Tail Latency," in *NSDI '19*. Boston, MA: USENIX Association, Feb. 2019, pp. 345–360, https://www.usenix.org/conference/nsdi19/presentation/kaffes.

[8] B. Li, T. Cui, Z. Wang, W. Bai, and L. Zhang, "Socksdirect: Datacenter Sockets Can Be Fast and Compatible," in *SIGCOMM '19*. ACM, Aug. 2019, pp. 90–103, https://doi.org/10.1145/3341302.3342071.

[9] Y. Moon, S. Lee, M. A. Jamshed, and K. Park, "AccelTCP: Accelerating Network Applications with Stateful TCP Offloading ," in *NSDI '20*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 77–92, https://www.usenix.org/conference/nsdi20/presentation/moon.

[10] J. Hwang, Q. Cai, A. Tang, and R. Agarwal, "TCP $\approx$ RDMA: CPU-efficient Remote Storage Access with i10 ," in *NSDI '20*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 127–140, https://www.usenix.org/conference/nsdi20/presentation/hwang.

[11] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro Load Balancing for Low-Latency Data Center Networks," in *SIGCOMM '17*. ACM, 2017, pp. 225–238, https://doi.org/10.1145/3098822.3098839.

[12] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient Datacenter Load Balancing in the Wild," in *SIGCOMM '17*. ACM, 2017, pp. 253–266, https://doi.org/10.1145/3098822.3098841.

[13] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving *et al.*, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," in *SIGCOMM '15*. London, United Kingdom: ACM, 2015, pp. 183–197, https://doi.org/10.1145/2829988.2787508.

[14] C. Raiciu and G. Antichi, "NDP: Rethinking Datacenter Networks and Stacks Two Years After," *SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 5, pp. 112–114, 2019, https://doi.org/10.1145/3371934.3371968.

[15] A. Akella, T. Benson, B. Chandrasekaran, C. Huang, B. Maggs, and D. Maltz, "A Universal Approach to Data Center Network Design," in *ICDCN '15*. Goa, India: ACM, 2015, pp. 1–10, https://doi.org/10.1145/2684464.2684505.

[16] A. Langley, J. Iyengar, J. Bailey, J. Dorfman, and I. Swett, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *SIGCOMM '17*, Los Angeles, CA, USA, Aug 2017, pp. 183–196, https://doi.org/10.1145/3098822.3098842.

[17] X. Gao, "libgquic," 2020, https://github.com/Gscienty/libgquic.

[18] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *SIGCOMM '11*. Toronto, Ontario, Canada: ACM, August 2011, pp. 50–61, https://doi.org/10.1145/2018436.2018443.

[19] K. Zheng, Y. Bai, and X. Wang, "FCTcon: Dynamic Control of Flow Completion Time in Data Center Networks for Power Efficiency," *IEEE Transactions on Cloud Computing*, 2019, 10.1109/TCC.2019.2912969.

[20] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-Aware Datacenter Tcp (D2TCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, Aug. 2012, https://doi.org/10.1145/2377677.2377709.

[21] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *NSDI '15*. Oakland, CA: USENIX Association, May 2015, pp. 455–468, https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/bai.

[22] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *SIGCOMM '18*. Budapest, Hungary: ACM, Aug 2018, pp. 561–575, https://doi.org/10.1145/3230543.3230544.

[23] J. Bao, D. Dong, B. Zhao, and S. Huang, "HyFabric: Minimizing FCT in Optical and Electrical Hybrid Data Center Networks," in *SIGCOMM Posters and Demos '19*. Beijing, China: ACM, 2019, pp. 57–59, https://doi.org/10.1145/3342280.3342306.

[24] R. B. Basat, G. Einziger, I. Keslassy, A. Orda, S. Vargaftik, and E. Waisbard, "Memento: Making Sliding Windows Efficient for Heavy Hitters," in *CoNEXT '18*. Heraklion, Greece: ACM, 2018, pp. 254–266, https://doi.org/10.1145/3281411.3281427.

[25] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-Aware Load Balancing at the Virtual Edge," in *CoNEXT '17*. Incheon, Republic of Korea: ACM, 2017, pp. 323–335, https://doi.org/10.1145/3143361.3143401.

[26] Q. Shi, F. Wang, and D. Feng, "Intflow: Integrating per-packet and per-flowlet switching strategy for load balancing in datacenter networks," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020, https://doi.org/10.1109/TNSM.2020.2990868.

[27] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "Tcp fast open," in *CoNEXT '11*. Tokyo, Japan: ACM, 2011, pp. 1–12, https://doi.org/10.1145/2079296.2079317.

[28] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and Mitigating Micro-Burst Traffic in Data Center Networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 98–111, 2019, https://doi.org/10.1109/TNET.2019.2953793.

[29] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with QUIC: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017, https://doi.org/10.1109/MIC.2017.44.

[30] L. Ye, L. Mhamdi, and M. Hamdi, "Efficient udp-based congestion aware transport for data center traffic," in *CloudNet '14*, Luxembourg, Luxembourg, Oct 2014, pp. 46–51, https://doi.org/10.1109/CloudNet.2014.6968967.

[31] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and Evaluation," in *CoNEXT '17*. Incheon, Republic of Korea: ACM, Nov 2017, pp. 160–166, https://doi.org/10.1145/3143361.3143370.

[32] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making QUIC Quicker With NIC Offload," in *EPIQ '20*. Virtual Event, USA: ACM, 2020, pp. 21–27, https://doi.org/10.1145/3405796.3405827.

[33] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat, "Swift: Delay is Simple and Effective for Congestion Control in the Datacenter," in *SIGCOMM '20*. Virtual Event, USA: ACM, 2020, pp. 514–528, https://doi.org/10.1145/3387514.3406591.

[34] R. Al-Saadi, G. Armitage, J. But, and P. Branch, "A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3609–3638, 2019.

[35] T. A. N. Nguyen, S. Gangadhar, and J. P. G. Sterbenz, "Performance evaluation of tcp congestion control algorithms in data center networks," in *CFI '16*. Nanjing, China: ACM, June 2016, pp. 21–28, https://doi.org/10.1145/2935663.2935669.