A Practical SDN-based Bicasting Method for Mitigating HOL Blocking in VPN over TCP

Iou Takeuchi*, Rei Nakagawa†, Tomoya Kawana‡, Nariyoshi Yamai§
Department of Electrical Engineering and Computer Science
Tokyo University of Agriculture and Technology
Tokyo Japan
Email: *s240919t@st.go.tuat.ac.jp, †rnakagawa@go.tuat.ac.jp, ‡s249086t@st.go.tuat.ac.jp, §nyamai@cc.tuat.ac.jp

***Abstract*—With the rise of remote work and increasing security concerns, VPN usage has expanded. VPNs typically rely on UDP or TCP tunnels; UDP is preferred for performance, whereas TCP is favored when the use of UDP is restricted or unstable. TCP-based VPNs in wireless networks suffer from Head-of-Line (HOL) blocking, where a lost packet delays subsequent packets, increasing latency and reducing throughput. To address this issue, we propose an SDN-based multicasting method for multi-homed environments that operates without requiring modification of the operating system. SDN enables flexible path control, and bicasting improves reliability while mitigating HOL blocking by sending duplicated packets over an alternative path. According to simulation results under both bidirectional and client-side unidirectional bicasting, the former reduces RTT up to 80% along with throughput improvement six times compared with that of unicast, and the latter reduces RTT up to 20% along with throughput improvement up to 8%. These results demonstrate the effectiveness of our proposed methods in poor network conditions.**

***Index Terms*—Multi-homed Network, Bicasting, SDN, Head-of-Line Blocking**

I. INTRODUCTION

Despite the recent resurgence of office-based work, 58% of employees still have the option to work remotely [1], making it an integral part of the modern workplace. In such contexts, users frequently connect to external wireless networks (e.g., public Wi-Fi or shared workspaces), which pose security threats such as eavesdropping and data tampering. To address these issues, Virtual Private Networks (VPNs) [2] are employed extensively.

VPNs typically use UDP or TCP for tunneling. UDP tunneling is lightweight, has small headers, and reduces latency and overhead compared with TCP [3], making it the default or recommended method in modern implementations(e.g., OpenVPN and WireGuard). However, corporate networks and government agencies often block UDP and non-standard ports, forcing users to use TCP tunneling. Furthermore, wireless environments can experience increased packet loss owing to hidden terminal problems [4] and other issues, and the lack of UDP retransmission control can significantly reduce throughput. In contrast, the TCP reliability mechanism provides an advantage over UDP in poor network environments, achieving a stable throughput, as shown in Fig. 1.

However, TCP tunnels suffer from Head-of-Line Blocking (HOLB) [5] issues owing to their reliability mechanisms. When a single packet is lost, the subsequent data delivery

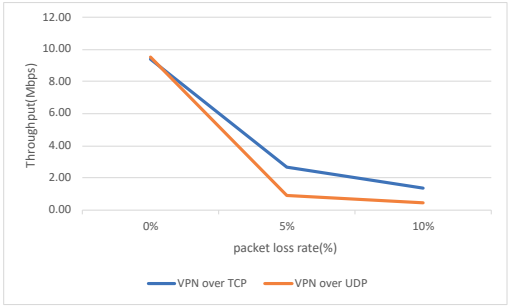


Fig. 1: Impact of Packet Loss on VPN Throughput: TCP vs. UDP Tunneling

is halted, and the delays propagate until the retransmission is complete. The impact of HOLB is particularly pronounced in high-RTT environments and wireless networks with high loss rates, which severely degrades the performance of VPN over TCP. Although some existing studies have proposed methods for mitigating performance degradation such as per-flow reconfiguration using PFRAS [6] and utilization of Multipath TCP (MPTCP) with multiple underlying TCP connections) [7], these approaches require OS kernel modifications or the introduction of dedicated protocols, making their practical implementation challenging. QUIC [8] has attracted attention as a new transport to avoid HOLB; however, its adoption rate is limited compared to HTTP/3's adoption rate of 34.6% [9]. Furthermore, since it is based on UDP, it cannot avoid UDP blocking and UDP-specific vulnerabilities in corporate networks.

To address these issues, an effective approach is to reduce the probability of HOLB by mitigating packet loss. In particular, in multihomed environments, where multiple communication paths can be used simultaneously, using different network paths to increase redundancy and reliability is effective. Specifically, bicasting, which allows the simultaneous transmission of the same packet over multiple routes, improves loss tolerance since it only requires successful packet delivery via at least one route.

Based on these findings, this paper proposes a software-based bicasting method to mitigate the performance degradation caused by the HOLB in a VPN over TCP. Because the proposed method is a purely software-based implementation that utilizes the Software-Defined Networking (SDN) [10]

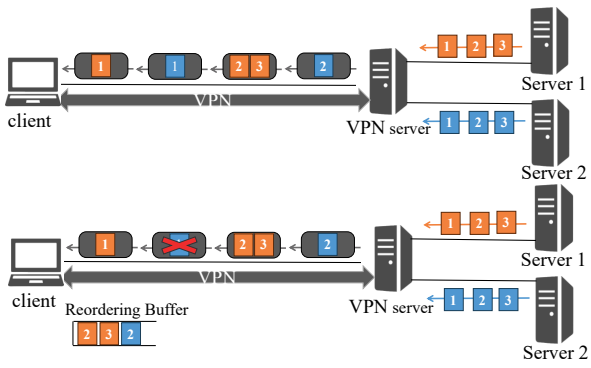


Fig. 2: HOLB in VPN over TCP

architecture and the flexible packet handling mechanism of Open vSwitch (OVS) [11], it does not require OS kernel modifications or the introduction of dedicated protocols, and can be applied without changing the existing VPN applications or TCP stacks.

The contributions of this study are as follows:

- **Software-based Bicasting Method for HOLB Mitigation:** By leveraging Software Defined Networking (SDN) and Open vSwitch (OVS), bicasting in multi-homed environments can be implemented entirely in software, without requiring any modifications to the operating system.
- **Bidirectional Bicasting and Unidirectional Bicasting:** To investigate the bicasting performance and its associated network overhead, two approaches are considered: bidirectional bicasting, applied in both the client–server and server–client directions, and client-side unidirectional bicasting, applied only in the client-to-server direction.
- **Empirical Validation:** Experimental evaluation using OpenVPN on real devices demonstrated that the proposed bidirectional bicasting substantially mitigates HOL blocking in VPN over TCP communication. Furthermore, client-side unidirectional bicasting improves performance under similar network conditions, indicating its potential for effective HOLB mitigation with reduced redundancy.

II. Background and Related Work

A. SDN

Software-defined networking (SDN) [10] virtualizes and centrally controls network devices such as routers through software. Unlike conventional networks, in which the control and data planes are integrated and configured per device, SDN separates these planes, enabling centralized and flexible management without modifying the host kernels.

The use of Open vSwitch (OVS) [11] and OpenFlow allows for dynamic path control and software-based deployment. SDN has been widely adopted in practical systems, such as data centers [12] and commercial routers [13], demonstrating its low overhead and applicability in real-world environments.

B. HOLB in VPN over TCP

Head-of-Line Blocking occurs when the first packet in a queue is delayed, preventing subsequent packets from being processed. This is a well-known issue at both the application and transport layers, although it is caused by different mechanisms of occurrence. At the application layer, HTTP/1.1 processes requests sequentially over a single TCP connection; therefore, a slow response blocks all the subsequent requests. This is known as the HOLB at the HTTP level.

HTTP/2 [14] mitigates this by multiplexing multiple streams over a single TCP connection. However, HOLB still occurs at the transport layer: if a packet is lost, TCP halts the processing of all streams until the missing packet is retransmitted.

Therefore, HOLB occurs in the VPN over TCP. VPNs use tunneling to prevent data from being sent and received from being viewed from the outside. By tunneling, packets sent from different servers are multiplexed within the same VPN tunnel and pass through a single route. Fig. 2 shows a case in which a HOLB occurs downstream of the VPN connection. Even if data orange 1 sent from server 1 reaches the client, if data blue 1 sent from server 2 are lost in the network, data orange 2 and 3 of the VPN client will remain in the queue until the retransmission is completed, and the processing of data orange will be kept waiting until the retransmission of data blue 1 is completed. Similarly, upstream ACK loss also causes HOLB.

C. Related work for mitigation of HOLB

Shikama et al. proposed the Per-Flow ReAssembling and reSequencing (PFRAS) method [6] to reduce HOLB when multiple flows are aggregated into a single TCP connection. In this method, flow identifiers and sequence information are added to packets, and reassembly is performed per-flow at the receiver using the ”urgent pointer” and ”frame pointer” in the TCP header. Although experiments have shown that this mitigates HOLB, it requires TCP stack modification, making it challenging to implement.

Another approach is to leverage Multipath TCP (MPTCP), an extension of TCP designed for multihomed environments. The redundant scheduler of the MPTCP can transmit identical data across multiple interfaces, thereby mitigating the HOLB by compensating for packet loss. However, its dependence on kernel-level support restricts its compatibility with limited operating systems, thereby reducing its practicality in diverse deployment scenarios.

QUIC, a UDP-based transport protocol developed by Google, mitigates HOL blocking by multiplexing streams with independent flow control, preventing loss in one stream from affecting others [8]. However, as a relatively new technology, it is not yet fully supported by the network infrastructure and servers. Because it runs on UDP, it still faces issues such as security concerns and communication blocking. Moreover, the adoption rate of the related HTTP/3 protocol remains limited to approximately 34.6% [9], restricting its practical deployment.

III. A Practical SDN-based Bicasting Approach for HOLB Mitigation

A. Proposed Architecture

This study proposes a practical and easily deployable method for mitigating HOLB in VPN over TCP, even under

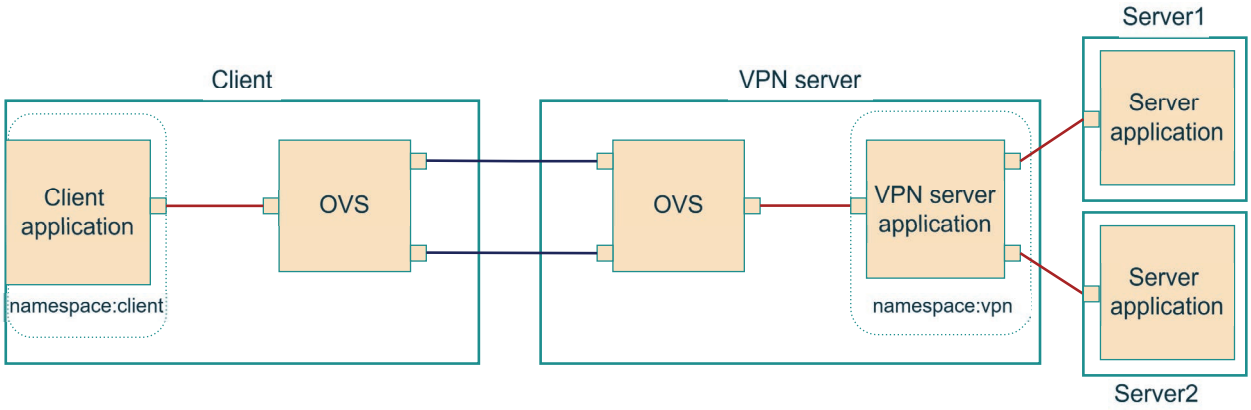


Fig. 3: Proposed network architecture

high packet loss conditions. Fig. 3 illustrates the proposed architecture, where the client is configured as a multi-homed client with two interfaces that are connected to the VPN server. The client–VPN server link is assumed to be a wireless connection prone to packet loss

The bicasting functionality is implemented using OVS, which is deployed directly beneath both client and VPN servers. The SDN controller dynamically manages the forwarding rules to realize bicasting over multiple interfaces. OVS is a flexible user-space software switch that allows fine-grained control of packet forwarding behavior and is easy to install and configure in a virtual environment.

In the experimental environment, Linux namespaces [15] were employed to isolate the network stacks of both the VPN client and OVS, enabling their co-location on the same host machine.

In this method, the architecture incorporates an SDN-based bicasting function into a multi-homed network environment. Bicasting refers to the simultaneous transmission of identical packets over multiple network paths. This redundancy ensures successful delivery as long as packet loss does not occur on all paths simultaneously, effectively reducing the packet loss and mitigating the HOLB. Although bicasting improves reliability, it also increases the number of transmitted packets, which may introduce an additional overhead. Thus, two approaches with different overhead trade-offs are considered:

- **Bidirectional bicasting:** Packets are bicasting in both directions—client to VPN server and VPN server to client.
- **Client-side unidirectional bicasting:** Only upstream packets (from client to VPN server). Client-side unidirectional bicasting reduces network overhead by bicasting only small packets, such as ACKs.

In this study, bicasting only from the server side was not employed because the data packets sent from the server to the client were relatively large, and server-side bicasting would increase the network load, which is undesirable.

This method is entirely software-based and can be implemented by simply installing and configuring OVS. It is highly portable and easy to integrate into existing VPN environments because it does not require any changes to the operating system.

Section III-B describes how the application and OVS are integrated into a single host using the namespaces. Finally, Section III-C provides a detailed explanation of the bicasting implementation.

B. Namespace integrated host

In the proposed network environment, as illustrated in Fig. 3, namespaces are used and OVS is placed on the client and VPN server. A Namespace is a mechanism that provides an independent resource space for each process. For example, using network namespaces, a virtual network environment can be built that allows communication with the host using virtual NICs. This allows the client to coexist with the OVS device without introducing new OVS equipment and can be implemented only by installing the software. Virtual interfaces are used to connect the client application and OVS on the client side, and the OVS and VPN server application on the VPN server side.

C. Bicasting function using SDN

Fig. 4 presents an overview of the system architecture for implementing bicasting using SDN. The mechanism for realizing bicasting with OpenFlow and Open vSwitch (OVS) is described as follows:

The OpenFlow switches correspond to the OVS instances shown in Fig. 3. OpenFlow switches are installed on both the client and VPN servers, and are interconnected through two network paths. Packets sent from the client (or VPN server) are forwarded to the counterpart through the OpenFlow switches. The OVS is used as the OpenFlow switch, and it is connected to an OpenFlow controller that performs control, and bicasting is performed by changing flow entries.

A group table function is used to realize bicasting in OpenFlow. The group table enables a single flow entry to specify multiple output ports, thereby allowing packets to be transmitted to multiple destinations simultaneously. Table I lists the group table configurations used in the implementation.

TABLE I: GROUP TABLE OF OpenFlow

Group ID	Type	Action Bucket
1	all	send to client port1 via port 1
		send to client port2 via port 2

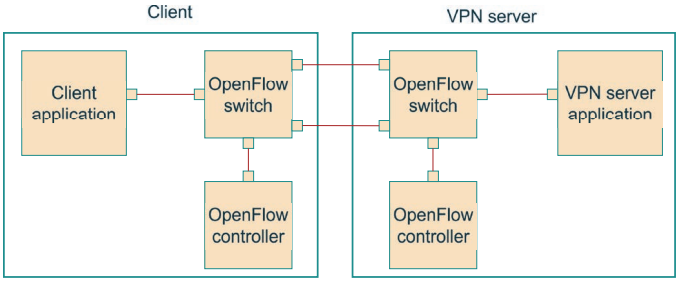


Fig. 4: Overview of Bidirectional Communication via Bicasting

Whereas a general flow table forwards packets to a single output port, the group table enables concurrent forwarding (bicasting) to multiple ports, thereby achieving redundancy and reliability of the system.

Fig. 5 illustrates the packet flow when bicasting occurs from the VPN server to the client. Packets sent from server1 and server2 are sent to the OVS via the network interface (vpn-veth → ovs-veth) using the VPN server application on the namespace vpn. The OVS replicates each packet and concurrently transmits the identical copies to the client's port1 and port2. Conversely, Fig. 6 illustrates the flow of packets when bicasting from the client to the VPN server. Packets sent from the client application are sent to the OVS via the network interface (client-veth → ovs-veth), duplicated by the OVS, and then sent to VPN server's port1 and port2. Bidirectional bicasting performs bicasting in both downstream and upstream directions. Downstream bicasting reduces the HOLB caused by data packet loss, whereas upstream bicasting mitigates the HOLB caused by ACK packet loss. In contrast, client-side unidirectional bicasting is a method of performing bicasting only in the upstream direction. A challenge in downstream bicasting is the increased communication overhead. To address this, HOL blocking can be reduced while suppressing additional traffic by restricting bicasting to the upstream direction.

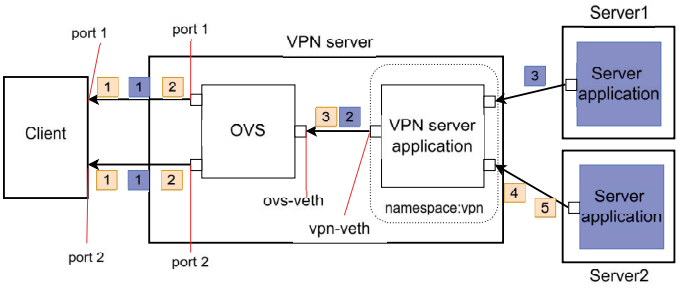


Fig. 5: Packet flow during bicasting from server to client

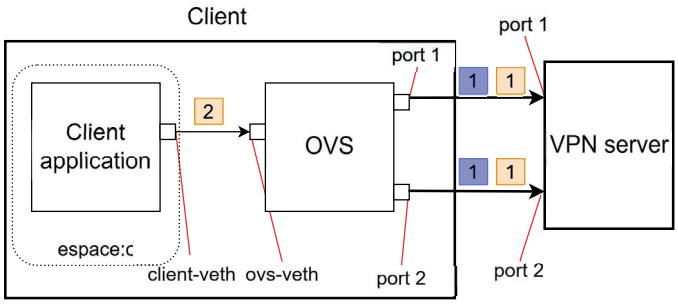


Fig. 6: Packet flow during bicasting from client to server

TABLE II: HARDWARE SPECIFICATIONS OF FIGURE3

	Client	VPNserver
CPU	Intel Core i7-8550U @ 1.80GHz	Intel Core i5-1035G1 @ 1.00Ghz
Memory	4GB×2 DDR4 2400MHz	8GB DDR4 3200MHz
OS	Ubuntu 22.04.6 LTS	Ubuntu 22.04.6 LTS

IV. Experimentation, Results and Analysis

A. Test Environment

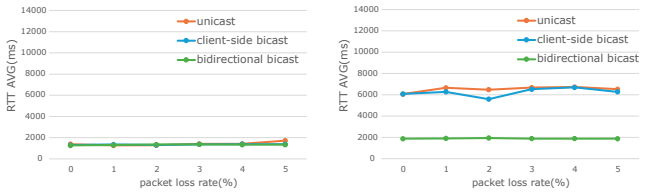
The experimental environment was implemented on real hardware based on the proposed architecture, as shown in Fig. 3. The specifications of the client and VPN server are summarized in Table II, and a Raspberry Pi 3 was used as the content server for the test. Both the client-VPN and VPN server-content server links were configured as wired connections.

To emulate the characteristics of wireless communication over wired links, the Linux Traffic Control (TC) tool [16] was used. Using the TC, link parameters such as the bandwidth, delay, and packet loss rate were adjusted according to the settings listed in Table III. This configuration allowed for the controlled reproduction of various wireless conditions while maintaining the experimental reproducibility. Open vSwitch (OVS) was installed on both the client and VPN server hosts, and packet forwarding was managed using the OpenFlow protocol. OpenVPN (version 2.5.1) was used to establish VPN tunnels between the client and the VPN server. The tunnels were configured in TCP mode by modifying the OpenVPN configuration files. The overall network configuration during the experiments consisted of the aforementioned components and settings, ensuring a controlled environment for evaluating the proposed bicasting mechanism under various network conditions.

- On the client, the OVS and the client application are executed in different network namespaces, and communication is performed to the VPN server via OVS. Similarly, on the VPN server, the OVS and VPN server applications were executed in different namespaces.
- Establish a VPN connection between the client application and the VPN server application, and perform data communication through the VPN tunnel.

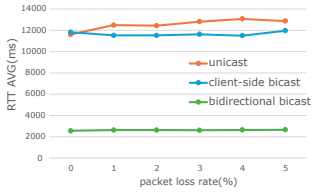
B. Test Method

The experimental environment described in the previous subsection was used to evaluate the performance of the proposed bicasting mechanism. Using this setup, the following

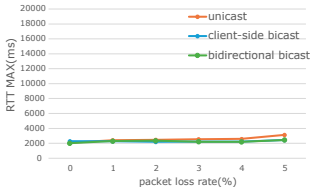


(a) RTT AVG with 0% server-side loss

(b) RTT AVG with 5% server-side loss

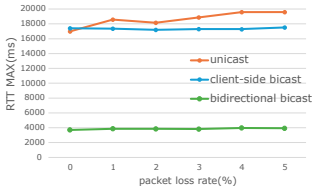


(c) RTT AVG with 10% server-side loss



(d) RTT MAX with 0% server-side loss

(e) RTT MAX with 5% server-side loss



(f) RTT MAX with 10% server-side loss

Fig. 7: Experiment RTT Results

TABLE III: Communication link setup for the experiment

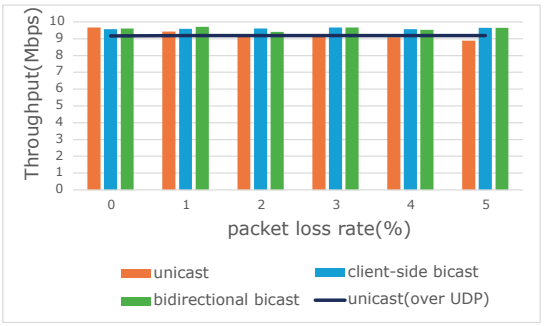
	Client-VPNserver	VPNserver-Server1, 2
RTT (ms)	10	10
Bandwidth (Mbps)	10	100
Loss rate (%)	0-5, 0-10 *	0

* from client to vpn server:0-5%, from vpn server to client:0-10%

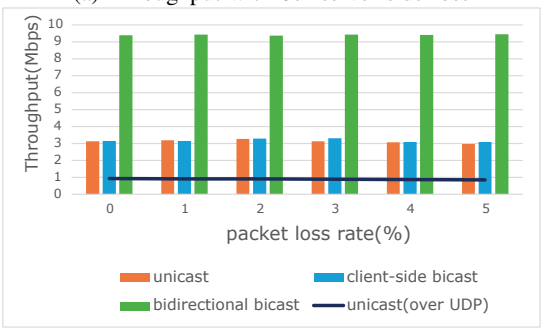
test methods were applied to measure the impact of the HOLB under various packet loss conditions.

To evaluate the effectiveness of the proposed method in mitigating HOLB in VPN over TCP, the performances of unicast, client-side unidirectional bicasting (c-bicast), and bidirectional bicasting (bi-bicast) transmissions were compared. The performance was evaluated using ping and iperf3 under various packet loss conditions. In the experiment, server2 transmitted a TCP data stream to the client using iperf3, while server1 measured the RTT to the client using ping. In this context, the downstream direction is defined as the traffic from the server to the client, and the upstream direction is defined as the traffic from the client to the server. The packet loss rates varied asymmetrically: 0–5% in the upstream direction (client → VPN server) and 0–10% in the downstream direction (VPN server → client). The asymmetric configuration reflects the fact that packet loss in the downstream direction (data packets) has a larger impact on the HOLB than the loss in the upstream direction (ACK packets).

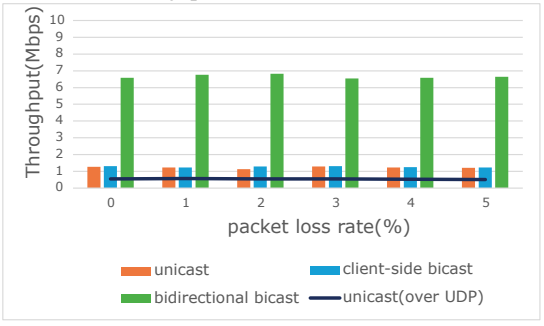
For each condition, RTT was measured by sending 30 ping packets from server1 to the client, and the average values were calculated. Simultaneously, the throughput was measured



(a) Throughput with 0% server-side loss



(b) Throughput with 5% server-side loss



(c) Throughput with 10% server-side loss

Fig. 8: Experiment Throughput Results

using iperf3 with the -R option to evaluate the downstream performance from server2 to the client. By analyzing the average and maximum RTT values and throughput, the impact of HOLB and the effectiveness of bicasting were assessed.

C. Result

In this experiment, the ICMP RTT occasionally exceeded 1000 ms, which was likely due to the buffer bloat caused by concurrent TCP (iperf3) traffic. Packet loss triggers congestion control and retransmission, delaying the ICMP packets in the shared buffer and resulting in large RTT values.

Fig. 7 shows the average and maximum RTTs of the ping packets between the client and Server1 under different packet loss conditions. When the client-side loss rate is 0%, the average RTT of unicast is 1389ms for a server-side loss rate of 0%, increasing to 6066 ms and 11602 ms for server-side loss rates of 5% and 10%, respectively. These results indicate that RTT increases proportionally with packet loss, thereby demonstrating the impact of HOLB on TCP-based VPN tunnels. Even when the server-side loss rate is 0%,

the average RTT increases from 1389 ms to 1720 ms as the client-side loss rate increases from 0% to 5%, suggesting that upstream HOLB occurs owing to the loss of ACK packets.

In contrast, bi-bicast effectively suppressed RTT growth under moderate and severe loss conditions. For instance, when the client-side loss rate is 5% and the server-side loss rate is 5%, the maximum RTT of unicast reaches 10046 ms, whereas the bi-bicast reduces it to 2450 ms, representing an approximately 75% reduction in the RTT. Similarly, when the client-side loss rate is fixed at 5% and the server-side loss rate increases to 10%, bi-bicast achieves 3925 ms compared to 19591 ms for unicast, corresponding to an approximately 80% reduction. These results demonstrate that redundant data transmission in both directions via the bi-bicast effectively mitigates the HOLB and stabilizes the delay characteristics under asymmetric packet loss.

Fig. 8 further shows that unicast throughput degrades sharply as packet loss increases, whereas the bi-bicast throughput remains stable under moderate to high loss rates. When the server-side loss rate is 5%, bi-bicast maintains approximately three times the throughput of unicast. Even under a 10% server-side loss rate, the bi-bicast achieves approximately six times the unicast throughput. This demonstrates that the bi-bicast effectively alleviates the retransmission delay and maintains higher data rates under adverse network conditions.

The performance of c-bicast, which performs unidirectional bicasting only in the upstream direction, is more modest but remains beneficial in specific conditions. When the server-side loss rate is 5%, c-bicast reduces the maximum RTT by approximately 10%. When the server-side loss rate is 0% and the client-side loss rate is 5%, the average RTT decreases from 3133 ms (unicast) to 2447 ms (c-bicast), representing a 20% reduction. Throughput improvements were also observed compared to unicast when the loss occurred only on the client-side path. However, the effect of c-bicast under high server-side loss is limited because data packets lost on the downstream path prevent the transmission of ACK packets required for TCP reliability. These observations align with the improved performance under loss-free server conditions, where ACK redundancy is most effective. Additionally, the TCP delayed ACK mechanism [17] partially mitigates the upstream HOLB by aggregating ACK packets into multiple data segments.

Overall, these results confirm that the bi-bicast significantly mitigates HOLB, achieving both low latency and high throughput even under severe packet loss conditions. Meanwhile, c-bicast offers a lightweight alternative that provides meaningful performance gains in asymmetric loss environments, particularly when the server-side path is stable and the upstream path is prone to packet loss.

V. CONCLUSION

This study proposes a practical SDN-based bicasting method in multi-homed environments to mitigate head-of-line blocking (HOLB) in VPN over TCP environments. By leveraging Open vSwitch and OpenFlow, this method does not require any OS modifications and can be readily deployed.

Under poor network conditions, data packet loss in the downstream direction from the server causes HOLB and significantly degrades communication quality. Bidirectional bicasting substantially improves communication quality by mitigating downstream packet loss, whereas client-side unidirectional bicasting provides performance gains by alleviating HOLB in the upstream direction. Future work will include evaluating the system overhead introduced by bicasting, particularly in terms of the additional network traffic and processing load.

Furthermore, because the effectiveness of bicasting depends on the packet loss rate, a selective bicasting mechanism that dynamically adjusts the redundancy levels according to real-time network conditions is explored to balance the performance gain with the communication overhead. In addition, the proposed method was compared with QUIC, which inherently mitigates HOLB, to evaluate the practical advantages and limitations of SDN-based bicasting under diverse network scenarios. Moreover, the experimental environment will be expanded to more realistic settings, including heterogeneous wireless networks, to validate the effectiveness and scalability of the proposed method for practical deployment.

REFERENCES

[1] B. Robinson, "Companies not keeping pace with hybrid work adoption in 2024, new study finds," 2024. Accessed: 2025-04-28.

[2] "Business VPN — Next-Gen VPN — OpenVPN." https://openvpn.net/.

[3] I. Coonjah, P. C. Catherine, and K. Soyjaudah, "Experimental performance comparison between tcp vs udp tunnel using openvpn," in 2015 International Conference on Computing, Communication and Security (ICCCS), pp. 1–5, IEEE, 2015.

[4] A. Tsertou and D. I. Laurenson, "Revisiting the hidden terminal problem in a csma/ca wireless network," IEEE Transactions on Mobile Computing, vol. 7, no. 7, pp. 817–831, 2008.

[5] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," in IEEE Globecom 2006, pp. 1–5, IEEE, 2006.

[6] T. Shikama, H. Watanabe, and T. Mizuno, "Effects of head of line (hol) blocking in aggregating multiple flows over a tcp connection," in Proc of DICOMO 2006 Symposium, 2006.

[7] "Multipath tcp – linux kernel implementation." https://www.multipath-tcp.org/, 2025. Accessed: 2025-10-16.

[8] A. M. Kakhki, S. Jero, and et al, "Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols," in Proc. of the 2017 internet measurement conference, pp. 290–303, 2017.

[9] W3Techs, "Usage report of http/3 broken down by web servers." https://w3techs.com/technologies/breakdown/ce-http3/web_server, Mar. 2025. Accessed: 2025-04-29.

[10] E. Haleplidis, J. Brouckaert, D. Clarke, and et al, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, 2015.

[11] B. Pfaff, J. Pettit, T. Koponen, and et al, "The design and implementation of open {vSwitch}," in Proc. of the USENIX, 2015.

[12] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data center multicast using software defined networking," in Proc. of the COMSNETS, pp. 1–8, 2014.

[13] NEC Corporation, "UNIVERGE IX3315," 2016.

[14] M. Thomson and C. Benfield, "HTTP/2." RFC 9113, June 2022.

[15] Man7 Documentation, Namespaces, 2024. Accessed: 2025-03-04.

[16] L. M. Pages, "tc(8) - linux manual page," 2025. Accessed: 2025-03-04.

[17] R. T. Braden, "Requirements for internet hosts - communication layers." RFC 1122, 1989.