IFIP Networking 2024 1571006283
2024 IFIP Networking Conference (IFIP Networking)

Enhancing Real-Time Streaming Quality through a Multipath Redundant Communication Framework

Koki Ito*, Jin Nakazato*, Romain Fontugne†, Manabu Tsukada* and Hiroshi Esaki*
* Graduate School of Information Science and Technology, University of Tokyo, Tokyo
{jin-nakazato, utmt0328, mtsukada}@g.ecc.u-tokyo.ac.jp,hiroshi@wide.ad.jp
†Internet Initiative Japan Inc., Tokyo, Japan
romain@iij.ad.jp

***Abstract*—Recently, as networks operate as the infrastructure of modern society, the demands placed on the network by applications have become more complex. In particular, there is an increasing annual demand for high-capacity and low-latency services, including real-time streaming. 5G has been launched to meet this demand, but its stability varies depending on location and time and can only sometimes be considered sufficient. One method to improve communication stability is multipath redundant communication, and much research has been conducted in this area. However, most of this research has focused on TCP-based communication and cannot be applied effectively to real-time UDP streaming. Hence, we propose a multipath redundant communication framework to improve the quality of real-time media streaming communication. Our proposed system is verified using multipath redundant communication and multiple mobile networks from a vehicle moving in an urban area. The experiments use a real-time streaming application based on WebRTC, and the framework significantly reduced packet loss and improved bitrate compared to existing multipath redundant communication systems without interfering with the congestion control mechanisms of the application.**

***Index Terms*—5G, multipath redundant communication, mobile network, vehicular communication, UDP streaming**

I. Introduction

Since their inception, networks have continued to evolve and become a critical infrastructure of modern society. As such, they support societal shifts like Industry 4.0 and Society 4.0 and are expected to play an increasingly crucial role in the upcoming Industry 5.0 and Society 5.0 [1]. A global traffic survey reports that video traffic accounts for two-thirds of all traffic, highlighting the diverse nature of video traffic and its varying technical demands. On-demand video streaming services like Netflix, Amazon Prime, and Disney+ require high bandwidth to deliver FHD and 4K resolution content. On the other hand, live streaming services such as YouTube Live and Instagram Live and online conferencing systems like Zoom and Microsoft Teams [2]–[4], which saw a surge in users during the COVID-19 pandemic, prioritize real-time communication over bandwidth. Furthermore, emerging services related to remote healthcare and remote control demand ultra-low latency and high reliability due to their direct impact on user safety. These scenarios underscore the need for network infrastructures capable of meeting the specific requirements of each application.

The diversification in the types and nature of media traffic and the environments in which these applications are used has advanced. Traditionally, network connections were made through fixed networks at offices or homes. However, the development of mobile networks has made it possible to use networks on the move or outdoors, mainly benefiting the IoT (Internet of Things), where various devices, not limited to PCs and smartphones, are connected to the network. Among these, CAVs (connected autonomous vehicles) have garnered significant attention. CAVs, which link to the internet and other devices, offer various functions, such as enabling sophisticated traffic systems through real-time data collection and analysis, providing safety features like emergency calls, and offering entertainment content. With the advancement

978-3-903176-63-8/24/$31.00 ©2024 IEEE

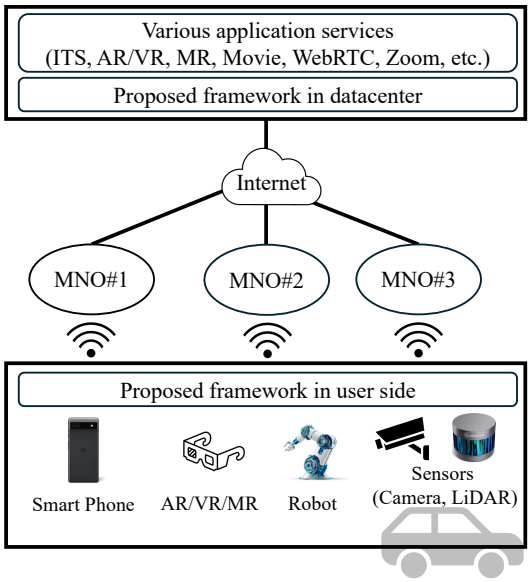


Fig. 1: Overview of this study.

of autonomous driving technology, the perception of vehicles as a second living space is emerging among vehicle manufacturers, increasing the demand for comfortable communication from within vehicles [5]. Indeed, a 2022 report from Zoom indicates that 43 % of people attending meetings from places other than their desks do so from within vehicles [6].

However, the quality of current mobile networks is only sometimes sufficient. This is particularly evident in urban areas, where communication quality can vary based on factors such as time, location, and environmental obstacles like crowded trains or buildings. Previous study [7] involved real-time media communication using WebRTC (Web Real-Time Communication) over mobile networks while driving through urban areas. This study reported that even within the same area, the signal conditions could vary between mobile carriers, leading to differences in communication quality, and that using multiple mobile networks simultaneously could improve communication quality. Furthermore, Lee et al. [8] studied using two mobile networks and WiFi to communicate from moving vehicles have shown that the best communication path can change within seconds, making it difficult to predict which path will have the best quality at the next moment. As a solution to the constant changes in network quality, redundant communication, where a device connects to multiple networks simultaneously and sends duplicated packets through each link, has been proposed. The receiving side then processes the first packet to arrive and discard any later ones. While this increases bandwidth usage by the number of networks connected, it enables communication over the best path at any given moment. Several studies [8]–[11] have extended MPTCP (Multipath TCP) [12] to perform redundant communication. However, these methods are tailored for TCP-based communication and are not directly applicable to media traffic, which frequently employs UDP. Furthermore, in media streaming using UDP, such as WebRTC, session management, and transport control are handled at the application layer, making it challenging to implement multipath without altering existing applications, requiring methods using proxies or tunnels. We employ a multipath UDP proxy as middleware in our research to facilitate redundant communication for media traffic, demonstrating its capability to reduce packet loss and delay. However, redundant communication can cause packet order inconsistencies, potentially misleading the application's congestion control and thus limiting bandwidth [13]. In this study, Kaneko et al. employed techniques based on RTP (Real-time Transport Protocol) header information for transmitting WebRTC media data, which do not apply to non-RTP media traffic.

Considering these challenges, this study proposes a framework for multipath redundant communication through tunneling at the IP layer, aiming to improve the quality of real-time communication without interfering with the application's congestion control regardless of the transport layer protocol as shown in Fig. 1. Our framework allows for redundant communication by duplicating packets on the sender's side and transmitting the same packet across multiple links. This approach can be applied to existing media streaming applications without modifications using a middleware approach. Additionally, to address the packet order inconsistencies caused by redundant communication, a buffering mechanism is introduced at the receiving end of the redundant communication, preventing bandwidth limitation due to misinterpretation by the application's congestion control. The effectiveness of the proposed framework is verified through real-world experiments involving a vehicle moving through an urban area using multiple mobile networks and a

real-time streaming application based on WebRTC. The contributions of this paper are summarized as follows:

- We propose a framework for multipath redundant communication through tunneling at the IP layer, which can be applied to existing media streaming applications without modifications.
- We resolve packet order inconsistencies caused by redundant communication by introducing a buffering function at the receiver's end.
- We evaluate the implemented system using multiple mobile networks in an urban area with an actual vehicle, demonstrating the effectiveness of the proposed framework.

The rest of this paper highlights related work in Section II, explaining the differences. Section III describes the proposed framework and its implementation. Section IV explains the software and hardware configurations used in the experiments of the implemented system, the experimental environment, and the evaluation results. Finally, the paper concludes with a summary and discussion of future challenges.

II. Related Work

This section discusses previous studies on multipath redundant communication to highlight the novelty of this research.

A. Study of end-to-end multipath communication

A notable protocol for enabling multipath communication is MPTCP. MPTCP is an extension of the TCP protocol that uses multiple paths simultaneously for data transmission. It is compatible with traditional TCP, offering the advantage of applying to existing applications without modification. Multipath communication improves throughput by utilizing multiple base stations (frequency bandwidths) and can maintain connections even if a failure occurs on one of the base stations (communication paths). However, conventional MPTCP distributes the application's data across multiple communication paths, which can throttle the overall communication if there are significant differences in quality or, specifically, latency among the utilized paths. Furthermore, packet loss always triggers retransmissions. Several studies have extended MPTCP to reduce packet loss and latency by sending duplicated packets across multiple communication paths. Frommgen et al. [9] proposed ReMP TCP, incorporating redundant communication into MPTCP, and demonstrated through simulations that it could halve the average round-trip delay. Their study employed TCP option headers to assign sequence numbers for packet identity discrimination. Wang et al. [10] proposed a set of extensions for MPTCP (called MPTCP-L) for latency-sensitive data communication in cloud data centers and evaluated it using actual traffic within a data center, demonstrating its effectiveness. Lee et al. [8] proposed a scheduler for MPTCP that performs redundant communication when network latency indicates low reliability and evaluated its performance from a moving vehicle. Guo et al. [11] also proposed a scheduler that observes path delays and performs redundant communication when quality deteriorates, showing performance improvements in mobile and WiFi networks. These studies demonstrate the effectiveness of redundant communication but they all rely on TCP, which presents challenges in applying them to UDP, commonly used for media traffic. QUIC [14], developed by Google in 2012, is an alternative transport protocol to TCP. While the replacement of TCP by QUIC, such as in HTTP/3, is underway, MP-QUIC is currently in the process of standardization and practical implementation [15], [16]. The standardization discussions include options for duplicating all packets or only those that require retransmission for redundant communication [17]. However, QUIC, being a reliable protocol with congestion and retransmission control, cannot directly replace the transport layer of WebRTC. Discussions are ongoing for extensions like RTP over QUIC, and methods for handling media traffic over QUIC, such as MOQT (Media over QUIC Transport) [18], [19]. The implementation of multipath video traffic communication using QUIC is still pending, warranting further discussion. Boutier et al. [20] adapted Mosh, a remote terminal application similar to SSH, for multipath communication, necessitating substantial additions and modifications due to its UDP-based.

B. Study of multipath communication

Protocols optimized for real-time media streaming, such as WebRTC, are tightly integrated across all layers, making modifications a considerable burden. Middlebox approaches excel in applicability to existing applications but hide the processing performed en route from the end host, complicating performance tuning, particularly when addressing

potential bottlenecks such as latency, jitter, and bandwidth. Moreover, middlebox approaches make it difficult to control applications using end-host statistics, increasing system complexity for flexible control. To our knowledge, several studies have explored middlebox techniques for multipathing UDP traffic, in contrast to MPTCP. Liu et al. [21] proposed an MPUDP (Multipath UDP) framework through overlay networks, where a central controller manages paths between nodes in the framework, distributing traffic based on usage. Lukaszewski et al. [22] proposed a VPN-based multipath framework and evaluated its performance using TCP and UDP as underlying protocols. Amend et al. proposed a framework using MP-DCCP (Multipath (Datagram Congestion Control Protocol)) and evaluated its performance through simulation. DCCP [23] is a transport layer protocol developed for applications where timely data delivery, such as real-time communication and online gaming, is more critical than reliability. SMPTE 2022-7, a standardization document in the broadcasting industry, defines redundant communication for RTP packets, typically implemented using specialized hardware within data center networks [24]. Kawana et al. [25] used QUIC-Proxy and OpenFlow for redundant communication, reducing packet loss. Kaneko et al. [13] implemented a system that redundantly transmits real-time video from WebRTC using multiple mobile networks, conducted measurements from a moving vehicle, and demonstrated that redundant communication improves WebRTC's communication quality. This method duplicates packets using an MPUDP-Proxy, with the receiving side's proxy employing RTP packet sequence numbers for duplicate resolution. Although it significantly reduced latency and packet loss, it reported a degradation in bitrate resulting from redundant communication. This degradation occurs when packets from a faster path are lost and recovered by a slower path, causing packet reordering. Hence, a framework capable of handling packet reordering is necessary.

III. Proposed Methodology

This research assumes a system designed to improve the quality of service for remote control services such as robots, autonomous driving, and UAVs. To ensure reliability within this system, it will connect to two mobile networks, where one operates in an unstable mobile network environment, while the other operates in a stable mobile network environment. The unstable mobile network environment is characterized by fluctuating communication delays and frequent packet losses due to handovers and weak signals, including bursts of losses occurring within seconds. The system in an unstable mobile network environment can connect to multiple mobile networks. The communication transmitted and received is real-time media traffic, where packet loss significantly adversely affects quality. It is assumed that the connecting mobile networks support IPv6 and consider the presence of stateful firewalls in the communication path.

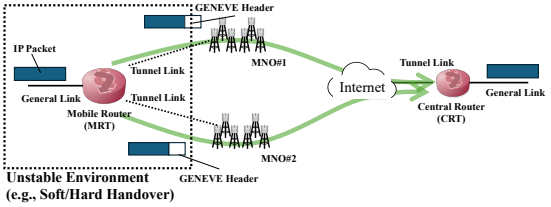


Fig. 2: Overview of the proposed framework.

A. System Framework

Fig.2 depicts the overview of the proposed framework. The proposed framework comprises an MRT (Mobile Router) deployed in an unstable mobile network environment and a CRT (Central Router) situated in a stable network environment. Each RT (router) is equipped with interfaces connected to one General Link and one or more Tunnel Links. RTs receive packets transmitted by regular hosts through the General Link and duplicate these packets. Duplicated packets are encapsulated using GENEVE (Generic Network Virtualization Encapsulation) [26] and transmitted through the Tunnel Links. Attaching sequence numbers to the GENEVE option header makes it possible to identify which packets are identical. The opposing RT receives the encapsulated packets through the Tunnel Link and checks the sequence numbers in the header. If the sequence number has already arrived, the packet is discarded; otherwise, it is decapsulated and inserted into the internal buffer. RT extracts the packet with the smallest sequence number from the buffer and transmits it through the General Link. If the packet's sequence number to be transmitted is more than two greater than the previous one, it waits for a certain period for the missing packets. It advances to the following sequence number if they

TABLE I: Software implementation in this study.

OS	Ubuntu 20.04.5 LTS
golang	v1.21.1 linux/amd64
gopacket	v1.1.20-0.20220810144506

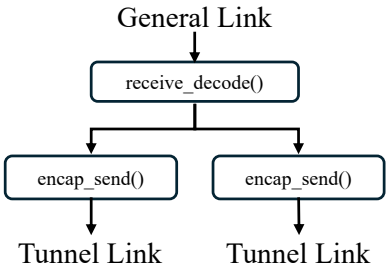


Fig. 3: Design of transmitter.

still do not arrive. This method achieves communication redundancy through middleboxes without necessitating changes to the application. Furthermore, the framework's tunneling encapsulates IP packets in GENEVE, making it independent of specific transport and higher-layer protocols. Since GENEVE operates over UDP, intermediate nodes treat it as UDP packets, allowing it to traverse existing infrastructure.

B. Implementation

The proposed framework is implemented to run a common program on the MRT and the CRT. In each router, two processes operate: Receiver and Transmitter. Data exchange between threads primarily occurs through channels, which trigger each thread's operation upon data arrival. This approach eliminates the need for busy loops across the system. The framework's implementation utilized Go (Go programming language), with all components developed in user space. Packet processing employed the gopacket library [27]. Implemented software versions, including OS (Operating System), are summarized in Table I.

The transmitter encapsulates packets from the General Link and sends them through the Tunnel Link. Its design is illustrated in Fig. 3. Upon startup, the Transmitter initializes a sequence number to zero and maintains it internally. The sequence number is incremented with each received packet. After decoding a packet, the thread `encap_send()` is launched for each Tunnel Link. In `encap_send()`, a capsule header is created using the associated Tunnel Link's IP and MAC addresses, enveloping the decoded packet for transmission. Destination IP addresses for each Tunnel Link are statically set via a configuration file, while other information is retrieved from interface data. Packet encapsulation and transmission, requiring significant processing, are parallelized across Tunnel Links.

Conversely, the receiver decapsulates packets received from the Tunnel Link, buffers them, and transmits them through the General Link, as depicted in Fig. 4. The receiver must maintain a sequence table to manage the sequence numbers of received packets. Since the sequence table is accessed by multiple threads, it requires exclusive control features, implemented in this case with an in-memory. Given the potential for multiple Tunnel Links, the thread `receive_decap()` for receiving packets is also plural. In `receive_decap()`, packets are decoded to obtain sequence numbers from the GENEVE option header. The sequence table is updated if the sequence number arrives for the first time. Subsequently, the packet, sequence number, and reception time tuple are passed to the thread `store()`. However, if the sequence number has already arrived, the packet is discarded, and processing moves to the next packet. `store()` asynchronously receives data from multiple threads, continuously inserting it into a buffer. The buffer, containing tuples of packets, sequence numbers, and reception times, is managed as a heap sorted by sequence number. As a heap, it rearranges its elements in logarithmic order upon each insertion or retrieval, allowing constant time access to the minimum value. Buffering timeout is handled by the thread `send()`, illustrated in Fig. 5 for processing a single packet. Initially, the buffer's minimum element (the one with the smallest sequence number) is checked (`GetMin()` without extraction. As described, if the sequence number of the minimum element follows immediately after the last transmitted packet's sequence number, it is sent without delay; if not, it is discarded if smaller or waits if larger. If waiting, a timer notifies after `waitTime`, during which the buffer's minimum value is awaited. `waitTime` is the lesser of `MaxBufferTime` (set at receiver startup) and `limitTime -- now`. `limitTime` starts with a sufficiently high value. If the buffer's minimum value updates before `waitTime` passes, `limitTime` updates to the packet's reception time + MaxBufferTime, and the process revisits the buffer's minimum element. This update prevents

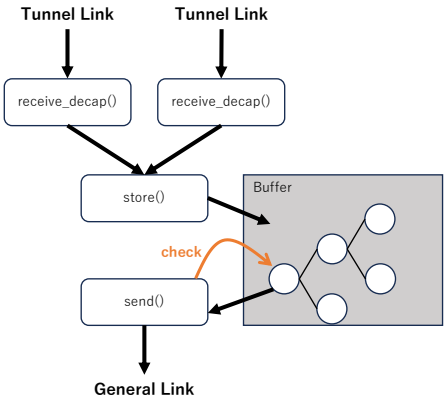


Fig. 4: Design of receiver

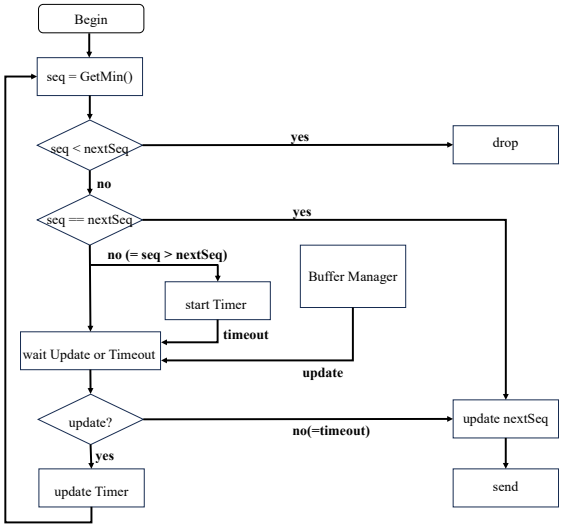


Fig. 5: Flow chart of buffering implementation.

delaying the transmission of already arrived packets for more than `MaxBufferTime` even if later packets arrive out of sequence. After waiting for `waitTime`, `limitTime` resets to a high value, and the packet is then sent. Besides, dummy packets are sent at a frequency of once per second for hole punching to maintain connectivity

IV. Evaluation

To verify that the proposed framework resolves the issue of interference with application congestion control, which was a problem with existing methods, live streaming communication was conducted from within a vehicle traveling through an urban area (Shinjuku, Tokyo, Japan) using the proposed framework. A live streaming system developed with

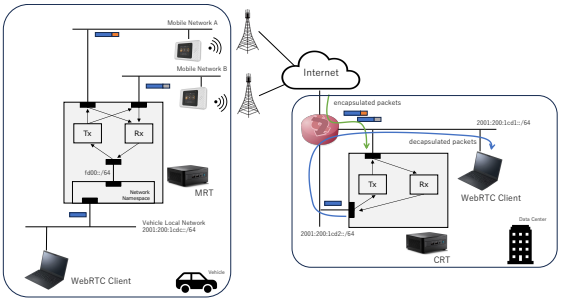
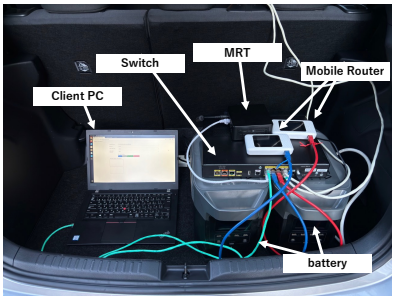


Fig. 6: Measurement network setup.

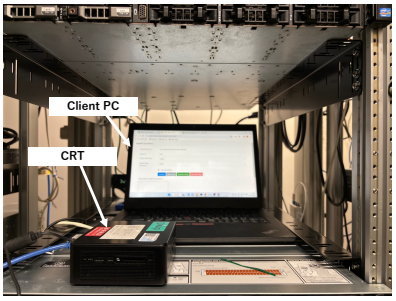
WebRTC that employs the proposed framework was used for the measurement, conducting peer-to-peer (P2P) communication between a PC in the vehicle and another PC in a data center. The proposed framework was evaluated based on statistical information obtained from WebRTC's API and information on packets processed by the proposed framework. For comparison purposes during the evaluation, software as conventional that omitted the packet reordering feature of the framework was prepared. This conventional software sends packets from the General Link to the client PC in the order they are received from multiple Tunnel Links. Consequently, from WebRTC's perspective, packet reordering occurs, causing congestion control to reduce the bitrate.

A. Experiment Setup

The measurement network setup is illustrated in Fig. 6. This figure shows a configuration connecting two mobile networks of 4G and 5G (Sub6 n 77/n 78/n 79 and millimeter wave n 257). The MRT operates with two network namespaces: the default namespace and an additional namespace called subNS. In the default namespace, the software components of the proposed framework, namely the transmitter and receiver, are active. The subNS provides internet connectivity to the client within the vehicle, functioning as a router. The MRT is connected to two different mobile networks via two Tunnel Links. The addresses for these interfaces are set via Router Advertisements sent by the mobile router. These two namespaces are connected by a virtual link, designated as the General Link. The virtual link's address uses the Unique Local Address space of fd00::/64. The local network within the vehicle uses the address space of 2001:200:0:1cdc::/64. In subNS, radvd is utilized



(a) Vehicle.



(b) Datacenter.

Fig. 7: Hardware configuration setup.

to advertise this prefix, and ip6tables is employed for MSS (Maximum Segment Size) clamping to ensure that the MSS for TCP communications is set to 1340. This adjustment results in a client PC's TCP communication MTU being 1400 bytes, with packets sent from the MRT at 1464 bytes, below the commonly acceptable size of 1500 bytes for internet transit. In subNS, the default route is directed towards the virtual link, ensuring that all traffic from the client within the vehicle's local network uses the proposed framework for communication. Meanwhile, the CRT has each Tunnel Link as well as General Link. The Tunnel Link has the address of 2001:200:0:1cd1::6666:92, and the General Link has the address of 2001:200:0:1cd2::6. Although the client PC also connects to the 2001:200:0:1cd1::/64 network, it only needs to ensure internet connectivity. The internet connectivity for the client PC is provided by an external router, meaning the CRT does not require multiple network namespaces. To facilitate communication from the internet to the vehicle's local network, routing was set up by the external router to direct traffic for 2001:200:0:1cdc::/64 to the General Link.

A separate service was established to facilitate WebRTC communication between a PC inside a vehicle and a PC within a data center. The system's architecture is illustrated in Fig. 8. All components of this system were deployed using Docker containers, which include a reverse proxy (nginx), a web server providing the WebRTC client program, and a signaling server. The nginx container serves multiple roles, including encryption for HTTPS and WebSocket over SSL/TLS, access logging, authentication, and site distribution based on domain. The WebRTC program is written in JavaScript, enabling the transmission of media traffic from devices running this frontend program. The web server lacks backend processing capabilities, serving only static HTML and JavaScript files to access hosts. The web server's interface is hosted using Vite. Ayame by Shiguredo [28] was employed for the signaling server. Hosts that receive the WebRTC client source from the web server connect to the signaling server, exchanging information to initiate P2P communication between hosts. The software versions used are listed in Table II. The WebRTC streaming service collected statistics for inbound–rtp, outbound–rtp, remote–inbound–rtp, and remote–outbound–rtp every second via `getStats()` API. VP9 was used for video compression, and Opus was used for audio. The video resolution options available were 1920×1080, 1280×720, 854×480, and 426×240, with only 1920×1080 used for this study's measurements. However, the resolution can automatically adjust based on communication quality. The frame rate is also automatically regulated by WebRTC's adaptive bitrate.

TABLE II: Software version of WebRTC.

docker	20.10.21
nginx	1.23.0
Vite	4.0.4
ayame–web–sdk	2022.1.0
ayame	2022.2.0
Chrome PC in vehicle	120.0.6099.71
Chrome PC in datacenter	120.0.6099.217

TABLE III: Comparison of packet statistics.

	MNO#1	MNO#2	Proposal
Transmitter packet count	170,445	170,445	170,445
Packet loss count	3542	25,462	4
Packet loss ratio [%]	2.078	14.938	0.002
Adopted packet count	123,619	46,822	—
Adopted packet ratio [%]	72.53	27.47	—

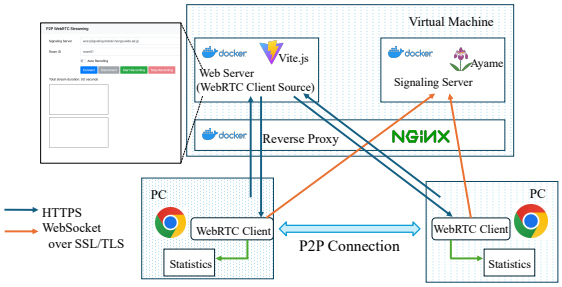


Fig. 8: Configuration of WebRTC streaming.

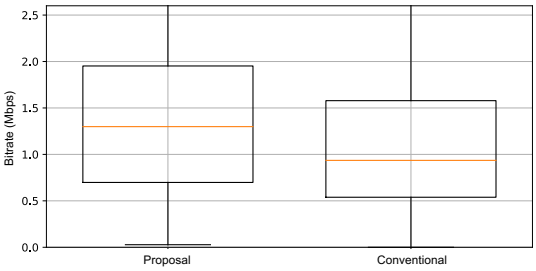


Fig. 9: Comparison of bitrate of WebRTC.

B. Results

This experiment conducted two verifications, comparing the proposed framework with traditional methods. The first verification involved operating two client PCs inside a vehicle, each communicating with the same client PC inside a data center. One client PC communicated using the proposed framework, while the other communicated using a system that lacked the proposed framework's packet reordering buffering feature (traditional method). The maximum buffer time was set to 300 ms. Since redundant communication was conducted on each client PC, two streams flowed through each mobile router. This experiment aimed to verify that (i) multipath redundant communication reduced packet loss and (ii) reordering improved the bitrate. Packet loss was evaluated based on packet logs passing through the CRT, and bitrate was assessed using WebRTC's statistical information. The second experiment involved operating only one client PC inside the vehicle, with the maximum buffer time varied for each loop. The maximum buffer times tested were 10 ms, 100 ms, 300 ms, 500 ms, and 1,000 ms. This experiment evaluated the relationship between maximum buffer time and bitrate. Both measurements were conducted around Shinjuku, Tokyo, Japan, repeating the same route. Each loop took approximately 15 minutes.

The first experiment's results, which used the proposed framework for communication, showed the number of packets sent to each mobile network, the loss rate, and the number of packets adopted, which are summarized in Table III. The "adopted packets" refer to the number of packets with the same sequence number that arrived first and were transmitted from the General Link in redundant communication. Packet loss in mobile network links showed about 15 % in MNO#1 network and 2–4 % in others. The proposed framework reduced the packet loss rate to 0.002 %. Besides, the results comparing the bitrate of WebRTC communications using the proposed framework and communications without reordering (conventional) are shown in Fig. 10. This figure shows that the bitrate is higher with the proposed framework than with the conventional method. This result confirmed that multipath redundant communication can lead to packet reordering and that the proposed framework can mitigate the bitrate reduction caused by such reordering.

The results of the second experiment, showing the bitrate for each maximum buffer time, are illustrated in Fig. 10. This figure indicates an improvement in bitrate compared to single carrier communication in all cases, attributed to the reduction in packet loss due to redundant communication. The higher bitrate compared to the first experiment is because, in the first experiment, two video streams were flowing through each mobile network. In contrast, in this experiment, there was only one stream.

The number of packet losses, timeouts, and packets discarded by the system for communications using different maximum buffer times are summarized in Table IV. The system-discarded packets refer to those arriving after a sequence number has been skipped due to timeouts and thus discarded. The measurement with a maximum buffer time of 300 ms was conducted twice, resulting in more transmitted packets. With a maximum buffer time of 10 ms, packet loss recovery for one link did not occur, resulting in 1027 packets being discarded. From the client PC's perspective, this discarding by the framework is equivalent to packet loss, contributing to a reduction in bitrate. In measurements with a 100 ms maximum buffer time, packet losses co-occurred on both MNO#1 and MNO#2 links, increasing the timeouts. Almost no timeouts occurred

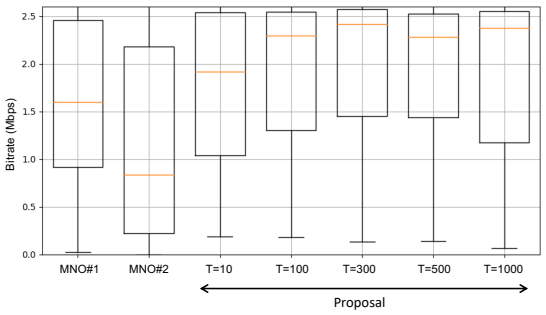


Fig. 10: Relationship between Maximum Buffer time (T) and bitrate of WebRTC.

TABLE IV: Maximum buffer time (T) vs. timeout occurrences count

Maximum buffer time (T)	10	100	300
Transmitter packet count	211694	249408	602361
Packet loss count	2	141	0
Timeout count	706	15	3
Discarded packet count	1027	23	14

when the maximum buffer time was set to 300 ms or more, and there was no significant difference in bitrate. These results suggest that providing a buffer of several 100 ms orders in multipath redundant communication can mitigate bitrate reduction.

V. Conclusion

This study proposed a redundant communication framework designed to improve the quality of real-time media streaming communications. It addresses the limitations of transport protocols in traditional redundant communication systems by employing GENEVE for tunneling at the IP layer. Additionally, it resolves interference with application congestion control due to packet order inconsistencies by implementing buffering on the receiver side. We demonstrated significant packet loss reductions and bitrate improvements compared to existing redundant communication systems by utilizing multiple mobile networks from a vehicle in an urban area. These results confirm the effectiveness of the proposed framework. However, this evaluation was conducted in a single environment, necessitating further evaluations in multiple settings to assess its universality. Additionally, an investigation into the overhead introduced by the implemented system is required. This is particularly important for real-time media communications, where an end-to-end delay of approximately 100 ms to 150 ms is sought, making overhead evaluation crucial. Moreover, this research focused solely on the data plane of redundant communication. The practical operation would require methods for exchanging IP addresses that serve as endpoints for the GENEVE tunnel and for communicating the maximum buffer time in response to communication quality. While this study showed that a short maximum buffer time could negatively affect communication bitrate, determining the most appropriate maximum buffer time requires further investigation.

Acknowledgment

This research was supported in part by JST, CREST Grant Number #JPMJCR22M4; and in part by JSPS KAKENHI under Grant #22H03574.

References

[1] The global internet phenomena report 2023. Technical report, Sandvine, 1 2023.

[2] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Eric Pujol, Igmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. Implications of the covid-19 pandemic on the internet traffic. In *Broadband Coverage in Germany; 15th ITG-Symposium*, pages 1–5, 2021.

[3] Cisco webex helps customers stay remotely connected and reimagine work. https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2020/m06/cisco-webex-helps-customers-stay-remotely-connected-and-reimagine-work.html, 6 2020. (Accessed on 01.22.2024).

[4] 2020 - zoom blog. https://blog.zoom.us/ja/2020ŒŁ/, 12 2020. (Accessed on 01.22.2024).

[5] Yu Asabe, Ehsan Javanmardi, Jin Nakazato, Manabu Tsukada, and Hiroshi Esaki. Autowarev2x: Reliable v2x communication and collective perception for autonomous driving. In *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, pages 1–7, 2023.

[6] Here's how you used zoom in 2022 - zoom blog. https://blog.zoom.us/how-you-used-zoom-2022/, 12 2022. (Accessed on 01.22.2024).

[7] Jin Nakazato, Kousuke Nakagawa, Koki Itoh, Romain Fontugne, Manabu Tsukada, and Hiroshi Esaki. Webrtc over 5g: A study of remote collaboration qos in mobile environment. *Journal of Network and Systems Management*, 32, 2023.

[8] HyunJong Lee, Jason Flinn, and Basavaraj Tonshal. Raven: Improving interactive latency for the connected car. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 557–572, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Alexander Frommgen, Tobias Erbshäußer, Alejandro Buchmann, Torsten Zimmermann, and Klaus Wehrle. Remp tcp: Low latency multipath tcp. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–7, 2016.

[10] Wei Wang, Liang Zhou, and Yi Sun. Improving multipath tcp for latency sensitive flows in the cloud. In *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, pages 45–50, 2016.

[11] Yihua Ethan Guo, Ashkan Nikravesh, Z. Morley Mao, Feng Qian, and Subhabrata Sen. Accelerating multipath transport through balanced subflow completion. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 141–153, New York, NY, USA, 2017. Association for Computing Machinery.

[12] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684, March 2020.

[13] Kaneko Naoya, Ito Takahiro, Katsuda Hajime, Watanabe Toshinobu, Abe Hiroshi, and Onishi Ryokichi. Applying and evaluating multipath redundant communication technology for webrtc-based video streaming. *Journal of digital practices*, 3(3):21–31, 2022.

[14] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[15] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-06, Internet Engineering Task Force, October 2023. Work in Progress.

[16] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. Multipath quic: A deployable multipath transport protocol. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

[17] Yunfei Ma, Yanmei Liu, Christian Huitema, and Xiaobo Yu. An Advanced Scheduling Option for Multipath QUIC. Internet-Draft draft-ma-quic-mpqoe-01, Internet Engineering Task Force, November 2022. Work in Progress.

[18] Joerg Ott, Mathis Engelbart, and Spencer Dawkins. RTP over QUIC (RoQ). Internet-Draft draft-ietf-avtcore-rtp-over-quic-07, Internet Engineering Task Force, October 2023. Work in Progress.

[19] Luke Curley, Kirill Pugin, Suhas Nandakumar, and Victor Vasiliev. Media over QUIC Transport. Internet-Draft draft-ietf-moq-transport-01, Internet Engineering Task Force, October 2023. Work in Progress.

[20] Matthieu Boutier and Juliusz Chroboczek. User-space multipath udp in mosh. *arXiv preprint arXiv:1502.02402*, 2015.

[21] Shaowei Liu, Weimin Lei, Wei Zhang, and Hao Li. Mpudp: Multipath multimedia transport protocol over overlay network. In *2017 5th International Conference on Machinery, Materials and Computing Technology (ICMMCT 2017)*, pages 731–737. Atlantis Press, 2017.

[22] Daniel Lukaszewski and Geoffrey Xie. Multipath transport for virtual private networks. In *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*, Vancouver, BC, August 2017. USENIX Association.

[23] Sally Floyd, Mark J. Handley, and Eddie Kohler. Datagram Congestion Control Protocol (DCCP). RFC 4340, March 2006.

[24] St 2022-7:2019 - smpte standard - seamless protection switching of rtp datagrams. *ST 2022-7:2019*, pages 1–11, 2019.

[25] Rei Nakagawa Tomoya Kawana, Nariyoshi Yamai. Communication multiplexing of server with quic and sdn in multihomed networks. In *The 38th International Conference on Information Networking (ICOIN2024)*, Ho Chi Minh City, Vietnam, 2024.

[26] Jesse Gross, Ilango Ganga, and T. Sridhar. Geneve: Generic Network Virtualization Encapsulation. RFC 8926, November 2020.

[27] Graeme Connell. gopacket. https://gitlab.com/google/gopacket.

[28] Shiguredo. OpenAyame. https://github.com/OpenAyame/ayame, December 2022.