Adapt compression rate and improve network packet utilization based on RTP video stream

Yucheng Jing
Department of Computer Science
KOOKMIN UNIVERSITY
Seoul, South Korea
jingyucheng@kookmin.ac.kr

Sang-Chul Kim
Department of Computer Science
KOOKMIN UNIVERSITY
Seoul, South Korea
sckim7@kookmin.ac.kr

***Abstract*—With the rapid development of the Internet and the widespread use of multimedia applications, real-time video transmission plays an increasingly important role in the field of network communication. We focus on real-time video transmission based on the Real-Time Transport Protocol (RTP), aiming to improve video transmission efficiency, reduce bandwidth requirements, and enhance user experience. Our emphasis lies in two aspects: adaptive adjustment of video compression rate and utilization of data packets. We devise an adaptive video compression rate algorithm that dynamically adjusts video compression rate in real time based on the current network congestion level to improve Quality of Service (QoS) and enhance user experience. Furthermore, to fully utilize network resources (data packets), we study and design an efficient data segmentation algorithm. By optimizing the video data packet encapsulation process of the RTP protocol, we aim to enhance the utilization of each data packet in the network. Our goal is to maximize bandwidth utilization while ensuring video quality, thereby improving video transmission efficiency. Finally, we will conduct experimental verification in a real network environment to assess the effectiveness of our proposed adaptive compression rate algorithm and data packet segmentation mechanism.**

*keywords:*Network,RTP,UDP,QoS,Packet,Adaptive

I. Introduction

In real-time video transmission scenarios, the User Datagram Protocol (UDP) over Local Area Network (LAN) is commonly employed, facilitated by the Real-time Transport Protocol (RTP), owing to UDP's numerous advantages in meeting low-latency transmission requirements. However, in highly congested network environments, severe packet loss occurs due to insufficient bandwidth allocation at the sender, resulting in significant degradation of user experience, manifested by screen tearing and severe stuttering at the client end. Furthermore, UDP protocol stipulates a maximum message length of 16 bits, with a default Maximum Transmission Unit (MTU) set at 1500 bytes, thus constraining the maximum payload size per data packet within a specified range. When the volume of transmitted data exceeds the maximum allowable packet size, the network layer automatically segments and packages large-sized data—a process beyond control. Moreover, as the data size of each frame may not be divisible by the packet size, packet wastage inevitably occurs.

Identify applicable funding agency here. If none, delete this.

Our Contribution: Addressing the aforementioned challenges, we propose a dynamic segmentation algorithm at the transport layer, which combines undersized data segments, not meeting the maximum effective payload of RTP data packets, with subsequent frames' data. This ensures that each data packet transmitted by the sender conforms to the maximum payload size. This method enhances network packet utilization to some extent, ensuring the transmission of maximum data with minimal network resources. Additionally, we compute the Round-Trip Time (RTT) for each data packet and dynamically adjust the video data compression rate based on real-time network state fluctuations, thereby achieving compression rate adaptation.

Through testing, employing this algorithm at the sender's end during the transmission of identical video data has demonstrated a reduction in the total number of transmitted data packets, enabling the transmission of more data with fewer packets and enhancing network transmission efficiency. Furthermore, under adaptive compression rates, as network congestion increases, the packet loss rate of video data significantly decreases. Instances of video playback stuttering and tearing are substantially reduced, thereby enhancing user experience.

II. Background

In this section, we delineate the role and characteristics of the User Datagram Protocol (UDP), followed by an exposition on the current state of UDP protocol and the forefront research in real-time video transmission domain. UDP protocol, as a lightweight and connectionless transport layer protocol, is engineered primarily to facilitate rapid and low-latency data transmission. Unlike its counterpart, the Transmission Control Protocol (TCP), UDP eschews reliability and sequencing mechanisms, endowing it with greater flexibility and suitability for applications demanding high real-time performance and tolerable data loss, such as real-time audio-video transmission and online gaming. The salient features of UDP encompass connectionlessness, the absence of congestion control, datagram-oriented transmission, and reduced communication overhead, rendering it an efficacious choice for transmitting small-sized data, notably applicable to applications necessitating swift responsiveness and streamlined communication

processes. At the network level, UDP streamlines data transmission for applications accentuating speed and real-time responsiveness by discarding redundant control mechanisms. Presently, in the realm of real-time transmission, the ubiquitously employed Real-time Transport Protocol (RTP) relies extensively on UDP, underscored by UDP's pivotal role in the transmission domain. The Optimal UDP Data Length in Ethernet [1]deliberates on the nuanced considerations involved in determining UDP packet length during video transmission. To alleviate CPU processing overhead, UDP packet segmentation transpires when the segment length eclipses the Maximum Transmission Unit (MTU), albeit at the expense of diminished transmission efficiency. Conversely, excessively brief UDP segments augment the packet count required for file transmission, constraining efficiency. Thus, fostering support for larger packet sizes emerges as pivotal for optimizing performance, within the confines of RTP packet length specifications, to ensure real-time imperatives are met sans undue elongation. Permutation-based TCP and UDP transmissions to improve goodput and latency in the Internet of Things [2]proffers a permutation-based encapsulation paradigm to redress underutilization of UDP packets, striving for seamless packet transmission and attaining elevated throughput and diminished latency. Galvanized by this proposition, we advance a solution within the video transmission sphere to counteract underutilization of UDP packets. Dynamic adaptive streaming over HTTP– standards and design principles [3]unveils the Dynamic Adaptive Streaming over HTTP (DASH) framework grounded on the TCP-based HTTP protocol, dynamically calibrating video resolution and ancillary settings predicated on real-time feedback from the receiving terminus. Encouraged by this paradigm, we extrapolate akin principles to optimize adaptive streaming within the purview of RTP video streams.

III. Method

In order to explore whether the above conjecture can improve network transmission efficiency and user experience, we designed two sets of experiments to verify the two methods respectively.

A. Optimize packet segmentation algorithm

We conducted experiments using the RTP protocol based on the UDP protocol, dividing the experiment into two parts: the sender and the receiver, for testing. (As this paper primarily focuses on video transmission, audio transmission is not discussed at this time.) We compared the consumption of network resources (packet count) when transmitting the same-sized video using both the original fixed packet segmentation algorithm and the dynamic packet segmentation algorithm. Since at the sender's end, we merged data from two frames into one packet, at the receiver's end, we re-divided the received data based on whether the packet contains the file ending marker bytes of the Joint Photographic Experts Group (JPEG) image byte stream, `b'\xff\xd9'`. We cached the received parts and decoded and reconstructed the image once a frame of data was successfully received, as depicted in Fig.1. Used

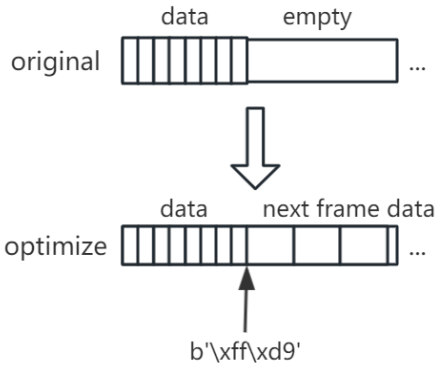

Fig. 1. Packet division

experimental equipment:

- Raspberry Pi 4B 4G
- PC host and CPU is i7-4770
- System is Ubuntu 22.04

1) sender: We use Raspberry Pi as the sender. We use a randomly found video in MP4 format. The video parameters are as follows:

- resolution 640×360
- frame rate $23.98FPS$
- total frame 19918
- video size 42.2M

When encapsulating data, we assess whether the data size is smaller than the maximum payload size that can be accommodated in a data packet (in our experiment, we assumed a maximum packet size of 1472 bytes). If the data size is smaller, we store this segment of data and retrieve the next frame of data. Before encapsulating the next frame of data, we first include the previously stored data, followed by the header content of the next frame of data, in a cyclic manner.

We employed both a fixed data packet segmentation algorithm and a dynamic data packet segmentation algorithm to transmit the same video. In the dynamic data packet segmentation algorithm, we defined a unique identifier for each data packet at the sender's end, denoted as $T_1...T_n$. Each data packet should have its own identifier, which is an incrementing value with a step size of 1. We embed T_i into the header of each data packet to facilitate packet loss rate calculation. We evaluated the total number of data packets consumed when transmitting the same data using these two different segmentation algorithms.

2) receiver: We utilized a PC host as the receiving terminal, strictly limiting the size of UDP data packets to minimize the occurrence of packet loss. To achieve this, we employed multithreading for receiving and processing data [4]. Since the sender continuously transmits video frame data, we need to segment the fragmented data received in each packet. We can accomplish this by parsing the received data packets based on the file ending marker (`b'\xff\xd9'`). The portion of

data containing the file ending marker and preceding bytes constitutes the end segment of the previous frame, which is appended to the previously received byte stream for playback of the frame (refer to Fig.1). The remaining portion represents the beginning of the next frame and is stored in a separate memory space until a subsequent data packet containing the file ending marker is received. Upon extraction of the byte stream corresponding to this frame from the packet, playback can commence, thus completing the cycle. Throughout this process, we define $R_1, R_2...R_n$ to record each received data packet, allowing us to describe the packet loss rate as such.

$$\frac{\sum_{i=1}^{n} R_i}{T_i} \tag{1}$$

B. Optimize user experience

We utilize a camera to capture real-time video data, which is then transmitted via the sender. Concurrently, we log the timestamp of each data packet transmission, denoted as $T_{s1}...T_{sn}$, along with the packet numbers $P_1..P_n$. Upon receiving a data packet, the receiver communicates back to the sender via the UDP protocol, indicating the receipt of the corresponding packet numbers. Upon reception of the acknowledgment packets containing the packet numbers, the receiver records the timestamp $T_{r1}...T_{rn}$ at the moment of each acknowledgment. Given the correlation between $T_{s1}...T_{sn}$ and $P_1..P_n$, we define the round-trip time(RTT) of a data packet as follows:

$$RTT_i = T_{ri} - T_{si} \tag{2}$$

we can establish multiple thresholds $T_{h1}, T_{h2}...t_{hn}$ and categorize the network status $status$ into different levels based on these thresholds. Depending on the network status level, we adjust the compression rate accordingly (Fig.2).

$$\text{Status}_i = \begin{cases} Status_1, & \text{if } RTT_i > T_{hi} \\ Status_2, & \text{if } RTT_i \leq T_{hi} \end{cases} \tag{3}$$

We artificially induce a congested network environment, for instance, by transferring ultra-large files within the same local area network, to facilitate testing and comparison between the dynamic compression rate setting and non-dynamic compression rate setting in terms of packet loss rate. We conduct tests under normal network conditions and calculate the packet loss rates for both transmission methods. Subsequently, we intentionally create a congested network environment to degrade network service quality (QoS), after which we conduct tests again and calculate the packet loss rates for both transmission methods under congested network conditions. We then analyze the results accordingly.

IV. Eveluation

In this section, we will evaluate two algorithms utilized in our experiments: the dynamic packet segmentation algorithm and the adaptive video compression rate algorithm.

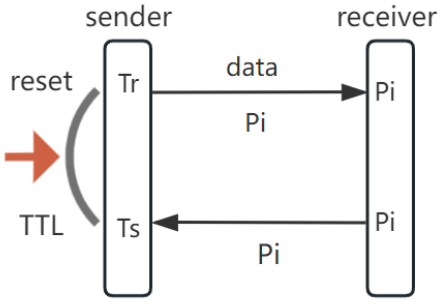

Fig. 2. Adaptive compression rate

A. dynamic packet segmentation

1) sender: The total number of data packets utilized for transmitting the same video twice amounted to 394782, as illustrated in Fig:3. Among these, the utilization of conventional packet segmentation consumed network resources totaling 202488, representing 52.6% of the total sent data packets. Following optimization through packet segmentation, the expenditure of network resources dwindled to 192294, constituting 47.4% of all transmitted data packets. Consequently, post-optimization resulted in savings of 10194 data packets, correlating to an improvement rate of approximately 5%.

2) receiver: On the receiver, it was observed that the size of the received packets remained constant at the predetermined value of 1472 bytes, effectively utilizing each packet to its fullest extent.

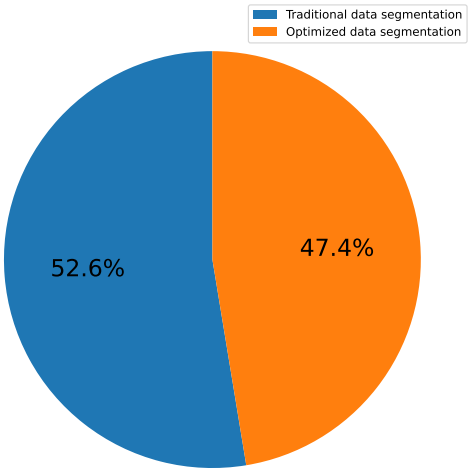

Fig. 3. Comparison of total data packet consumption

B. compression rate adaptive algorithm

In congested network environments, we conducted tests on both constant compression rates and adaptive compression

rates for video stream transmission, and subsequently evaluated the packet loss rates for each method. As illustrated in Fig:4, employing adaptive video compression rates resulted in a lower packet loss rate compared to the conventional constant compression approach for video streams.

Through the aforementioned assessments, it has been substantiated that employing packet segmentation algorithms and adaptive video compression rate algorithms can enhance the efficiency of video transmission and improve user experience (QoS) to a certain extent.

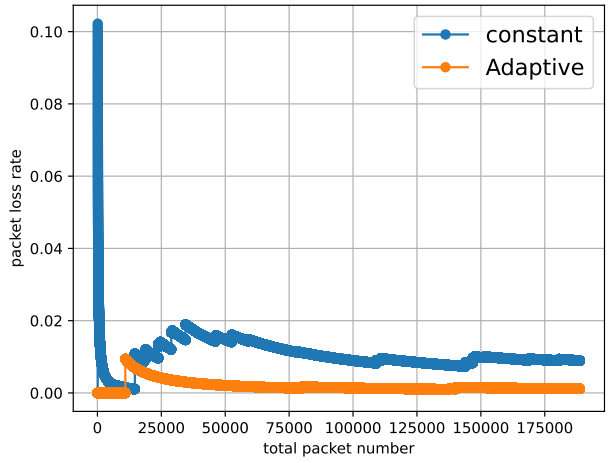

Fig. 4. Comparison of packet loss rates between constant compression rate and adaptive compression rate in the same high-voltage network environment

V. CONCLUSION

Through experimentation, it has been ascertained that augmenting packet utilization indeed yields a marginal increase in network efficiency, thereby enhancing transmission efficacy by 5% without incurring supplementary overhead. Taking the test footage as an exemplar, this improvement is evident. Nevertheless, this approach harbors certain drawbacks, notably concerning packet loss. Due to the potential amalgamation of data from two video frames within our optimized packets, a single packet loss may result in the forfeiture of two frames, thereby impeding seamless playback. Consequently, it is advocated to primarily employ this transmission method for video dissemination within local area networks.

Regarding video compression experimentation, it is concluded that under more intricate network conditions, employing adaptive video compression algorithms can effectively mitigate packet loss. However, the factors influencing compression ratio assessment based on network state are myriad, including the receiver's buffer size and the video playback buffer size. Currently, a definitive standard for evaluating network state is lacking, thus prompting a desire to employ machine learning models for prognosticating current network conditions and categorizing them based on their severity, utilizing extant network latency data.

Hitherto, our optimization endeavors have exclusively pertained to video transmission. Nonetheless, these methodologies bear relevance to other domains reliant on real-time network transmission, such as audio and file dissemination, warranting further exploration.

ACKNOWLEDGMENT

This research was supported by the MIST(Ministry of Science, ICT), Korea, under the National Program for Excellence in SW), supervised by the IITP(Institute of Information & communications Technology Planning & Evaluation) in 2022"(2022-0-00964). This research was supported by the MIST(Ministry of Science, ICT), Korea, under the National Program for Excellence in SW), supervised by the IITP(Institute of Information & communications Technology Planning & Evaluation) in 2022"(2022-0-00964)

REFERENCES

[1] C. Li, L. Fang, S. Sun, T. An, and C. Wang, "The optimal udp data length in ethernet," in *Signal and Information Processing, Networking and Computers: Proceedings of the 8th International Conference on Signal and Information Processing, Networking and Computers (ICSINC)*. Springer, 2022, pp. 1050–1057.

[2] Y. Yang and L. Hanzo, "Permutation-based tcp and udp transmissions to improve goodput and latency in the internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 14 276–14 286, 2021.

[3] T. Stockhammer, "Dynamic adaptive streaming over http– standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, 2011, pp. 133–144.

[4] Y. Yu and S. Lee, "Remote driving control with real-time video streaming over wireless networks: Design and evaluation," *IEEE Access*, vol. 10, pp. 64 920–64 932, 2022.