Reducing Latency in Interactive Live Video Chat Using Dynamic Reduction Factor

YangXin Zhao [*], Anfu Zhou [*], Xiaojiang Chen[†]
{zhaoyangxin, zhouanfu}@bupt.edu.cn, zhongsheng.cxj@taobao.com
[*]Beijing University of Posts and Telecommunications, Beijing, China
[†]Tao Bao (China) Software Co.,Ltd, Hangzhou, China

***Abstract*—In best-effort packet networks, a buffer is commonly used to eliminate network jitter and enable smooth video playback at the expense of additional delay (*i.e.* jitterdelay). In this work, we examine how jitter buffer performs in the Web Real-Time Communications (WebRTC), which is the de-facto standard used in interactive multimedia applications. We collect a dataset from a live video streaming service provider, which adopts WebRTC. After an in-depth analysis of the dataset, we find that jitter buffer can dynamically adjust the jitterdelay but is too conservative, resulting in a very slow decline of jitterdelay. To address the issue, we analyze the control logic of the jitter buffer and find that the reason lies in the use of a fixed reduction factor, known as psi (ψ). We propose an enhanced jitter buffer adaptation mechanism called JTB-ψ, which dynamically adjusts ψ according to frame size and frame duration, to reasonably speed up the decline of jitterdelay. Practical testbed experiments show that JTB-ψ achieves a 41.5% lower jitterdelay and improves receiving frame rate, quantization parameter (QP) and sending bit-rate under different network conditions, compared to the fixed-ψ approach.**

***Index Terms*—WebRTC, Live Video, Jitter Buffer, Jitterdelay, Low-latency**

I. Introduction

Recent years have witnessed the rapid development of the real-time audio and video field. Video content providers need to provide a seamless multimedia experience across different client devices. WebRTC provides real-time audio and video processing functionality for the implementation of PeerConnection and is supported by IETF RTCWEB Working Group [1] to standardize protocols and APIs. WebRTC can realize unified real-time communications across multiple web browsers, platforms, and devices, and has been provided in mainstream browsers and mobile platforms. At the same time, WebRTC is adopted by many applications, such as Google Hangouts, Whatsapp and Facebook Messenger. Despite the rapid development of WebRTC in the world, improving WebRTC's quality of service (QoS) under a best-effort network is still a major challenge that needs to be addressed [2].

The real network environment is usually complicated, resulting in packets arriving at the receiving end with different delays (*i.e.* jitter). If each packet received is played out immediately, as long as the network changes slightly, the video will accelerate or decelerate or even get stuck. We need to buffer the incoming packets for a smoother video, which introduces a latency (called jitterdelay). A greater jitterdelay will do a better job of eliminating the effects of network jitter, but it will increase network latency and reduce the real-time performance of the video. According to the ITU-T G.114 [3] standard, a target of 150ms one-way latency is critical in interactive RTC, so jitter buffer should increase or decrease the jitterdelay as needed. However, user feedback from the live video streaming service provider indicates that sometimes there is a sense of latency even when the network is in good condition. The issue is the motivation for this work.

In this paper, we use a fraction of our dataset, the filtered 16129 live streaming sessions to analyze the performance of the jitter buffer. The data shows that even if the network quality is good, the jitterdelay drops very slowly after the sudden increase, which is very unnecessary. Jitter buffer evaluates the jitterdelay from two aspects, the transmission time variation, and the network jitter. The former is affected by frame size differences. Maximum frame size L_{max} is maintained from the beginning of the session. When the current frame size is greater than L_{max}, the current frame is considered to be a large frame, otherwise L_{max} decreases at a fixed rate. From the design idea, jitter buffer only considers that the jitterdelay increases rapidly with the arrival of large frames and changes with the network jitter, without considering the fast and reasonable reduction of jitterdelay. However, we hypothesize that since the fixed ψ cannot adapt to the current relevant information, adapting ψ dynamically should lead to better performance.

For better performance of jitter buffer, we propose a model JTB-ψ for adapting ψ, the reduction factor of L_{max}, allowing L_{max} to decrease at a faster rate and thus reducing jitterdelay quickly. It is worth noting that JTB-ψ does not affect the impact of network jitter on jitterdelay. We design ψ to adjust dynamically according to two aspects: (i) the difference in size between the large frame and the normal frame, which indicates the amount of the large frame that needs to be decreased in size and determines the initial value of ψ. The difference is larger, the initial value is smaller, making the jitterdelay drop faster. (ii) the duration of the large frame, which causes ψ to slowly decrease with the duration of the large frame.

We implement our model in the WebRTC framework and conduct a series of experiments. Since there are no other algorithms to improve the jitter buffer of WebRTC, we compare the experimental results with the default jitter buffer. The results show that the overall average jitterdelay is reduced by approximately 41.5%. However, under weak networks, the performance of JTB-ψ is more prominent.

978-1-7281-3106-1/20/$31.00 ©2020 IEEE

The rest of the paper is structured as follows: Section.II reviews the relevant literatures on latency control for real-time video communications. Section.III introduces the adjustment of jitterdelay and describes our model JTB-ψ. Section.IV analyzes the network quality of collected trace dataset and also presents our implementation and custom-built testbed. We evaluate JTB-ψ in Section.V and conclude this paper in Section.VI.

II. Related Work

To provide the best quality of service (QoS), real-time applications should have high throughput and low latency while not incurring congestion, which is a huge challenge over the error-prone Internet [4], [5]. At present, research on WebRTC in academia and industry [6], [7] mainly focuses on congestion control and QoS measurement, but no further research and optimization for the jitter buffer mechanism in WebRTC. In particular, the IETF working group RMCAT designed three congestion control algorithms on top of RTP. The goal of these algorithms is to converge to the maximum available bandwidth while maintaining the queuing delay as low as possible.

Reducing latency is important for the QoS of real-time applications. We find that in other real-time applications, many methods have been proposed to reduce latency. In addition to reducing the queuing delay, the coding latency [8] and buffer latency can also be reduced. VPAP [9] dynamically calculates the minimum possible playback buffer required according to the estimated network throughput and allows an earlier start to playback while avoiding buffer under-runs. Dynamic jitter buffers adapt quickly to changing conditions and predict how long it should wait for packets before considering them lost. Many researchers have studied in jitter management areas. Literature [10]–[12] use current network condition (bitrate, delay variation) of received packets to evaluate the suitable latency. Literature [13] proposes an adaptive jitter buffer control algorithm based on network traffic prediction.

The jitter buffer in the WebRTC framework has the process of reducing buffer latency (jitterdelay), but it reduces the jitterdelay by a fixed ratio, which is too conservative and unnecessary. Inspired by the above literature, especially VPAP, it minimizes the playback buffer while avoiding playback interruptions. We aim at reducing the jitterdelay reasonably and quickly and propose a model JTB-ψ to adjust the falling speed of jitterdelay dynamically.

III. JTB-ψ Design

In this section, we introduce the jitter buffer used in WebRTC and focus on how to adjust the jitterdelay. The jitter buffer collects and constructs incoming media packets. Once a frame is complete, the buffer estimates how long the frame should wait in the buffer before passing it to the decoder and playback. The waiting time is jitterdelay, which affects end-to-end latency and session quality. If the jitterdelay is too large, the real-time performance will be degraded. If the jitterdelay is too small, the eliminating effect on jitter will be weakened and the video playback will be unstable. Latency is increasingly

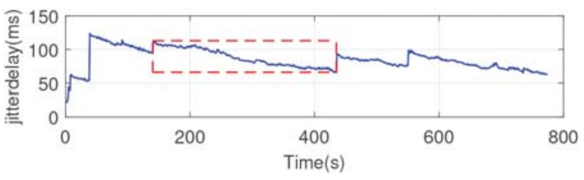

Figure 1. Drawbacks of the Jitter Buffer

becoming a performance bottleneck for RTC applications. It is necessary to adjust the jitterdelay reasonable. Next, We first illustrate the performance issue of jitterdelay control in WebRTC under the practical application. We then locate the problem by analyzing the control logic for jitterdelay in the WebRTC codebase. Finally, we design a new model to solve the problem.

A. Jitterdelay Problem in WebRTC

Jitterdelay is greatly affected by network jitter. We first observe the traces under good network conditions to understand the trend of jitterdelay. For a better description, we select a typical trace from the dataset as an example. We extract the first 800 seconds of this session. The network quality of this session is quite good, with an average RTT of 26 ms and zero loss, but the jitterdelay is not particularly good as shown in Figure 1. As can be seen from the dotted box in the figure, it takes nearly 300s (*i.e.*, from 140s to 435s) to reduce the jitterdelay by only 45 ms. Our view on this phenomenon is that under such a good network, the jitterdelay drops too slowly and it should recover to the average delay more quickly. To find out the root cause of the issue, we analyze the logic of jitterdelay adjustment below.

B. Locating the Root Cause for the Problem

Jitterdelay is considered to have two components: (i) the transmission time variation and (ii) network jitter $J(t_i)$, both units are milliseconds. The former adapts to frame's size difference and link capacity. The calculation of jitterdelay in the source code is as follows:

$$jitterdelay = \frac{L_{max} - L_{avg}}{C(t_i)} + J(t_i) \tag{1}$$

where L_{max} is *the maximum frame size received since the beginning of the session*, L_{avg} is *the average frame size of current time*, $1/C(t_i)$ is the current *the bottleneck link capacity*, the unit of $1/C(t_i)$ is B/ms. The factors $1/C(t_i)$ and $J(t_i)$ are directly and indirectly obtained by the Kalman filter [14]. Below we focus on the impact of frame size on jitterdelay.

The effect of frame size: It can be seen from Eq.(1) that the jitterdelay is affected by L_{max}. Large-sized frames require more time to travel through the link than small-sized frames. This reflects the jitter of the frame size will affect the jitter of the arrival delay. To eliminate this part of jitter and smooth the jitterdelay, jitter buffer does not consider the difference between the size of the current frame and the previous frame but considers the difference between the maximum frame size L_{max} and the average frame size L_{avg}, the maintenance of

Table I
EFFECTS UNDER DIFFERENT FIXED ψ

ψ	$0.9999(original)$	0.9995	0.999	0.998
Iteration times	13863	2772	1386	693
Time(s)	554	110	55	27

Algorithm 1: Dynamic Reduction Factor ψ

```
Input: $L_{max}$, $L_{avg}$, $D_t$
Output: The reduction factor of $L_{max} : \psi$.
$p = (L_{max} - L_{avg})/L_{max}$;
$k_l \leftarrow$ decline ratio based on $p$ from Eq.(5);
while $L_{max} \times \psi > L(t_i)$ do
    $k_t(t_i) \leftarrow$ decline ratio based on duration from Eq.(6);
    $L_{max} \leftarrow$ update from Eq.(3);
    $\psi \leftarrow$ final ratio from Eq.(4);
end
```

these two values will be described later. When a frame arrives, the average frame size L_{avg} is updated only if the frame size is within two standard deviations of a normal distribution. The weight ϕ is fixed at 0.997, and the update of L_{avg} is as follows:

$$L_{avg} = \phi \times L_{avg} + (1 - \phi) \times L(t_i) \tag{2}$$

where $L(t_i)$ is the current frame size. Unlike L_{avg}, L_{max} is updated every frame. When $L(t_i)$ is greater than $L_{max} \times \psi$, L_{max} will be updated to $L(t_i)$, then we consider the current frame as a large frame, otherwise, it will decrease by a fixed ratio of ψ. But $\psi = 0.9999$, which means that after a large frame appears, L_{max} will take a long time to fall to L_{avg}.

$$L_{max} = MAX(\psi \times L_{max}, L(t_i)) \tag{3}$$

C. *Solving the Problem*

1) Basic Idea: From the above analysis of the jitterdelay, the main reason for the slow decline in jitterdelay is the fixed decline rate of ψ, which is set to be 0.9999. In this scenario, L_{max} has to decrease at a fixed rate ψ, regardless of the size and duration of a large frame, causing L_{max} to slowly decrease, then leads to a slow decline in jitterdelay. Intuitively, using a fixed ψ is not optimal. Dynamically adjusting the ratio ψ should effectively accelerate the decline of jitterdelay and have a positive impact on overall performance.

2) Algorithm Design: We note that ψ is affected by two factors: the difference in size between L_{max} and L_{avg} and the duration of the large frame. These two factors determine the value of k_l and $k_t(t_i)$, respectively, and their values belong to $(0, 1]$. In particular, the adaptive ψ is defined as follows:

$$\psi(t_i) = \alpha \times (k_l + k_t(t_i)) + \psi_{min} \tag{4}$$

where α is the weight of the two factors, ψ_{min} is the lower limit of ψ. The determination of the parameter values, k_l and $k_t(t_i)$ will be explained in detail below.

Determination of parameter values: We determine ψ_{min} by comparing the effects of different ψ. Taking the trace in Sec.III-A as an example, the average values of L_{max} and L_{avg} are 60 KB and 15 KB, respectively, and the average receiving frame rate is 25 fps. We set different ψ and record the number of iterations and time required from L_{max} down to L_{avg}. As we can see from Table I, the values of these two metrics are too large under the original value (0.9999), and they will drop sharply as ψ decreases. We can infer that the smaller ψ is better, but if we set ψ to 0.998, this will cause the jitterdelay to drop too fast and possibly increase the jitter frequency. Finally, we set ψ_{min} to 0.999 and set 0.9999 to the upper limit. Because the upper limits of k_l and $k_t(t_i)$ are both 1, in order to adjust ψ within the interval $(0.999, 0.9999]$, we take α as 0.0005.

Effect of frame size: k_l reflects the extent to which L_{max} will be reduced, affecting the initial value of ψ. k_l is updated according to the magnitude of the difference between L_{max} and L_{avg}, which is defined as $p = (L_{max} - L_{avg})/L_{max}$, $p \in [0, 1)$. Since the natural exponential function e^x does not require additional factors, when $x \le 0$, $e^x \in (0, 1]$. The calculation of k_l is as follows:

$$k_l = e^{-p} \tag{5}$$

$k_l \in (1/e, 1]$, k_l decreases with the increase of p. Therefore, the larger the difference between L_{max} and L_{avg}, the smaller the value of k_l, allowing ψ to be 0.9997 or 0.9995 rather than using 0.9999 in the fixed ψ case. In other words, k_l allows a significant reduction in L_{max} at a smaller initial rate, which leads to faster jitterdelay reduction.

Effect of frame duration: It is natural to know that the rate of decline needs to increase as *the duration of large frame* increases, since L_{max} will get closer and closer to L_{avg}. D_t records the duration of the large frame in seconds, then $D_t/60$ indicates how many minutes the large frame has lasted. Based on this idea, we get the formula for $k_t(t_i)$ as follows:

$$k_t(t_i) = e^{-D_t/60} \tag{6}$$

Note that $k_t(t_i)$ is frame-independent, so when the next large frame has not yet appeared, L_{max} continues to drop, and the reduction of ψ is the same at the same D_t.

Here we can get the overall design of JTB-ψ as shown in Alg.1. It is noteworthy that $k_t(t_i)$ decreases over time but k_l is not updated until the next large frame appears. In other words, during the descent of L_{max}, only the initial value of ψ is different, which is affected by k_l, and then decreases due to the existence of $k_t(t_i)$.

IV. DATASET AND TESTBED

Here we analyze the quality of general network conditions in the collected dataset and introduce the experimental testbed for comparative evaluation.

A. *Dataset Analysis*

With the permission of the live video chat service provider, we visit its internal data analysis platform and crawl its global

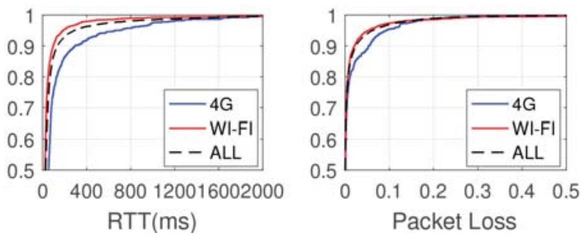

Figure 2. RTT and Loss CDF Compare

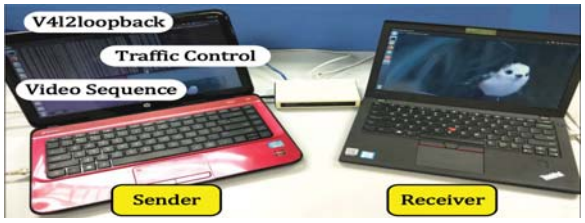

Figure 3. Testbed Employed for the Experimental Evaluation

Table II
EXPERIMENTAL EVALUATION METRICS

$Metric$	$Definition$
Delay (ms)	jitterdelay, frame buffer time.
L_{max} (KB)	current maximum frame size.
D_t (s)	duration of the large frame.
ψ	reduction factor of L_{max}.
QP	degree of image compression.
RTT (ms)	round-trip time of a packet.
Loss	the proportion of packets lost.
Bitrate (Mbps)	the rate of data that the sender sends.
FPS	the rate of receiving frame at receiver.

live data. For each session, the trace records streaming traffic data per second, such as RTT, packet loss rate and jitterdelay. We run the crawler and capture traffic data of all live sessions on June 29th, 2018. To present the overall trend of jitterdelay, we screen out sessions that are longer than 10 minutes, and finally, collect 16129 sessions for a total of 2537390 hours.

Session Quality Analysis: Some measurements are performed to learn the network quality of the selected sessions. According to statistics, about 75% of the sessions use Wi-Fi and the rest use 4G. We take the average RTT and average packet loss rate for each session as quality criteria and analyze their cumulative distribution under 4G, Wi-Fi, and all sessions. As illustrated in Figure 2, it can be observed that in all sessions, 90.9% have an average RTT less than 150 ms and 88.5% have an average packet loss below 2%. The probability that average RTT less than 150 ms of 4G and Wi-Fi sessions is 82.4% and 93.5%. Similarly, the probability that average packet loss below 2% of 4G and Wi-Fi sessions is 84.0% and 89.6%. Therefore, we can infer that: (i) the majority of the sessions have a good network quality, (ii) Wi-Fi sessions have better communication quality than 4G sessions.

B. *Exprimental Testbed*

Figure 3 shows our testbed, which is composed of two computers (*i.e.*, a pair of sender and receiver) connected through a wired network with zero loss and 1-3 ms average RTT. In the WebRTC source code, we configure the upper limit of the sending bit-rate to 2 Mbps to match the traces and add the logic of dynamically adjusting ψ in the jitter buffer. We compile WebRTC on the Ubuntu Linux operating system. For the reproducibility of the experiment, we select a publicly available video named Piper [15] from Youtube lasting about six minutes and transmit this video sequence in the experiment. The sender is a workstation and undertakes the following tasks: 1) It uses Traffic Control [16] to control the RTT and packet loss rate based on the traffic data of the dataset to simulate the real network environment. 2) It uses V4l2loopback [17] to create a virtual video device. Model JTB-ψ is implemented in the receiver to adjust the value of ψ and log the metrics of the jitter buffer. To evaluate the performance of JTB-ψ and the original jitter buffer (JTB), we compute the metrics for each session as shown in Table II. It is noteworthy that the quantization parameter (QP) regulates how much spatial detail is saved and ranges from 0 to 51. The smaller the QP, the more details are retained and the video quality is higher.

V. PERFORMANCE EVALUATION

To evaluate JTB-ψ, we use the testbed described in Sec.IV-B and randomly select 100 sessions from the dataset, of which 4G and Wi-Fi sessions each accounted for half. For each session, we use the first six minutes to simulate the real network environment and compare these two models with the metrics defined in the previous section. For convenience, we use JTB (Jitter Buffer) to represent the original model.

A. *Performance Evaluation*

To analyze the overall performance of the two models, we calculate the average of the corresponding metrics for the 100 sessions. Figure 4 shows the average of jitterdelay (Delay), receiving frame rate (FPS), quantization parameter (QP) and sending bit-rate (Bitrate) generated by JTB (in blue) and JTB-ψ (in yellow). Compared with JTB, the jitterdelay drops from 149.76 ms to 87.64 ms, by about 41.5%, while the other three metrics do not change much. The receiving frame rate and the sending bit-rate increase by 2.14 fps and 0.26 Mbps. The QP drops by 0.36, which means more spatial data is retained. Overall, JTB-ψ can improve real-time video quality.

Exploring the scenario with better performance: To better understand the performance advantage of JTB-ψ, we obtain the average jitterdelay of both JTB and JTB-ψ models for each session, and the difference between the two values is the jitterdelay descent. We plot the cumulative distribution of jitterdelay descent according to 4G and Wi-Fi type, as shown on the left side of Figure 5. It is observed that the average jitterdelay descent in 4G sessions (91.3 ms) is higher than that in Wi-Fi sessions (62.1 ms), and the 20% jitterdelay drops by more than 96 ms and 156 ms under 4G and Wi-Fi sessions, respectively. Combined with Figure 2, it further shows that the

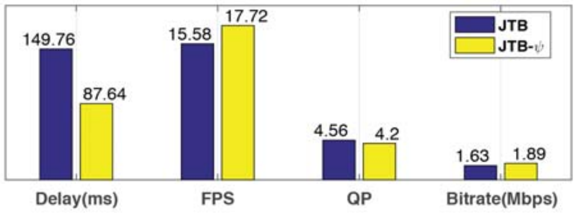

Figure 4. The Overall Average of Metrics Compare

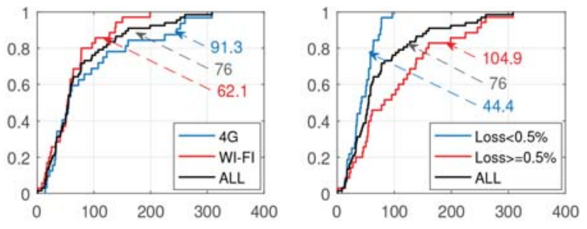

Figure 5. The CDF of Jitterdelay Decline Classified by Signal Type (Left) and Network Packet Loss Rate (Right)

4G signal is weaker than Wi-Fi, and we infer that the jitterdelay decreases more in weak networks. Then, we choose the packet loss rate to reflect the network quality and calculate the loss distribution of all 100 sessions, and the average loss of 50% sessions is less than 0.5%. We divide the 100 sessions into two groups according to the 0.5% loss and plot the right figure in Figure 5. The figure shows that the jitterdelay of the 100 sessions (in black) is reduced by 76 ms, and for sessions with an average loss less than 0.5% (in blue) and greater than 0.5% (in red), the jitterdelay is reduced by 44.4 ms and 104.9 ms. The result indicates that jitterdelay decreases more significantly for sessions with a higher packet loss rate, and the CDF of jitterdelay descent based on packet loss rate is more obvious than that based on signal type.

From the above analysis, we conclude that the proposed dynamic ψ mechanism effectively reduces the jitterdelay without affecting video quality, and has better performance in weak networks.

B. Detailed Performance Comparison

To better illustrate the effect of the dynamic ψ on the performance of JTB-ψ, we select two typical sessions from the experiments, one under strong network with 25 ms average RTT and zero loss, and one under unstable weak network with 40 ms average RTT and 2.9% loss, as shown in Figure 6. Figure 7 illustrates the jitterdelay, the reduction factor ψ, the maximum frame size L_{max} and the duration of large frame generated by JTB (in blue) and JTB-ψ (in red), while Figure 8 shows the other metrics (quantization parameter, receiving frame rate and sending bit-rate) for the two sessions.

Jitterdelay comparison: From the similarity of the four metric trends between strong and weak networks in Figure 7, we observe that 1) JTB-ψ has a lower jitterdelay than JTB, In the strong network, the average value of jitterdelay drops from 106.43 ms to 92.16 ms, a decrease of 13.4%. In the weak network, the average value of jitterdelay drops from 242.65 ms to 105.32 ms, a decrease of 56.6%. 2) dynamic ψ decreases as D_t increases, then JTB-ψ has a sharper drop in L_{max} and jitterdelay. For instance, in the case of a strong network session, during the period from $T1 = 190s$ to $T2 = 360s$, JTB took 170 s to reduce L_{max} by 20 KB and the jitterdelay by 45 ms, while JTB-ψ spent 125 s to reduce L_{max} by 50 KB and the jitterdelay by 120 ms.

QoS comparison: Figure 8 shows the curve of the other three metrics overtime in the strong and the weak networks, reflecting the video quality, from which we can extract the following information: 1) The average value of QP under the JTB-ψ does not change much compared with the JTB, which is reduced by 1.6 in the strong network and increased by 0.1 in the weak network. 2) In the strong network, the value and range of the receiving frame rate are basically the same as that of JTB. In the weak network, although the frame rate is slightly reduced by 2 fps under JTB-ψ, the jitter of the frame rate is reduced. 3) In the strong network, the sending bit-rate under JTB-ψ remains at the upper limit (2 Mbps). In the weak network, the sending bit-rate under JTB drops frequently due to the large jitter of RTT and packet loss rate. However, JTB-ψ suppresses the drop of the sending bit-rate and increases the average sending bit-rate from 1.16 Mbps to 1.66 Mbps. It can be seen that in the strong network, the overall performance of both JTB and JTB-ψ is similar, and the QP under JTB-ψ is improved. In the weak network, the performance of QP and the receiving frame rate is slightly degraded, with little effect, but the sending bit-rate is greatly improved.

In summary, JTB-ψ adaptively adjusts the reduction factor ψ, remarkably reduces jitterdelay under various network conditions and improves other metrics to varying degrees. In particular, JTB-ψ performs more prominent under weak networks.

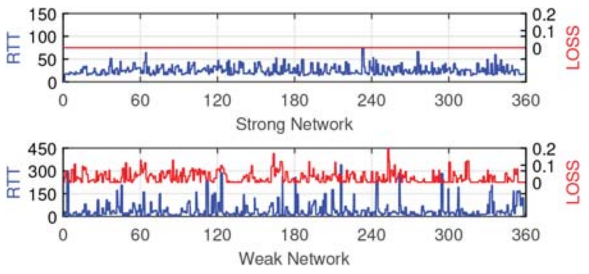

Figure 6. Two Typical Network Traces

VI. Conclusions

In this paper, we have investigated the performance issue of jitter buffer for low-latency interactive live video chat applications. We have analyzed the control logic of jitter buffer and a real trace dataset from a live video service provider, which proves that it is necessary to use an adaptive ψ in jitter buffer to speed up the decline of jitterdelay. To address the issue, we have proposed a model JTB-ψ to dynamically set the reduction factor ψ and experimentally compared with the default jitter buffer. The results show that model JTB-ψ can improve live video performance under various network conditions.

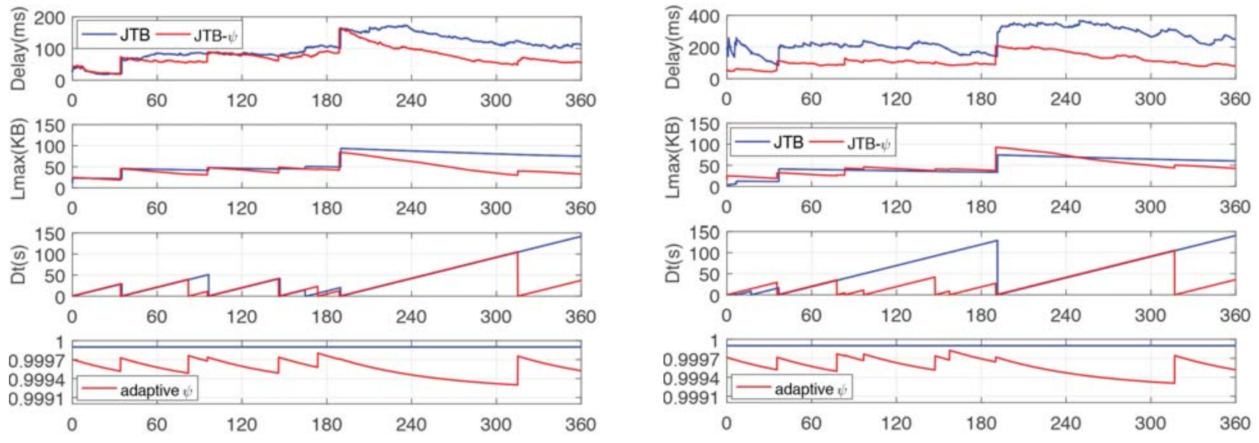

Figure 7. The Change Trend of Jitterdelay, L_{max}, D_t and psi Under Strong Network (left) and Weak Network (right). The Average Jitterdelay in JTB-ψ is Decreased by 13.4% and 56.6% Under the Two NetWorks, respectively.

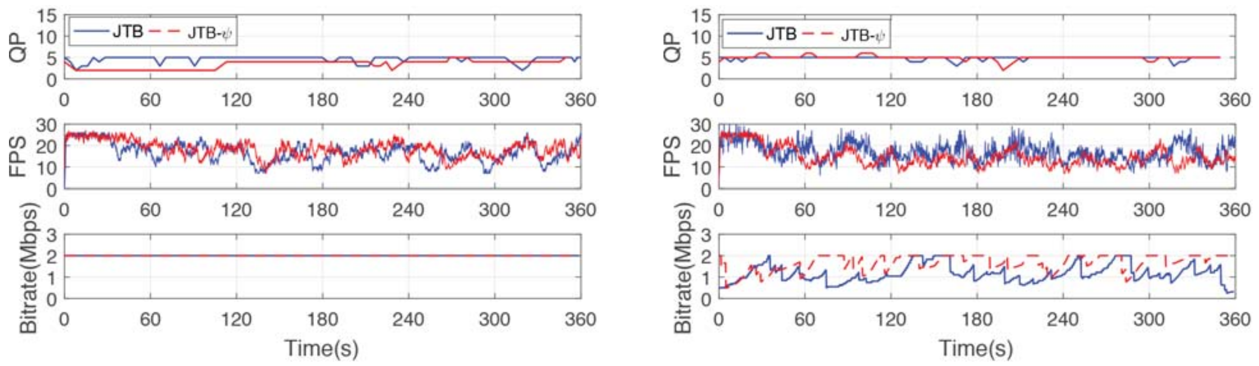

Figure 8. The Change Trend of quantization parameter, Receiving Frame Rate and Sending Bit-rate Under Strong Network (left) and Weak Network (right). The Average Sending Bit-rate in JTB-ψ is Increased by 43.1% Under the Weak Network, and Other Metrics are Not Changed Much.

VII. Acknowledgment

The work reported was supported by National Key R&D Program of China 2017YFB1003000, National Natural Science Foundation of China (61772084, 61722201, 61720106007), the 111 Project (B18008) and Alibaba Innovative Research (AIR) program.

References

[1] "Webrtc1.0," 2019. [Online]. Available: https://www.w3.org/TR/webrtc/

[2] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2149–2196, 2016.

[3] ITU, "Itu-t, 2003. recommendation g.114 – one-way transmission time." 2003. [Online]. Available: http://www.itu.int/rec/T-REC-G.114/en

[4] Y. Song, L. Liu, H. Ma, and A. V. Vasilakos, "A biology-based algorithm to minimal exposure problem of wireless sensor networks," *IEEE Transactions on Network and Service Management*, vol. 11, no. 3, pp. 417–430, Sep. 2014.

[5] L. Liu, Y. Song, H. Zhang, H. Ma, and A. V. Vasilakos, "Physarum optimization: A biology-inspired algorithm for the steiner tree problem in networks," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 818–831, March 2015.

[6] L. Wu, A. Zhou, X. Chen, L. Liu, and H. Ma, "Gcc-beta: Improving interactive live video streaming via an adaptive low-latency congestion control," in *ICC*, 2019.

[7] A. Zhou, H. Zhang, G. Su, L. Wu, R. Ma, Z. Meng, X. Zhang, X. Xie, H. Ma, and X. Chen, "Learning to coordinate video codec with transport protocol for mobile video telephony," in *MobiCom*, 2019.

[8] M. Mody, P. Swami, and P. Shastry, "Ultra-low latency video codec for video conferencing," in *IEEE International Conference on Electronics*, 2014.

[9] L. Ariyasinghe, Z. Huang, Haibo Zhang, and D. Eyers, "Vpap: Vbr pattern aware playback buffering for video streaming," in *ITNAC*, Dec 2016, pp. 53–58.

[10] L. Repele, R. Muradore, D. Quaglia, and P. Fiorini, "Improving performance of networked control systems by using adaptive buffering," *IEEE TIE*, vol. 61, no. 9, pp. 4847–4856, Sep. 2014.

[11] U. Premaratne, "Empirical network jitter measurements for the simulation of a networked control system," in *ICTer*, Dec 2014, pp. 235–240.

[12] H. Dbira, A. Girard, and B. Sansò, "On the relationship between packet jitter and buffer loss probabilities," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, Sep. 2016, pp. 95–100.

[13] M. Yuan, "Jitter buffer control algorithm and simulation based on network traffic prediction," in *International Journal of Wireless Information Networks*, vol. 26, Sep 2019, pp. 133–142.

[14] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2629–2642, Oct 2017.

[15] "Piper." [Online]. Available: https://www.youtube.com/

[16] B. Hurbert, "Linux advanced routing & traffic control." [Online]. Available: http://lartc.org/howto

[17] "A kernel module to create v4l2 loopback devices." [Online]. Available: https://github.com/umlaeute/v4l2loopback