Reducing First-Frame Delay of Live Streaming by Simultaneously Initializing Window and Rate

Bo Wu[†], Tong Li[‡]*, Cheng Luo[†], Xu Yan[‡], Fuyu Wang[†], Changkui Ouyang[†], Linfeng Guo[†], Haiyang Wang[§], and Ke Xu[¶]
Tencent[†], Renmin University of China[‡], University of Minnesota at Duluth[§], Tsinghua University[¶]

Abstract— **The first-frame delay is an essential indicator for evaluating the performance of cloud CDN vendors and affects the client-side QoE of live streaming. Instead of the traditional way of tuning the initial congestion window (cwnd) for all connections to a fixed value based on expert experience, this paper explores the using of transport signals unique to each connection (e.g., application-layer framing, historical QoS metrics) to initialize the sending parameters for each connection. Thus we propose Wira, a first-frame optimization mechanism that adjusts both initial cwnd and initial rate, which are two key parameters for decreasing the first-frame completion time (FFCT). Particularly, Wira provides cross-layer *Frame Perception* that parses frames and adapts the initial cwnd to the first-frame size. Meanwhile, Wira introduces the *Transport Cookie* to enable cloud-client collaborations, in which the historical QoS metrics from the clients can be reported and reused by rate initialization in the stateless cloud. This assures the initial rate matches the actual network conditions while avoiding non-trivial storage overhead in the cloud. We implement Wira upon QUIC and evaluate it via real-world deployments of commercial services. Results demonstrate the profitability of Wira, in which the average and 90th-percentile FFCT are reduced by 10.6% and 16.7%, respectively.**

Index Terms—**Transmission Control, First Frame Optimization, Frame Perception, Transport Cookie**

I. Introduction

The live-streaming services have become a critical part of our lives [1], [2], whose first-frame delay reflects the client-side waiting time from sending out the request packet to displaying the first screen. For example, the TikTok Live users served by our provided CDN service in Southeast Asia, have to wait 200ms∼400ms for the first-frame streaming. This delay is always regarded as an essential metric to evaluate the performance of CDN vendors, whose larger value will deteriorate the quality of experience (QoE) and decrease the revenue of both application providers (e.g., Twitch and TikTok Live) and CDN vendors (e.g., Amazon AWS and Google Cloud). By contrast, if the incurred first-frame delay is larger than the threshold (e.g., 1s) that is declared by application providers, the live-streaming users tend to leave the live room or even close the application.

This work was supported in part by the NSFC project under No. 62202473, the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under No. 62221003, the Key Program of the National Natural Science Foundation of China under No. 61932016 and No. 62132011, the National Science Foundation for Distinguished Young Scholars of China under No. 61825204, University of Minnesota Duluth, Grant-In-Aid 2024-2026, and funding from Tencent. Tong Li is the corresponding author (tong.li@ruc.edu.cn).

The existing schemes primarily emphasize the development of improved control policies to reduce the first-frame completion time (FFCT). Specifically, the focus is on setting up the correct congestion window (cwnd), with particular emphasis on its initial value (init_cwnd), to enhance the sender-side responses during the startup phase [3]–[8]. However, these approaches employ fixed parameter settings for all users, failing to adapt to the diversity of first-frame sizes (FF_Size), which vary from 6KB to 250KB based on our comprehensive measurements (§II-A). Generally, a smaller init_cwnd may introduce more Round-Trip Time (RTT) for the first-frame delivery, while a larger one can trigger congestion due to increased in-flight traffic data. Besides, these approaches overlook the significant impact of the pacing rate on FFCT, despite the proven benefits of pacing-based congestion controls such as BBR [9], TIMELY [10] and Copa [11]. In this paper, we argue that the initial pacing rate (init_pacing) should also be carefully configured for the FFCT optimizations, as evidenced by our testbed experiments(§II-B). For example, a higher init_pacing can lead to unacceptable packet losses, thus elongating the time required for successful recovery [12], [13]. Based on these observations, we infer that a finer-grained initialization of both cwnd and pacing rate is essential for minimizing first-frame delays.

To set parameters dynamically, some solutions employ user-group divisions and train machine-learning (ML) based models for each user group [14]–[24]. The central concept is to treat the network condition of the entire group as the condition encountered by each user within the group. However, we argue that these ML-based ways are coarse-grained and costly. **First**, the FF_Size of each flow varies within a user group. ML-based solutions face challenges in achieving per-flow parameter configurations due to the unknown FF_Size. **Second**, the group-based Quality of Service (QoS) estimation is less than ideal (§II-C). This is because a QoS metric (e.g., minimum RTT) of the whole user group only reflects the overall level of samples. Our large-scare measurement study has shown that the historical QoS (Hx_QoS) metrics, such as the minimum RTT and the maximum available bandwidth measured in the last connection between the same origin-destination (OD) pair, have a much lower dispersion degree than the group-based QoS metrics (§II-D). Thus, for a specific OD pair that is generating a new flow, Hx_QoS is more worthy of reference than the group-based QoS. **Third**, the frequentlyleveraged ML model, for each user group [15], will introduce non-trivial overhead (e.g., higher CPU load), especially when facing a large number of user groups (e.g., 8000+ in Brazil [25]), which also limits the real deployability of these methods.

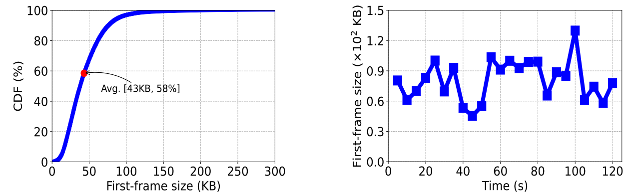
Some transport signals unique to each connection including FF_Size [26], [27] and Hx_QoS [28], [29] can be utilized for initializing sending parameters for each connection. In this paper, we propose `Wira`[1], a first-frame optimization mechanism that combines both FF_Size and Hx_QoS to enable effective initializations for cwnd and pacing rate. In particular, init_cwnd is set by referring to the actual FF_Size while the init_pacing is configured based on OD-pair's Hx_QoS. However, to achieve the mentioned initializations, several challenges that cannot be ignored should be addressed, as follows. **First**, the transport layer does not have awareness of the specific information at the application layer. As a result, the current transport protocols do not inherently support the awareness of FF_Size. **Second**, maintaining the Hx_QoS records in the cloud (sender) for each OD pair would result in excessive overhead, making it impractical to quickly retrieve this information for initializing the pacing rate. **Third**, it will deteriorate the FFCT if the parameter initializations, such as initializing cwnd according to FF_Size and initializing pacing rate according to Hx_QoS, are not carefully handled.

The means of Wira to address the aforementioned challenges are also divided into corresponding three steps. **First**, Wira introduces the *Frame Perception (FP)*, a cross-layer scheme that identifies the first frame and gets FF_Size before delivering it to the sending module (§IV-A). **Second**, For fast Hx_QoS acquisition without incurring non-trivial storage overhead, Wira proposes the *Transport Cookie (TC)*, a cloud-client collaboration scheme that synchronizes Hx_QoS between the *stateless* cloud-side server and its user-side clients. Particularly, during the connection establishment, the client reports the desired Hx_QoS in the handshake packets (§IV-B). **Third**, once obtaining FF_Size and Hx_QoS, the sending module of Wira will regard them as essential signals for initializing both cwnd and pacing rate. The main objective is to ensure that the first frame can be successfully transmitted without causing excessive congestion or packet losses. This is achieved by configuring the init_cwnd parameter to an appropriate value and adapting the init_pacing rate based on the available bandwidth in Hx_QoS (§IV-C).

The contributions are summarized as follows.

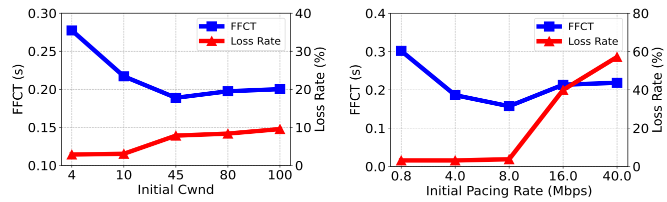
- We construct large-scale measurements and testbed experiments that demonstrate both init_cwnd and init_pacing should be highly required for per-flow's FFCT.
- We propose a first-frame optimization mechanism named Wira that can combine both FF_Size and Hx_Qos to achieve more appropriate initialization for cwnd and pacing rate.
- We introduce accurate cross-layer perceptions, whose frame parser implemented in L4 can identify the first frame and get its size. Besides, a lightweight collaboration scheme is designed for quickly obtaining Hx_QoS without incurring non-trivial overhead at the sender side.
- We implement Wira prototype upon QUIC protocol [30], [31] and evaluate it through real-world deployments for 6 months. The experimental results show the average and 90th-percentile FFCT values can be lowered by 10.6% and 16.7%, respectively. Besides, the first-frame loss rate can be reduced from 8.8% to 6.4%, on average.

[1]Wira is named after initializing window and rate, simultaneously.



(a) Inter-stream first-frame sizes (b) Intra-stream first-frame sizes.

Fig. 1. Diverse first-frame sizes in our measured live streams.



(a) FFCT vs. init_cwnd (b) FFCT vs. init_pacing.

Fig. 2. FFCT varies with init_cwnd and init_pacing.

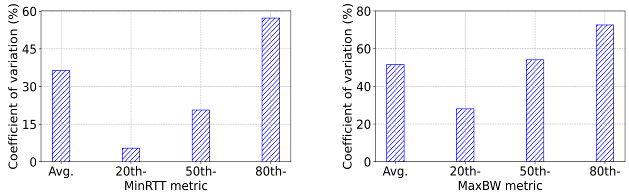
The remainder of this article is organized as follows: In §II, we describe the motivation through our performed large-scale measurements and testbed experiments. Then, the overview and design details of out proposed Wira are depicted in §III and §IV, respectively. §V discusses the implementation and §VI shows the experimental evaluation. Then §VII discusses the transport cookie security and the first-frame playback conditions. §VIII gives an overview of related works. Finally, §IX concludes the paper.

II. Motivation

In this section, we motivate Wira based on our performed testbed experiments and large-scale measurements in the real product network.

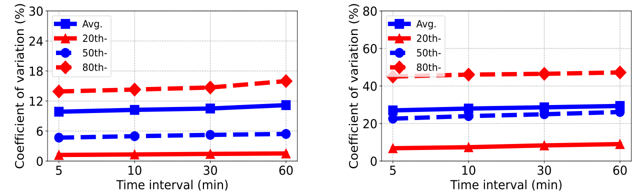
A. Diverse First-Frame Sizes Require Dynamic Initial Cwnd

Due to varying resolution ratios of different live streams, the FF_Size of different live streams is generally different. Additionally, within the same live stream, the live video picture changes over time, and the complexity of the picture affects the size of the video frame. Therefore, even when requesting play of the same live stream at different times, the FF_Size may vary. To better explore the actual FF_Size, large-scale measurements have been performed, which collected 100+ million streams of a famous live platform that is supported by our CDN service. Fig. 1(a) shows the obvious difference of inter-stream FF_Size with the average value of 43.1KB. Besides, 20% live streams (i.e., 80th-percentile value) hold



(a) MinRTT CV in the same UG. (b) MaxBW CV in the same UG.

Fig. 3. The differences of MinRTT and MaxBW within the same UG.



(a) MinRTT CV of same OD pairs. (b) MaxBW CV of same OD pairs.

Fig. 4. The differences of MinRTT and MaxBW of the same OD pairs.

the first-frame size of >60KB compared to <30KB in 30% live streams. Meanwhile, we also make testbed experiments, in which the requested live stream can be pulled from our live CDN and then will be transmitted to another server. We find the FF_Size (even in the same live stream) always changes at different viewing timestamps. Fig. 1(b) depicts the actual FF_Size values when we view some live stream every 5s, which range from 45KB to 130KB. This is affected by the complexity of the first-frame picture. Fig. 2(a) shows the FFCT of some live stream with the FF_Size of 66KB in our performed testbed experiment[2]. We can learn a smaller value (e.g., 4 and 10) will incur larger FFCT as more transmission RTTs are incurred while the larger ones (e.g., 80 and 100) can suffer from non-trivial packet losses due to network congestion. By contrast, the init_cwnd that is adapted to the FF_Size (i.e., init_cwnd = 45) will gain much better FFCT. Therefore, the actual cwnd should be dynamically initialized as the diversity of inter-/intra-stream FF_Size values.

B. Simultaneously Adjusting Initial Rate and Initial Window Helps

The pacing rate is also regarded as a significant indicator for high-performance transmissions while only configuring init_cwnd is far away from enough, in which an affable init_pacing can achieve much better first-frame optimization. Fig. 2(b) shows the FFCT under various init_pacing configurations, which is based on our performed 1000 testbed experiments with init_cwnd = FF_Size. We can learn both the smaller and the larger init_pacing values can all result in unsatisfied FFCT while the configured value that adapts to the maximum available bandwidth (MaxBW) can introduce much better FFCT. For example, the init_pacing with 0.8Mbps and 4Mbps can introduce the FFCT of 302ms and 186ms, respectively. However, the employed 16Mbps and 40Mbps will both incur the FFCT of 210ms+ and the loss rate of >40%. By contrast, 8MBps init_pacing that adapts to MaxBW can result in a much lower FFCT (i.e., 157ms) with a smaller loss rate (i.e., 3.8%). Therefore, the pacing rate should also be carefully initialized for further optimizing our focused FFCT.

C. User-Group Modes Cannot Accurately Control Each Flow

Even though user group (UG) oriented schemes (including their ML solutions) can be leveraged to optimize grouped connections (in §VIII), these methods fail to achieve more fine-grained controls for each flow. On the one hand, their enabled init_cwnd configurations are based on the measured network QoS, which cannot well adapt to the diverse FF_Sizes in §II-A. On the other hand, the actual network conditions in the same UG still present obvious differences that can be learned from our real-network measurements. In this paper, we employ coefficient of variation (CV) [32], as formula 1 shows, to depict the differences between the actual values of some QoS metric (e.g., MinRTT and MaxBW).

$$\mathsf{CV} = \frac{1}{\mathsf{N} \cdot \mathsf{v}_{\mathsf{avg}}} \cdot \sqrt{\sum_{i=1}^{N} (\mathsf{v}_i - \mathsf{v}_{\mathsf{avg}})^2} \quad (1)$$

where N is the amount of live-streaming connections, and v_i ($\mathsf{v}_{\mathsf{avg}}$) represents the (average) value of some QoS metric. As Fig. 3 shows, the average CV values (within 5mins) of both MinRTT and MaxBW of 1000+ UG[3] in Southeast Asia are 36.4% and 51.6%, respectively.

Besides, ∼50% MinRTTs have already become >20.0% while only 12.8% MaxBW values introduce 20.0%. This also shows the UG-based network estimations cannot accurately reflect the actual transmission quality between the sender and some receiver, especially compared to the Hx_QoS measurements in §II-D. Thus, initializing the pacing rate for all connections that belong to the same UG is not enough for optimizing FFCT. In addition, UG-powered control schemes might become impracticable, especially when the UG amount becomes intolerable (e.g., over 8000 UGs in Brazil [25]). In this case, deploying a DRL model for each user group will introduce huge overhead (e.g., higher CPU load), which will affect the stability of the CDN servers. Besides, the live-streaming traffic in the top 10 UGs only accounts for 11%, lacking finer-grained controls for each flow. Therefore, the UG-based or ML-powered solutions cannot be well scaled out in the real product network.

D. Network QoS Performs Similarly with The Same OD Pair

The Hx_QoS with the same OD pair outperforms UG-based network estimations. To further explore the Hx_QoS, we make large-scale measurements for over 10 million live-streaming connections with the same OD pair, in which the

[2] In this testbed experiment, the network condition is configured as 8Mbps bandwidth, 3% loss rate, 50ms RTT and 25KB network buffer.

[3] Two users will be divided into the same UG if they have the same network type (e.g., WIFI, 3G, 4G, and 5G), geographic location (i.e., country, province and city) and AS number.

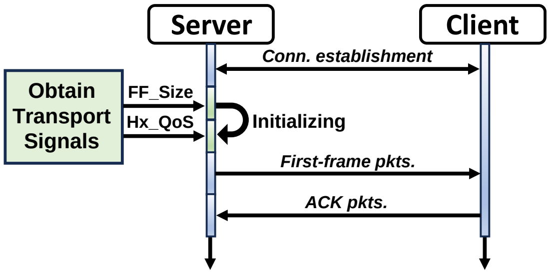


Fig. 5. The Overview of Wira.

client-side network type such as WIFI, 4G, and 5G has been considered. Fig. 4 shows the OD-pair CV values in terms of both MinRTT and MaxBW with different time intervals. We can learn the following observations. (i) The MinRTT metric of the same OD-pair session will become slightly differentiated when the time interval tends to become larger. As Fig. 4(a) shows, the average MinRTT CV values are 9.9%, 10.2%, 10.5%, and 11.2% under the time intervals (min) of (0, 5], (0, 10], (0, 30] and (0, 60], respectively. (ii) A high proportion of MinRTTs does not show significant changes under different time intervals. Within 5-min interval, ∼80% connections with the same OD pair keep the MinRTT CV metric of 13.9% while the MinRTTs of ∼80% connections can still keep insignificantly-changed (i.e., CV = 16.0%) with the interval of (0, 60]. (iii) Compared to MinRTT, the MaxBW exhibits significant differences, whose 50th-percentile CV has exceeded 22.6%, as Fig. 4(b) shows. (iv) We also find that both MinRTT and MaxBW of the same OD-pair connections are stable compared to the values in the same UG. For example, the average MinRTT and MaxBW CV values are 9.9% and 27.0% within the intervals of 5 minutes, whose changing rates are lower than 36.4% and 51.6% which are shown in the same UG. Therefore, the Hx_QoS within the same OD pair can reflect the realistic network condition more accurately compared to the UG-based network estimations.

III. Overview

In this section, we will provide high-level descriptions of Wira that aim to optimize the FFCT of live streaming by fully considering both FF_Size and Hx_QoS.

This paper proposes Wira that can take both FF_Size and Hx_QoS into full consideration for the first-frame optimizations of large-scale live streaming, which can be shown as Fig. 5. In particular, more fine-grained transmission controls for per-flow initialization can be realized by carefully configuring the initial parameters based on FF_Size and Hx_QoS. Concretely, Wira refers to the actual FF_Size and enables more appropriate init_cwnd, in which fewer RTTs will be required for first-frame deliveries. For better pacing the first frame, Wira senders leverage Hx_QoS for initializing the sending rate that adapts to the real network conditions. To optimize per-flow's first frame, Wira should follow the next design principles for further lowering the FFCT of live streaming.

Algorithm 1: First-frame parsing pseudo code.

```
Input: Live streaming
if FF_Complete then
    return -1;
end
Obtain PtlType;
if PtlType ∉ PtlSet then
    return -1;
end
Obtain HeaderLen;
FF_Size = HeaderLen;
FF_Size += PreviousTagSizeLen;
Num_VF = 0;
for each frame do
    Obtain FrameType;
    Obtain FrameSize;
    if FrameType is Video then
        Num_VF++;
    end
    FF_Size += FrameSize;
    FF_Size += PreviousTagSizeLen;
    if Num_VF == Θ_VF then
        FF_Complete = 1;
        return FF_Size;
    else
        continue;
    end
end
```

- **Principle 1: The init_cwnd should adapt to the actual FF_Size.** The smaller init_cwnd will consume more RTTs to complete the first-frame delivery while the larger one might cause a more congested transmission path due to incurring excessive in-flight packets.
- **Principle 2: The init_pacing should match the real network conditions.** The proper init_pacing can efficiently transmit the first frame to its client, whose smaller value will slow down the delivery while the larger one can introduce non-trivial packet losses that require extra time for their recoveries.

However, the following challenges are still faced when utilizing both FF_Size and Hx_QoS for optimizing per-flow FFCT. (i) L4 keeps agnostic to FF_Size due to lacking the ability to parse live streaming that is to be transmitted, in which the fixed init_cwnd cannot cover the diverse FF_Size (§II-A). (ii) The Hx_QoS is hard to gain or store locally on traffic senders for configuring highly-required init_pacing (§II-B) because the non-trivial storage overhead will be introduced, especially facing millions of live streams. (iii) The initialization should be carefully performed for both init_cwnd and init_pacing as some negative effects (e.g., larger loss rate and RTT) will be incurred if some parameter has not been better configured.

To address the mentioned-above challenges, Wira supports accurate cross-layer perceptions, in which the frame parser is designed and implemented in L4. In this case, the FF_Size can be obtained before it has been transmitted (§IV-A). For efficiently estimating the cold-start transmission condition, Wira introduces a lightweight client-cloud collaboration scheme that

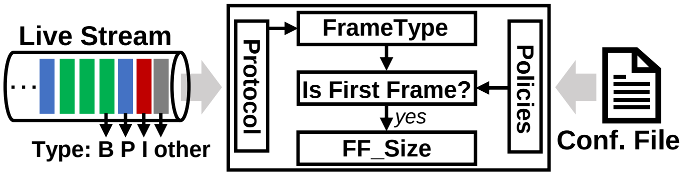


Fig. 6. The sketch of Wira's parsing module.

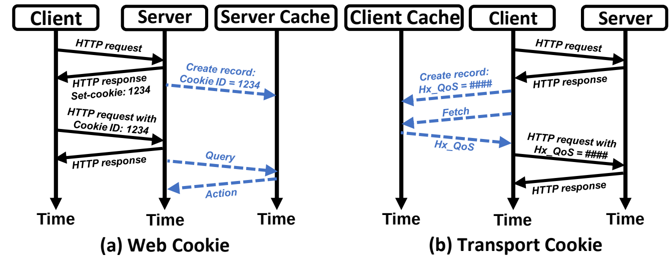


Fig. 7. The analogy between Web Cookie and Transport Cookie.

enables each request packet to carry historical information of the last sessions, further offloading server-side storage overheads (§IV-B). To better configure the initial parameters for each live streaming, Wira enables more appropriate init_cwnd based on its parsed FF_Size while initializing init_pacing according to its obtained Hx_QoS (§IV-C).

IV. Design Details

In this section, we will describe the design details of Wira that support cross-layer frame parsing, stateless transport cookie, and sending parameter initializations, as Fig. 9 shows.

A. Frame Perception

To obtain the desired FF_Size and achieve per-flow's control, Wira introduces a parsing module that enables accurate cross-layer perceptions in L4, which can identify the first frame as well as its size. The reason why we chose to perform frame perception in L4 is that it allows us to obtain the first-frame size without modifying any L7 applications that attempt to configure the initial window based on the first-frame size. In other words, Wira is transparent to the upper-layer applications. Figure 6 depicts the sketch of parsing module that is implemented in L4. When receiving a request packet from some client, the traffic sender will fetch one or more Group of Pictures (GOP) of the requested live streaming, which contain I, P, B, and other (e.g., audio and script data) frames[4]. Then, these fetched frames will be input into the parsing module before they are sent out. Finally, the desired FF_Size will be output to the sending module (described in §IV-C). The parsing process is as Algorithm 1 shows.

When receiving a live stream, the Wira parser will firstly determine whether the FF_Size obtaining has been completed based on FF_Complete, which is initialized to 0 once receiving a new request packet and changed to 1 when FF_Size has been output. Only FF_Size = 0 can Wira sender perform the following frame parsing, which is also leveraged for identifying the current frame that belongs to our focused first frame. Then, the type of live-streaming protocols (PtlType), e.g., Flash Video (FLV), HTTP Live Streaming (HLS), and Real Time Messaging Protocol (RTMP), will be identified, which is required as different fields are located in their header and body structure. For example, if the signature on the protocol header is 'FLV', the parsing module will perform frame parsing based on the existing FLV structure. Next, Wira parser will accumulate the size of live-streaming data until the number ($\mathsf{Num}_{\mathsf{VF}}$) of parsed video frames has reached its threshold Θ_{VF} that is set to 1 (by default).

In this paper, Wira parser regards some video frames as the end of the first frame, so that the size of previous information, e.g., protocol header, audio frame, and script data, will be considered as part of FF_Size. This is because they are also critical for successfully displaying the first frame on the client side. Take a live stream in the real network as an example. When receiving a request packet, the traffic sender will sequentially transmit script data, audio, an I frame, a P frame, and three B frames to its client, whose sizes are S_{script}, S_{audio}, S_{I}, S_{P}, S_{B1}, S_{B2} and S_{B3}, respectively. In this case, FF_Size = $S_{\mathsf{script}} + S_{\mathsf{audio}} + S_{\mathsf{I}}$ when $\Theta_{\mathsf{VF}} = 1$, whose value will become $S_{\mathsf{script}} + S_{\mathsf{audio}} + S_{\mathsf{I}} + S_{\mathsf{P}} + S_{\mathsf{B1}}$ when $\Theta_{\mathsf{VF}} = 3$. The presented Wira enables more appropriate init_cwnd by taking the parsed FF_Size as a vital transport signal for first-frame optimizations (§IV-C).

B. Transport Cookie

Wira regards the last session's Hx_QoS of the same OD pair as essential transport signals for initializing control parameters. In particular, Wira introduces a cookie module that supports stateless transport cookie acquisition, in which the newly-measured Hx_QoS of current live streaming will be periodically synchronized (as transport cookie) from our Wira server to its clients. Besides, Wira enables its clients to insert the obtained Hx_QoS metrics into the handshake packets, which can be quickly extracted by Wira server during their connection establishment in the future. Thus, Hx_QoS can be fully considered for first-frame optimizations without introducing non-trivial storage overhead on the server side.

The presented transport cookie can result in the following benefits, which operate on a similar principle to the existing web cookie [33], [34], as Fig. 7 shows. On one hand, network QoS metrics (e.g., MinRTT and MaxBW) can be synchronized in the transport cookie, which can help the server configure initial transmission parameters more appropriately. On the other hand, different from the web cookie, the transport cookie allows the Wira server to offload the collected Hx_QoS to the cache of its clients, which greatly reduces the storage and retrieval pressure on the server.

To achieve the above stateless transport cookie, Wira designs and implements a client-server collaboration scheme and

[4] Actually, presentation timestamps (PTS) of these frames are actually earlier than the receiving time of this request packet so that enough frames can be transmitted without suffering from more application limitations.

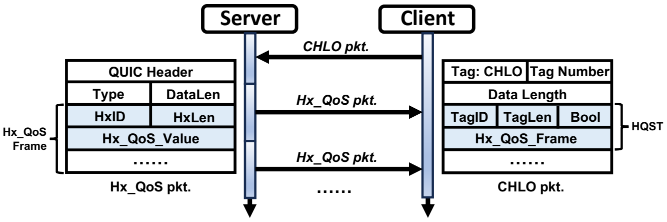


Fig. 8. Hx_QoS synchronization and acquisition.

its packet format upon user-space QUIC protocol, as Fig. 8 shows, in which the interaction process is as follows.

Collaboration declaration. Wira enables each handshake stage to declare whether the client supports Hx_QoS synchronization in the follow-up live-streaming transmission. This can be realized by adding a new tag (called HQST) in the CHLO packet of QUIC , as Fig. 8 shows. The field TagLen depicts the length of newly-created HQST while the field Bool = 1 reflects this client support Hx_QoS synchronization in this live-streaming connection.

Periodic synchronization. When receiving a CHLO packet with Bool = 1, the Wira sender will periodically transmit its collected network QoS (e.g., MinRTT and MaxBW) of the current connection to its client. Unless otherwise declared, the synchronization period is set to 3s. To achieve the above operation, Wira introduce Hx_QoS packet that is built on the QUIC protocol, as Fig. 8 shows, whose "type" is set to 0x1f that can differentiate the existing values in QUIC. In particular, a new frame (called Hx_QoS frame) is carried in this Hx_QoS packet, which contains one or more <HxID, HxLen, Hx_QoS_Value> triples. Note that HxID and HxLen are the identification and the length of each Hx_QoS tuple, respectively, whose value is shown in the Hx_QoS_Value field of Hx_QoS packets. In Wira, the clients will extract the newly-introduced Hx_QoS frame from their received Hx_QoS packets and then update Hx_QoS metrics stored locally. Meanwhile, the timestamp is also recorded when receiving an Hx_QoS packet, which will be carried in the next CHLO packets. Therefore, the Wira sender is not required to save these Hx_QoS values for each OD pair by offloading this non-trivial storage overhead to its clients.

Lightweight Hx_QoS obtaining. The proposed Wira enables its sender to quickly obtain the last session's Hx_QoS metrics with the same OD pair, which is carried in the Hx_QoS_Frame field of the CHLO-packet HQST tag. Concretely, the Hx_QoS_Frame will keep available only when Bool = 1 and the TagLen is larger than the sum of sizes of TagID, TagLen and Bool. In Wira, the Hx_QoS_Frame can be encrypted using the sender-side symmetric key, which cannot be decrypted on the receiver side. This can also prevent the measured Hx_QoS metric from being eavesdropped by unreliable clients and man-in-the-middle attacks [35].

In Wira, the required Hx_QoS metrics include MinRTT and MaxBW that have been demonstrated not to change significantly (§II-D), especially compared to UG-based network estimations (§II-C). Besides obtaining the last session's MinRTT and MaxBW from CHLO packets, the introduced cookie module also keeps collecting network QoS metrics over some time and periodically delivers them to the sending module that will construct Hx_QoS packets and synchronize them with the client. The synchronized Hx_QoS is used by the Wira sender to configure more appropriate init_pacing before the first frame is sent out (§IV-C).

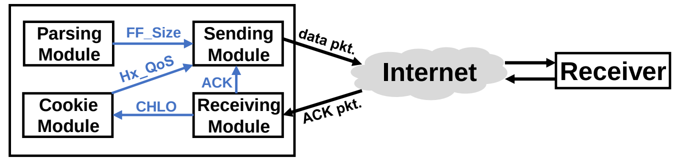


Fig. 9. Wira-powered parameter initialization.

C. Initial Parameter Configuration

Wira enables the sending module to regard the obtained transport signals (i.e., FF_Size in §IV-A and Hx_QoS in §IV-B) as an essential reference for initializing both cwnd and pacing rate to optimize per-flow's FFCT. To ensure the first frame can be successfully sent out without suffering from restricted resources (e.g., cwnd and network), the sending module will configure the init_cwnd for trying to adapt to the parsed FF_Size and network conditions. For better pacing the first frame and avoiding non-trivial packet losses, the transport cookie (i.e., MaxBW) obtained from the synchronized Hx_QoS frame will be leveraged for setting per-flow's init_pacing, as follows.

$$\mathsf{init\_pacing} = \mathsf{MaxBW} \tag{2}$$

Meanwhile, the insignificantly-changed MinRTT (that has been demonstrated in §II-D) can be utilized to compute bandwidth-delay product (BDP) that will be based to further adjust the init_cwnd value, as follows.

$$\mathsf{init\_cwnd} = \min\{\mathsf{FF\_Size}, \mathsf{MaxBW} \times \mathsf{MinRTT}\} \tag{3}$$

Even though we impose limitations on the sending rate, if there is no window constraint and queuing occurs in the bottleneck buffer, we would send data exceeding MaxBW × MinRTT within one RTT, which would prevent the queue from being emptied and result in significant network latency. Therefore, Wira adopts a conservative strategy, which enables init_cwnd to be properly configured by fully considering both the parsed FF_Size and actual transmission condition (that can be depicted by BDP) between the OD pair.

Corner case 1. When the requested live-streaming data is being delivered to L4, the parsing module might not get FF_Size in time before the first few bytes should have been sent out. Take the HTTP-FLV protocol as an example, the FLV header, script data, and audio frame will be delivered to L4 in turn before the I frame has been pulled. In this case, the FF_Size cannot be gained, causing the init_cwnd will not be successfully configured. To address this issue,

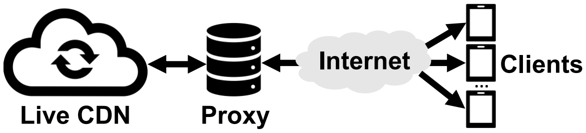


Fig. 10. Real-world deployment based evaluations.

Wira will temporarily leverage the init_cwnd_exp to replace FF_Size in Eq. 3 and compute a temporary init_cwnd. The setting of init_cwnd_exp is user-customized. It can either use the empirical fixed value (e.g., 10 [36]) or the experimental value that is computed by conducting specific A/B tests. For example, we can set the init_cwnd_exp as the average FF_Size collected from all connections during one week. Our years of real-world deployment experience demonstrate that the experimental one is more robust the the fixed value, which is also set as the baseline in this paper (see §VI). Once the first-frame parsing is completed, the init_cwnd will be updated to the minimum value of FF_Size and BDP.

Corner case 2. When leveraging Hx_QoS (i.e., MinRTT and MaxBW) for parameter initializations, the sending module will first check the timestamp of the last Hx_QoS synchronization. If the time interval T exceeds its threshold Δ, i.e., $\mathsf{T} > \Delta$, the synchronized Hx_QoS will become unavailable. In this paper, the Δ is set to 60mins unless otherwise declared. In this case, the init_cwnd will be configured to FF_Size and the init_pacing can be computed as $\mathsf{init\_pacing} = \frac{\mathsf{FF\_Size}}{\mathsf{init\_RTT\_exp}}$, where the init_RTT_exp is an experimental value similar to init_cwnd_exp. Specifically, the init_RTT_exp is set as the average MinRTT collected from all connections during one week through A/B tests.

V. Implementation

We implement Wira described in §IV upon NGINX architecture (with nginx 1.17.3) [37] and user-space QUIC protocol (with LSQUIC Q043) [38], which acts as an essential component for optimizing our provided live-streaming services. Our implementation consists of 1000+ lines of code.

In this paper, the Wira sender will (i) perform frame parsing and gain FF_Size (§IV-A), (ii) extract and synchronize Hx_QoS metrics as transport cookie (§IV-B), and (iii) initialize both cwnd and pacing rate (§IV-C). Meanwhile, the clients have also upgraded to support Hx_QoS can be synchronized and stored locally, which will be carried in its CHLO packets when requesting some live-streaming resource. For frame parsing, we enable a new function ngx_quic_send_data in nginx to load our developed Wira parser and then parse frames and obtain FF_Size. When the received frame is incomplete, the newly-introduced function ngx_quic_flv_parser_parse_or_send will temporarily save a portion of this frame until frame type and size can be learned. The obtained FF_Size will be delivered to the critical component, i.e., send controller of LSQUIC. Meanwhile, the newly implemented function parse_hs_data can extract our desired Hx_QoS from CHLO packets and pass it to send controller. Finally, Send Controller will perform the initialization for both cwnd and pacing rate based FF_Size and Hx_QoS.

TABLE I
Parameter configurations of init_cwnd and init_pacing.

Scheme	init_cwnd	init_pacing
Baseline	init_cwnd_exp	init_cwnd/init_RTT_exp
Wira(FF)	FF_Size	init_cwnd/init_RTT_exp
Wira(Hx)	BDP	MaxBW
Wira	min{FF_Size, BDP}	MaxBW

VI. Real-Network Evaluations

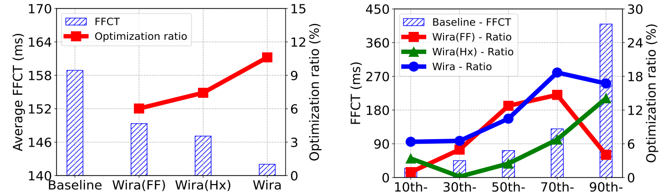
This section describes the performed experiment evaluations that are based on our real-world deployments. As Fig. 10 shows, the proxy server can pull the requested live-streaming data from our live CDN, and then respond to its clients based on Wira-powered initialization. In our CDN services, the live-streaming data is decoded using HTTP-FLV protocol.

Comparison schemes. We select the control policy with init_cwnd = init_cwnd_exp and init_RTT = init_RTT_exp (described in §IV-C) as the baseline method instead of Google recommended init_cwnd = 10 [4] or UG-based cwnd initialization [15] through our real-network A/B tests and measurements, as follows. (i) The init_cwnd = 10 always incurs unsatisfied FFCT, whose average (and 90th-percentile) FFCT value is 201.0ms (476.5ms). By contrast, our selected baseline method can optimize these two values to 158.9ms and 409.6ms, respectively, which is based on our performed 1000+ A/B tests. (ii) We make large-scale measurements and find each CDN proxy server (e.g., in Southeast Asia and Latin America) always serves over 1000 UGs so that it is unacceptable to run an ML model for each UG. Besides, we find top 5 UGs serve $<30\%$ of live streams, in which deploying 5 ML models for these UGs can only optimize a small number of live streams [25]. This reveals that the UG-based solutions cannot be well-scaled.

To further evaluate FFCT benefits of Wira, we also decouple the FF_Size based cwnd initialization and the Hx_QoS enabled init_pacing configuration from Wira, and then construct Wira(FF) and Wira(Hx) as two other comparison schemes, respectively. In this section, the init_cwnd and init_pacing of all above comparison schemes are configured as Table I shows.

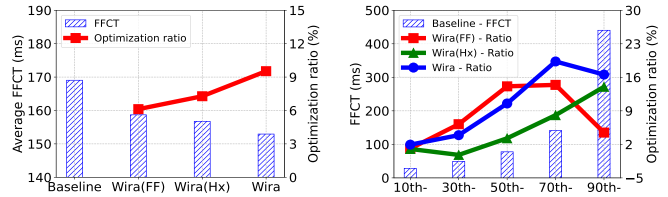
This paper mainly focuses on exploring more appropriate initializations of both cwnd and pacing rate, in which we select the BBR (with version 1) scheme [9] to support the above-parameter configurations.

The differences in first-frame transmission between 0-RTT and 1-RTT. When the server and client consume 1 RTT to establish a connection, the server measures the accurate RTT and uses it, instead of the configured initial RTT, along with other initial parameters we have configured to calculate new values for cwnd and pacing rate. These new values are then used for transmitting both the first-frame data and subsequent live-streaming data.

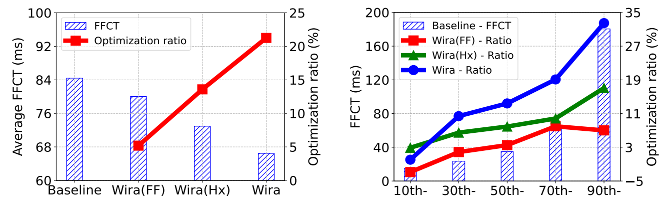


(a) Average FFCT benefits. (b) The quantiles of FFCT benefits.

Fig. 11. The real-network FFCT benefits of all live streams.



(a) Average FFCT with 0-RTT. (b) FFCT quantiles with 0-RTT.



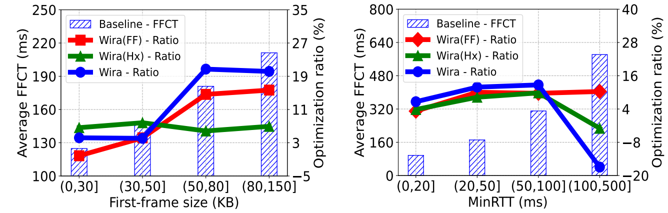
(c) Average FFCT with 1-RTT. (d) FFCT quantiles with 1-RTT.

Fig. 12. The real-network FFCT benefits of 0- and 1-RTT streams.

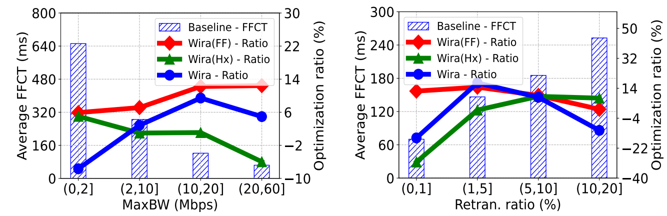
On the other hand, when the server and client consume 0 RTT to establish a connection, the server relies entirely on the initially configured parameters to send some or all of the first-frame data. It continues to use these parameters until an accurate RTT or bandwidth measurement is obtained, at which point it begins updating the sending parameters for the transmission of unsent data.

A. Overall Performance

The CDN proxy server that deploys Wira has been running steadily for over 6 months to serve a famous live application. Fig. 11 depicts the FFCT benefits and optimization ratios of Wira as well as its two variants, i.e., Wira(FF) and Wira(Hx), in which we can learn the following results. (i) Wira outperforms other three schemes, whose average FFCT value (142.0ms) can be lowered by 10.6% compared to the baseline (158.9ms), as Fig. 11(a) shows. (ii) Both Wira(FF) and Wira(Hx) can also introduce FFCT benefits, on average, that can be optimized by 6.0% and 7.4%, respectively. (iii) Wira can realize further optimizations for the high-quantile FFCT, whose 70th- and 90th-percentile value is reduced to 105.6ms (from 130.0ms) and 341.1ms (from 409.6ms) with the ratios of 18.7% and 16.7%, respectively, as Fig. 11(b) shows. (iv) Wira(FF) can obviously optimize the 70th-percentile FFCT values (with the ratio of 14.7%) while Wira(Hx) mainly reduce the 90th-percentile FFCT values (with the ratio of 14.1%). These results demonstrate the practicability and profitability



(a) FFCT varies with FF_Size. (b) FFCT varies with MinRTT.



(c) FFCT varies with MaxBW. (d) FFCT varies with retran. ratio.

Fig. 13. The real-network FFCT benefits under different network conditions.

of our proposed Wira for first-frame optimizations with more reduced FFCT.

To further explore the FFCT benefits, we classify the current live streams according to whether their connection establishments belong to 0-RTT that accounts for ∼90% based on our large-scale measurements. As Fig. 12(a) shows, the average FFCT (169.0ms) of 0-RTT streams can be lowered to 158.7ms (6.2%), 156.7ms (7.3%) and 152.9ms (9.5%) by employing Wira(FF), Wira(Hx) and Wira, respectively. Besides, the 90th-percentile FFCT can be reduced from 440.3ms (baseline) to 367.4ms (Wira) with a ratio of 16.6%, as Fig. 12(b) shows. The Hx_QoS enabled controls, i.e., Wira(Hx), enable more optimizations for 90th- and 95th-percentile FFCT values with the ratio of $>$14.0%.

Compared to 0-RTT connections, the FFCT values of 1-RTT connections can be reduced by employing Wira as well as its decoupled variants. This is because 1-RTT connections can obtain the accurate MinRTT so that the pacing rate can be updated to be a more appropriate value before the first frame is sent out. In Fig. 12(c) and Fig. 12(d), the average FFCT under 1-RTT streams can be optimized by 21.3% from 84.4ms (baseline) to 66.5ms (Wira), whose 90th-percentile value is lowered by 32.5% from 180.4ms to 121.8ms. Different from FFCT benefits under 0-RTT streams, both Wira(FF) and Wira(Hx) can achieve significant optimizations (7.0% and 17.1%) for the 90th-percentile FFCT. More importantly, we can also discover that Wira(Hx) always performs better than Wira(FF) under all, 0-RTT and 1-RTT live streams. This is because Wira(Hx) can realize the initialization of both cwnd and pacing rate, in which the init_cwnd will be configured to the BDP (i.e., MaxBW$\times$MinRTT). By contrast, Wira(FF) mainly focuses on the init_cwnd configuration, without leveraging the obtained transport cookies for carefully setting its init_pacing.

B. Benefits in Different Conditions

Wira can introduce different FFCT benefits under diverse conditions of both first frames and transmission networks.

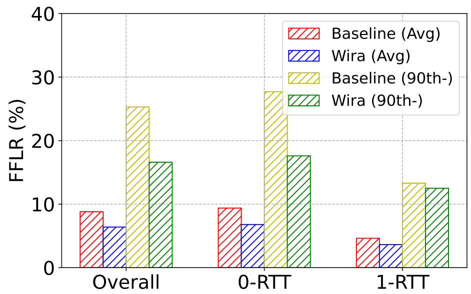


Fig. 14. The loss rate of the first frame.

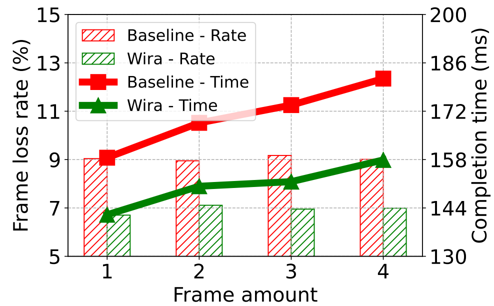


Fig. 15. The performance of previous frames.

Through real-world deployments of Wira, we can learn the following observations. (i) Wira can achieve more obvious optimizations for FFCT when the actual FF_Size becomes larger. For example, FFCT can be only lowered by 4.1% with the FF_Size of (30, 50], which will be reduced by 20.2% (from 211.2ms to 168.6ms) with the FF_Size of (80, 150], as Fig. 13(a) shows. (ii) Under larger FF_Size, FFCT will benefit from FF_Size enabled cwnd initialization compared to Hx_Qos based init_pacing configuration. For example, the FFCT values of 178.1ms and 196.5ms can be realized by Wira(FF) and Wira(Hx), respectively. (iii) Within 100ms MinRTT, Wira can optimize FFCT by 6.6% $\sim$ 12.7%, which will become deteriorated when MinRTT $>$ 100ms, as Fig. 13(b) shows. This is mainly affected by Wira(Hx) whose Hx_QoS metrics (e.g., MinRTT) might become inaccurate, causing more inappropriate configuration for init_cwnd. (iv) Wira performs much better in larger-MaxBW conditions compared to under smaller MaxBW. For example, FFCT can be reduced by 9.4% and 4.9% under the MaxBW of (10Mbps, 20Mbps] and (20Mbps, 60Mbps], respectively, which will becomes $<$2.8% with the MaxBW of (0Mbps, 10Mbps], as Fig. 13(c) shows. By contrast, Wira(Hx) enabled FFCT optimizations tend to become worse with larger MaxBW values. (v) When the retransmission ratio is in (1%, 10%], FFCT can be optimized by 8.6% $\sim$ 17.2% while Wira(FF) can keep stable benefits, i.e., with the ratio of 1.4% $\sim$ 14.7% under the retransmission ratio of $\sim$20%, as Fig. 13(d) shows.

C. First Frame Loss Rate

To better evaluate the performance of Wira, we will next analyze the first-frame loss rate (FFLR) that is incurred during traffic transmissions. Fig. 14 depicts the average and 90th-percentile FFLR when performing real-network deployments. We can learn Wira can reduce the average FFLR from 8.8% (baseline) to 6.4% with the optimization ratio of 27.3%. Besides, the 90th-percentile FFLR can be lowered by the ratio 34.4% from 25.3% (baseline) to 16.6%. For 0-RTT streams, their FFLR values are significantly lower than the values in 1-RTT streams. This is because 1-RTT streams measure the accurate RTT during the connection establishment process and use this value to calculate more accurate sending parameters (§VI). The average FFLR optimization ratios (that are incurred by Wira) for 0- and 1-RTT streams have reached 27.6% and 21.4%, respectively. In addition, for 0- and 1-RTT streams, the FFLR at the 90th percentile has been optimized by 36.5% and 6.0% respectively. By fully considering the actual transmission condition between the OD pair (§IV-C), Wira can achieve obvious optimizations for the first-frame loss rate in various network environments.

D. Influences on Follow-Up Frame Transmissions.

The proposed Wira introduces negligible influences on follow-up frame transmissions when lowering the value of FFCT. In this section, the completion time and loss rate of the first 1 $\sim$ 4 video frames will be leveraged to evaluate the performance of follow-up frame transmissions.

Completion time. The proposed Wira mechanism does not slow down the transmissions of follow-up video frames. As Fig. 15 shows, Wira enables the FFCT to be reduced by 16.5ms (from 158.5ms to 142.0ms) while 150.3ms, 151.6ms, and 157.9ms will be taken for completing the transmissions of first 2 $\sim$ 4 frames (since the first live-streaming packet is sent out), respectively, incurring stable optimization ratios (i.e., 10.9% $\sim$ 13.0%) compared to the baseline. Thus, we can learn the transmission performance (in terms of frame completion time) of follow-up video frames has not been affected by Wira-powered first-frame optimizations. In other words, the Wira-powered FFCT optimizations do not deteriorate the completion time of follow-up 2 $\sim$ 4 video frames.

Frame loss rate. Wira does not incur significant congestion in the transmission network, in which the loss rate of the follow-up video frames is demonstrated to not deteriorate when the proposed Wira is employed. As Fig. 15 shows, the Wira-incurred loss rate of follow-up video frames remains 6.7% $\sim$ 7.1%, compared to the ratios of 9.0% $\sim$ 9.2% that are introduced by baseline. Thus, we can know there is no significant negative effect on the transmissions of follow-up frames during Wira-enabled first-frame optimizations.

VII. Discussion

Transport cookie security. The proposed Wira supports more secure Hx_QoS synchronization and acquisition, in which the Hx_QoS frame in Hx_QoS packets can be encrypted using a server-side secret key that can be decrypted only by traffic server(s). In this case, each client cannot understand its received transport cookies that can not be easily fabricated as a non-existent Hx_QoS value for either obtaining more efficient transmissions or launching attacks on the targeted server. Wira enables its servers to verify the consistency between the sent and received Hx_QoS and then leverage the authentic values for initializing both cwnd and pacing rate.

First-frame playback conditions. This paper mainly focuses on the required delay of the first I frame in live streams. The first-frame playback conditions are related to client-side policies, which can be configured as (i) the buffered time length exceeds its threshold (e.g., 3s) or (ii) the amount of received video and audio frame satisfies its requirements, etc. Fortunately, the presented Wira can adapt to differentiated first-frame playback conditions by configuring the number ($\mathsf{Num}_{\mathsf{VF}}$) of parsed video (audio) frames, whose reached its threshold Θ_{VF} indicates FF_Size can be obtained for its init_cwnd initialization.

VIII. Related Work

Initial parameter optimization. To decrease the first-frame delay of live streaming, the initial transmission parameters are usually configured for shortening its FFCT. On the one hand, the initial cwnd of TCP is recommended to be settable, which is increased from $2 \sim 4$ to 10 segments through large-scale Internet experiments [3], [4]. Besides, Halfback [5] employs both larger initial cwnd and other supplementary methods (e.g., loss recovery) for optimizing short-flow performance. Further, JumpStart [6] abandons the initial cwnd configuration by skipping the startup stage and enables transmissions at the rate they deem appropriate. On the other hand, the initial sending rate can also be configured for well utilizing the available bandwidth and mitigating the severe packet losses caused by traffic bursts at the cold-start phase [7], [8]. This can be achieved by setting the initial values of both cwnd and minimum RTT (minRTT), especially for the pacing-based congestion controls, e.g., BBR [9], TIMELY [10] and Copa [11]. However, these schemes focus on initializing the fixed value for one of the sending parameters, which is far away from enough for FFCT optimizations (§II-A and II-B).

Dynamic parameter adjustment. To further explore better-performed configurations, several methods leverage machine learning to achieve dynamic adjustments for the transmission parameters [14]–[24]. For example, Orca [19] and AUTO [22] can adaptively determine the global weights when configuring cwnd and sending rate, respectively. However, they all focus on the in-process adjustments, ignoring further considerations for the parameter initialization. NeuroIW [14] enables DRL-powered selection for initial cwnd values under SDN-based mobile edge computing (MEC) while TCP-DRL [15] introduces dynamic initial cwnd configurations for each divided user group. However, these schemes are designed for all (a set of) connections and can only realize the coarse-grained transmission controls, which are unable to adapt to diversified first-frame sizes and differentiated network conditions (§II-A). Meanwhile, it is hard for frequently-utilized DRL to achieve more fine-grained (e.g., per-flow) parameter initialization due to its well-known instability and Heavyweight (§II-C).

Transport signals based optimization. It is well-studied that some transport signals can be leveraged to assist video-streaming optimizations [39]. On the one hand, the cross-layer message can be obtained by the transport layer for constructing better scheduling policies [40]. For example, both VOXEL [26] and DTP [27] can parse the received frame types that will be used for adjusting each packet's (re)transmission priority. On the other hand, the historical QoS can also be learned to configure the newly-established connection [28] [29]. For example, PCP can use the history for choosing the initial probe rate [41] while Antelope can predict the most suitable congestion control mechanism based on IP-related historical information [42]. Under the observation that Hx_QoS with the same OD pair performs similarly (described in §II-D), transport signals are fully considered for a more fine-grained first-frame control paradigm.

IX. Conclusion

This paper proposes the first-frame optimization scheme named Wira, which takes both the first-frame size and historical transmission QoS for initializing per-flow's control parameter of live streaming. In particular, Wira supports cross-layer frame parsing at the transport layer to accurately perceive the first-frame size that will be leveraged to configure more appropriate initial cwnd. Besides, a lightweight client-cloud collaboration is carefully designed, which enables the historical transmission QoS to be quickly obtained to set the initial pacing rate, offloading the intolerable sender-side storage overhead to Wira clients. We implement Wira and evaluate it via real-world deployments, whose average and 90th-percentile FFCT values are reduced by 10.6% and 16.7%, respectively. Currently, the presented Wira has been deployed on our CDN services [43] and our edge products (named EdgeOne) [44], one of the world's largest CDN vendors , serving thousands of millions of live-streaming users worldwide.

References

[1] Technavio. Live streaming market by product, end-user, and geography - forecast and analysis 2023-2027. https://www.technavio.com/report/live-streaming-market-industry-analysis.

[2] Stream Hatchet. 2022 yearly video game live streaming trends report. https://streamhatchet.com/blog.

[3] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's initial window. In *RFC 3390*. IETF, 2013.

[4] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM Computer Communication Review*, 40(3):26–33, 2010.

[5] Qingxi Li, Mo Dong, and P Brighten Godfrey. Halfback: Running short flows quickly and safely. In *ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–13, 2015.

[6] Dan Liu, Mark Allman, Shudong Jin, and Limin Wang. Congestion control without a startup phase. In *Proc. PFLDnet*, 2007.

[7] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the performance of tcp pacing. In *IEEE INFOCOM*, volume 3, pages 1157–1165, 2000.

[8] Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer. TCP friendly rate control (TFRC): Protocol specification. In *RFC 5348*. IETF, 2008.

[9] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

[10] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, et al. Timely: RTT-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[11] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 329–342, 2018.

[12] Tong Li, Wei Liu, Xinyu Ma, Shuaipeng Zhu, Jingkun Cao, Senzhen Liu, Taotao Zhang, Yinfeng Zhu, Bo Wu, and Ke Xu. Art: Adaptive retransmission for wide-area loss recovery in the wild. In *IEEE ICNP*, pages 1–11. IEEE, 2023.

[13] Xu Yan, Tong Li, Bo Wu, Cheng Luo, Fuyu Wang, Haiyang Wang, and Ke Xu. Poster: Too: Accelerating loss recovery by taming on-off traffic patterns. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 1147–1149, 2023.

[14] Ruitao Xie, Xiaohua Jia, and Kaishun Wu. Adaptive online decision method for initial congestion window in 5g mobile edge computing using deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(2):389–403, 2019.

[15] Xiaohui Nie, Youjian Zhao, Zhihan Li, Guo Chen, Kaixin Sui, Jiyang Zhang, Zijie Ye, and Dan Pei. Dynamic tcp initial windows and congestion control schemes through reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 37(6):1231–1247, 2019.

[16] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 395–408, 2015.

[17] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling data-driven quality of experience optimization using Group-BasedExploration-Exploitation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[18] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[19] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *ACM SIGCOMM*, pages 632–647, 2020.

[20] Salma Emara, Baochun Li, and Yanjiao Chen. Eagle: Refining congestion control by learning from the experts. In *IEEE INFOCOM*, pages 676–685, 2020.

[21] Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, and Guihai Chen. A unified congestion control framework for diverse application preferences and network conditions. In *CoNext*, pages 282–296, 2021.

[22] Xu Li, Feilong Tang, Jiacheng Liu, Laurence T Yang, Luoyi Fu, and Long Chen. Auto: Adaptive congestion control based on multi-objective reinforcement learning for the satellite-ground integrated network. In *USENIX Annual Technical Conference*, pages 611–624, 2021.

[23] Zhiyuan Pan, Jianer Zhou, Xinyi Qie, Weichao Li, Heng Pan, and Wei Zhang. Marten: A built-in security drl-based congestion control framework by polishing the expert. In *INFOCOM*, pages 1–10, 2023.

[24] Wenzheng Yang, Yan Liu, Chen Tian, Junchen Jiang, and Lingfeng Guo. Gemini: Divide-and-conquer for practical learning-based internet congestion control. In *INFOCOM*, pages 1–10, 2023.

[25] Bo Wu, Tong Li, Cheng Luo, Changkui Ouyang, Xinle Du, and Fuyu Wang. Autoplex: inter-session multiplexing congestion control for large-scale live video services. In *Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration*, pages 1–6, 2022.

[26] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K Sitaraman. Voxel: Cross-layer optimization for video streaming with imperfect transmission. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 359–374, 2021.

[27] Hang Shi, Yong Cui, Feng Qian, and Yuming Hu. Dtp: Deadline-aware transport protocol. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, pages 1–7, 2019.

[28] Jing Chen, M Zhang, and Qiang Meng. A network congestion control algorithm based history connections and its performance analysis. *Journal of Computer Research and development*, 40(10):1470–1475, 2003.

[29] Zhigang Chen, Xiaoheng Deng, Lianming Zhang, and Biqing Zeng. A new parameter-config based slow start mechanism. *Journal of Communication and Computer*, 2(5):56–62, 2005.

[30] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *ACM SIGCOMM*, 2017.

[31] Jana Iyengar, Martin Thomson, et al. QUIC: A UDP-based multiplexed and secure transport. In *RFC 9000*. IETF, 2021.

[32] Wikipedia. Coefficient of variation. https://en.wikipedia.org/wiki/Coefficient_of_variation, 2023.

[33] John Giannandrea and Lou Montulli. Persistent client state: HTTP cookies. October 1994.

[34] David Kristol and Lou Montulli. RFC2109: HTTP state management mechanism. In *RFC 2109*. IETF, 1997.

[35] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[36] HK Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. Rfc6928-increasing tcp's initial window. 2013.

[37] Ivan Ovchinnikov et al. Nginx. https://github.com/nginx.

[38] LiteSpeed Tech. LiteSpeed QUIC and HTTP/3 Library. https://github.com/litespeedtech/lsquic.

[39] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Tack: Improving wireless transport performance by taming acknowledgments. In *ACM SIGCOMM*, pages 15–30, 2020.

[40] Li Li, Ke Xu, Tong Li, Kai Zheng, Chunyi Peng, Dan Wang, Xiangxiang Wang, Meng Shen, and Rashid Mijumbi. A measurement study on multi-path tcp with multiple cellular carriers on high speed rails. In *ACM SIGCOMM*, pages 161–175, 2018.

[41] Thomas E Anderson, Andy Collins, Arvind Krishnamurthy, and John Zahorjan. PCP: Efficient endpoint congestion control. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[42] Jianer Zhou, Xinyi Qiu, Zhenyu Li, Gareth Tyson, Qing Li, Jingpu Duan, and Yi Wang. Antelope: A framework for dynamic selection of congestion control algorithms. In *IEEE International Conference on Network Protocols (ICNP)*, pages 1–11, 2021.

[43] Tencent cloud CDN. https://www.tencentcloud.com/products/cdn.

[44] Tencent cloud edgeone. https://www.tencentcloud.com/products/teo.