Tackling Bit-Rate Variation of RTC through Frame-Bursting Congestion Control

Zhidong Jia, Yihang Zhang, Qingyang Li, Xinggong Zhang*
Peking University
{jiazhidong, zhangyihang, liqingyang, zhangxg}@pku.edu.cn

***Abstract*—Interactive video applications signal widespread interest in Real-time communication (RTC), yet issues like frame delay, rebuffering, etc. remain a common complaint. We argue that this is mainly due to legacy network-oriented congestion control (CC), which assumes a continuous stream of packets is being sent. But this assumption doesn't hold in RTC since the video encoder exhibits inherent bit-rate variation: (1) Bursty spiking bit-rate leads to packets waiting in the sending buffer, which increases frame delay. (2) Low bit-rate causes insufficient packets available for sending, making current CCs hard to detect available bandwidth.**

In response, we propose BurstRTC, a novel paradigm for RTC transport protocol. Each frame is emitted as a whole, and the video bit-rate is directly controlled by network congestion feedback. BurstRTC uses frame-bursting to estimate available bandwidth efficiently regardless of bit-rate variation. Considering the impact of bit-rate variation on network congestion, BurstRTC models frame size as a Gaussian distribution instead of a fixed size and further derives its frame delay, preventing suboptimal performance of purely network-oriented designs. An analytic method for determining the target bit-rate replaces the trial-and-error updates of gradient-based methods, ensuring fast convergence to the available bandwidth.

We evaluated the performance of BurstRTC and found that, compared with GCC, BurstRTC achieves up to 59.8% higher bit-rate and up to 48.9% lower frame delay. Further, compared with SQP and Pudica, BurstRTC can also reduce tail frame delay by up to 89.2%, and improve average bit-rate by up to 15.6%.

***Index Terms*—congestion control, real-time communication, bit-rate variation**

I. Introduction

The popularity of real-time communication (RTC), exemplified by the substantial growth of video conferencing, cloud gaming, and XR/VR applications, etc. is evident in its rapid ascent [1]. However, the continuous advancement of network and RTC technologies still fails to ensure consistent Quality of Service, thereby affecting the Quality of Experience (QoE) [2]–[5]. Serious QoE degradation, such as freezing, interactive delay, and low image quality, are often complained about by users [6]–[10]. According to reports by [8], [11], RTC users experience video freeze for 3.6% of their usage time. Over 25% of the time, RTC users experience noticeable latency, and over 5% of the time, RTC users deem the application unusable due to latency. It is still an open question how to improve the QoE of RTC for both academia and industry.

* The corresponding author.

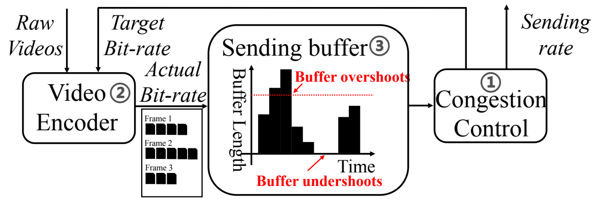


Fig. 1. RTC overshooting and undershooting occur due to bit-rate variations and network-oriented CC. Overshooting results in higher buffering delay, while undershooting leads to incorrect bandwidth detection.

We argue that the **legacy network-oriented congestion control** mechanisms are to blame. Most RTC applications employ congestion control (CC) algorithms, including BBR [12], GCC [13] and etc. to decide the packet sending rate and video bit-rate (Fig. 1 ①). These algorithms share a simple assumption with their historical TCP-variant predecessors [14]–[18]: there is a continuous stream of packets waiting to be sent. They assume there are always enough packets for sending and ignore sender-buffering delay. However, this assumption is no longer valid in RTC since video frames are discrete, packets are encoded in real-time, and the RTC delay starts from the time packets are encoded.

Video bit-rate variation is inherent in RTC. In RTC applications, the server encodes the real-time raw video and pushes the encoded video packets into the sending buffer (Fig. 1 ②). Due to the inherent spatial and temporal complexities of the video content, the actual bit-rate produced by the encoder often varies around the target bit-rate [2], [4], [5], [19].

Bit-rate variation results in RTC's performance degradation, as shown in Fig. 1 ③. (1) **Overshooting:** When the actual bit-rate excesses CC's sending rate and the sending buffer contains more than 150ms[1] of data packets, newly incoming packets wait in the buffer, resulting in high frame delay. (2) **Undershooting:** When the actual bit-rate is below CC's sending rate and the sending buffer is empty, CC doesn't have enough packets to estimate the network's available bandwidth.

To mitigate the above problem, we propose the insight of **Frame-Bursting Congestion Control**. Each frame is emitted as a whole, and the video bit-rate is directly controlled by network congestion feedback. In response to buffer undershooting, frame-bursting CC can probe the bandwidth using the packet train algorithm [21], avoiding the assumption of sufficient data. In response to buffer overshooting, frame-

[1]RTC's tolerable latency requirement [20]

bursting CC optimizes video encoding by using one control loop to determine the target bit-rate, simultaneously accounting for network congestion and bit-rate variation, resulting in improved performance over traditional dual-loop methods.

To design an effective frame-bursting CC, we still have the following challenges to overcome: (1) **How to estimate available bandwidth accurately while competing with the background traffic.** Purely bursting frames with the packet train algorithm can detect link capacity efficiently, but it fails to detect the volume of background traffic, which is critical to controlling frame delay and ensuring fairness. (2) **How to decide the encoding video bit-rate to match the available bandwidth.** As compressed video bit-rate varies around the target bit-rate, directly treating the available bandwidth as the target bit-rate may insert extra data packets into the network, putting the bottleneck link at risk of serious congestion. (3) **How to estimate frame delay.** We need to consider the frame delay when determining the target bit-rate because of some encoders' rate control interval limitations [5] and network delayed feedback. (4) **How to decide the video target bit-rate with fast convergence.** We need an analytic method that directly determines the appropriate target bit-rate, enabling fast convergence to the fluctuating available bandwidth.

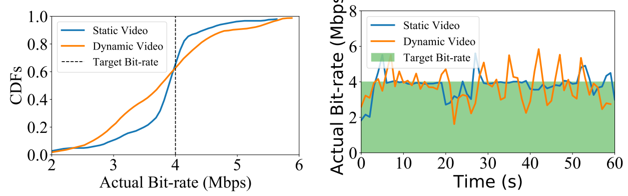
In this paper, we propose **BurstRTC**, a frame-bursting congestion control with four key design blocks, to address the aforementioned challenges: **The bandwidth sampler** aims to estimate the bandwidth and background traffic per frame by alternating between *bursting and pacing* methods. **The frame size model** aims to match video bit-rate with the available bandwidth by predicting frame size distribution. **The frame delay model** aims to predict the frame delay of future frames based on their size or distribution. **Analytic rate control** aims to decide an optimal target video bit-rate directly by the above analytic models instead of traditional AIMD or gradient-CC, to achieve fairness and fast convergence.

We evaluated the performance of BurstRTC and found that, compared with WebRTC, BurstRTC achieves up to 59.8% higher bit-rate and up to 48.9% lower frame delay. The tail frame delay of BurstRTC is reduced by up to 143%, and the freezing time is reduced by up to 82.9%. Further, compared with the latest algorithms, SQP [22] and Pudica [23], BurstRTC can also reduce tail frame delay by up to 89.2% and 82.2%, respectively, and improve average bit-rate by up to 13.7% and 15.6%. It also ensures good fairness.

In summary, this paper makes the following contributions:

1) We identify the problem of overshooting and undershooting and analyze how it affects RTC applications.
2) We design BurstRTC, a frame-bursting congestion control to address the above issue.
3) We implement BurstRTC and conduct several evaluations, achieving the expected results.

This paper is organized as follows. Firstly, we present our motivation for addressing the overshooting and undershooting problem of legacy network-oriented CC in II. Next, we provide our design of BurstRTC in III and its implementation in IV-A1. Then, we present the experimental results of BurstRTC in IV and our conclusion in V.



(a) Distribution of bit-rate. (b) Bit-rate varies around the target over time.

Fig. 2. Bit-rate varies around the target bit-rate of 4Mbps due to spatial and temporal complexity.

II. Motivation

A. Network-Oriented Congestion Control

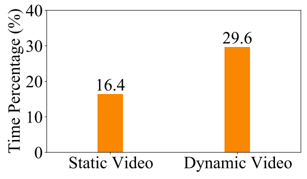
Real-time communication (RTC) spans applications like video conferencing and cloud gaming, which, despite varying architectures, share common transport mechanism in Fig. 1. This paper focuses on the challenges of RTC's real-time video generation, encoding, and transmission, rather than individual use cases. Today's RTC applications utilize two primary control loops: Congestion Control (CC) for bandwidth probing and flow fairness, and Video Bit-Rate Control (VBC) for encoding rate adjustment. Numerous CC algorithms have evolved over three decades, with RTC favoring delay-based methods like BBR, GCC, and SCReAM [24] to manage sending and encoding rates concurrently. However, these legacy CCs may not excel in performance. [19], [22], [23]

While legacy CCs are designed with the goals of ensuring high throughput, low delay, and flow fairness simultaneously, they are purely **network-oriented**, treating the data packets to be sent as an endless byte stream, without considering the patterns in the generation of these packets. This poses no issue in traditional FTP/HTTP data transmission, where data is generated in advance and awaits transmission. However, in RTC, users aim to consume the dynamically generated frames with minimal delay, making the **packet patterns and the length of the sending buffer** crucial. Therefore, in the next subsection, we focus on how RTC packets are generated based on the complexity of real-time videos.

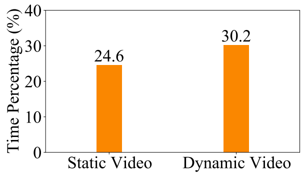
B. Bit-Rate Variation

Even when the video encoder is given a fixed target bit-rate, the compressed video bit-rate demonstrates significant variation due to the spatial and temporal complexity of video content. These complexities can cause the actual bit-rate to vary around the target bit-rate as the encoder struggles to accurately represent the visual information within the constraints of the compression algorithm. Despite advancements in modern encoding technology, the bit-rate variation is inevitable.

As shown in Fig. 2, a video encoded with a target bit-rate of 4 Mbps (encoding 1080p 30fps videos using x264) demonstrates actual bit-rate variations from 2 Mbps to 6 Mbps, with a 50% variation. The encoding parameters are set as follows: enabling frame intra-refresh, with 'preset' set to "faster" and

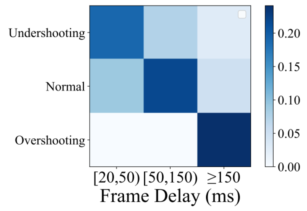


(a) Percentage of overshooting.

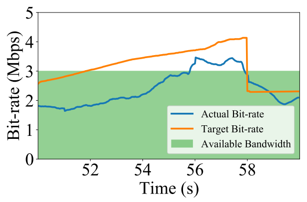


(b) Percentage of undershooting.

Fig. 3. Overshooting and undershooting commonly occur in both static and dynamic videos.



(a) Frame delays over 150 ms are mainly caused by overshooting.



(b) Incorrect bandwidth detection (target bit-rate) when undershooting.

Fig. 4. Overshooting induces high frame latency. Undershooting leads to incorrect bandwidth estimation.

'tune' set to "zerolatency", while all other parameters are kept at their default values. We refer static video as content with minimal changes between frames, such as animated films like "Big Buck Bunny [25]" where the scene composition remains relatively constant. While dynamic video content, on the other hand, encompasses videos with frequent scene changes, rapid motion, and high visual complexity, such as game streaming or live-action sports, such as "Fallout 4 [26]". Comparing the bit-rate of a static video to a video with movement, we further conclude that when encoding videos with more movement, the actual bit-rate exhibits more pronounced variation.

Certain encoding strategies can limit bit-rate exceedance, but this often comes at the cost of prolonged periods where the bit-rate falls short of the target [22]. Simultaneously avoiding both bit-rate exceeding and falling below the target is quite challenging. Moreover, bit-rate variation can result from various factors, including instability in video encoding intervals due to device performance [4].

C. Overshooting and Undershooting

As the encoder's actual bit-rate varies around the target bit-rate (II-B), the actual bit-rate may exceed or fall below the CC's sending rate. When the actual bit-rate exceeds the sending rate, additional data packets will overflow the sending buffer, with large buffering delay. In contrast, when the actual bit-rate is lower than the sending rate, the sending buffer will be gradually emptied. Thus, considering the status of the sending buffer, two problems arise:

(Overshooting): When there are too many packets in the sending buffer, CC fails to send all packets out instantly, causing large buffering delays.

When the size of queued packets in the sending buffer exceeds 150 ms, each incoming data packet incurs a delay of at least 150 ms, and the sending buffer is considered overshooting at this point. We choose 150 ms as the threshold because a typical RTC application, such as a video conference, demands consistent latency below 150 ms [20].

We validate that the occurrence of buffer overshooting is common under commercial 4G networks, as shown in Fig. 3(a). We set up a WebRTC [27] server and client based on a open source project Razor [28] to stream videos in II-B under commercial 4G networks and collect statistics on the percentage of time the sending buffer is overshooting. The sender and server are hosted on two Ubuntu desktops, connected by a CPE [29] which is connected to the Internet using a 4G data card in between. Each video is transmitted for 60s, and this process is repeated three times. The result in Fig. 3(a) shows that during at least 16% of the time during playback time, the sending buffer is overshooting. The percentage even reaches nearly 30% for a video with dynamic content.

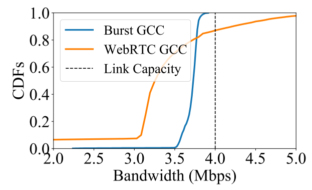
Overshooting is the primary cause of significant frame delay, as every video packet needs to spend extra time in the sending buffer waiting to be sent out. In Fig. 4(a), we draw a heat map to show the correlation between the frame delay and the buffer status when the first packet of this frame reaches the sending buffer. The result shows that frame delays over 150 ms are mainly caused by overshooting, despite the fact that 4G networks exhibit fluctuations in network latency [30].

(Undershooting): When there are insufficient packets in the sending buffer, CC fails to estimate the available bandwidth, further resulting in an incorrect video bit-rate.

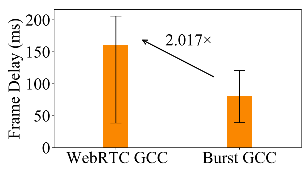
The occurrence of sending buffer undershooting is not uncommon in video playback. We use the same experimental settings before and collect statistics on the percentage of time of the sending buffer is undershooting. The result in Fig. 3(b) shows that the sending buffer is undershooting for at least 24% of the time during playbacks of different videos. For the playback of dynamic videos, this percentage is raised to 30%.

When the buffer undershoots, CC's sending rate is strictly limited by the actual bit-rate produced by the encoder. This prevents CC from accurately observing true network conditions such as delay and receiving rate, since existing CC algorithms assume sufficient data flow. For example, as show in Fig. 4(b), the actual bit-rate remains below the target bit-rate of GCC from 50s to 58s. During this period, the sending buffer is undershooting, causing CC to overestimate the network capacity and resulting in an incorrect target bit-rate.

In-Depth Analysis: Here we analyze the reasons for the above phenomenon. There are conflicts between network-oriented CC and bit-rate variation. Initially, these schemes assess the current pacing rate (or window size) against congestion signals like packet delay. In the absence of congestion signals, as in GCC, the pacing rate is incremented. However, insufficient data availability undermines the relationship between low latency and an underestimated pacing rate. Secondly, these designs estimate bandwidth using the receiving rate, which is limited by the actual sending rate, as in GCC and BBR. When data is insufficient, the receive rate consistently falls short of expectations. Thirdly, some algorithms [18] aim to manage network queue occupancy. Yet, this objective is challenging to achieve when the bit-rate varies. Finally, these



(a) Bursting out can estimate bandwidth more efficiently.



(b) Bursting out every frame instantly can reduce the average frame delay by half.

Fig. 5. Potentials of burst-driven bandwidth estimation and rate adaptation.

controls confine the overall sending rate, resulting in buffer accumulation and elevated latency when data surplus exceeds sending rate.

D. Insight

In this paper, we come up with the idea of **Frame-bursting Congestion Control.** Each frame is emitted as a whole, and the video bit-rate is directly controlled by network congestion feedback. The insight arises for two reasons:

Firstly, when buffer undershoots, the frame-bursting CC, specifically the packet train algorithm [21], avoids the assumption of sufficient data, probing the bandwidth regardless of undershooting and overshooting. Secondly, when the buffer overshoots, frame-bursting rate control optimizes video encoding by using a single control loop to determine the optimal target bit-rate, simultaneously accounting for network congestion and bit-rate variation, resulting in improved performance over traditional dual-loop methods.

To reveal the potential of our insight, we carry out a preliminary experiment with WebRTC. We use the WebRTC server and client with default settings to stream video under a simulated network environment (bandwidth: 4 Mbps, delay: 20 ms, controlled using TC [31]) and record its video bit-rate trace. WebRTC GCC sends out packets with the pacing. To show the potential of Frame-Bursting, we replay the video bit-rate trace in the same environment, but burst out all packet of a frame out at the time they are encoded (named Burst GCC). For Burst GCC, we employ the packet train algorithm as shown in (1) at the receiver to estimate bandwidth.

$$\hat{C} = \frac{\sum_{i=2}^{n} P_i}{rt_n - rt_1}, \tag{1}$$

where $\hat{C}$ is the estimated bandwidth, P_i is the i-th packet size of the frame, rt_i is the time the i-th packet arrivals the receiver and n is the number of packets included in the frame.

Firstly, frame-bursting bandwidth estimation holds great promise in RTC scenarios. Fig. 5(a) shows CDF plots of the WebRTC GCC and Burst GCC's estimated bandwidth. Due to the bit-rate variation, even in such a simple simulation scenario, WebRTC GCC struggles to effectively probe the upper limit of link capacity, exhibiting prolonged underestimation and occasional overestimation of bandwidth. In contrast, Burst GCC consistently demonstrates superior bandwidth estimation, unaffected by bit-rate variation.

Secondly, due to GCC's inability to consistently provide accurate bandwidth estimation, the consequences of buffering packets in the sending buffer are even more severe. Fig. 5(b) shows the average, 75th, and 25th percentile values of frame delay of WebRTC GCC and Burst GCC. The result shows that bursting out every frame immediately at the time they are encoded does reduce the frame delay, specifically by half of the average frame delay.

E. Design challenges

To design an effective frame-bursting CC, we still have the following challenges to overcome:

How to estimate available bandwidth accurately while competing with the background traffic. While bursting frames with packet train algorithm, as used by Salsify [19], SQP [22] and Pudica [23], can detect link capacity efficiently, purely bursting out the whole frame fails to detect the volume of **background traffic**, which is critical to controlling the frame delay and ensuring fairness. It's necessary to delicately design the bursting process and corresponding algorithms to sample both the network capacity and background traffic.

How to decide the encoding video bit-rate to match the available bandwidth. While existing works directly treat the available bandwidth as the target bit-rate, we argue that this is problematic. As compressed video bit-rate varies around the target bit-rate, these works may insert extra data packets into the network when the former exceeds the latter, putting the bottleneck link at risk of serious congestion. Thus, it is critical to determine an optimal target bit-rate that makes the compressed video bit-rate match the available bandwidth.

How to estimate the long-term frame delay. In the frame-bursting operational mode, delay is no longer divided into sending buffer delay and network delay; instead, bit-rate variation is directly reflected in the network delay. We need to consider the long-term frame delay when determining the target bit-rate for the following two reasons: (1) Due to the software and hardware limitations of the commercial video encoders, changing the target bit-rate has at least an L-frames adjustment interval limitation [5], [32], (2) Due to network delayed feedback, only the delays of frames that were sent one-RTT ago are accessible [33], [34].

How to decide the video target bit-rate with fast convergence. Existing works [19], [22], [23] make their target bit-rate converge to the available bandwidth through trial and error, using gradient-based methods or AIMD. However, since available bandwidth features huge fluctuations in real-world networks [30], we argue that this kind of trial-and-error method fails to utilize the available bandwidth effectively and keep frame latency low. In contrast, we need an analytic method that directly determines the appropriate target bit-rate with a network model, enabling fast convergence to the fluctuating available bandwidth.

III. System Design

A. Design Overview

In this paper, we propose **BurstRTC**, a frame-bursting congestion control, to address the aforementioned challenges. BurstRTC is structured around four key design blocks:

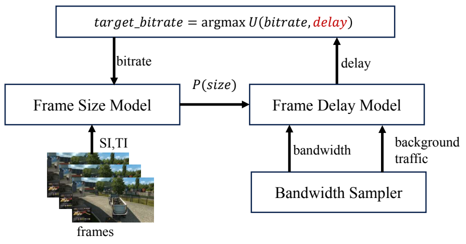


Fig. 6. BurstRTC's design overview.

(1) **Sampling available bandwidth within one frame.** (III-B) This design block aims to estimate the bandwidth and background traffic per frame. The frame is divided into two parts: the pacing part and the bursting part. The pacing part is transmitted at a rate higher than the available bandwidth to infer background traffic by observing competition. The bursting part is sent at maximum speed to determine the upper limit of the network capacity. (2) **Modeling frame size as a distribution.** (III-C) This design block aims to match video bit-rate with the available bandwidth. Considering bit-rate variation, we model the frame size as a Gaussian distribution. We then predict its mean and variance based on content complexity and the given target bit-rate. (3) **Modeling long-term frame delay.** (III-D) This design block aims to predict the long-term frame delay of based on their size or distribution. We model the relationship between the frame delays of adjacent frames and use an iterative method to predict their frame delay. (4) **Determining target bit-rate with an analytic method.** (III-E) This design block aims to determine an optimal target bit-rate directly at any time to achieve fast convergence. We construct a utility function based on target bit-rate and the corresponding frame delay, and determines the optimal target bit-rate by maximizing the utility function. This approach ensures fast convergence and avoids the inefficiencies of gradient-based methods or AIMD.

Fig. 6 shows the relationship between the blocks of BurstRTC. BurstRTC determines the target bit-rate by maximizing a utility function, which is constructed based on the bit-rate and the corresponding frame delay:

$$target\_bitrate = \arg\max_{bitrate} U(bitrate, delay) \tag{2}$$

In order to achieve direct **analytical solutions** for (2), BurstRTC needs to determine the relationship between frame delay and the target bit-rate provided to the encoder. Considering the bit-rate variation, BurstRTC uses a **frame size model** to calculate the frame size distribution corresponding to any target bit-rate based on the video content. The **frame delay model**, constructed according to the link capacity and the background traffic measured by the **bandwidth sampler**, then predicts future frame delay based on the this distribution.

B. Bandwidth Sampler

1) Block Design: The bandwidth sampler utilizes the packet train algorithm to probe available bandwidth. However, the straightforward packet train algorithm can only estimate the link capacity, not the available bandwidth. In this paper, we employ a variant of the packet train algorithm to probe available bandwidth, which is capable of measuring both the link capacity and the background traffic within a single frame.

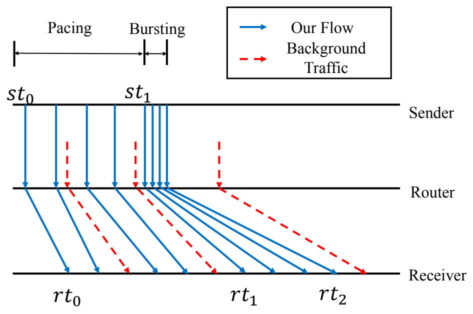


Fig. 7. Schematic diagram of the bandwidth sampler.

Specifically, we divide a frame into two parts as shown in Fig. 7: the pacing part and the bursting part. These two parts are responsible for different tasks. The bursting part is sent at full speed without any control, and uses traditional methods to obtain an upper bound on the link capacity:

$$\hat{C} = \frac{\sum_{i=2}^{n} P_i}{rt_2 - rt_1}, \tag{3}$$

Existing work [35] has already demonstrated that when the burst length is sufficient, the link capacity can be estimated accurately (discussed in IV-A1). The packets of pacing part and the background traffic arrive at the bottleneck router alternately and then the link capacity is allocated proportionally. Then, let the sending rate S to be $\hat{C}$ measured in the previous turn, and the receiving rate R is calculated as :

$$R = \frac{\sum_{i=2}^{m} P_i}{rt_1 - rt_0}, \tag{4}$$

where m is the number of packets sending by the pacing rate S. Assuming the router adheres to a first-come, first-served (FIFO) scheduling policy, the link capacity should be allocated in proportion, which means:

$$R = \frac{S}{S+B} \times C, \tag{5}$$

where C is the current link capacity and B is the rate of background traffic. Thus, we can estimate the rate of background traffic using:

$$\hat{B} = S \times (\frac{\hat{C}}{R} - 1). \tag{6}$$

To further enhance robustness, we use the average values over the past 10 frames to calculate all the rates mentioned above.

2) Validation of Model: In Fig. 8, we validated the effectiveness of our bandwidth sampler under a simulated experiment. In this experiment, we used a square wave with a period of 10 seconds that fluctuates between 2 Mbps and 4 Mbps to limit the link capacity, and added an additional 20 ms of delay as well as a UDP background traffic that constantly sends at 1 Mbps. BurstRTC shares the link capacity with the background

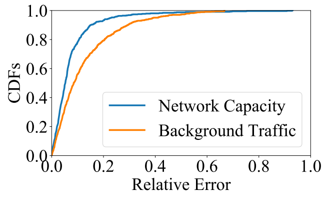


Fig. 8. The bandwidth sampler can accurately estimate the link capacity and background traffic.

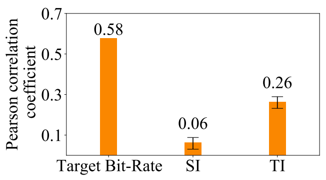


Fig. 9. The variation in frame size is correlated with the target bit-rate, SI, and TI.

traffic. The bandwidth sampler continuously estimates the link capacity and the background traffic. The relative error of the estimated link capacity and background traffic calculated by the bandwidth sampler is shown in Fig. 8, representing the ratio of the difference between the estimated value and the true value to the true value. The average estimation errors for link capacity and background traffic are 0.21 Mbps and 0.18 Mbps, respectively, illustrating the effectiveness of this block.

C. Frame Size Model

1) Block Design: The frame size model is responsible for matching video bit-rate with the available bandwidth. It achieves this by providing the distribution of frame sizes corresponding to any target bit-rate. The frame sizes output by the encoder is modeled as a Gaussian distribution [36]:

$$P(F \le x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt, \tag{7}$$

where F is the frame size. Thus we use the Gaussian distribution to represent the size of the frames, which means that we need the mean (μ) and variance (σ) of the frame sizes.

Note that the mean frame size (μ) is not always equal to the target bit-rate. Prior researches [2] has demonstrated that this primarily results from the content of the video, as the size of encoded frames is influenced by factors such as frame visual complexity (Spatial perceptual Information SI, and Temporal perceptual Information TI) and target bit-rate (T). Inspired by this, we explored the correlation between these quantities and frame size. We selected 10 gaming videos [26], each lasting for 1 minute with 30 fps, and encoded each video three times. During each encoding process, the target bit-rate was randomly modified every 5 seconds, and we collected 54,000 relevant data points. The experimental results are shown in Fig. 9. The target bit-rate can roughly determine the size of the frames. Therefore, we calculated the Pearson correlation coefficient between the target bit-rate and frame size using all the data, which is 0.576. When the target bit-rate is the same, the variation in frame size is primarily caused by SI and TI. Hence, we calculated the correlation coefficients between frame size and SI, as well as frame size and TI, when the target bit-rate is the same, which are 0.062 and 0.263, respectively. Since the encoder primarily uses residual coding, it is expected that TI exhibits a stronger correlation than SI. This preliminary analysis validates the usability of these variables. Thus, our prediction model is formulated as follows:

$$\mu = W \times [T, SI, TI]'. \tag{8}$$

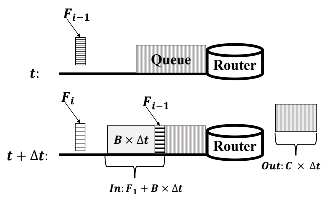


Fig. 10. Schematic diagram of the frame delay model.

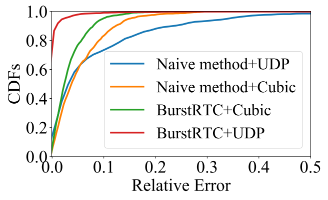


Fig. 11. The frame delay model predicts frame delay precisely.

Here, $W \in R^{1\times3}$ represents the parameter matrix, and R is the target bit-rate. In practice, we record the information for the last 30 frames and then use the least squares method to fit the parameter matrix W. The variance σ is updated using a moving average approach:

$$\sigma_i = \sqrt{\alpha \times \sigma_{i-1}^2 + (1-\alpha) \times (\mu - F_i)^2}, \tag{9}$$

where $\alpha = 0.8$ is the weight. We discretize the above probabilities to get the probability that the frame size is equal to a certain value, $P(F = x) = P(F \le x) - P(F \le x - 1)$.

2) Validation of Model: We validate the usefulness of the frame size model through a statistical experiment. We use the encoding data collected in III-C1. For each frame, we predict the mean size based on its SI, TI, and target bit-rate using the method described above. The experimental results show that the relative error between the predicted mean frame size and the actual frame size is 10% at the 50th percentile and 25% at the 80th percentile, confirming the effectiveness of this approach for predicting frame size.

D. Frame Delay Model

1) Basic Model: Given the probed link capacity ($\hat{C}$) and background traffic ($\hat{B}$), this block is responsible for predicting the frame delay of a frame size distribution (μ, σ). Because the different schemes (bursting and pacing) used by the bandwidth sampler for frames can also affect the overall frame delay (from frame encoding completion to the frame being fully received by the receiver), we define the frame delay (D) as the time from frame encoding completion to the first packet of the frame being received by the receiver.

Consider the following scenario: at time t, frame i-1 with the frame size of F_{i-1} arrives the bottleneck router and its frame delay is D_{i-1}. After a frame interval Δt, frame i arrives at the bottleneck. The schematic diagram of the queue changes during this period is shown in Fig. 10. During Δt, all packets of frame i-1 and several packets of background traffic arrive at the bottleneck, totaling $F_{i-1} + B \times \Delta t$ packets. At the same time, at most $C \times \Delta t$ packets are being transferred out. Thus, the frame delay of frame i can be calculated as:

$$D_i = max\{D_{min}, D_{i-1} + \frac{(F_{i-1} + B \times \Delta t)}{C} - \Delta t\}, \tag{10}$$

where D_{min} is the propagation delay from the sender to receiver (discussed in IV-A1). The above relationship can be written as $D_i = g(D_{i-1}, F_{i-1})$. When F_{i-1} is a Gaussian distribution, frame delay D_i will also follow a distribution.

2) Frame Delay Considering Encoder Adjustment Interval and Delayed Feedback: The use of the frame delay model primarily requires consideration of the two issues: encoder adjustment interval and feedback delay. Due to the software and hardware limitations of commercial video encoders, changing the target bit-rate has at least an L-frames adjustment interval limitation. Therefore, BurstRTC adjusts the target bit-rate at fixed intervals large than L frames. In this case, predicting the delay of just one frame would not yield the optimal target bit-rate. We need to predict the delays of all L frames. We assume that the distribution of frame sizes of L frames does not change. Therefore, the transition probability of their delays remains constant:

$$\begin{gathered} P(D_i = d_i | D_{i-1} = d_{i-1}) = P(F = x), \\ P(D_i = D_{min} | D_{i-1} = d_{i-1}) = P(F \leq y), \\ i = 1, ..., L \end{gathered} \tag{11}$$

where x satisfies $d_i = g(d_{i-1}, x)$, and y satisfies $D_{min} = g(d_{i-1}, y)$. Therefore, the frame delay after any k frames can be represented as:

$$P(D_k) = P(D_0) \prod_{i=1}^{k} P(D_i | D_{i-1}), \tag{12}$$

L is set to one by default, and depends on encoder.

Feedback delay means that the delay of the frames that have been sent out is unknown until at least one round-trip delay later. When we need to calculate the frame delay of frame v, due to feedback delay, the latest known frame delay is from frame $v - u$. Since we know the frame size of the intermediate u frames, we can use an iterative calculation method to determine the frame delay of frame v. First, we calculate the delay of frame $v - u + 1$, then calculate that of frame $v - u + 2$, and so on, until that of frame v is calculated.

3) Validation of Model: We devised a validation experiment to evaluate the usefulness of this model. Our experimental design aims to predict the frame delay 30 frames ahead based on the currently known information. That means both the encoder adjustment interval (L equals 30) and feedback delay are considered. We conducted tests in environments with different background traffic, Cubic and UDP, and used TC to control the link capacity to 4 Mbps and the propagation delay to 20 ms. We ran BurstRTC and recorded the difference between the predicted frame delay and the actual frame delay. To highlight the effectiveness, we introduced a naive method for comparison. The naive method directly uses the latest known frame delay to represent the frame delay 30 frames later. The results shown in Fig. 11 demonstrates a significantly higher prediction accuracy of our frame delay model compared to the naive method, affirming its effectiveness.

E. Analytic Solution

We utilize a direct analytical approach to solve for the optimal target bit-rate, achieving faster convergence than inefficient gradient-based methods. To ensure fairness and minimize frame delay, our utility function is as follows:

$$U = log(T) - \frac{\beta}{k} \sum_{i=1}^{k} (E(D_i) - D_{min}), \tag{13}$$

where T is measured in $kbps$; k is the number of frames; $\beta = 0.15$ (based on experience) is the weight of frame delay; and $E(D)$ is the expected frame delay calculated from frame delay distribution, measured in ms. Whenever we need to make a target bit-rate decision, we first calculate the utility function at $T = C - B$. Then, we gradually decrease T in steps of 100 kbps. When the utility function stops increasing, we set T as the target bit-rate for the next interval.

IV. Evaluation

We evaluate the performance of BurstRTC from the following aspects: (1) Whether BurstRTC effectively addresses the issue of bit-rate variation (IV-B)? (2) How does BurstRTC performs in adverse network conditions (IV-C)? (3) Whether BurstRTC can achieve fairness between flows (IV-D)? (4) How does BurstRTC performs under WI-FI (IV-E)?

A. Setting

1) Implementation Environment.: We adapted BurstRTC from Razor, substituting GCC with our protocol and adjusting the pacer queue to align with our bandwidth sampler. BurstRTC operates over UDP, managing send rates without congestion window limits. Post-camera capture, we compute SI and TI, encode the image, and log the encoded frame size for the frame size model.

Bursting Length. To ensure the accuracy of probing, we first limit the minimum number of packets in the pacing part and the bursting part to 4 packets each. If a frame has fewer than 8 packets, we switch to pacing odd-numbered frames and bursting even-numbered frames. Additionally, to prevent long bursts from causing packet loss, we limit the maximum burst length to 20 packets. If the burst length exceeds this threshold, the excess packets are allocated to the pacing part. Beyond the aforementioned restrictions, frames are, by default, divided into two equal parts and transmitted separately using distinct modes, without establishing initial values.

Network Delay and Packet Loss Handling. The propagation delay D_{min}, a critical parameter in the frame delay model, is not statically set to the minimum observed value due to potential fluctuations. Instead, we treat the propagation delay as a symmetric distribution, using the mean of the longest sequence satisfying this criterion within a 10-second frame delay queue as D_{min}. D_{min} is set to the first packet delay of the first frame when BurstRTC starts. We estimate the packet loss rate $Loss$ using the ratio of the number of bytes received to the number of bytes sent within a history of several frames. When no frames are received, $Loss$ is set to 0. Consequently, the target bit-rate is adjusted to $T \times (1 - Loss)$ to accommodate potential losses.

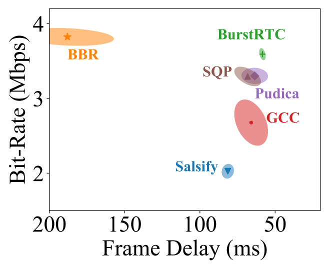


Fig. 12. BurstRTC achieves high bit-rate and low frame delay when transmitting static videos.

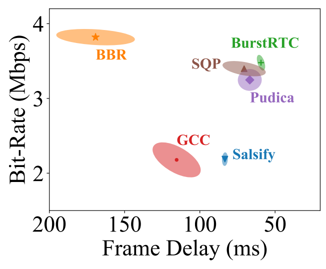


Fig. 13. BurstRTC achieves high bit-rate and low frame delay when transmitting dynamic videos.

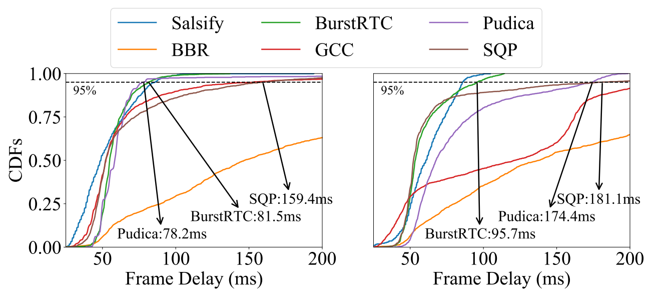


Fig. 14. BurstRTC achieves lower tail frame delay when transmitting static videos (Left) and dynamic videos (Right).

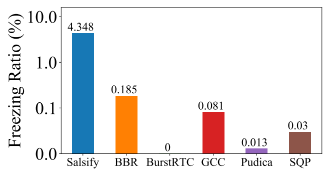


Fig. 15. BurstRTC achieves a lower freezing time ratio when transmitting static videos.

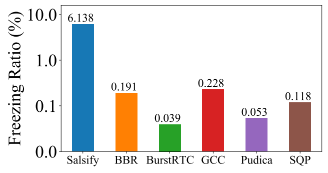


Fig. 16. BurstRTC achieves a lower freezing time ratio when transmitting dynamic videos.

2) Testbed: We compared five leading-edge algorithms: GCC [13], BBR [12], Salsify [19], Pudica [23], and SQP [22]. GCC and BBR are two network-oriented congestion control algorithms and are already deployed in WebRTC. Salsify is designed for tighter integration between the video codec and the transport protocol, with open-source implementations. SQP and Pudica are newest packet-train based RTC algorithms. We deployed the Pudica and SQP algorithms on the WebRTC demo. In addition to the parameters explicitly given in their papers, we specified Pudica's maximum step size for AI (Additive Increase) as 800 kbps, and the minimum step size as 100 kbps. We used two Ubuntu desktops as the sender and receiver, respectively. In the simulation experiments, to test different network conditions, we employed mahimahi [37] to simulate network traces from our design or Lumous 5G [30]. The static video transmitted in the following experiments is the animation video "Big buck bunny" [25] and the dynamic video is the game video "Fallout 4" [26]. We also used the other videos from this resource, totaling 10 game videos. All videos are adjusted to 1080P and 30fps. In the subsequent experiments, we primarily compared metrics such as bit-rate, frame delay, freezing time ratio, and fairness. To prevent confusion, we clarify that the frame delay in the following description refers to the time from a frame's encoding completion to the frame being fully received by the receiver, which is different from the frame delay used in BurstRTC as mentioned earlier. We define any interval between two frames that exceeds 100 ms as freezing time. The proportion of freezing time to the total transmission time is referred to as the freezing time ratio. The fairness metric we use is Jain's fairness index [38].

B. Addressing Variable Bit-Rate

1) Overall Performance: To evaluate the performance of BurstRTC in addressing the challenge of bit-rate variation, we first conduct a controlled experiment for transmitting videos with different content (static video and dynamic video). We use mahimahi to control the link's capacity to 4 Mbps and the propagation delay to 20 ms. The average frame delay and average bit-rate per second are shown in Fig. 12 and Fig. 13. The ellipse represents the 25% boundary. BurstRTC maintains high bit-rate and low frame delay across both scenarios, surpassing other algorithms. BurstRTC's frame size distribution modeling and adaptive response to bit-rate variation yield slightly better throughput and lower delay than SQP and Pudica. Network-oriented GCC suffers higher frame delay and lower throughput, especially with dynamic video, due to suboptimal bandwidth estimation and slow convergence. BBR's aggressive bandwidth approach significantly increases frame delay. Salsify, despite encoder enhancements, has reduced encoding efficiency, limiting the frame rate to 11.6 fps (versus 30 fps for others), thus lowering the average bit-rate.

2) Tail Latency: Tail latency is a significant optimization of BurstRTC after considering bit-rate variation. We further analyzed the tail latency (95th percentile frame delay) from the experiment in IV-B1, and the results are shown in Fig. 14. We focus on the three algorithms with the best overall performance: SQP, Pudica, and BurstRTC. For static video, Pudica and BurstRTC offer similar low tail frame delays of 78.2 ms and 81.5 ms, respectively. In dynamic video, Pudica's delay increases to 174.4 ms, while BurstRTC remains low at 95.7 ms due to its precise consideration of bit-rate variation. SQP consistently has higher delays at 159.4 ms for static and 181.1 ms for dynamic video.

3) Freezing Time Ratio: Inappropriate handling of bit-rate variation can further affect application-level metrics, primarily manifesting as freezing. We further analyzed the freezing time ratio from the experiment in IV-B1, and the results are shown in Fig. 15 and Fig. 16. For both static and dynamic videos, BurstRTC achieved extremely low freezing times, specifically 0% and 0.039% (23.4 ms in 60 s), respectively, benefiting from modeling of bit-rate variation and properly adjusting the target bit-rate. Network-oriented GCC and BBR, in addition to network delay, also have a significant portion of sending buffer delay, resulting in a higher overall freezing ratio. SQP and Pudica, in their target bit-rate decision-making, do not consider bit-rate variation, which introduces a slightly higher risk of freezing compared to BurstRTC.

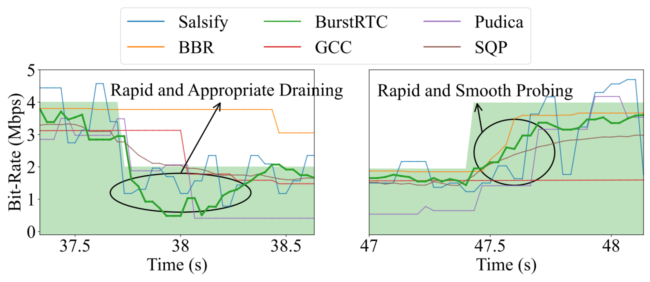


Fig. 17. BurstRTC achieves better responsiveness when link capacity decreases (Left) and increases (Right).

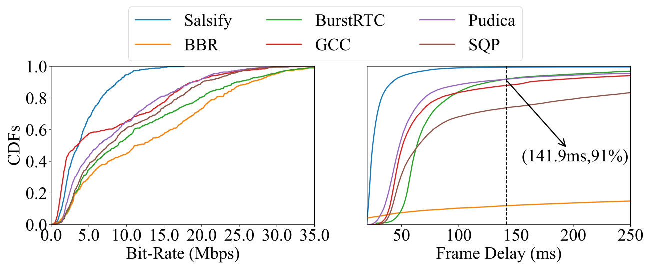


Fig. 18. BurstRTC achieves the highest bit-rate under network trace, except for BBR, which has high frame delay (Left). BurstRTC achieves a low frame delay, especially tail frame delay, under network trace (Right).

4) *Responsiveness:* This experiment tests the responsiveness of algorithms to network capacity changes, primarily manifested by the efficiency of burst-driven bandwidth estimation and our analytical solution. We use the same configuration as IV-B1 but reduce the link capacity to 2 Mbps at approximately 38 seconds and restore it 10 seconds later. The transmitted video is the dynamic video. The decision sequence of target bit-rates for the algorithms is shown in Fig. 17. To accurately reflect Salsify's response speed, we set the frame rate of Salsify to 12, while the rest of the settings remain the same. (According to the above experiments, Salsify's processing capability is insufficient at high frame rates.) BurstRTC exhibits the best responsiveness to changes in network capacity. When the capacity drops, the rate required for draining the queue can be accurately calculated, and when the capacity rises, its target bit-rate is smooth and converges within 10 frames (because we use the average bandwidth of 10 frames in III-B). In contrast, Pudica's AIMD-based detection mode cannot accurately calculate the target bit-rate needed to clear the congestion queue, and the trial-based method makes its decision sub-optimal and unstable. Gradient-based SQP can ensure smoothness, but its responsiveness to bandwidth drops is slow, resulting in a larger congestion queue. Network-oriented GCC is slow when the bandwidth increases, and BBR is slow when the bandwidth decreases. MIMD-based Salsify's target bit-rate fluctuation is very large.

C. Handling Bad Cases in Network Condition

1) *Network Capacity Trace:* In this experiment, we focus on the ability of the algorithms to respond to continuous changes in network capacity. We select the

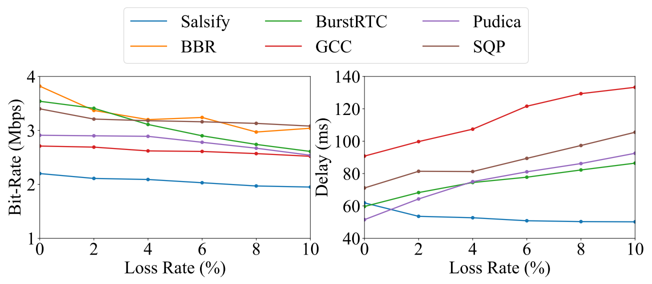


Fig. 19. BurstRTC maintains a high bit-rate (Left) and low frame delay (Right) in packet loss scenarios.

"4G_trace_walking_60049_a" trace in Lumous, which lasts about 10 minutes and has a link capacity ranging from 0.5 Mbps to 90.5 Mbps, averaging 16.52 Mbps. The reason for this selection is that algorithm differences cannot be detected if we choose a high link capacity. On top of that, we add an additional 20 ms of propagation delay. We splice all the game videos together into a 10-minute video, which is transmitted under the above network trace. The experimental results are shown in Fig. 18. Except for BBR, BurstRTC maintains the highest bit-rate, averaging 11.35 Mbps, 15.6% higher than Pudica and 13.7% higher than SQP. This is because it can always find the best target bit-rate in the case of bit-rate variation of the video encoder and avoid unnecessary target bit-rate decreases caused by high delay. Although Fig. 18 shows a slight increase in its average delay, this is mainly due to the high frame delay of larger frames, which is exacerbated by unpredictable link capacity fluctuations. However, due to its good responsiveness and consideration of bit-rate variation, the tail frame delay after 91% of BurstRTC remains minimal.

2) *Random Loss:* This experiment tests the ability of the algorithms to resist random packet loss. Although packet loss resistance is not a primary contribution of BurstRTC, we have made a corresponding response. We use the same configuration as in IV-B1 to transmit the game video spliced in IV-C1. In addition, we add random packet loss ranging from 2% to 10% to the network. Note that in packet loss scenarios, the frame delay of the algorithms includes the delay of packet retransmission. The experimental results are shown in Fig. 19. The results show that the average bit-rate of BurstRTC is slightly reduced in the face of random packet loss, but it can still maintain its characteristics. This demonstrates the effectiveness of BurstRTC in dealing with packet loss. Additionally, BurstRTC reserves enough bandwidth for retransmission packets to ensure their timely arrival, so the frame delay is not significantly increased. Because of Salsify's unique design, the frame delay does not increase, but decreases due to the drop in bit-rate. BBR, however, has a frame delay of more than 140 ms.

3) *Propagation Delay Jitter:* This experiment tests the ability of the algorithms to resist propagation delay jitter. Delay jitter resistance is also not a primary contribution of BurstRTC, but we have made a corresponding response. We use the same configuration as in IV-B1 to transmit the game video spliced

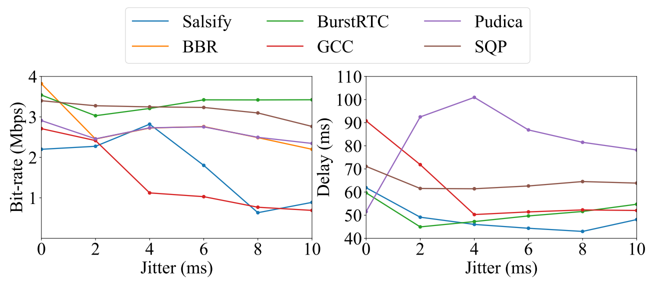


Fig. 20. BurstRTC maintains a high bit-rate (Left) and low frame delay (Right) in propagation delay jitter scenarios.

TABLE I
JAIN'S FAIRNESS WHEN TWO FLOWS COMPETE.

	Salsify	**BBR**	**BurstRTC**	**GCC**	**Pudica**	**SQP**
Fairness	0.980	0.9667	0.949	0.840	0.986	0.985

in IV-C1. In addition, we add propagation delay jitter ranging from 2ms to 10ms to the network. The experimental results are shown in Fig. 20. The results show that BurstRTC is almost unaffected by delay jitter. Meanwhile, both Salsify and GCC experience a significant bit-rate decrease. AIMD-based Pudica, due to its congestion signal no longer being accurate, has a larger frame delay.

D. Converging to Fairness

This experiment verifies whether the algorithm can fairly allocate bandwidth when competing with flows carrying the same algorithm. We use the same configuration as in IV-B1 to transmit fixed-size frames according to the target bit-rate. We do this to exclude the effect of bit-rate variation on fairness. The experimental results are shown in Tab. I. All algorithms achieve good fairness in this scenario. Although SQP can fairly share bandwidth, it does not guarantee convergence of frame delay in more complex scenarios [23]. This is mainly because, in order to ensure competitiveness, it reduces the measurement window for propagation delay. GCC's slightly lower fairness mainly comes from its slower convergence. To more intuitively show how BurstRTC converges, we further present the timing diagram of its bit-rate and frame delay in Fig. 21 and Fig. 22. Since BurstRTC directly calculates the target bit-rate without a gradient-based convergence process, it cannot smoothly converge to fairness, and the bit-rate is not stable after convergence. However, due to the fairness guaranteed by the utility function, it still ensures fairness.

E. Performance under WI-FI

In this experiment, video transmission is conducted over a Wi-Fi, with the algorithm configuration and video content identical to previous tests. An Ubuntu desktop with a public IP is connected to the internet via wired connection. A CPE [29], also wired to the internet, converts this connection into a 5GHz 802.11ax (WiFi 6) signal, utilizing a 160MHz channel bandwidth. The sender accesses the internet through this WiFi and connects to the server. Each algorithm is run for 5 minutes. The results shown in Tab. II indicate that BurstRTC maintains a consistently high bitrate and low latency. The average bitrate is slightly lower than SQP due to Wi-Fi fluctuations. SQP achieves the highest throughput due to its aggressiveness (manifesting as a slower reduction in response to high frame delay), at the cost of increased frame delay. The AIMD-based Pudica exhibits instability, resulting in a wide range of bitrates and frame delays, despite having a comparable average bitrate.

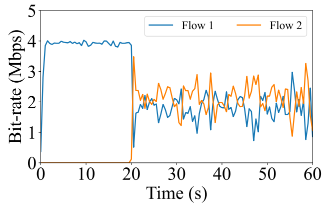


Fig. 21. Two flows carrying BurstRTC converge to fairness.

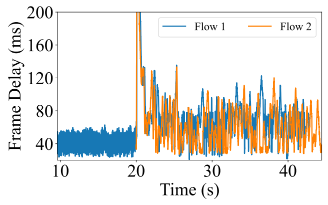


Fig. 22. The frame delay converges in the competitive scenario.

TABLE II
BURSTRTC ACHIEVES GOOD PERFORMANCE UNDER WIFI.

	Average Bit-Rate (Mbps)	Frame Delay (ms)	95% Frame Delay (ms)
Salsify	5.14	43.16	72.74
BBR	7.70	1787.4	2990.99
BurstRTC	**19.40**	**32.35**	**73.21**
GCC	6.84	16.07	44.26
Pudica	15.12	44.66	104.75
SQP	23.13	77.68	279.47

V. CONCLUSION

This paper introduces BurstRTC, a frame-bursting congestion control algorithm aimed at mitigating the challenges posed by overshooting and undershooting. The conflicts between the encoder's bit-rate variation and network-oriented congestion control severely affect the experience of RTC applications. We deeply analyze the irrationality of this design in the RTC scenario and propose a frame-bursting congestion control to solve these problems. To fully implement the frame-bursting design, we overcome the shortcomings of existing research and propose sampling available bandwidth within one frame, modeling frame size and frame delay, and determining the target bit-rate with an analytical method. The necessity of these designs is also discussed. Our experimental results suggest that BurstRTC effectively resolves our concerns and achieves better performance.

ACKNOWLEDGMENT

We sincerely thank our shepherd Aaron Striegel and anonymous reviewers for their valuable feedbacks. This work was sponsored by the NSFC grant(U21B2012), Huawei Grant(TC20230914045). We gratefully acknowledge the support of State Key Laboratory of Media Convergence Production Technology and Systems, Key Laboratory of Intelligent Press Media Technology, State Key Laboratory of General Artificial Intelligence, Peking University, Beijing, China.

References

[1] I. Global Industry Analysts, "Web Real-Time Communication (WebRTC) - Global Strategic Business Report." [Online]. Available: https://www.researchandmarkets.com/reports/4806411/

[2] Z. Zhang, H. Chen, X. Cao, and Z. Ma, "Anableps: Adapting Bitrate for Real-Time Communication Using VBR-encoded Video," in *2023 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2023, pp. 1685–1690.

[3] Y. Li, Z. Zhang, H. Chen, and Z. Ma, "Mamba: Bringing Multi-Dimensional ABR to WebRTC," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 9262–9270.

[4] Y. Li, H. Chen, B. Xu, Z. Zhang, and Z. Ma, "Improving Adaptive Real-Time Video Communication via Cross-layer Optimization," *arXiv preprint arXiv:2304.03505*, 2023.

[5] 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 *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[6] K. MacMillan, T. Mangla, J. Saxon, and N. Feamster, "Measuring the Performance and Network Utilization of Popular Video Conferencing Applications," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 229–244.

[7] H. Lim, J. Lee, J. Lee, S. D. Sathyanarayana, J. Kim, A. Nguyen, K. T. Kim, Y. Im, M. Chiang, D. Grunwald *et al.*, "An Empirical Study of 5G: Effect of Edge on Transport Protocol and Application Performance," *IEEE Transactions on Mobile Computing*, 2023.

[8] J. He, M. Ammar, and E. Zegura, "A Measurement-Derived Functional Model for the Interaction Between Congestion Control and QoE in Video Conferencing," in *International Conference on Passive and Active Network Measurement*. Springer, 2023, pp. 129–159.

[9] A. Hassan, A. Narayanan, A. Zhang, W. Ye, R. Zhu, S. Jin, J. Carpenter, Z. M. Mao, F. Qian, and Z.-L. Zhang, "Vivisecting Mobility Management in 5G Cellular Networks," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 86–100.

[10] X. Tian, S. Jia, P. Dong, T. Zheng, and X. Yan, "An Adaptive Bitrate Control Algorithm for Real-Time Streaming Media Transmission in High-Speed Railway Networks," in *2018 10th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2018, pp. 328–333.

[11] J. Eickhoff, J. Donkervliet, and A. Iosup, "Meterstick: Benchmarking Performance Variability in Cloud and Self-hosted Minecraft-like Games Technical Report," *arXiv preprint arXiv:2112.06963*, 2021.

[12] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[13] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and Design of the Google Congestion Control for Web Real-time Communication (WebRTC)," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, pp. 1–12.

[14] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," *Technical report*, 1990.

[15] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," Tech. Rep., 2004.

[16] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[17] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[18] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 329–342.

[19] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein, "Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 267–282.

[20] P. S. Alan Jones and R. Wetzel., "Requirements for Effective Video Conferencing to Support Work from Home and eLearning — NetForecast." 2021. [Online]. Available: https://www.netforecast.com/wp-content/uploads/NFR5137-Videoconferencing_Internet_Requirements.pdf

[21] R. Jain and S. Routhier, "Packet Trains–Measurements and a New Model for Computer Network Traffic," *IEEE journal on selected areas in Communications*, vol. 4, no. 6, pp. 986–995, 1986.

[22] D. Ray, C. Smith, T. Wei, D. Chu, and S. Seshan, "SQP: Congestion Control for Low-Latency Interactive Video Streaming," *arXiv preprint arXiv:2207.11857*, 2022.

[23] S. Wang, S. Yang, X. Kong, C. Wu, L. Jiang, C. Xu, C. Zhao, X. Yang, J. Xiao, X. Liu *et al.*, "Pudica: Toward Near-Zero Queuing Delay in Congestion Control for Cloud Gaming," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 113–129.

[24] A. S. Jagmagji, H. D. Zubaydi, and S. Molnar, "Exploration and Evaluation of Self-Clocked Rate Adaptation for Multimedia (SCReAM) Congestion Control Algorithm in 5G Networks," in *2022 45th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2022, pp. 230–237.

[25] T. Roosendaal, "Big Buck Bunny," in *ACM SIGGRAPH ASIA 2008 computer animation festival*, 2008, pp. 62–62.

[26] C. Montgomery *et al.*, "Xiph. org Video Test Media (Derf's collection), The Xiph Open Source Community, 1994," *Online, https://media. xiph. org/video/derf*, vol. 3, no. 5, 2021.

[27] A. B. Johnston and D. C. Burnett, *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, 2012.

[28] yuanrongxi. (2024) A Google's Congestion Control Algorithm. Accessed on: 2024-9-7. [Online]. Available: https://github.com/yuanrongxi/razor

[29] L. Huawei Technologies Co. (2024) HUAWEI 5G CPE Pro 2. Accessed on: 2024-9-7. [Online]. Available: https://consumer.huawei.com/en/routers/5g-cpe-pro-2/

[30] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao *et al.*, "A Variegated Look at 5G in the Wild: Performance, Power, and QoE Implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 610–625.

[31] M. P. Stanic, "TC–Traffic Control," *Linux QOS Control Tool*, 2001.

[32] R. Globisch, K. L. Ferguson, Y. Sanchez, and T. Schierl, "Step Response Metric for Video Encoder Rate Control Characterisation," in *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2016, pp. 1–6.

[33] L. Massoulie, "Stability of Distributed Congestion Control with Heterogeneous Feedback Delays," *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 895–902, 2002.

[34] S. Shakkottai, R. Srikant, and S. P. Meyn, "Bounds on the Throughput of Congestion Controllers in the Presence of Feedback Delay," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, pp. 972–981, 2003.

[35] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-Dispersion Techniques and a Capacity-Estimation Methodology," *IEEE/ACM Transactions On Networking*, vol. 12, no. 6, pp. 963–977, 2004.

[36] M. Lecci, M. Drago, A. Zanella, and M. Zorzi, "An Open Framework for Analyzing and Modeling XR Network Traffic," *IEEE Access*, vol. 9, pp. 129 782–129 795, 2021.

[37] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate Record-and-Replay for HTTP," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 417–429.

[38] R. K. Jain, D.-M. W. Chiu, W. R. Hawe *et al.*, "A Quantitative Measure of Fairness and Discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, vol. 21, p. 1, 1984.