Received 13 June 2020; revised 22 July 2020; accepted 11 August 2020. Date of publication 14 August 2020; date of current version 15 September 2020.

Digital Object Identifier 10.1109/OJCOMS.2020.3016768

PIBES - A Competing-Flow-Aware Protocol for Real-Time Video Applications

LIOBA HEIMBACH[1], LINGFENG GUO[2], RUDOLF K. H. NGAN[2], AND JACK Y. B. LEE[2] (Senior Member, IEEE)

[1]Department of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland

[2]Department of Information Engineering, Chinese University of Hong Kong, Hong Kong

CORRESPONDING AUTHOR: J. Y. B. LEE (e-mail: yblee@ie.cuhk.edu.hk)

ABSTRACT With the recent explosive growth in online classes and virtual meetings, real-time video communication has quickly become essential to everyday life. Despite its widespread deployment, our investigation revealed that current protocols, ranging from industry standards such as WebRTC to state-of-the-art research such as Salsify, frequently perform sub-optimally in the presence of competing flows at the same bottleneck. For example, WebRTC's throughput can degrade from 73% to a mere 8% of the available bandwidth when competing with just two TCP flows. We tackle this problem in this work by introducing a novel PIBES protocol for real-time video applications to operate in the presence of competing TCP traffic. PIBES employs a new inband bandwidth estimation method that can quickly and accurately measure the bottleneck link bandwidth even with competing flows. Moreover, PIBES can detect the absence or presence of competing flows, which enables it to maximize video quality when there is no competing flow and to maintain acceptable video quality while sharing bandwidth with competing flows. Experiments demonstrate that PIBES achieves throughput and delay comparable to the state-of-art protocols, but outperforms them significantly in the presence of competing TCP flows.

INDEX TERMS Real-time, video communication, congestion control, bitrate control, bandwidth estimation.

I. INTRODUCTION

ALREADY comprising the largest share of Internet traffic with 75% in 2017, the proportion of video traffic is expected to increase even further to 82% by 2022 [1]. In particular, live video traffic increases progressively – making up a more substantial fraction of video traffic. Real-time video traffic is predicted to increase 15-fold between 2017 and 2022 [1]. The growing popularity of video conferencing applications such as Skype, FaceTime, Google Hangouts, and Zoom, as well as live video streaming applications such as Facebook Live, Instagram Live, and Periscope, are drivers behind this increase in real-time video traffic.

Real-time video traffic presents additional challenges as its performance is affected by both throughput and delay. However, the Transmission Control Protocol (TCP), which represents 85% to 90% of Internet traffic [2], has been optimized for goodput [3]. While suitable for Web browsing and file transfer, as well as traffic with weak real-time characteristics such as on-demand video streaming systems, TCP's loss-based congestion control algorithm and retransmission mechanism are inherently not suitable for throughput and delay-sensitive traffic such as real-time video.

In light of this, video conferencing applications primarily streamed video over UDP with custom-designed congestion control schemes implemented inside the application [4]. To enhance interoperability, Google recently developed the WebRTC [5] framework that supports both standalone and browser-based real-time applications, further fueling their growth. However, our investigation uncovered an often-neglected issue when operating real-time applications in practice.

It has recently become common for multiple users to share an Internet connection at home via a home network. Consequently, real-time video traffic must coexist and compete for bandwidth against other data traffics such as Web or non-real-time video streaming. In one experiment, we built

TABLE 1. Impact of competing traffic on WebRTC and salsify.

System	Bandwidth Utilization in the Presence of		
	0 TCP flows	1 TCP flow	2 TCP flows
WebRTC	73%	24%	8%
Salsify	92%	37%	35%

a testbed with a shared Internet link of 4Mbps bandwidth. We initiated one real-time video flow together with zero to two competing TCP flows to measure their performance. The results, summarized in Table 1, demonstrated a severe limitation of current real-time protocols – their throughput performance can degrade significantly even in the presence of just a single TCP flow. While WebRTC can achieve a reasonably high bandwidth utilization of 73% alone, it declined to a mere 24% with one, and 8% with two competing TCP flows at the bottleneck link. This sharp decline in link utilization is not limited to WebRTC; we repeated the experiment with Salsify [6] – a recently presented state-of-the-art real-time protocol and observed similar performance degradations in the presence of competing TCP flows.

The implication is that video quality in real-time applications will significantly degrade whenever there are competing traffics. In severe cases, the achievable throughput can drop below the minimum required by the video encoder, leading to loss of the video feed entirely. In retrospect, this does occur in practice, but our experiments showed that lack of bandwidth is not the only cause.

In this work, we tackle this challenge by developing a novel *packet inter-arrival time bandwidth estimation for short latency* protocol (PIBES) for transporting real-time video traffic in the presence of competing cross-traffic. PIBES employs a new inband bandwidth estimation method that can quickly and accurately measure the bottleneck link bandwidth even with competing flows. Moreover, PIBES can detect the absence or presence of competing flows, which enables it to maximize video quality when there is no competing flow and to maintain acceptable video quality through explicit control of bandwidth sharing with competing flows. Experiments demonstrate that PIBES achieves throughput and delay comparable to the state-of-the-art protocols and outperforms them significantly in the presence of competing traffics.

II. BACKGROUND AND RELATED WORK

Traditional loss-based TCP congestion control [7]–[9] is unsuited for real-time applications. Bandwidth probing introduces periodic queuing delays, as the bottleneck queue is continuously filled up and drained. Thus, inducing significant latency – making it inapt for delay-sensitive traffic.

In a quest to avoid the oscillations inherent to loss-based protocols, various delay-aware congestion control algorithms have surfaced in recent years. TCP Vegas [10] and Fast TCP [11] rely on round trip time (RTT) to detect congestion. However, reverse traffic can reduce link utilization of RTT-based congestion control algorithms [12]–[14].

Congestion control algorithms based on one-way delay do not suffer from this issue. TCP Santa Cruz [15] uses delay estimation along the forward path to infer congestion. However, when competing against loss-based flows, one-way delay-based congestion control algorithms often lose bandwidth to their loss-based counterparts [16]. LEDBAT [17] also uses one-way delay for congestion detection but is affected by the *latecomer effect* [18], where the first flow can be starved by a second flow entering the same bottleneck link.

WebRTC [5] was initiated to standardize protocols and APIs for real-time services among Web browsers. WebRTC has been incorporated into major Web browsers, including Chrome and Firefox. Three algorithms have been designed for streaming live video using RTP/RTCP over UDP as part of the WebRTC initiative: SCReAM [19], NADA [20], and GCC [21], [22]. SCReAM is based on LEDBAT, similar to its predecessor, SCReAM has short latency but low channel utilization [23]. NADA uses a composite congestion signal, which consists of delay, loss, and explicit congestion notification markings. While NADA achieved the highest channel utilization of the three algorithms, it suffers from the latecomer effect [23]. Similar to SCReAM, GCC largely relies on one-way delay measurements. GCC achieves good fairness property and maintains a reasonable packet sending rate in the presence of losses, but is slow in tracking bandwidth variations [23].

Apart from delay, another approach to congestion control is via bandwidth estimation. TCP Westwood [24] first uses available bandwidth estimations from packet inter-arrival times for congestion control. However, TCP Westwood faces problems in the presence of reverse traffic due to ACK compression. More recently, Google proposes TCP BRR [25], which also uses available bandwidth estimation to regulate its sending rate. A recent study [26] showed that fairness amongst TCP BBR flows could be affected by flow RTTs and bottleneck buffer size. Moreover, TCP BBR tends to starve competing TCP flows when the bottleneck buffer is small [27].

Sprout [28] was designed for interactive applications requiring high throughput and low delay, and uses packet inter-arrival times at the receiver to estimate how many bytes the sender can send. In doing so, Sprout assumes that the sender always has something to send, which is not the case in real-time video applications [29]. Most recently, Salsify [6] expanded Sprout-EWMA to become video-aware. Additionally, Salsify integrates the transport protocol with the video encoder's rate-control algorithm to achieve more precise rate control.

While many of the previous work on real-time transport can achieve good throughput with low delay, they can only do so in the *absence* of competing flows. As discussed in Section I, competing traffic is now the norm rather than the exception. Therefore, a real-time protocol's behavior in the presence of competing flows is essential to its performance in practice.

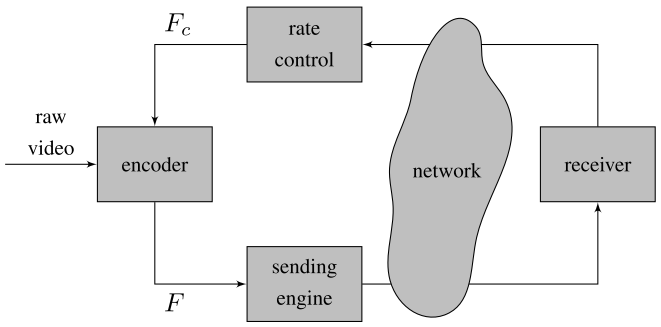


FIGURE 1. Congestion and rate control architecture.

III. SYSTEM DESIGN

Designed for high-quality real-time video flows, PIBES aims at achieving high utilization while keeping video frame delay short in the absence of competing flows. On the other hand, when competing flows are presence at the shared bottleneck link, we want PIBES to reasonably share the bandwidth with its counterparts while avoiding starvation. PIBES employs an inband method to continuously estimate the available bandwidth and sense the presence/absence of competing traffic to dynamically adapt the video bitrate and sending rate to achieve the above goals.

The congestion and rate control architecture in PIBES is depicted in Figure 1. The receiver records the incoming packets' timestamps to compute a smoothed inter-arrival time and detect competing flows sharing the bottleneck link. This information is returned to the sender through ACKs. With this information, the sender calculates the target frame size F_c for the video encoder and a sending rate R_c for pacing the outgoing packets. The video encoder captures and encodes video frames according to a preset frame rate. Note that the encoded frame size, denoted by F, may deviate slightly from the target in practice. The sending engine packetizes the video frame into UDP datagrams and then transmits them in a two-part transmission schedule. We present details of the system components in the following subsections.

A. PACKET INTER-ARRIVAL TIME BANDWIDTH ESTIMATION

As reviewed in Section II, bandwidth estimation has been employed in numerous previous work, e.g., TCP Westwood [24], BBR [25], Sprout [29], and Salsify [6]. Nevertheless, our investigation reveals that existing bandwidth estimation algorithms often converge slowly towards the available bandwidth and may not be able to adapt to the changing network condition quickly. This is because outgoing packets are often paced implicitly (via TCP ACKs) or explicitly (via sending rate control) so that it takes considerable time for the measured bandwidth to converge towards the available bandwidth.

Another approach is to transmit packets in a burst at a high datarate to allow more accurate measurement of the link bandwidth. This approach is particularly suitable in real-time video communication as packets in a frame are generated

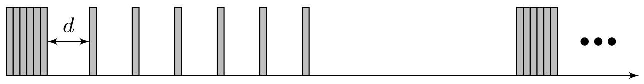


FIGURE 2. Two-part transmission scheduler.

simultaneously by the video encoder at the sender, which can then be transmitted in a burst. However, with today's increased video quality (e.g., 720P or 1080P), sending a large video frame in a single transmission burst could cause router buffer overflow, leading to unnecessary congestion losses. Therefore, we propose a novel two-part transmission scheduler depicted in Figure 2 to enable accurate bandwidth estimation via initial short transmission burst, followed by paced transmission to prevent congestion loss.

Specifically, a fixed number of packets, n_b, are sent out in a burst at the beginning of each frame. Here,

$$n_b = \max\left(n_{\min}, \min\left(\left\lceil \frac{n_f}{2} \right\rceil, n_{\max}\right)\right), \tag{1}$$

where n_f is the number of packets in the frame, and $[n_{\min}, n_{\max}]$ is the range of the number of packets in a burst.

The lower limit $n_{\min}$ is to ensure sufficient accuracy in bandwidth estimation in case the video frame is too small. The upper limit $n_{\max}$ is to keep the burst size small for large video frames so that the burst transmission will not cause unnecessary network congestion.

After the initial burst, the sender paces the remaining packets (Figure 2) with an inter-departure time of d seconds, where d is the most recent smoothed inter-arrival time the sender received – see (2). Note that the pacing rate is set to the estimated link bandwidth instead of video bitrate so as to minimize frame delivery delay while preventing congestion at the bottleneck link.

At the receiver, the initial packet burst in each frame is then used as a packet train to maintain an exponential moving average of the packet inter-arrival time: with each arrival the receiver calculates the smoothed inter-arrival time, d_i, from

$$d_i = \begin{cases} t_i & i = 0 \\ \alpha \cdot t_i + (1-\alpha) \cdot d_{i-1} & i > 1, \end{cases} \tag{2}$$

where $i \in \mathbb{N}$ enumerates the time interval observations of burst packets across frames and t_i is the most recent observed time interval.

The smoothed packet inter-arrival time is then returned to the sender via ACKs. There, the rate control block periodically estimates the path bandwidth,

$$B_k = \frac{P}{d}, \tag{3}$$

from the most recent smoothed inter-arrival time, d, reported by the receiver, and the packet size P. $k \in \mathbb{N}$ identifies the frame to be encoded at the sender.

B. COMPETITION-AWARE RATE ADAPTATION

The target sending rate is set by the rate control block using the estimated link capacity and knowledge about the presence of competing flows at the bottleneck link. While most

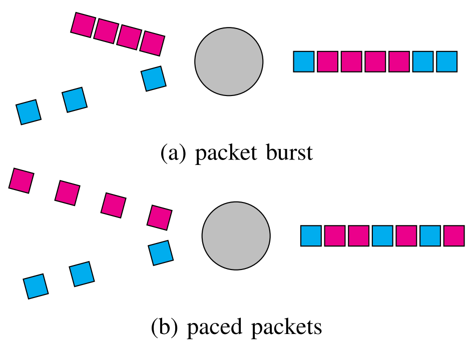


FIGURE 3. **Packets from competing flows (blue) are more likely to interject in-between paced packets than burst packets.**

previous competition detection algorithms relied on monitoring changes in queue build-up speed [30]–[32], the two-part transmission scheduler enables a far simpler and quicker way to detect competing traffic.

In particular, the presence of competing flows can be directly inferred from packet inter-arrival times of the paced packets monitored by the receiver. The intuition is that packets from competing flows will more likely interject between paced video packets due to their lower data rate (Figure 3). The interjected packets will then increase the inter-arrival times of the paced video packets. Thus, by comparing the smoothed inter-arrival times of paced versus burst packets, the presence of competing flows can then be detected.

Specifically, we can compute the smoothed inter-arrival times for paced packets, denoted by $\tilde{d}_j$, similar to (2), from

$$\tilde{d}_j = \begin{cases} \tilde{t}_j & j = 0 \\ \alpha \cdot \tilde{t}_j + (1-\alpha) \cdot \tilde{d}_{j-1} & j > 1, \end{cases} \tag{4}$$

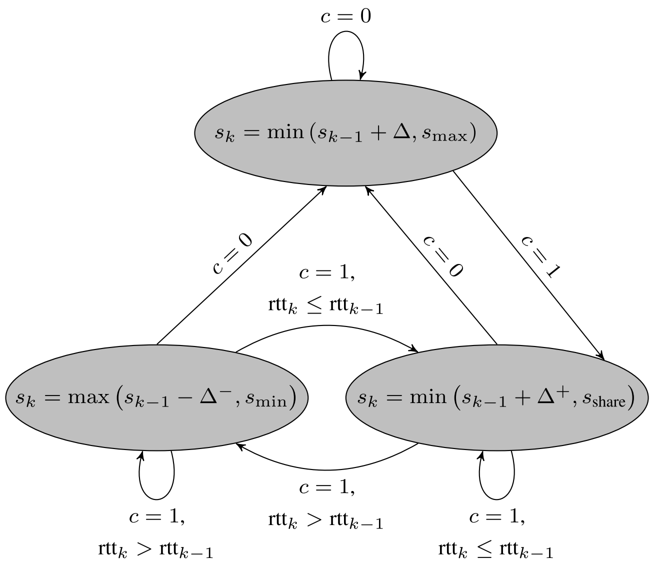
where $\tilde{t}_j$ is the observed inter-arrival time for paced packets and $j \in \mathbb{N}$ enumerates the observations across frames. Let $\tilde{d}$ be the latest smoothed inter-arrival time for paced packets. If the $\tilde{d}$ for paced packets exceeds the d for burst packets by more than β (e.g., 10%) then the receiver declares the presence of competing flows and informs the sender via ACK packets.

Combining the knowledge of the path bandwidth and the presence of competing traffic at the bottleneck link, the rate control block computes the target sending rate from

$$R_k = \max(R_{\min}, \min(s_k \cdot B_k, R_{\max})). \tag{5}$$

Here, $k \in \mathbb{N}$ corresponds to the frame number, $[R_{\min}, R_{\max}]$ is the rate range of the encoder, and $s_k \in [0, 1]$ is the proportion of link capacity the sender intends to use which is governed by the Finite State Machine (FSM) in Figure 4.

The FSM is where competition-awareness is implemented. Specifically, let $c = \{0, 1\}$ be a binary variable representing the presence ($c = 1$) or absence ($c = 0$) of competing flows. The target bandwidth share is initialized to $s_0 = s_{\max}$ and is updated whenever a new frame is encoded. The FSM operates in two modes. If there is no competing traffic at the beginning, it simply transmits video at the maximum bandwidth share of $s_{\max}$. We set $s_{\max}$ to be slightly smaller than 1



$c = 0$
$s_k = \min(s_{k-1} + \Delta, s_{\max})$
$c = 0$
$c = 0$
$c = 1$
$c = 1$, $\mathrm{rtt}_k \le \mathrm{rtt}_{k-1}$
$s_k = \max(s_{k-1} - \Delta^-, s_{\min})$
$s_k = \min(s_{k-1} + \Delta^+, s_{\mathrm{share}})$
$c = 1$, $\mathrm{rtt}_k > \mathrm{rtt}_{k-1}$
$c = 1$, $\mathrm{rtt}_k > \mathrm{rtt}_{k-1}$
$c = 1$, $\mathrm{rtt}_k \le \mathrm{rtt}_{k-1}$

FIGURE 4. **Finite state machines to adapt the target bandwidth share s_k.**

(e.g., 0.95) to prevent overshooting the link capacity due to bandwidth estimation errors. Being slightly conservative can also minimize latency when there is no competing traffic.

Once a competing flow is detected ($c = 1$), s_k will be reduced to s_{share}, where $s_{\mathrm{share}} < s_{\max}$. Intuitively, s_{share} controls the target proportion of bandwidth to share with the competing flows. However, if the competing traffic is intense, congestion may still occur, leading to long delays and packet losses. Therefore, PIBES will continuously monitor the RTT and then further adapt the sending rate according to the intensity of the competing traffic.

Specifically, let rtt_k be the average round trip time of packets in frame k. If it has increased since the last frame, i.e., $\mathrm{rtt}_k > \mathrm{rtt}_{k-1}$, then it indicates that the combined traffic overshoots the link capacity. To prevent congestion, PIBES will decrease the send proportion by Δ^- per frame, down to a lower limit of $s_{\min}$ to maintain acceptable video quality.

On the other hand, if rtt_k decreases, then PIBES will progressively increase the send proportion by Δ^+ per frame until the target bandwidth share of s_{share} is restored. As soon as the competing flow leaves the network ($c = 0$), PIBES will further increase the send proportion by Δ per frame up to $s_{\max}$ again.

With s_k determined, the target sending rate R_k can be obtained from (5) which enables the rate control block to compute the target frame size F_{c_k} from

$$F_{c_k} = \left\lceil \frac{R_k \cdot T}{P} \right\rceil \cdot P, \tag{6}$$

given the target sending rate R_k, the frame interval T, and the packet size P. Note that PIBES implements *forward erasure correction* (FEC) to recover from packet losses, so the target frame size F_{c_k} is inclusive of FEC overheads. FEC is chosen over retransmission as the latter incurs more delay which is undesirable in real-time applications. While PIBES

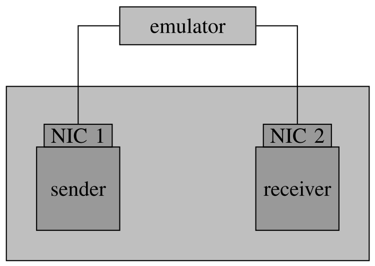


FIGURE 5. Experiment testbed.

also implements retransmission as a last resort, we found that with a modest amount of FEC (e.g., 10%) it is rarely needed at all.

IV. PERFORMANCE EVALUATION

In this section, we evaluate PIBES's performance and compare it to WebRTC and Salsify using the testbed depicted in Figure 5. Both sender and receiver were in the same machine so that they are clock-synchronized, which is required for measuring one-way frame delay. Note that this is purely for measurement purposes, PIBES itself does not need or rely on clock synchronization. The sender/receiver machine was a Dell OptiPlex 990 desktop computer running Linux Ubuntu 18.04. The bottleneck link is emulated using a modified version of Dummynet.[1]

As mentioned in Section III, the actual frame size produced by the video encoder may deviate slightly from the target frame size F_{c_k}. To simulate frame size variations, we added a random error to the frame size F_k as follows:

$$F_k = F_{c_k} \cdot (1 + \varepsilon_k), \tag{7}$$

where $\varepsilon \sim \mathcal{N}(0, 0.025)$.

The parameters for the FSM in Figure 4 are as follows: $s_{\max} = 0.95$, $\Delta = 0.05$, $s_{\text{share}} = 0.8$, $s_{\min} = 0.5$, $\Delta^+ = 0.01$ and $\Delta^- = 0.05$. For the experiments, we ran Salsify using the authors' source code.[2] WebRTC was executed through AppRTC. [3] GCC [21], [22] was employed as the congestion control algorithm in WebRTC.

A. BANDWIDTH ESTIMATION ACCURACY

We first evaluate the accuracy of PIBES' inband bandwidth estimation method. We conducted an experiment with bottleneck link bandwidth of 5Mbps to examine the choice of parameters and its sensitivity to competing traffic. There are around 15,000 bandwidth estimation samples, collected over 40 seconds, for each data point.

Two parameters, namely the size of the packet burst and the value of α for the exponential moving average calculation of the packet inter-arrival times, can affect estimation accuracy. PIBES' two-part transmission scheduler has a fixed number of packets at the beginning of each frame sent in a burst, followed by paced packets. Only packet inter-arrival times from packets in the initial burst are used for bandwidth estimation. Figure 6 compares the bandwidth estimation accuracy with respect to the burst size. We observe that the estimated bandwidth converged to the actual bandwidth with a burst size of three packets and stabilized beyond a burst size of six packets. Thus, we set $n_{\min} = 3$ to ensure sufficient bandwidth estimation accuracy - if the video frame comprises fewer than three packets then bandwidth estimation will be skipped. We set $n_{\max} = 6$ to improve estimation accuracy for larger video frames while keeping the burst size small to avoid congestion.

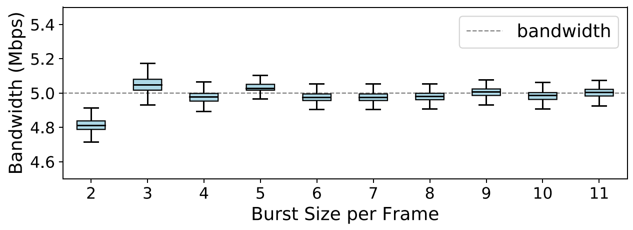


FIGURE 6. Impact of burst size on bandwidth estimation accuracy. α was set to 0.1.

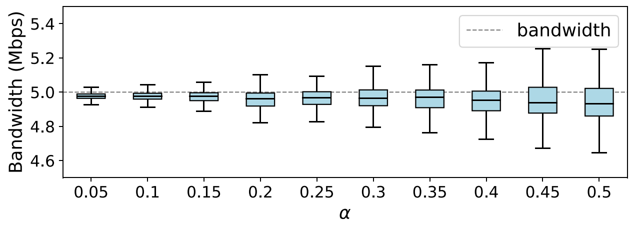


FIGURE 7. Impact of α on bandwidth estimation accuracy. Burst size was set to 6.

The impact of the smoothing factor α used in the calculation of the smoothed packet inter-arrival times is visualized in Figure 7. Here, one can see that the accuracy and precision of the estimation decrease as α increases. We adopted $\alpha = 0.1$ in the rest of the experiments.

Finally, we evaluate the impact of competing TCP traffic on bandwidth estimation accuracy in Figure 8, as well as assessing the flow detection accuracy in Figure 9. We ran two experiments, one with reverse TCP flows (Figure 8(a)) and another with forward competing TCP flows (Figure 8(b)). In both cases, an additional TCP flow entered the bottleneck link every 5 seconds. The results demonstrate that competing TCP flows in either direction have a negligible impact on bandwidth estimation accuracy. Figure 9 further shows PIBES's competing flow detection accuracy over time. We see that PIBES detected the competing flow shortly after its introduction and maintained reasonably consistent detection performance throughout the latter's presence. While there were a few isolated instances of incorrect detection, their impact is negligible.

B. LIVE VIDEO QUALITY

To evaluate the quality of the live video stream, we consider both throughput and video latency. Latency is evaluated on

1. https://github.com/mclab-cuhk/netmap-ipfw
2. https://github.com/excamera/alfalfa
3. https://appr.tc/

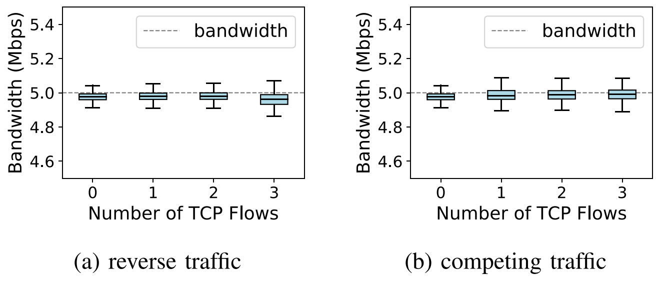


FIGURE 8. **Impact of competing traffic on bandwidth estimation accuracy.**

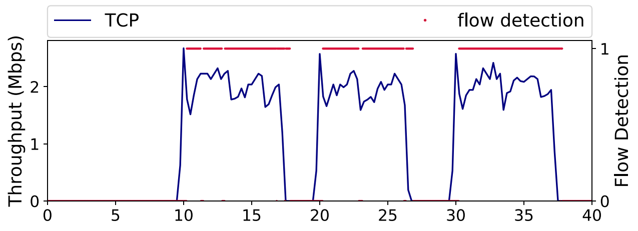


FIGURE 9. **Competing traffic detection performance over time.**

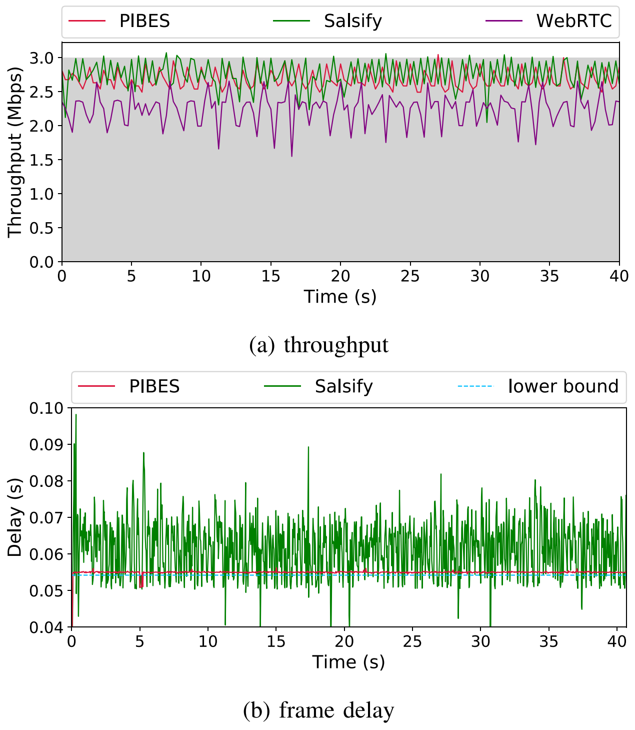


FIGURE 10. **Performance of PIBES, Salsify, and WebRTC over a fixed capacity link of 3Mbps and one-way delay of 0.02 seconds. The lower bound for frame delay is the minimum delay for transmitting video data at 95% of the link bandwidth, same as s_{max} configured in PIBES.**

a per-frame basis. The application on the receiver side can only display a frame once all packets have been received. Thus, for live video applications, delay matters per frame as opposed to per packet. We define frame delay as the time it takes for a complete video frame to be transported from the sender to the receiver, excluding video encoding and decoding times. The duration of each of the following experiments is 40 seconds, and the link's one-way delay is 0.02 seconds.

a) *Fixed bandwidth network:* We first compare the performance of PIBES, Salsify, and WebRTC over a fixed-bandwidth link of 3Mbps and no competing traffic in Figure 10. Surprisingly, WebRTC's link utilization is lower than expected. We could not determine the cause of this, but we conjecture that the implementation may have an internal cap on the maximum video bitrate. PIBES and Salsify, on the other hand, both achieved high link utilization.

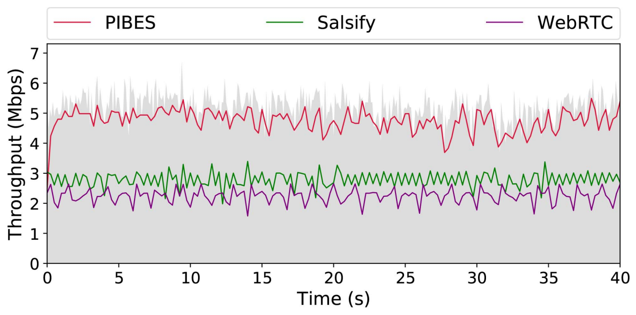
Next, we compare frame delay for PIBES and Salsify in Figure 10(b). We were not able to obtain frame delay measurements for WebRTC due to implementation limitations. Nevertheless, in a recent work, Fouladi *et al.* [6] compared WebRTC's frame delay to Salsify and found that WebRTC's 95th-percentile frame delay was 10.5 times higher than Salsify.

In Figure 10(b), we also show the minimum frame delay when sending at 95% of the link capacity. The results show that PIBES' average frame delay is slightly lower than Salsify and very close to the lower bound. Salsify's frame delay also exhibited more significant variations, presumably due to larger fluctuations in frame size.

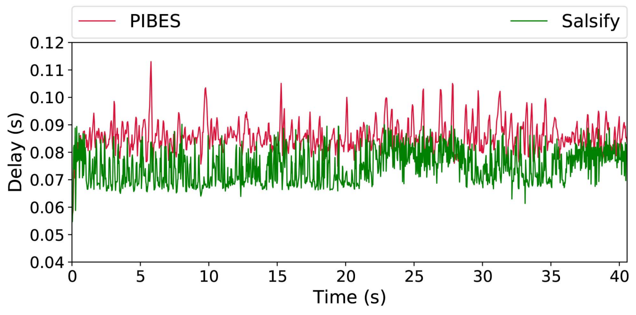
b) *Mobile networks:* Next, we evaluate the three protocols performance over mobile networks. We captured bandwidth trace data from production 3G and 4G networks and then feed the bandwidth trace data into our customized dummynet emulator to reproduce the same bandwidth variations in our experiments.

We first consider performance over 3G network in Figure 11(a) and Figure 11(b). The shaded region represents the emulated bandwidth at the bottleneck link. We observe that PIBES closely tracked and sent data along with the fluctuating bandwidth, maintaining a small gap from the link bandwidth due to the $s_{max} = 0.95$ setting. In comparison, both Salsify and WebRTC were not able to fully utilize the available bandwidth, presumably due to their internal bandwidth cap. In terms of frame delay (Figure 11(b)), Salsify exhibited slightly lower frame delay than PIBES due to its lower bandwidth utilization.

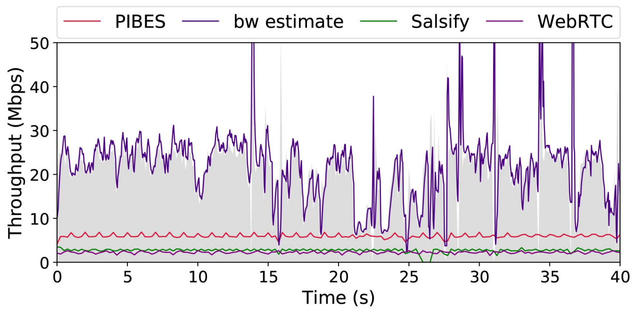
4G networks generally have much higher bandwidth. For example, under normal network conditions, the 4G network we measured offered mean bandwidth over 20Mbps. As such, there is abundant bandwidth even for live video streaming. Therefore, all three protocols hit their video bitrate cap under such high-bandwidth 4G networks (Figure 11(c)). On the other hand, we found that 4G network often exhibits larger bandwidth fluctuations, with occasional bandwidth drops to a very low bandwidth level. Such severe bandwidth drops will impact delay performance as depicted in Figure 11(d), resulting in occasional spikes in the frame delay. In one instance, around 26s to 27s in Figure 11(d), Salsify temporarily suspended video transmission altogether due to the bandwidth fluctuation. This result reveals a new challenge - despite



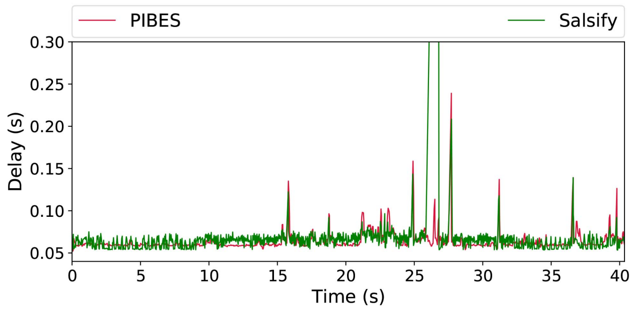
(a) throughput over 3G mobile network link



(b) frame delay over 3G mobile network link



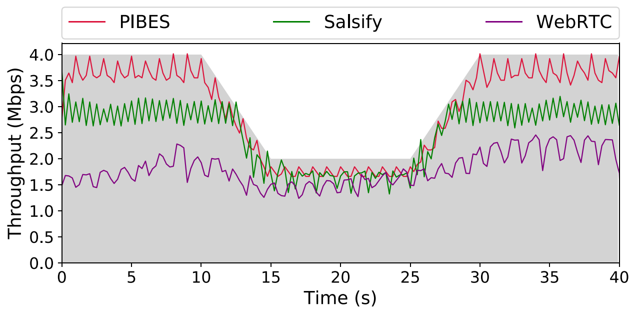
(c) throughput over 4G mobile network link



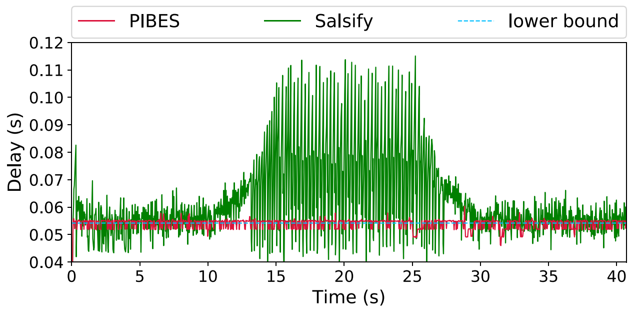
(d) frame delay over 4G mobile network link

FIGURE 11. **Performance of PIBES, Salsify, and WebRTC over 3G (Figures 11(a) and 11(b)) and 4G (Figures 11(c) and 11(d)) mobile networks. The link capacity is shaded grey in Figures 11(a) and 11(c). The one-way delay is 0.02 seconds.**

the high bandwidth, severe bandwidth fluctuations could hinder the achievable performance of live video streaming applications. This problem may very well require new cross-layer approach to tackle the challenge of sudden bandwidth drought, e.g., by dropping video data queued at the base station, which warrants further investigation.



(a) throughput



(b) frame delay

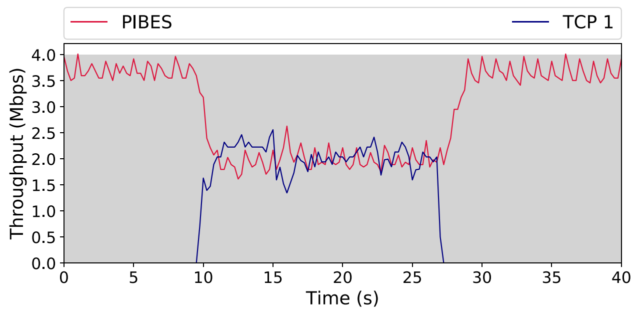
FIGURE 12. **Performance of PIBES, Salsify, and WebRTC on an emulated capacity ramp link. The link capacity is shaded grey in Figures 12(a). The one-way delay is 0.02 seconds. The lower bound for the delay of a frame is the minimum delay for a 95% sending rate. PIBES attempts to send at a 95% sending rate in our experiments.**

Due to the relatively low video bitrate caps observed in WebRTC and Salsify, we configure the testbed's link bandwidth to match their operating range in the rest of the experiments to allow for a fair comparison with PIBES.

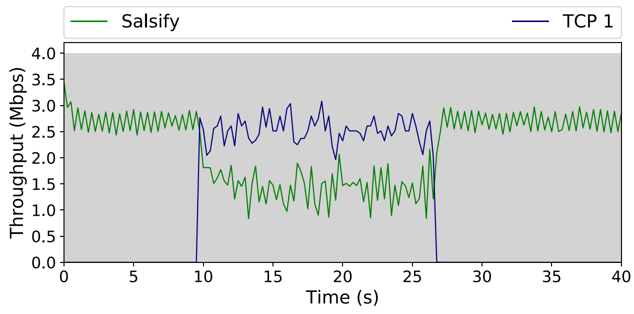
c) *Bandwidth ramps:* In this experiment, we explicitly ramped down and then up the link bandwidth between 2Mbps and 4Mbps to investigate the protocols' behavior in terms of throughput (Figure 12(a)) and frame delay (Figure 12(b)). In line with the previous experiment results, WebRTC exhibited the lowest throughput. It did adapt its sending rate in accordance with the changing bandwidth, but due to its internal bitrate cap, it was not able to fully utilize the available bandwidth.

Salsify achieved significantly higher throughput than WebRTC during the high-bandwidth regions, and it also quickly reacted to the bandwidth decrease from time 10 seconds to 15 seconds by adjusting its sending rate downwards accordingly. Moreover, it can also quickly ramp up its sending rate when link bandwidth was increased from 25 seconds to 30 seconds. In comparison, PIBES closely tracked the available bandwidth throughout the experiment and responded swiftly to the bandwidth ramps. In terms of frame delay in Figure 12(b), PIBES exhibited substantially more consistent frame delay than Salsify during the bandwidth trough from 15 seconds to 25 seconds.

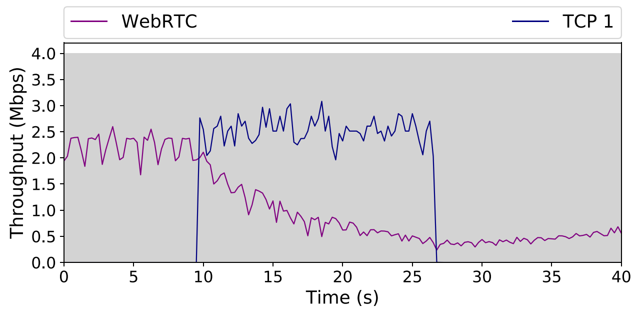
d) *Competing TCP traffic over a fixed capacity link:* In this experiment, we evaluate the protocols' behavior in the



(a) PIBES

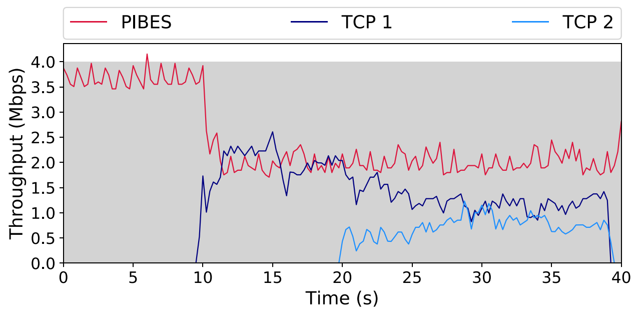


(b) Salsify

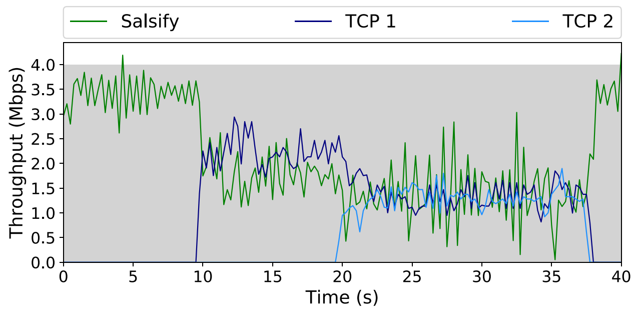


(c) WebRTC

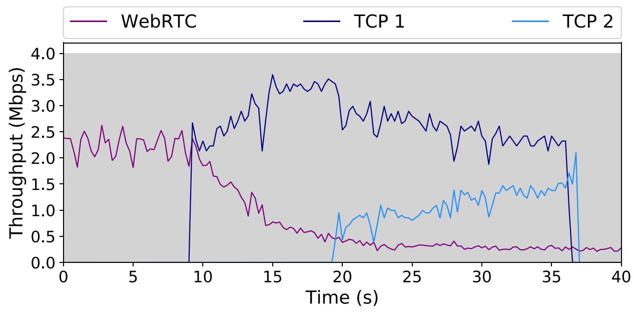
FIGURE 13. **Bandwidth-sharing with one TCP flow over a fixed capacity link. The link capacity is shaded grey.**



(a) PIBES



(b) Salsify



(c) WebRTC

FIGURE 14. **Bandwidth-sharing with two TCP flows over a fixed capacity link. The link capacity is shaded grey.**

presence of forward competing TCP traffic over a fixed-bandwidth bottleneck link. As shown in Figure 13, a TCP flow entered the network at 10 seconds and terminated at 27 seconds. The throughput plot shows that PIBES reacted quickly to the presence of the TCP flow by reducing its transmission rate towards the minimum bandwidth share setting of $s_{\min} = 0.5$.

Salsify also reacted quickly to the competing TCP flow. It lowered its transmission rate even further to around 1.5Mbps, leaving more bandwidth to the competing TCP flow (approximately 2.5Mbps). A key difference from PIBES is that Salsify does not control the amount of bandwidth to share with the competing flow. Thus, it may suffer from severe video quality degradation in lower link bandwidth or heavy competing traffic scenarios. Finally, WebRTC again exhibited some unexpected behavior. While it also reacted to the competing TCP flow, its transmission rate kept on decreasing over time and recovered very slowly even after the TCP flow was terminated. A slow recovery is undesirable as competing TCP flows are often short but may happen from time to time, leading to unnecessarily low video quality in the case of WebRTC.

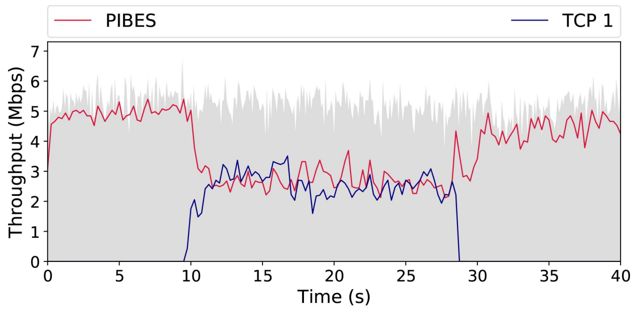
To further investigate the impact of competing flows, we performed another set of experiments with two TCP flows entering one after the other in 10 second intervals (Figure 14). For WebRTC, as shown in Figure 14(c), the throughput dropped further to below 0.5Mbps once the second TCP flow was started. By contrast, the two competing TCP flows grabbed the majority of the available bandwidth. This strongly suggests that WebRTC is overly conservative in the presence of competing traffic, which is highly detrimental for real-time video applications. Salsify performed significantly better than WebRTC by maintaining higher throughput when the second TCP flow was started. Nevertheless, the average throughput is still lowered by the second TCP flow, and it also exhibited much more significant fluctuations, presumably due to interaction with the competing TCP flows. Finally, PIBES maintained consistent throughput at around 2Mbps irrespective of whether there are one or two competing TCP flows. Moreover, its throughput variation was not impacted significantly by competing flows as in Salsify. Smaller throughput variations will result in more consistent video quality as the number of concurrent TCP flows is likely to vary rapidly in a shared network.

e) *Competing TCP traffic over a varying-bandwidth link:* We further experimented with competing TCP flow over 3G network. The results, depicted in Figure 15, are largely in line with those in Figure 13 and Figure 14. WebRTC slowly lost most of the link bandwidth to the TCP flow and recovered very slowly. In this experiment, Salsify also lost most of the bandwidth to the TCP flow. Compared to the case in Figure 13, Salsify was not able to hold onto a significant bandwidth proportion. By contrast, despite the bandwidth fluctuations, PIBES consistently shared the available bandwidth with the TCP flow according to the prescribed minimum bandwidth share setting.

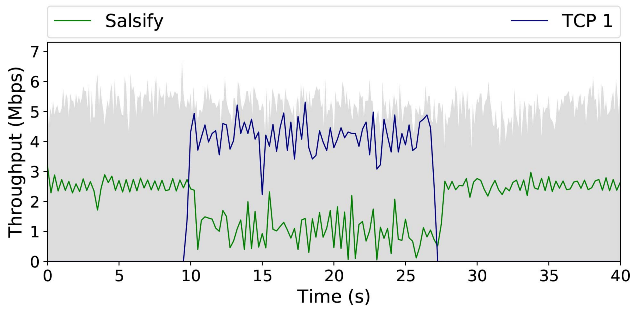
We observe similar behavior for WebRTC and Salsify under bandwidth ramps as shown in (Figure 16). Both protocols were dominated by the competing TCP flow, especially during the bandwidth trough from 15 seconds to 20 seconds. In contrast, PIBES closely tracked the minimum bandwidth share setting from 10 seconds to 25 seconds. When the bandwidth was ramped up from 25 seconds, PIBES was able to quickly utilize the increased bandwidth while the TCP flow kept at around the same throughput, showing that PIBES can explore and use additional bandwidth more quickly than TCP.

f) *Maintaining a minimum desired video quality:* PIBES was able to share the link bandwidth with competing traffics according to the prescribed minimum bandwidth share in the previous experiments. Conceivably, if the link bandwidth is too low, then even with the minimum bandwidth share, it may still result in throughput too low for acceptable video quality. For this reason, PIBES was designed with a configurable minimum video bitrate $R_{\min}$ in (5) to guarantee the minimum video quality. To illustrate this, we set $R_{\min}$ to 1.5Mbps and repeated the competing traffic experiment. As shown in Figure 17, PIBES maintained the minimum video datarate at 1.5Mbps even when the link bandwidth dropped to 2Mbps. Naturally, the share of bandwidth for the competing TCP flow was reduced, which is preferable in this case as TCP is designed for transporting elastic traffic.

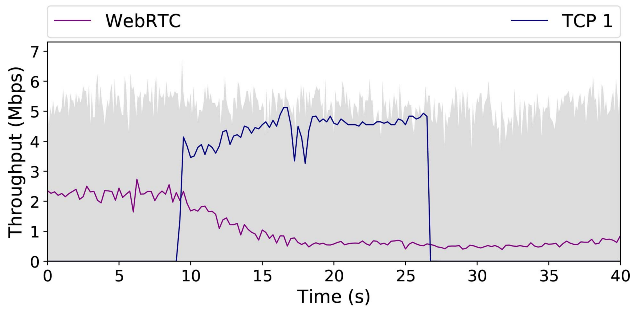
g) *Competing PIBES traffic:* So far, the experiments were run with one PIBES flow at a time. While this is likely to be the case under most circumstances, it is also possible for more than one PIBES users to share the same bottleneck link, e.g., two users attending different online classes simultaneously at home. Clearly, with a minimum bandwidth share setting of $s_{\min} = 0.5$, two PIBES flows will likely congest the bottleneck link.



(a) PIBES



(b) Salsify



(c) WebRTC

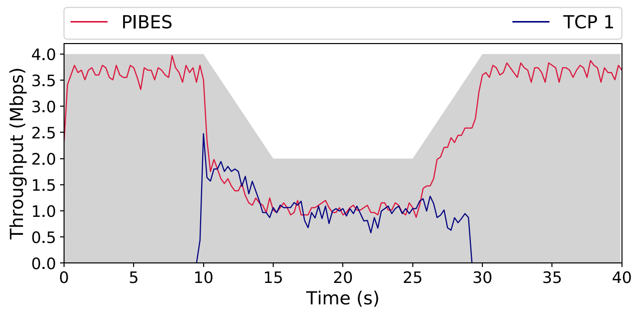
FIGURE 15. **Bandwidth-sharing over a 3G mobile network. The link capacity is shaded grey in (b).**

One solution is to adjust $s_{\min}$ according to the number of PIBES flows sharing the bottleneck. For example, we set $s_{\min} = s_{\max}/N$, where N is the number of concurrent PIBES flows sharing the bottleneck and conducted an experiment with two PIBES flows in Figure 18. We observe that the two PIBES flows shared the available bandwidth reasonably fairly as expected.

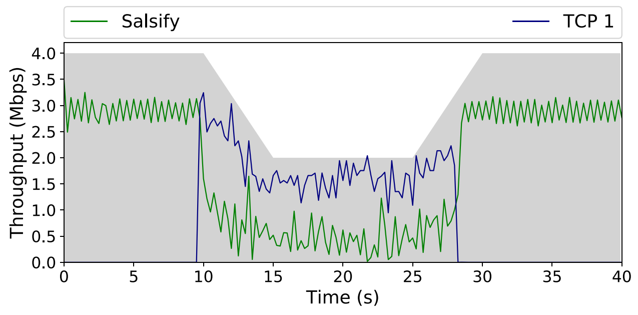
The above solution assumes that the number of PIBES flows sharing a bottleneck link is known. This could be implemented by the service provider, e.g., by comparing the public IP addresses of the two clients or via the clients' own discovery of one another using standard protocols such as UPnP. The design of such mechanisms and the further refinement of adaptation of the $s_{\min}$ setting are subjects that warrant further research.

V. SUMMARY AND FUTURE WORK

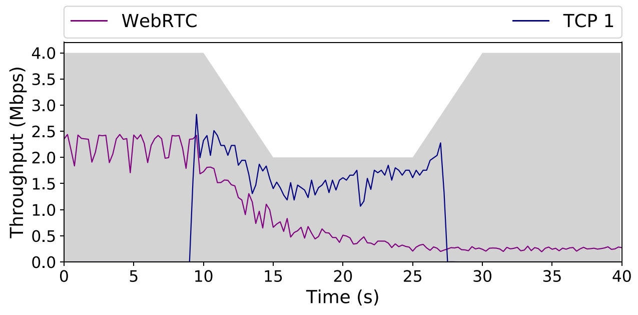
We developed PIBES in this work as a transport protocol for real-time video communications. In the absence of competing



(a) PIBES



(b) Salsify



(c) WebRTC

FIGURE 16. **Bandwidth-sharing on a varying bandwidth link. The link capacity is shaded grey in (b).**

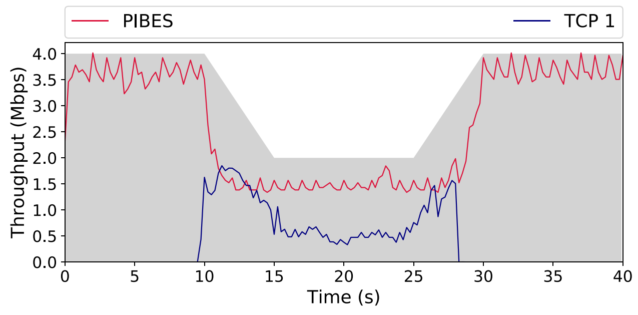


FIGURE 17. **Maintaining minimum acceptable video quality (set at 1.5Mbps) when competing with a TCP flow. The link capacity is shaded grey.**

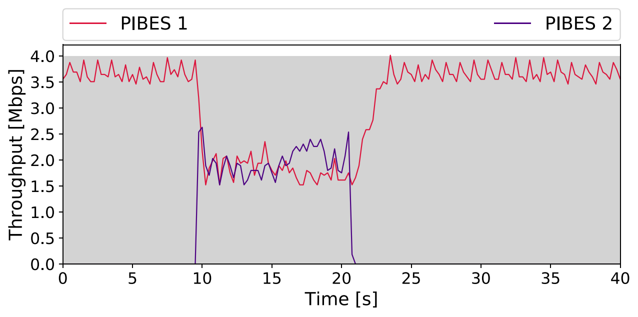


FIGURE 18. **Bandwidth-sharing between two PIBES flows over a fixed capacity link. The link capacity is shaded grey.**

traffic, PIBES performs equal to or better than existing protocols such as WebRTC and Salsify. Once competing TCP flows are introduced, PIBES performs substantially better in terms of throughput as well as delay variations. More importantly, PIBES offers the user/application comprehensive control of the video flow in terms of target bandwidth share, minimum bandwidth share, and minimum video datarate. This opens up new ways for applications to fine-tune its video flow to match its desired quality of experience.

This study is a first step in understanding the significant impact of competing traffics on live video streaming services. There remain many open problems which warrant further investigation, e.g., the impact of different TCP variants, the impact of non-TCP competing traffics, the challenge of sudden and deep bandwidth troughs in high-bandwidth mobile networks, the challenge in streaming ultra-high-definition live video (4K), and the design of more sophisticated bandwidth sharing schemes for concurrent live video flows.

Last but not least, although PIBES was initially designed for real-time video communications, its key components, including the bandwidth estimator, the competing flow detector, and the bitrate adaptation algorithm, could potentially be applied to other types of real-time or semi-real-time services to improve their performance in the presence of competing traffic. Further research is warranted to explore the potential applications.

ACKNOWLEDGMENT

The authors thank the Associate Editor and the anonymous reviewers for their insightful comments and suggestions in improving this article to its final form.

REFERENCES

[1] "Cisco visual networking index: Forecast and trends, 2017–2022," Cisco, San Jose, CA, USA, White Paper, 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html
[2] "TCP optimization: Opportunities, KPIs, and considerations—An industry whitepaper," Sandvine, Plano, TX, USA, White Paper, 2016. [Online]. Available: https://www.sandvine.com/hubfs/downloads/archive/whitepaper-tcp-optimization-opportunities-kpis-and-considerations.pdf
[3] B. Briscoe *et al.*, "Reducing Internet latency: A survey of techniques and their merits," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2149–2196, 3rd Quart., 2016.
[4] Y. Xu, C. Yu, J. Li, and Y. Liu, "Video telephony for end-consumers: Measurement study of Google+, iChat, and Skype," in *Proc. Internet Meas. Conf.*, 2012, pp. 371–384.

[5] H. Alvestrand, "Overview: Real time protocols for browser-based applications," Internet Eng. Task Force, Internet-Draft, 2018. [Online]. Available: https://tools.ietf.org/id/draft-ietf-rtcweb-overview-19.html

[6] 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 *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 267–282.

[7] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, vol. 4. Hong Kong, China, 2004, pp. 2514–2524.

[8] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, 2003.

[9] S. Ha, I. Rhee, and L. Xu, "Cubic: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[10] L. S. Brakmo and L. L. Peterson, "TCP vegas: End to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[11] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, vol. 4. Hong Kong, China, 2004, pp. 2490–2501.

[12] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP reno and vegas," in *Proc. IEEE INFOCOM Conf. Comput. Commun. 18th Annu. Joint Conf. Comput. Commun. Soc.*, vol. 3. New York, NY, USA, 1999, pp. 1556–1563.

[13] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of westwood+, new reno, and vegas TCP congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 25–38, 2004.

[14] S. Mascolo and F. Vacirca, "The effect of reverse traffic on the performance of new TCP congestion control algorithms," *presented at the Int. Workshop Protocols Fast Long Distance Netw. (PFLDnet)*, 2006.

[15] C. Parsa and J. Garcia-Luna-Aceves, "Improving TCP congestion control over Internets with heterogeneous transmission media," in *Proc. IEEE 7th Int. Conf. Netw. Protocols*, Toronto, ON, Canada, 1999, pp. 213–221.

[16] Ł. Budzisz, R. Stanojević, A. Schlote, F. Baker, and R. Shorten, "On the fair coexistence of loss- and delay-based TCP," *IEEE/ACM Trans. Netw. (TON)*, vol. 19, no. 6, pp. 1811–1824, Dec. 2011.

[17] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (LEDBAT)," Internet Eng. Task Force, RFC 6817, 2012.

[18] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for LEDBAT fairness," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Miami, FL, USA, 2010, pp. 1–6.

[19] I. Johansson, "Self-clocked rate adaptation for conversational video in LTE," in *Proc. ACM SIGCOMM Workshop Capacity Sharing Workshop*, 2014, pp. 51–56.

[20] X. Zhu and R. Pan, "NADA: A unified congestion control scheme for low-latency interactive video," in *Proc. IEEE 20th Int. Packet Video Workshop*, San Jose, CA, USA, 2013, pp. 1–8.

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

[22] 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 *Proc. 7th Int. Conf. Multimedia Syst.*, 2016, p. 13.

[23] S. Zhang, W. Lei, W. Zhang, and Y. Guan, "Congestion control for RTP media: A comparison on simulated environment," in *Proc. Int. Conf. Simulat. Tools Techn.*, 2019, pp. 43–52.

[24] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. 7th Annu. Int. Conf. Mobile Comput. Netw.*, 2001, pp. 287–297.

[25] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[26] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in *Proc. IEEE IFIP Netw. Conf. (IFIP Networking) Workshops*, Zürich, Switzerland, 2018, pp. 1–9.

[27] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Toronto, ON, Canada, 2017, pp. 1–10.

[28] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement.*, 2013, pp. 459–472.

[29] N. Hermanns and Z. Sarker, *Congestion Control Issues in Real-Time Communication 'Sprout' as an Example*, Internet Congestion Control RG, Fremont, CA, USA, 2013. [Online]. Available: https://datatracker.ietf.org/meeting/88/materials/slides-88-iccrg-3.pdf

[30] A. Baiocchi, A. P. Castellani, and F. Vacirc, "Yeah-TCP: Yet another highspeed TCP," in *Proc. Int. Workshop Protocols Future Large Scale Diverse Netw. Transp. (PFLDnet)*, vol. 7, 2007, pp. 37–42.

[31] S. Cho and R. Bettati, "Adaptive aggregated aggressiveness control on parallel TCP flows using competition detection," in *Proc. 15th Int. Conf. Comput. Commun. Netw.*, Arlington, VA, USA, 2006, pp. 237–244.

[32] S. Cho and R. Bettati, "Gen05-5: Use of competition detection in TCP for fair and effective utilization of network bandwidth," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, San Francisco, CA, USA, 2006, pp. 1–6.

LIOBA HEIMBACH received the B.Sc. degree in electrical engineering and information technology from ETH Zurich, Switzerland, in 2020, where she is currently pursuing the M.Sc. degree in electrical engineering and information technology. She has worked on Internet protocols and financial technologies.

LINGFENG GUO received the B.Eng. degree in software engineering from Sun Yat-sen University, Guangzhou, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Information Engineering, Chinese University of Hong Kong, where he participated in the research and development of Internet protocols.

RUDOLF K. H. NGAN received the B.Sc. degree in physics from the Chinese University of Hong Kong in 1995 and the M.Sc. degree in computer science from the City University of Hong Kong in 2016. He has been working as a Research Assistant with the Department of Information Engineering, Chinese University of Hong Kong since 2000.

JACK Y. B. LEE (Senior Member, IEEE) received the B.Eng. and Ph.D. degrees in information engineering from the Chinese University of Hong Kong, Hong Kong, in 1993 and 1997, respectively, where he is with the Department of Information Engineering. His research group focuses on research in multimedia communications systems, mobile communications, protocols, and applications. He specializes in tackling research challenges arising from real-world systems. He works closely with the industry to uncover new research challenges and opportunities for new services and applications. Several of the systems research from his lab have been adopted and deployed by the industry.