A Measurement-Derived Functional Model for the Interaction between Congestion Control and QoE in Video Conferencing

Jia He, Mostafa Ammar, and Ellen Zegura

Georgia Institute of Technology, Atlanta, GA, USA
{jhe332@,ammar@cc.,ewz@cc.}gatech.edu

Abstract. Video Conferencing Applications (VCAs) that support remote work and education have increased in use over the last two years, contributing to Internet bandwidth usage. VCA clients transmit video and audio to each other in peer-to-peer mode or through a bridge known as a Selective Forwarding Unit (SFU). Popular VCAs implement congestion control in the application layer over UDP and accomplish rate adjustment through video rate control, ultimately affecting end user Quality of Experience (QoE). Researchers have reported on the throughput and video metric performance of specific VCAs using structured experiments. Yet prior work rarely examines the interaction between congestion control mechanisms and rate adjustment techniques that produces the observed throughput and QoE metrics. Understanding this interaction at a functional level paves the way to explain observed performance, to pinpoint commonalities and key functional differences across VCAs, and to contemplate opportunities for innovation. To that end, we first design and conduct detailed measurements of three VCAs (WebRTC/Jitsi, Zoom, BlueJeans) to develop understanding of their congestion and video rate control mechanisms. We then use the measurement results to derive our functional models for the VCA client and SFU. Our models reveal the complexity of these systems and demonstrate how, despite some uniformity in function deployment, there is significant variability among the VCAs in the implementation of these functions.

1 Introduction

Video Conferencing Applications (VCAs) represent an increasingly significant amount of Internet application and bandwidth usage [35]. Schools and businesses have turned to video conferencing in support of safe COVID practices and also for efficiency and convenience. In VCAs, a signaling server first allows clients to coordinate their activities and perform session management. Once a session is established the clients transmit video and audio to each other, either directly in peer-to-peer mode or indirectly through a video bridge. The dominant type of commercial video bridge is a Selective Forwarding Unit (SFU), which forwards videos from each client to other clients without decoding [44]. SFUs can manipulate videos prior to forwarding using techniques such as subsampling and layer

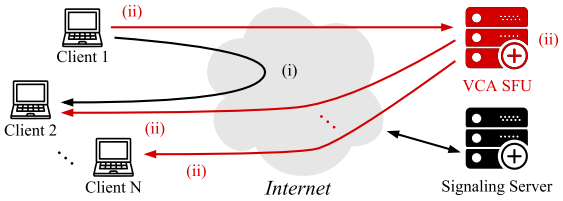


Fig. 1: High-level architecture of a video conference with N clients. Video traffic originating at Client 1 shown: (i) peer-to-peer connection between two clients (ii) in SFU mode, the SFU will forward the video from Client 1 to all other clients.

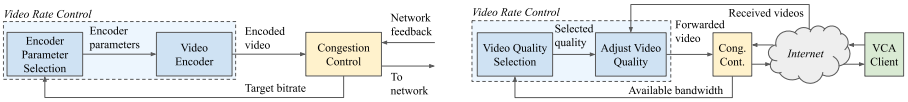
selection which can be applied to the encoded video. A high-level overview of a video conferencing session is given in Figure 1.

VCAs can be bandwidth intensive because of their extensive use of video. Commercial systems carry the video on top of UDP, with congestion control implemented in the application. Recognizing the important role of congestion control, there is a long history of effort dedicated to building application layer congestion control on top of UDP (e.g., [4,11,15]) with more recent efforts geared towards specific use in VCAs [13]. Congestion control mechanisms present in commercial VCAs are typically not shared publicly, though some open-source, production-quality VCAs do exist [3,25].

The main goal of congestion control is to determine an application sending rate that does not congest the shared network path used by the application. After congestion control functions determine an acceptable target sending rate, the application must enforce this target rate through *video rate control*. The application can adjust the rate of the video being sent using video encoding at the sending client or through layer selection and/or subsampling at the SFU[1]. Congestion control in VCAs, therefore, has a direct impact on video quality and, in turn, the VCA user's quality of experience (QoE). Understanding the user-perceived QoE is important for network operators as it helps with efficient bandwidth provisioning, though in this paper we show that congestion control and video rate control can differ between VCAs, potentially complicating the matter of estimating QoE. Researchers have reported on the throughput and video metric performance of specific VCAs using structured experiments. Yet prior work rarely examines the interaction between congestion control mechanisms and rate adjustment techniques that produces the observed throughput and QoE metrics. Understanding this interaction at a functional level paves the way to explain observed performance, to pinpoint commonalities and key functional differences across different VCAs, and to contemplate opportunities for innovation. This is the aim of this paper.

Our starting point is the high-level interaction shown in Figure 2 between congestion control and video rate control in both the client and in the SFU. Figure 2(a) illustrates that the target rate determined by the congestion control

[1] In principle, the total client sending rate includes video, audio, and control data (e.g., RTCP packets). In this paper, we focus only on video data as it is the most bandwidth intensive.



(a) Client (b) SFU

Fig. 2: Interaction between congestion control and video rate control.

function at a VCA client is used to determine video encoding parameters. These parameters are then fed into the video encoder which outputs video at or below the desired target rate. Figure 2(b) shows the interaction between congestion control and video rate control within an SFU. The SFU receives video from all clients and forwards them to other clients according to their available download bandwidth. As coarse-grained rate control, all SFUs we examine can select which videos to forward to each client [19]. After this selection, the SFU uses additional mechanisms to manage the forwarded video rate without decoding.

This paper aims to deepen understanding of this interaction between congestion control, video rate control, and end user video quality in VCAs beyond the high-level structure shown in Figure 2. The interaction between these three is complicated, and the proprietary nature of commercial VCA systems restricts visibility into key details. We know that WebRTC [13] and the open source Jitsi [25,26] use Google Congestion Control (GCC). Prior work [9,28,30] has ascertained the presence of congestion control in VCA clients and was able to fit their operation within the general structure of the GCC mechanisms [8]. We add to this literature in two ways. First, we conduct a measurement study, outlined in Section 3, of several VCAs (WebRTC/Jitsi, Zoom, and BlueJeans) to fully understand the different congestion control methods (Section 4) and video rate control methods (Section 5). The measurements are extended to include multiple clients and device types in Section 6.

Altogether, the results obtained from our measurement study allow us to develop functional models of the VCA client and VCA SFU, which is one of the primary contributions of this paper. The results allow us to make several observations key to the functional models including, among other things, the different congestion control algorithms employed by the VCAs, such as how BlueJeans is unresponsive to changes in latency but highly sensitive to packet loss, while Zoom is insensitive to packet loss and does respond to latency. Similarly, the VCAs prioritize different video quality metrics at highly constrained bandwidth. For example, Zoom commits to a maximum quantization parameter (QP), while WebRTC/Jitsi prioritize maximizing the frame rate.

The differences among VCAs observed from our measurements highlight the need for a generalized functional model. To that end, in Section 7, we derive the generalized functional models of the VCA client and VCA SFU from the measurement results, along with data from publicly available documents and source code. Our models reveal additional details and complexity in these systems and demonstrate how, despite some uniformity in function deployment, there is significant variability among the VCAs in the implementation of these functions. We believe our more detailed models can better serve the research community as we continue to investigate the design and performance of VCAs.

1.1 Ethical Considerations

The experiments presented in this work were highly controlled and did not involve any real users. There was no personal or private information sent or received as part of any experiment. This work raises no ethical concerns.

2 Related Work

The surge in video conferencing use in recent years allow us to split prior work into two groups; those from before this surge [1,3,16,24,42,43], and those from after it [9,27,28,30,34]. In general, previous studies cover VCAs that were popular at the time of publication. For example, Xu et al. [42] and Yu et al. [43] from 2012 and 2014 cover Skype and FaceTime, whereas more recent work studies VCAs like Zoom [9,27,28,30,34], Microsoft Teams [27,30], and WebEx [9,28]. "Legacy" VCAs, such as Skype, are typically not subject to examination in more recent work, reflecting their decline in use in favor of newer products such as Zoom and Microsoft Teams. Notably, interest in WebRTC remains consistent in both groups of prior work [1,3,24,28,30].

VCA measurements typically involve subjecting various VCAs to different network conditions and recording the results. These results are often presented as measurements of throughput and video metrics, typically the video resolution and frame rate. A recent study aimed to infer models for congestion control [28] but without exploring the fully functional dependence between congestion control and video QoE. In typical studies, measurements are often taken in a highly controlled laboratory environment, though Fund et al. [16] perform their measurement campaign in two different outdoors settings, one urban and one suburban/rural, with the user devices connected to WiMAX base stations in the vicinity. Varvello et al. [39] evaluate the performance of Zoom, Webex and Google Meet using clients distributed around the world.

The VCAs studied and measurements taken in this paper result are most directly related to work by MacMillan et al. [30]. In that work, the authors study Zoom and WebRTC-based VCAs and consider measurements of the video metrics without considering the underlying architectures. However, in our work we focus on measurement for the purpose of building a generalized understanding of critical parts of the VCA, rather than to measure performance of specific VCAs. Sander et al. [34] focus on improving Zoom's performance in the presence of competing flows by implementing different queuing policies at the bottleneck, and discuss Zoom's insensitivity towards packet loss and queuing delay. We observe similar results for packet loss, but show that Zoom will respond to network latency above a certain threshold. This paper focuses on developing an understanding of the congestion control to explain the competition behaviors, with some examples in Appendix A. Furthermore, while we consider multiple VCAs, direct comparisons of the benefits or drawbacks of each [27] is outside of the scope of this work. To achieve the desired understanding of VCA behavior, we consider the time-varying effects of bandwidth limitation along with packet loss and latency on the sending rate and video metrics. The video metrics we consider

include the quantization parameter (QP) used by the Zoom encoder, which to the best of our knowledge has not been previously studied.

In summary, we believe this paper to be the first to present a generalized functional structure of VCAs, using measurements designed to determine the specifics of the interaction of the congestion control and video rate control, the two key components governing overall VCA QoE. We anticipate that the models presented in this work will aid research into VCAs and end user QoE by explaining behavior seen but not elaborated on in prior work.

3 Measurement Design

In this section we describe the measurement design. We first provide a breakdown of the possible test conditions and describe those relevant to our goal of building the generalized models. Second, we describe the experimental testbed that will enable measurement under the defined set of test conditions.

3.1 Test conditions

The space of possible VCA measurements is multidimensional. We consider the following dimensions: (i) choice of VCA, (ii) type of video, (iii) network conditions, (iv) type of data collected, (v) number of clients, (vi) background traffic, and (vii) device type. A summary of these dimensions and selected values is given in Table 1, and further details are provided in Sections 4.1 and 5.1.

We perform measurements with Zoom, BlueJeans, Jitsi, and a custom, basic WebRTC program. Zoom and BlueJeans are commonly used closed-source production VCAs. Jitsi and the custom application provide production and baseline examples, respectively, of systems built around the open-source WebRTC. Jitsi itself is also open-source. Each VCA uses one of two encoders (H264 or VP8), and provides different stream subsampling methods (described in Appendix B) for the SFU, as well as different congestion control and video rate control methods. We send a"talking head" video through the VCA during measurements, which represents a typical video from a user webcam.

We apply different levels of adverse bandwidth limitation, packet loss, and latency to provoke a response from the VCA congestion control or video rate control. These network conditions could be experienced by users in under-provisioned locations, such as rural areas [14], or in highly loaded access networks. Even in well-served areas, cellular providers are known to shape video traffic via bandwidth limitation to reduce congestion at the radio edge [12]. To study the response of the congestion or video rate control, we collect packet traces and video metrics available in the VCA statistics or in conference recordings.

Two laptop clients are used for the majority of measurements, as this is sufficient to understand the congestion control and video rate control on the client side. To fully understand the SFU, we take measurements with additional clients to examine how the SFU works when multiple videos are to be forwarded to a single client. The background traffic on each client is kept to a minimum to

Dimension	Selected Values
VCA	Zoom, BlueJeans, WebRTC/Jitsi
Video Type	Low-motion "talking head"
Network Conditions	Adverse conditions for bandwidth, latency, and loss
Collected Data	Packet traces, in-app video metrics
Number of Clients	2 for most measurement, 4–6 for others
Background Traffic	Minimize on the measurement devices
Device Type	Laptop, phone, tablet

Table 1: Summary of the considered test conditions and the values selected for each.

avoid competition with other flows. Measurements with competing TCP flows are presented in Appendix A, but they do not help directly inform the functional models. Lastly, as it is known that device type can impact video conferencing performance and QoE [9,39,40], we use phone and tablet clients alongside the laptops in SFU mode to understand their effect.

3.2 Testbed

The testbed used for measurements reflects the high-level architecture in Figure 1 and must support two clients, with support for more in select experiments. Furthermore, the testbed supports video conferencing in both peer-to-peer and SFU-mediated modes, as the two are important for fully understanding the VCA client and SFU, respectively. For measurements taken in peer-to-peer mode, only Clients 1 and 2 and the Signalling Server, typically located within the network owned by the VCA developer [31], are involved. The signalling server is only used to establish the peer-to-peer video call and is uninvolved after this. The SFU measurements make use of Clients 1 and 2 as well as the VCA SFU for forwarding videos. Some measurements will also include additional clients.

While we desire minimal background traffic on the devices themselves, this is not necessarily a realistic environment in practice. Therefore, to provide a semi-realistic environment that we can model the VCAs within, the measurements were taken with the two laptops connected to an uncontrolled WiFi access point (AP). This access point was found to deliver a minimum of 10 Mbps upload and download to each device at all times, which is sufficient for all of the considered VCAs to send and receive video at maximum bitrate. Furthermore, packet loss at the AP is minimal, and the latency is stable even during peak periods, such as evenings and weekends. These properties ensure that all effects under the various applied network conditions can be reliably observed, which is further guaranteed using repeated measurements.

We are not able to control the traffic condition at the Zoom or BlueJeans SFU, though we assume in our measurements that the SFU experiences minimal congestion. We believe this to be a fair assumption, given the large cloud-based networks that Zoom and BlueJeans control. We find that the clients typically connect to Zoom SFUs located in or around New York City, as reported by the Zoom application. The precise location of the BlueJeans SFU could not be ascertained, though we suspect it is the eastern USA from traceroutes to the

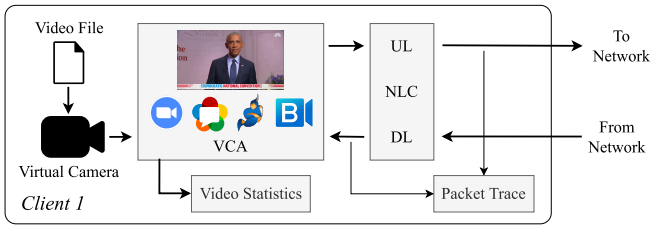


Fig. 3: Architecture of Client 1. Client 2 only uses the VCA and virtual camera input.

SFU IP address. The Jitsi SFU is set up on a Google Cloud VM located in the `us-east1-b` datacenter region in South Carolina. As the Jitsi SFU runs on a VM which we control, we can ensure background traffic is kept to a minimum.

Both Clients 1 and 2 run Mac OS with native Zoom and BlueJeans applications (version 5.10.4 and 2.35.0, respectively), and Chrome version 100, which is used for Jitsi and WebRTC. We note that VCAs are subject to frequent software updates, which may bring slightly different behavior to the VCAs over time. Given the relatively short time period over which measurements were taken, we do not anticipate such effects to impact the accuracy of the measurements or derived models. Different operating systems can also influence the behavior of native VCA apps [30,39]. In addition, we observed over the course of our measurements that the administrator-controlled settings for the enterprise VCAs, particularly Zoom, could impact how the VCAs can be used. For example, over Summer 2022, we noticed that peer-to-peer mode calls could not be established on the Zoom accounts provided by our institution, but accounts from a different institution would permit peer-to-peer calls under the same conditions.

All measurements are taken on Client 1, which has additional software running as shown in Figure 3 to support automated measurement and data processing. We apply the network conditions on Client 1; we use Network Link Conditioner (NLC), available as part of the Xcode developer tools on Mac OS. NLC allows the bandwidth, packet loss rate, and latency to be changed on upload and download independently. Both clients use a virtual camera to display a talking head video, a still from which is shown in Figure 3. This video was chosen for its typicality and realistic resolution of 1280x720. The VCAs are operated in full-screen mode during the two-client measurements. The measurement process was fully automated using the Selenium library [32] and the Chrome WebDriver [10] for WebRTC/Jitsi, and using the PyAutoGUI library [38] for Zoom and BlueJeans. The automation code is made open source on GitHub [20].

Client 1 performs two types of data logging. The first is collecting packet traces using Tshark. We note that packets are collected after NLC operates in the uplink (UL) and the downlink (DL); this operating principle is important for putting the results in later sections into context. Secondly, video statistics data is logged. This logging varies depending on the VCA; for WebRTC and Jitsi, the data is collected from the Chrome instance running the VCA using the developer tools at `chrome://webrtc-internals`. Zoom and BlueJeans provide statistics within the application itself; these are extracted via recording the screen area

containing the statistics and processing this screen recording with the Tesseract text recognition software [17], with appropriate error-checking and correction to ensure accurate reconstruction of results. Zoom provides statistics on the frame rate, resolution, and bitrate; BlueJeans only provides the resolution.

Zoom's built-in local recording feature produces videos that are high-fidelity representations of the exact video that was sent or received, and thus can be used to analyze the quantization parameter (QP) as well. The recording is processed using an FFMpeg-based tool [33] to extract the QP for each frame; the tool also outputs whether the frame is an I-frame or a P-frame. While BlueJeans supports cloud-based recordings, they are post-processed to add black borders when the video degrades from maximum resolution, and thus it is impossible to determine if the QP is faithful to the encoder's configuration.

4 Congestion Control Study

We begin developing the functional models by examining the VCA congestion control algorithms. In this section we present the results of measurements taken with Zoom and WebRTC operating in peer-to-peer mode, and then the results for Zoom, Jitsi, and BlueJeans operating in SFU mode.

4.1 Measurements

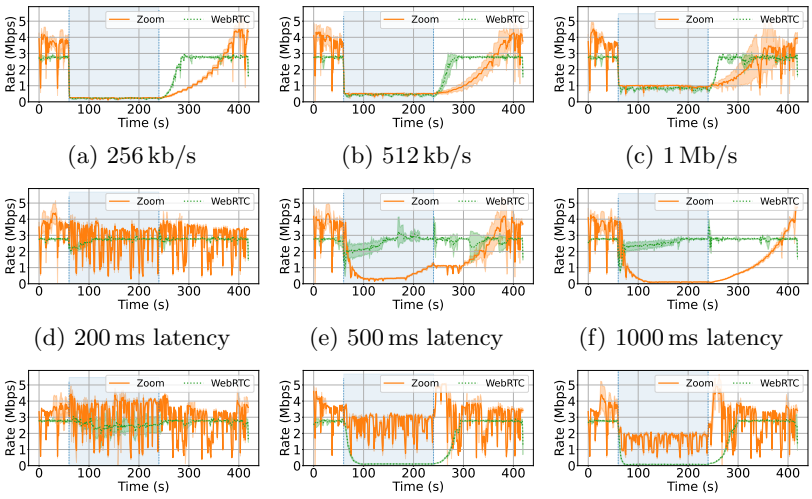
We measure congestion control by applying a network condition to an otherwise stable network environment, and then removing it. This gives insight into two critical parts of the congestion control: what network conditions elicit a response, and how the sending rate increases once the network condition is removed.

The three considered network conditions are the available bandwidth, packet loss rate, and latency. For each condition, we specify four values, for a total of 12 experiments. During each experiment, the VCA is started and left to stabilize under no applied condition for two minutes, after which the measurement begins. During these two minutes, the VCA will achieve the maximum sending rate.

After $t = 60$ seconds from the start of measurement, the network condition is applied, after which it is removed at $t = 240$ seconds. The measurement continues for three more minutes until $t = 420$ seconds. This gives seven-minute measurements, which are repeated five times for each network condition. When studying the VCA client's congestion control, the network conditions are applied to Client 1's upload; to study the SFU congestion control we apply the network condition to Client 1's download. Using the five repeats, we are able to compute an average and standard deviation of the VCA sending rate over time. Thus, each graph with results represents 35 minutes of measurement for each VCA.

4.2 Peer-to-Peer Mode Congestion Control

Figure 4 shows throughput plots for Zoom and WebRTC under the bandwidth, latency, and packet loss conditions. To conserve space, nine out of the 12 network conditions are shown.



(a) 256 kb/s (b) 512 kb/s (c) 1 Mb/s

(d) 200 ms latency (e) 500 ms latency (f) 1000 ms latency

(g) Loss rate 0.05 (h) Loss rate 0.2 (i) Loss rate 0.5

Fig. 4: Response of Zoom and WebRTC peer-to-peer clients to different network conditions. (a)-(c) upload bandwidth limits, (d)-(f) additional upload latency, (g)-(i) upload loss rates.

We note that there are differences in the sending rate for each VCA even in the absence of induced network conditions. Zoom's sending rate varies considerably over time with peaks between 3.5 and 4 Mb/s, while WebRTC has a much more stable sending rate that rarely exceeds 2.8 Mb/s.

Bandwidth. Removal of the 512 kb/s bandwidth limit in Figure 4(b) shows that Zoom takes over 160 seconds to return to its original sending rate of around 3.7 Mb/s; WebRTC returns to its original peak sending rate in only around 40 seconds. Zoom also exhibits a step-like sending rate increase.

Latency. Unlike WebRTC's GCC which responds to delay variation, Zoom responds to the actual delay value, and only does so above a certain threshold, at most 500 ms. The response is severe, with the sending rate dropping to less than 300 kb/s when 500 ms of latency is added.

Packet Loss. While WebRTC's GCC uses a loss rate threshold of 10% before reducing the sending rate, we find a small response at 5% loss rate. This may be due to NLC's probabilistic packet drop mechanism causing packet loss rates above 10% at some points. Figures 4(g) and 4(h) show how GCC's sending rate decreases further when the packet loss rate is higher.

We note that the observed drop in Zoom's sending rate is equal to the loss rate. Given that the packet trace is recorded after NLC as described in Section 3.2 and shown in Figure 3, we are observing only the packet loss enforced by NLC, rather than Zoom lowering its sending rate. The lack of the step-like increase upon removal of the packet loss is further evidence that Zoom's congestion control algorithm is insensitive to packet loss, even at a loss rate of 50%.

The ability of Zoom to function in the presence of such extreme loss rates may be due to forward error correction (FEC) data that is included with the video stream [29]. Observation of the received video shows intermittent freezing, but the picture quality and resolution remain high.

Overall, the measurements taken for WebRTC match well with the GCC algorithm as described in the literature. We learn that Zoom in peer-to-peer mode exhibits three notable behaviors: (i) a slow, step-like sending rate increase function with a step multiplicative factor of 1.2, (ii) a severe response to latency above 500 ms, and (iii) no response to extremely high packet loss rates. We note that properties (i) and (iii) are highly similar to the TCP BBR algorithm [7], which has been gaining popularity on the Internet [6]. Though the time between each bandwidth probe event for Zoom is much longer than for BBR, Zoom's probing gain value of 1.2 is very similar to BBR's 1.25. It seems likely that Zoom uses a BBR-like congestion control algorithm, which contradicts studies that find a fit between Zoom's congestion control and GCC [28].

Zoom's lack of response to packet loss means it behaves essentially opposite to loss-based TCP congestion control. Therefore, in the presence of competing TCP flows, Zoom is likely to starve the TCP flows, as long as the latency does not increase beyond the threshold at which Zoom lowers its sending rate. This effect has been noted in previous work [9,28], and we confirmed this behavior with measurement shown in Appendix Section A. Appendix Figure A.1 shows how the Zoom network flow is unhindered by up to ten concurrent TCP flows started at the same time, instead managing to *increase* its sending rate while the TCP flows are active.

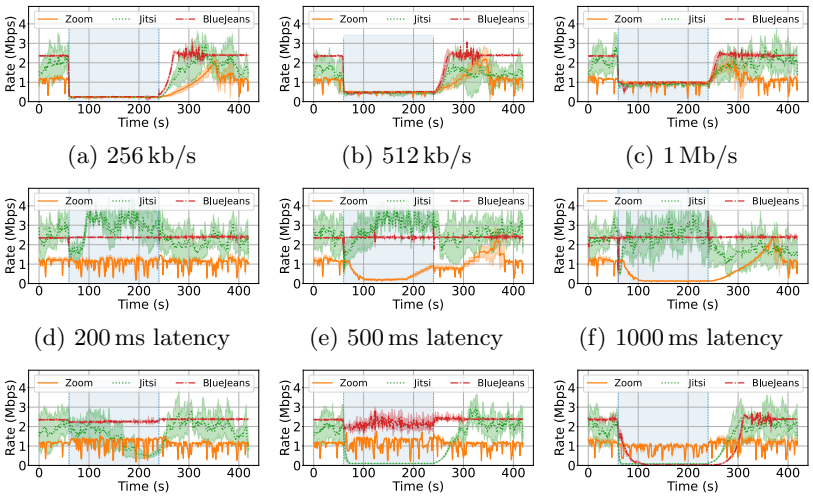
Lastly, Zoom takes a considerably longer time to recover its sending rate compared to WebRTC, as can be seen most clearly in Figure 4(a). At face value, this means the user needs to wait almost 90 seconds longer for Zoom to reach the same sending rate as WebRTC after recovery from a drop in network bandwidth.

4.3 SFU Mode Congestion Control

We run identical measurements to study the congestion control when operating in SFU mode, using Zoom, Jitsi, and BlueJeans. We consider the congestion control at the client and SFU separately.

Client Behavior We run an identical set of measurements, changing upload bandwidth, latency, and packet loss as in Section 4.2. The results for nine out of the 12 conditions are shown in Figure 5. For Jitsi, clients make use of the browser WebRTC implementation using GCC.

Bandwidth. Zoom shows a similar step-like increase function as previously observed. Compared to Zoom, Jitsi and BlueJeans both have a faster rate increase. BlueJeans has the fastest recovery time, around ten seconds faster than Jitsi and over a minute faster than Zoom. Additionally, BlueJeans has a highly fluctuating sending rate for 40 seconds after it returns to the original sending rate when the upload limit is more severe.



(a) 256 kb/s (b) 512 kb/s (c) 1 Mb/s

(d) 200 ms latency (e) 500 ms latency (f) 1000 ms latency

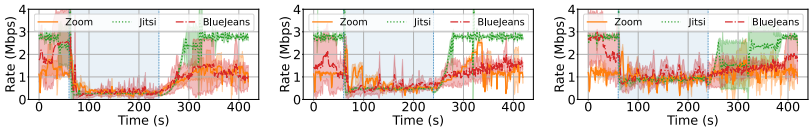
(g) Loss rate 0.05 (h) Loss rate 0.2 (i) Loss rate 0.5

Fig. 5: Response of Zoom, Jitsi, and BlueJeans clients to network conditions while in SFU-mediated mode. (a)-(c) upload bandwidth limits, (d)-(f) additional upload latency, (g)-(i) upload loss rates.

Latency. Zoom has a similar response to the peer-to-peer case when 500 ms latency is added, including how the sending rate will begin to recover before this additional latency is removed. This particular effect is further investigated from the perspective of the video metrics in Section 5.2. This may occur as Zoom realizes the latency has not decreased as a result of reducing the sending rate. Jitsi has a similar transient response to latency changes as WebRTC has in peer-to-peer mode. BlueJeans is notable as it has no observable response to latency changes, except for small transients when the latency is added and removed.

Packet Loss. When the packet loss rate increases, there is an increase in Zoom's sending rate, as opposed to remaining unchanged in the peer-to-peer case. This is possibly because Zoom is adding a larger amount of FEC code for resilience to packet loss. The ability of Zoom to add FEC was noted previously in [30] and is explained further in patents held by Zoom [29]; these results show that the Zoom client itself is also capable of adding FEC. Jitsi starts to show a response to loss above 5%, which becomes more severe as the loss rate increases. BlueJeans does not have a significant response to packet loss except for the 50% case; before this point we are measuring the reduction in effective sending rate due to NLC's packet drops, as explained in Section 3.2 and Section 4.2 for Zoom. At 50% packet loss in Figure 5(i), there is a pronounced reduction in BlueJeans' sending rate. This figure also shows that after packet loss, BlueJeans takes over 30 seconds longer than Jitsi to start recovering sending rate.

BlueJeans stands out among the three VCAs in that it has a purely loss-based congestion control. Zoom and Jitsi have responses to both loss and delay when the SFU is being used.



(a) 256 kb/s (b) 512 kb/s (c) 1 Mb/s

Fig. 6: VCA SFU responses to client download bandwidth.

SFU Behavior The SFU also performs congestion control on forwarded videos, leveraging the SVC or simulcast mechanisms described in Appendix B. By limiting the download bandwidth of Client 1, we can measure the response of the SFU's congestion control.

Figure 6 shows the results of measurements taken similarly to the ones in Sections 4.2 and 4.3. We can see that the Jitsi SFU uses a similar multiplicative rate increase as the client uses, and also takes around 40 seconds to complete the rate increase. However, with higher download limits, we observe some variance in the measurement results. This is likely caused by the SFU's video selection algorithm, which is choosing a different resolution simulcast stream or different frame rate subsampling for each of the five measurement runs.

The Zoom SFU behaves almost identically to the client, whereas the BlueJeans SFU does not keep a steady sending rate, and takes longer to recover from a bandwidth constraint than the client. The SFU takes around one minute to recover to the original sending rate; the client takes around 20 seconds.

Remarks The first notable difference is how Zoom in SFU mode uses a significantly lower sending rate when no network conditions are applied. The reason for this is likely to reduce congestion at the SFU, though we do observe that this is an institutional configuration setting, as accounts from a different institution were capable of the same sending rate in both SFU and peer-to-peer mode. We report the results for the reduced sending rate case to best illustrate this markedly different mode of operation. BlueJeans uses a maximum resolution of 1280x720 even though it also operates in SFU mode like Zoom. Consequently, the bandwidth usage is almost double that of Zoom.

We see that whether the video conference is held in peer-to-peer mode or via the SFU, the Zoom and WebRTC/Jitsi clients have mostly the same congestion control, aside from the lower sending rate for Zoom in SFU mode. Instead, we see significant differences between the VCAs; BlueJeans stands out as having no response to latency increase, and Zoom stands out having little response to packet loss, with the client *increasing* its sending rate when in SFU mode.

5 Video Rate Control Study

We now focus on the video rate control methods used by the VCAs. We again carry out two separate studies of VCAs operating in peer-to-peer mode and SFU mode, to examine what differences the two modes of operation and the different

VCAs have. Our goal as before is to reveal the presence of various functions and their interactions.

5.1 Measurements

Unlike the measurements described in Section 4.1, we keep a constant network condition applied for the duration of the measurements. By adjusting the available bandwidth or the latency of this constant condition, we are able to study how the three video quality metrics are affected: (i) frame rate, (ii) resolution, and (iii) quantization parameter (QP). These will give insight into how the video rate control is responding to different target encoding bitrates provided to it by the congestion control.

Repeated measurements were taken for a range of upload bandwidth limits between 96 kb/s and 2 Mb/s on Client 1. We also took measurements with upload latency values between 200 and 1,200 ms for Zoom in particular, as the results in Sections 4.2 and 4.3 show that Zoom responds to high, long-term latency. Each measurement consists of five minutes with the network condition held in steady state; we take five repeats for each test condition, leading to each plotted point representing 25 minutes of data. Each measurement repeat is given two minutes to stabilize before data is collected. The average and standard deviation for each video metric is computed over the entire 25 minutes of data.

We observe that the receiver's window size impacts the quality of the received video for Zoom. Therefore, in a call with Client 1 and Client 2, if Client 2 changes its window size, it can cause Client 1 to send a lower maximum video quality. This occurs in both peer-to-peer and SFU mode. We ensure the Zoom window on both clients is of sufficient size to allow for the maximum video quality.

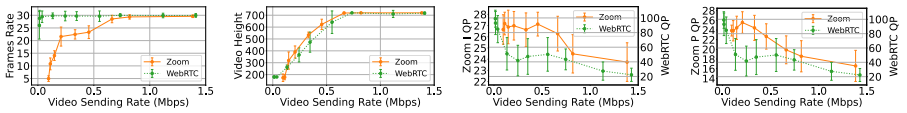
5.2 Peer-to-Peer Mode Video Rate Control

Here, we present the results for measurements with limited upload bandwidth and different upload latencies.

Upload Bandwidth Limit Figure 7 presents the video metrics as a function of the measured video sending bitrate. Note, the minimum possible video sending rate for Zoom was measured to be 150 kb/s; if this bandwidth is not available, Zoom sends no video.

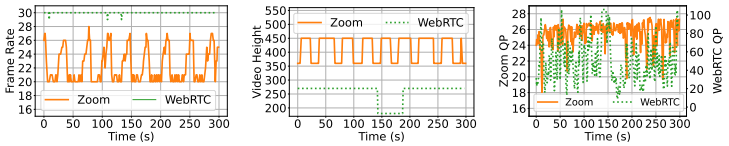
Frame rate. WebRTC clearly prioritizes keeping the frame rate at 30 (same as the source video) as much as possible, with only a slight decrease observed at extremely low sending rates below 100 kb/s. Zoom on the other hand starts to reduce the frame rate when the video sending rate is below 500 kb/s.

Resolution. Both Zoom and WebRTC show a very similar, almost-linear relationship between the average resolution and video sending rate. Zoom requires a slightly lower sending rate for a given resolution. Both VCAs reach the maximum video resolution of 1280x720 (same as the source video) at around 750 kb/s video sending rate.



(a) Frame rate (b) Resolution (c) I-frame QP (d) P-frame QP

Fig. 7: Client sent video quality metrics in peer-to-peer mode, when adjusting the available upload bandwidth.



(a) Frame rate (b) Resolution (c) QP

Fig. 8: Video quality metrics over time with upload bandwidth limited to 256 kb/s.

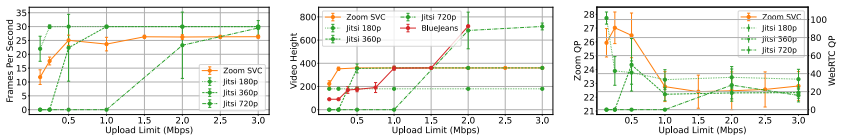
QP. Note that Zoom and WebRTC use different encoders with different QP scaling. As a consequence of WebRTC's preference of maintaining 30 FPS, the QP can reach values above 90, leading to a significant loss of picture quality. Zoom reaches a plateau around 27 for I-frames and 25 for P-frames, achieved by reducing the frame rate as seen in Figure 7(a).

Figure 8 illustrates how the frame rate, resolution, and QP change over time with an upload bandwidth of 256 kb/s. The frame rate and QP can change by small amounts over time, most clearly seen in Zoom's frame rate and WebRTC's QP. However, these small changes will occur around a clear target value. For example, Figure 8(a) shows two distinct frame rate levels for Zoom.

Conversely, the resolution for Zoom and WebRTC takes discrete values, and typically changes on a much longer time scale compared to the frame rate and QP. This suggests that, given a specific target bitrate, the encoder is able to minutely adjust the frame rate and QP to meet it as best as possible, but will settle on one of a small set of available resolutions.

Remarks We note that Zoom and WebRTC have taken different approaches on how the encoder responds to low target bitrates. The video metrics measurements in Figure 7 show that WebRTC considers frame rate to be more important than QP at low bandwidths, whereas Zoom considers the opposite. Therefore, these two VCAs behave very differently in low bandwidth regimes, with WebRTC offering smooth frame rate but extremely low picture quality, and Zoom offering low frame rate and moderate picture quality.

Figure 8 shows how Zoom's frame rate and resolution have a periodic character at the 256 kb/s upload bandwidth limit. This is also shown by the relatively high variance at lower sending rates in Figure 7. Furthermore, the periods of higher frame rate appear to coincide with the periods of lower resolution, and vice versa. Periodic changes in quality are not ideal, and this behavior may be caused by a combination of the low bandwidth limit and a limited set of target



(a) Frame rate (b) Resolution (c) I-frame QP

Fig. 9: Client sent video quality metrics in SFU mode.

resolutions and frame rates that Zoom chooses from. Specifically, the upload bandwidth of 256 kb/s seems to lie between the minimum bandwidths which support 640x360, 26 frame/s and 800x450, 20 frame/s targets.

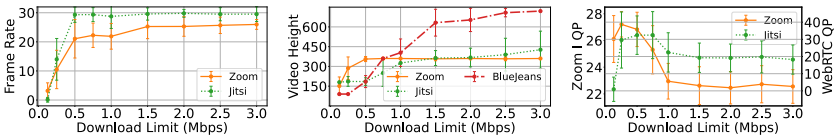
Lastly, results for Zoom under different latency conditions are presented in Appendix Section C. The discrete levels of average sending rate as a function of latency shown in Figure A.2(d) suggest that the Zoom congestion control has four modes of sending rate reduction in response to latency: (i) no reduction, (ii) a limited, temporary reduction, (iii) an increased but still temporary reduction, and (iv) a severe and long-term reduction. Each of these operation modes has a clear threshold in terms of the additional upload latency. Furthermore, within operation mode (iii), the time taken for the sending rate to recover varies as a function of the additional latency.

5.3 SFU Mode Video Rate Control

Client Behavior Figure 9 shows the metrics for video sent by the VCA clients as a function of the available upload bandwidth. Zoom's SVC stream is shown, along with Jitsi's three simulcast streams. Chrome's built-in WebRTC statistics page provides metrics for each of Jitsi's simulcast streams. The BlueJeans application provides information regarding the resolution of the current video; it is likely this represents the resolution of the highest bitrate simulcast stream. The individual simulcast streams cannot be measured for BlueJeans.

Figures 9(a) and 9(b) show that the Zoom client will send video with a maximum resolution of 640x360 and frame rate of 26 frames/s when the upload bandwidth is 512 kb/s or greater. In the peer-to-peer case, the sent video reaches 1280x720 and 30 frames/s. Figure 9(c) shows how Zoom uses additional bandwidth above 512 kb/s to reduce the QP.

The three simulcast streams for Jitsi each turn on at certain upload bandwidths. The 320x180 stream is always on, while the 640x360 stream is enabled at 512 kb/s and the 1280x720 stream is enabled at 2 Mb/s. Interestingly, Figure 9(a) shows that the video streams require additional bandwidth to reach 30 frames/s beyond the amount needed to enable them. This is a different result compared to the WebRTC peer-to-peer case, where 30 frames/s was achieved in all but the lowest bandwidth cases. The QP for the 640x360 and 1280x720 streams also does not exceed 50, showing that the bandwidth thresholds for enabling the simulcast streams are chosen to achieve some minimum picture quality rather than achieve the maximum frame rate. We found that the 320x180 and 640x360 streams have a maximum sending rate of around 200 kb/s and 700 kb/s respectively, and each



(a) Frame rate (b) Resolution (c) I-frame QP

Fig. 10: Client received video quality metrics in SFU mode.

successive stream will not turn on until the lower quality stream is sent at its maximum rate. The resolution of the highest bitrate BlueJeans stream in Figure 9(a) shows a step increase, demonstrating how successive simulcast streams are enabled as bandwidth increases.

SFU Behavior To understand how the SFU decides what video quality to forward, we take similar measurements to Section 5.3 with the download rate limited on Client 1. Figure 10 shows the video metrics as a function of the applied download limit. Overall, the quality metrics for the forwarded video from Zoom's SFU are similar to those for the encoded video from the Zoom client. The main difference is a slight reduction in frame rate and resolution at lower bandwidths. This suggests that the Zoom SFU has almost the same flexibility via subsampling as the client does via encoder parameter selection.

Jitsi achieves a stable 30 frames/s at 512 kb/s download bandwidth as seen in Figure 10(a), which is higher than the frame rate achieved by Zoom. However, at 128 kb/s, the frame rate is almost zero on average, while Zoom will still be able to forward video. This is likely a consequence of the limited frame rate subsampling options available to Jitsi's SFU [26]. The zero frame rate is also what causes the close-to-zero measured QP in Figure 10(c). Figure 10(b) shows that the average resolution does eventually exceed Zoom's maximum of 640x360, which is expected as Jitsi has a 1280x720 simulcast stream. The high variance suggests that the received video resolution changes frequently over time, which implies that the Jitsi SFU is switching between simulcast streams very often.

BlueJeans' resolution is also prone to switching as shown by the variance in Figure 10(b), however it is not as prevalent as in Jitsi. The BlueJeans SFU is able to forward the maximum resolution stream consistently once the download bandwidth reaches 3 Mb/s.

Remarks We find Zoom's SFU behaves very similarly to the Zoom client in the presence of bandwidth limits, as seen in Figures 9 and 10. This shows that the SVC mechanism used by Zoom allows for stream selection at the SFU which is almost as flexible as the client's encoder. The same does not hold for the simulcast systems Jitsi and BlueJeans; for example, the resolution sent by the BlueJeans client at 1.5 Mb/s is lower than the resolution that can be received at 1.5 Mb/s, likely due to the overhead from sending multiple simulcast streams.

The received video quality metrics measurements in Figure 10 show that the Jitsi SFU is unstable even at 3 Mb/s client download rate. The client should be

able to download the maximum bitrate simulcast stream at this point, but the Jitsi SFU still switches it with a lower bitrate stream. This unintended behavior manifests as a received video that changes quality often, possibly degrading QoE. All SFU will be prone to such a problem, especially if the network bandwidth is close to the threshold between two available bitrate streams.

6 SFU with Multiple Users

The experiments in Sections 4 and 5 involve only two users in the setup described in Figure 1. In this section, we consider video conferencing sessions in SFU-mediated mode with more than two users. These experiments demonstrate certain behaviors of the SFU, including how it considers different device types, and how it chooses which videos to forward to a client with constrained download bandwidth. These insights cannot be provided by the two-user experiments.

6.1 Experimental Setup

The additional devices available are an older laptop from 2014 running Mac OS, a 2017 iPad Pro, and a 2015 iPhone 6S. Every device used the native VCA available for the platform, and was connected to the same access point. These devices reflect a range of commonly used form-factors and different device age.

Client 1 is used similarly to Figure 3, with a few modifications. First, the packet trace is not collected, as unknown packet formats make it difficult to filter the trace to correlate packets and clients[2]. Secondly, the video statistics collection is performed manually for Zoom, as it only displays the statistics for the *currently focused video*. We continue to use NLC to adjust the download rate on Client 1, as in Section 5.3. The SFU will measure the available download rate for the client in order to decide which videos to send and at what quality.

VCAs display videos to users in one of two modes; single-speaker or gallery mode. The *currently focused video* is the video which is shown in single-speaker mode, or the current speaker's video in gallery mode. Therefore, we can collect the Zoom video statistics for a specific device by unmuting it and keeping all other devices muted. The same method does not work for BlueJeans', which seems to report the highest resolution out of all displayed videos. Jitsi is able to use the same measurement collection method used previously, as `chrome://webrtc-internals` provides one report for each received video stream.

6.2 Observations

We consider two effects in our experiments. The first is the impact of device type on the recorded statistics. Secondly, we try to understand how the SFU decides which videos to send, and at what quality to send those videos.

[2] The Zoom packet structure determined by Michel et al. [31] could aid this process.

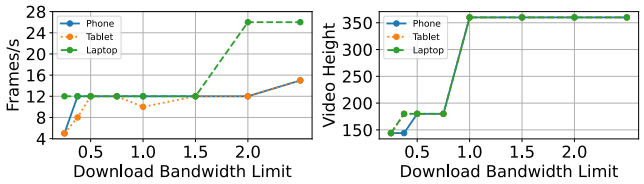


Fig. 11: Received frame rate and resolution with different device types on Zoom.

Impact of Device Type Device type may have an impact on VCA performance as described in [9,40]. Therefore, we can evaluate whether there are any differences between the three device types: laptop, tablet, and phone.

Zoom. Figure 11 shows the received frame rate and video resolution for the three additional devices on a Zoom call with Clients 1 and 2. Client 1 was kept in gallery mode for this experiment. The video resolution received from each device type generally follows the same trend, though at the lowest download bandwidths, it appears that the laptop gets some priority in bandwidth allocation. However, for frame rate, there is a significant difference; even at higher bandwidths, the phone and tablet will only be received at 15 frames/s maximum.

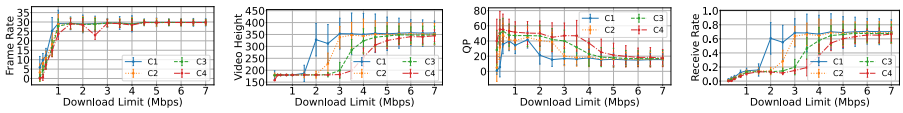
In single-speaker mode, we found that all devices had the same behavior. This shows that the phone and tablet are capable of sending at the maximum 26 frames/s that was found in the two-client video call for Zoom. However, the SFU intervenes to limit the forwarded video from the phone and tablet to 15 frames/s if the client receiving the forwarded video is in gallery mode.

BlueJeans. While the reported video statistic for BlueJeans is difficult to control, we observed that a 320x180 resolution video was received from the phone client in gallery mode, while all others were 640x360. The frame rate, while not reported in the BlueJeans application, appeared to be the same for all devices.

Jitsi. There was no observed impact of device type for Jitsi.

SFU Decisionmaking Section 5.3 describes how the SFU adjusts the video quality metrics of the forwarded video streams in response to download bandwidth constraints at the receiving client. If multiple client videos are available for forwarding, the SFU must also make decisions on which videos to forward to a receiving client. Combined with the adjustment of per-stream video quality metrics, this represents the full congestion control at the SFU. Known decision making processes such as Last-N [19] are based on the measured download bandwidth for the receiving client, as well as a priority queue based on when each client last spoke (produced sound through their microphone).

In all three VCAs, we found that the SFU will only decide which videos to forward when the receiving client is in gallery mode, displaying multiple other client videos at once. In single-speaker mode, all SFUs will forward the currently focused video at maximum possible quality. Typically, they will forward only thumbnail videos at the lowest possible resolution for the remaining video participants, which may be displayed as small insets in the VCA interface.



(a) Frame rate (b) Resolution (c) QP (d) Receive rate

Fig. 12: Jitsi received video quality metrics as a function of download bandwidth when receiving four client videos (C1–C4) simultaneously.

Zoom. The limited information available within the application and packet traces mean an observational approach must be taken. Client 1's download bandwidth is limited between 0.5 and 7 Mbps with the other five available devices connected with video on. Notably, Zoom does not turn off any received videos; at lower bandwidth, some videos may freeze for an extended time but will remain visible. At moderate bandwidths, we observed that Zoom will allocate the maximum possible quality to the currently focused video, and then share remaining bandwidth fairly between all others.

BlueJeans. As BlueJeans also provides limited information in the application and packet trace, the same approach as Zoom is taken. Like Zoom, BlueJeans does not turn off received videos even at very low bandwidths. However, unlike Zoom, BlueJeans has a more clear prioritization of videos; the currently focused video will receive the most bandwidth possible, but then successive videos will receive a bandwidth share in order of when they were last active in audio. So the remaining bandwidth after the currently focused video is not shared equally, it is prioritized similarly to Jitsi's Last-N algorithm.

Jitsi. Jitsi's Last-N algorithm is well described [19]. Furthermore, the Jitsi source code describes how bandwidth is allocated to videos according to the Last-N prioritization [25,26]. Altogether, the decisionmaking process is very similar to BlueJeans. However, the `chrome://webrtc-internals` statistics allow us to evaluate how Last-N impacts the received video quality metrics.

Figure 12 shows how the video metrics for each of the four received video streams change as a function of download bandwidth. The Last-N algorithm is clear in the plots for resolution, QP, and video bitrate; each video stream increases in quality in turn. However, for the frame rate, all three videos are being received at the maximum 30 frames/s as soon as 1.5 Mbps download rate is reached. This matches the general behavior of WebRTC and Jitsi as described in Sections 5.3 and 5.3 where the frame rate is maximised with greater priority than the resolution and QP.

Altogether, the three studied SFU-mediated VCAs have similar behavior when deciding which videos to forward and at what quality. All use a prioritization based on which client was last active in audio; this client becomes the currently focused video in gallery mode. Zoom will share the remaining bandwidth fairly among other clients, whereas BlueJeans and Jitsi will allocate bandwidth to clients prioritized according to audio activity.

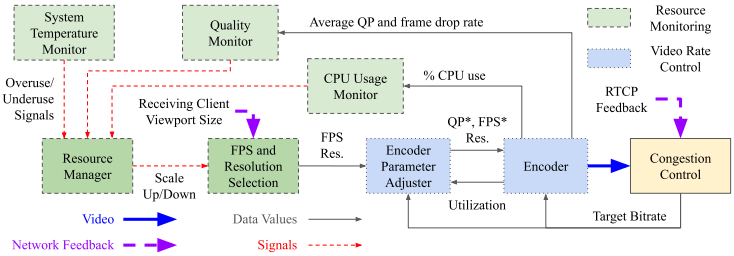


Fig. 13: Functional model for the peer-to-peer VCA client.

7 VCA Functional Models

In this section, we leverage the results presented in Sections 4, 5, and 6 to derive the functional models for a VCA client and VCA SFU. These functional models will expand on the high-level depictions in Figure 2. In addition to the measurement results, we will also include findings from a study of the Chromium WebRTC and Jitsi open-source code [18,25].

7.1 VCA Client

Using our results from Sections 4 and 5, along with a review of the WebRTC open-source code, we can elaborate on the simple functional diagram of the client depicted in Figure 2(a). In this section, we focus specifically on the *peer-to-peer* client, as it does not involve simulcast or SVC-based encoding, a complexity which we cover in Section 7.2.

Figure 13 shows the outcome of piecing together the measurement results and code survey into a coherent functional model. There are three distinct components in the functional model: (i) *resource monitoring*, indicated by the green boxes with dashed outline, (ii) *congestion control*, indicated by the yellow box with solid outline, and (iii) *video rate control*, indicated by the blue boxes with dotted outline.

Resource Monitoring The resource monitoring component relies on three "monitors" which constantly measure certain system parameters. These three monitors are explicitly defined in Chromium WebRTC, and we postulate that they are also used in other VCAs. The first monitor is the *system temperature monitor*, which can generate an overuse signal to the VCA to reduce video quality to ease processing load and reduce device temperature. Curiously, in the Chromium code, we find that this monitor is enabled by default only on Chrome OS and Mac OS, further highlighting potential differences between VCA behavior on different operating systems.

The second is the *CPU usage monitor*, which provides similar feedback to the system temperature monitor. While Zoom's source code is unavailable for review, we note that the Zoom statistics view does report CPU usage, suggesting that this parameter is at least monitored by Zoom. CPU utilization on mobile

devices can reach 100% on two cores in typical VCA usage [9], meaning that the CPU usage monitor, like the system temperature monitor, is particularly important for mobile devices.

The third monitor is the *quality monitor*, which monitors the average QP and frame drop rate reported by the encoder. The encoder may drop frames when it has insufficient bitrate to encode frames at the target video metrics. If the frame drop rate or QP are above or below certain thresholds, this monitor will generate an overuse or underuse signal. The Chromium WebRTC source specifies an upper threshold for frame drop rate of 0.6 and a QP of 95 for signaling overuse, and a lower QP threshold of 29 for signaling underuse. This monitor is particularly important as it ultimately derives from the target bitrate provided by the congestion control, meaning that under typical VCA use, this is the monitor that governs much of the VCA operation.

All three monitors feed their signals into the *resource manager*, which aggregates the incoming signals and produces a scale up or scale down signal, ensuring only the most important signal is processed if multiple are active. The scale up/down signal is used by the *FPS and resolution selection* to adjust the target frame rate and video resolution used by the encoder. While the architecture of this resource monitoring component is inferred in large part from the Chromium WebRTC source, we believe that a functionally similar system is employed by Zoom and BlueJeans, and VCAs generally. This is indicated largely by the results in Sections 5.2 and Sections 5.3, which show that the VCAs have a discrete set of target frame rates and resolutions.

Congestion Control The congestion control component of the model maintains the high-level view as in Figure 2. While a unified VCA congestion control model based on GCC has been proposed [28], we find through the results in Section 4 that the congestion control behavior is vastly different between VCAs. Zoom experiences reduced sending rates in the presence of stable high latency and BlueJeans has no response to latency at all; these modes of operation are incompatible with the GCC model, which responds to the *change* in latency [8,13,22,24]. A similar point can be made for packet loss response; Zoom effectively does not have any, but BlueJeans and WebRTC/Jitsi have a very clear reduction in sending rate for moderately high levels of packet loss.

Another distinction between the VCAs is how their sending rates increase over time when network conditions improve. As noted in Section 4.2, Zoom has a step-like increase in sending rate over time. Furthermore, as seen in Figure 5(b), this stepped increase function can exceed the typical maximum video sending rate, suggesting that Zoom is actively probing for bandwidth, rather than employing a multiplicative rate increase function as used by GCC.

Altogether, the differences in measured congestion control responses in Section 4 demonstrate the infeasibility of generalizing VCA congestion control. Even within only the subset of specific VCAs covered in this study, we observed highly incongruent techniques for congestion control. Therefore, we include the conges-

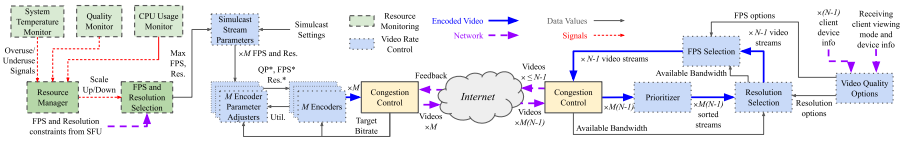


Fig. 14: Functional model for the VCA SFU and client.

tion control as a unified block in the functional model, which uses RTCP network feedback to make adjustments to the target video encoding bitrate.

Video Rate Control The video rate control is split into two parts, the *encoder parameter adjuster*, and the *encoder* itself. The *encoder parameter adjuster* accepts a target frame rate and resolution from the *FPS and resolution selection* and makes adjustments to the targets as well as generates the QP. The objective of this is to most closely match the target bitrate that is provided by the congestion control. As seen in Figure 8, the QP and frame rate can be finely adjusted over time to match the target bitrate as best as possible, but the resolution takes a set of discrete values and does not change on such a small timescale. Therefore, as illustrated in Figure 13, the resolution emerges from the *encoder parameter adjuster* unchanged, whereas the final frame rate may be different from the target, indicated by the asterisk notation.

The *encoder* uses the resolution, frame rate, and QP provided by the *encoder parameter adjuster* to produce the video that will be sent over the network. The encoder also uses the target bitrate produced from the congestion control to compute the utilization, which it feeds back to the *encoder parameter adjuster*. This forms the first of three closed feedback loops that the encoder drives. The second feedback loop involves the encoder's CPU utilization, and the third feedback loop involves the frame drop rate and the QP. These two feedback loops connect back to the system monitors.

Section 5 shows how Zoom and WebRTC/Jitsi share a similar control for the video resolution as a function of the available bandwidth, but opposite behavior for frame rate and QP. At low bandwidths, Zoom will drop the frame rate to maintain a reasonable QP, while WebRTC/Jitsi will maximize the QP to maintain a high frame rate. Therefore, while each VCA may employ a different policy or algorithm to control the video metrics for the video rate control, they all make use of the general functionality described in Figure 13.

7.2 VCA SFU

The VCA SFU model also derives from measurement results in Sections 4 and 5. In addition, the observations from Section 6 will be vital for understanding some specific components of the SFU. Figure 14 presents the complete functional model of a VCA client and SFU operating in tandem. This functional model represents the singular VCA SFU and one of N connected clients, all of whom are functionally identical. The VCA client follows the same structure as Figure 13,

with some differences specific to operation in SFU mode. We develop the SFU side of the model to be as analogous to the client side as possible, and so it is split into the *congestion control* and *video rate control* components, with the associated boxes colored and bordered as in Figure 13.

Changes to the Client Model The client now includes M encoders and encoder parameter adjusters to support simulcast streams at M different resolutions. Zoom, with its single SVC stream, has $M = 1$. In our measurements and examination of the source code, we observe $M = 3$ for Jitsi. We are not able to determine specific parameters for BlueJeans. Additionally, the client may also receive feedback from the SFU providing additional constraints for what resolution and frame rate to send. These constraints are based on the SFU's knowledge of how the client's video is being displayed by other clients. As observed for Zoom, if all other clients are displaying a particular client's video in a small viewport, then that client has no need to send high quality video and the SFU will provide parameters to constrain the quality of the video being sent.

We note that the measurement results in Sections 4.3 and 5.3 show that the congestion control and video rate control mechanisms on the client side are generally identical between peer-to-peer and SFU modes. Therefore, we retain the same congestion control and video rate control components as in the peer-to-peer model.

SFU Congestion Control Section 4.3 shows that the SFU also performs congestion control when forwarding video to a client. As on the client side, the SFU must implement congestion control by adjusting the video metrics for the videos it forwards. Furthermore, the SFU can choose to not forward all videos if there is insufficient bandwidth to do so. Therefore, the foremost job of the SFU congestion control is to generate an available bandwidth for the video rate control to use for decision making. This is equivalent to the target encoding bitrate generated by the congestion control on the client side.

The specific behaviors of the SFU congestion control seen in Section 4.3 typically mirror those of the client, and so we are left with the same conclusion that generalizing the SFU congestion control is infeasible. With the observation that each VCA's SFU will implement its own specific congestion control algorithm, typically similar to the client's, we use the same unified block as on the client.

SFU Video Rate Control The video rate control on the SFU side of the model can be viewed as a step-by-step procedure. First, the $M(N-1)$ received client videos are processed by the *prioritizer*, which orders the videos according to a specific criteria. For Jitsi, the Last-N algorithm which prioritizes by most recent speakers is used [19], and we observed similar behavior for BlueJeans as described in Section 6.2. Zoom uses a similar algorithm, but instead attempts to share the remaining bandwidth fairly after allocating the most recent/pinned speaker as much bandwidth as possible. We note the prioritization process can

be done in a centralized manner by the SFU and applied to the video rate control for all receiving clients.

After video prioritization, the SFU will run *resolution selection* for each of the $N-1$ client videos that are to be forwarded using the available bandwidth estimated by the congestion control. For VCAs which use simulcast, this will involve the selection of one out of the M simulcast streams that were received. For VCAs that use SVC-based encoding, this will involve a subsampling procedure. In any case, the number of videos which emerge from the *resolution selection* will be $N-1$. The SFU will then run *FPS selection* for each of the $N-1$ video streams, again taking the available bandwidth into account. *FPS selection* can typically be performed via subsampling on all VCAs, as it is supported by H.264, VP8, and VP9, the most common encoders used.

The available options for resolution and frame rate are determined by several factors, including the devices used by the $N-1$ sending clients and receiving client, and the receiving client's viewing mode. All will be known by the SFU as a result of signaling operations. The effect of different device types for Zoom is shown in Figure 11, and the video metrics for each of four received streams at a bandwidth-limited client are shown in Figure 12. This figure clearly shows the prioritization method, with four distinct traces for each client. We note that the SFU must perform all of its video rate control on the encoded video streams, as the decoding would put an unscalable computational load on the SFU.

8 Concluding Remarks

VCAs deploy congestion control and video rate control functionality at both the client and SFU. In both instances, target rates are determined by congestion control functions and then used to influence the rate of video transmitted by the clients and the SFU. The adjustment of the video rate has direct consequences on the video quality metrics such as frame rate and resolution. Given this baseline level of understanding, we constructed more detailed functional models for the VCA client and SFU which are based predominantly off of a directed measurement campaign using a subset of commonly used VCAs. We believe these models, along with the accompanying measurement results, provide a level of understanding which was previously unavailable in related literature.

We expect the functional models will serve to inform further research into video conferencing. We will use the functional models in our future work to relate the congestion control mechanisms employed by different VCAs to end user quality of experience, which is an important functionality for network operators in order to best provision network services to maximize end user QoE.

9 Acknowledgements

This work was supported by NSF grant NETS-1909040. We thank the PAM reviewers and our shepherd for their valuable feedback.

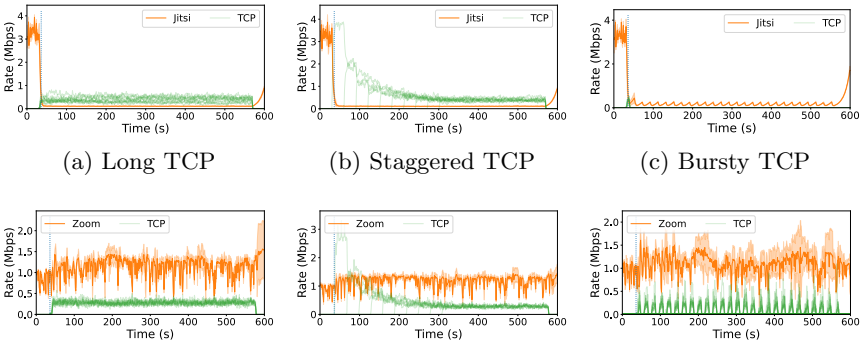
References

1. Amirante, A., Castaldi, T., Miniero, L., Romano, S.P.: Performance Analysis of the Janus WebRTC Gateway. In: Proc. ACM Workshop on All-Web Real-Time Systems (2015)
2. Amon, P., Li, H., Hutter, A., Renzi, D., Battista, S.: Scalable Video Coding and Transcoding. In: Proc. IEEE International Conference on Automation, Quality and Testing, Robotics (2008)
3. André, E., Le Breton, N., Lemesle, A., Roux, L., Gouaillard, A.: Comparative Study of WebRTC Open Source SFUs for Video Conferencing. In: Principles, Systems and Applications of IP Telecommunications (IPTComm) (2018)
4. Arif, A.S.M., Hassan, S., Ghazali, O., Nor, S.A.: The Relationship of TFRC Congestion Control to Video Rate Control Optimization. In: Proc. IEEE International Conference on Network Applications, Protocols and Services (2010)
5. BlueJeans: Technical Specifications for Services (2021), https://support.bluejeans.com/s/article/BlueJeans-Technical-Specifications
6. Cao, Y., Jain, A., Sharma, K., Balasubramanian, A., Gandhi, A.: When to Use and When Not to Use BBR: An Empirical Analysis and Evaluation Study. In: Proc. ACM Internet Measurement Conference (2019)
7. Cardwell, N., Cheng, Y., Gunn, C.S., Yeganeh, S.H., Jacobson, V.: BBR: Congestion-Based Congestion Control. ACM Queue pp. 20 – 53 (2016)
8. Carlucci, G., de Cicco, L., Holmer, S., Mascolo, S.: Analysis and Design of the Google Congestion Control for Web Real-Time Communication (WebRTC). In: Proc. ACM International Conference on Multimedia Systems (2016)
9. Chang, H., Varvello, M., Hao, F., Mukherjee, S.: Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet. In: Proc. ACM Internet Measurement Conference (2021)
10. Chromium: ChromeDriver - WebDriver for Chrome (2022), https://chromedriver.chromium.org/home
11. Chundong, S., Chaojun, L., Shaohua, L.: Research on Congestion Control Algorithms for Real-time Audio and Video Stream. In: Proc. IEEE International Conference on Computer and Communications (2018)
12. Cisco: Cisco Ultra Traffic Optimization (CUTO) Data Sheet (2021), https://www.cisco.com/c/en/us/products/collateral/wireless/ultra-traffic-optimization/datasheet-c78-744385.html
13. De Cicco, L., Carlucci, G., Mascolo, S.: Congestion Control for WebRTC: Standardization Status and Open Issues. IEEE Communications Standards Magazine **1**(2), 22–27 (2017)
14. Federal Communications Commission: FCC Proposes Higher Speed Goals for Small Rural Broadband Providers (2022), https://www.fcc.gov/document/fcc-proposes-higher-speed-goals-small-rural-broadband-providers-0
15. Floyd, S., Handley, M., Padhye, J., Widmer, J.: Equation-Based Congestion Control for Unicast Applications. In: Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (2000)
16. Fund, F., Wang, C., Liu, Y., Korakis, T., Zink, M., Panwar, S.S.: Performance of DASH and WebRTC Video Services for Mobile Users. In: Proc. IEEE Packet Video Workshop. pp. 1–8 (2013). https://doi.org/10.1109/PV.2013.6691455
17. Google: Tesseract Open Source OCR Engine (2022), https://tesseract-ocr.github.io/

18. Google Git: WebRTC Native Code Package (2022), https://webrtc.googlesource.com/src/
19. Grozev, B., Marinov, L., Singh, V., Ivov, E.: Last N: Relevance-Based Selectivity for Forwarding Video in Multimedia Conferences. In: Proc. ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (2015)
20. He, J., Ammar, M., Zegura, E.: Automation Code (2023), https://github.com/jh4001/PAM2023_Automation
21. High Scalability: A Short On How Zoom Works (2020), http://highscalability.com/blog/2020/5/14/a-short-on-how-zoom-works.html
22. Holmer, S., Lundin, H., Carlucci, G., de Cicco, L., Mascolo, S.: A Google Congestion Control Algorithm for Real-Time Communication (2017), https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02
23. International Telecommunications Union: H.264: Advanced video coding for generic audiovisual services (2016), https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-H.264-201602-S!!PDF-E&type=items
24. Jansen, B., Goodwin, T., Gupta, V., Kuipers, F., Zussman, G.: Performance Evaluation of WebRTC-Based Video Conferencing. ACM SIGMETRICS Performance Evaluation Review **45**(3), 56–68 (2018)
25. Jitsi: Github repository (2022), https://github.com/jitsi
26. Jitsi: Selective Forwarding Unit implementation of the Jitsi Videobridge (2022), https://github.com/jitsi/jitsi-videobridge/blob/master/doc/sfu.md#bandwidth-estimations
27. Kumar, R., Nagpal, D., Naik, V., Chakraborty, D.: Comparison of Popular Video Conferencing Apps Using Client-Side Measurements on Different Backhaul Networks. In: Proc. ACM Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (2022)
28. Lee, I., Lee, J., Lee, K., Grunwald, D., Ha, S.: Demystifying Commercial Video Conferencing Applications. In: Proc. ACM International Conference on Multimedia (2021)
29. Liu, Q., Jia, Z., Jin, K., Wu, J., Zhang, H.: Error resilience for interactive real-time multimedia application. U.S. Patent #10348454 (2019)
30. MacMillan, K., Mangla, T., Saxon, J., Feamster, N.: Measuring the Performance and Network Utilization of Popular Video Conferencing Applications. In: Proc. ACM Internet Measurement Conference (2021)
31. Michel, O., Sengupta, S., Kim, H., Netravali, R., Rexford, J.: Enabling Passive Measurement of Zoom Performance in Production Networks. In: Proc. ACM Internet Measurement Conference (2022)
32. Muthukadan, B.: Selenium with Python (2022), https://selenium-python.readthedocs.io/
33. Robitza, W., Goring, S., Lebreton, P., Trevivian, N.: ffmpeg_debug_qp (2022), https://github.com/slhck/ffmpeg-debug-qp
34. Sander, C., Kunze, I., Wehrle, K., Rüth, J.: Video Conferencing and Flow-Rate Fairness: A First Look at Zoom and the Impact of Flow-Queuing AQM. In: Proc. Passive and Active Measurement. pp. 3–19 (2021)
35. Sandvine: 2022 Global Internet Phenomena Report (2022)
36. Schwarz, H., Marpe, D., Wiegand, T.: Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. IEEE Transactions on Circuits and Systems for Video Technology **17**(9), 1103–1120 (2007)

37. Streaming Media: Q&A: Zoom CTO Brendan Ittelson (2021), https://www.streamingmedia.com/Articles/Editorial/Featured-Articles/QA-Zoom-CTO-Brendan-Ittelson-145024.aspx
38. Sweigart, A.: PyAutoGUI - GitHub (2022), https://github.com/asweigart/pyautogui
39. Varvello, M., Chang, H., Zaki, Y.: Performance Characterization of Videoconferencing in the Wild. In: Proc. ACM Internet Measurement Conference (2022)
40. Vucic, D., Skorin-Kapov, L.: QoE Assessment of Mobile Multiparty Audiovisual Telemeetings. IEEE Access **8**, 107669–107684 (2020)
41. WebRTC Glossary: Temporal Scalability (2022), https://webrtcglossary.com/temporal-scalability/
42. Xu, Y., Yu, C., Li, J., Liu, Y.: Video Telephony for End-Consumers: Measurement Study of Google+, IChat, and Skype. In: Proc. ACM Internet Measurement Conference (2012)
43. Yu, C., Xu, Y., Liu, B., Liu, Y.: "Can you SEE me now?" A measurement study of mobile video calls. In: Proc. IEEE Conference on Computer Communications (2014)
44. Zoom: Zoom: Architected for Reliability (2019), https://explore.zoom.us/docs/doc/Zoom_Global_Infrastructure.pdf
45. Zoom: Here's How Zoom Provides Industry-Leading Video Capacity (2022), https://blog.zoom.us/zoom-can-provide-increase-industry-leading-video-capacity/

A Competition with TCP



(a) Long TCP (b) Staggered TCP (c) Bursty TCP

(d) Long TCP (e) Staggered TCP (f) Bursty TCP

Fig. A.1: Competition with different types of TCP flows. (a-c) Jitsi, (d-f) Zoom in SFU mode.

As Zoom has been reported to take a majority share of the bandwidth when competing with TCP [28,30,34], we took measurements with different types of TCP flows to compare and explain results using the functional models and measurements in Section 4.

VCA	Simulcast	SVC	Subsample FPS	Subsample Res.	Codec	Alternate Codecs
Zoom	No	Yes	Yes	Yes	H.264	None
Jitsi	Yes	No	Yes	No	VP8	VP9, H.264
BlueJeans	Yes	No	Yes	No	VP8	None

Table 2: Summary of subsampling methods for the considered SFU-mode VCAs.

The testbed was used in two-person SFU mode with Zoom and Jitsi as in Section 4.3. In all cases, a total bandwidth limit of 4 Mb/s was applied to the upload of Client 1. After 30 seconds of measurement, ten TCP flows begin in various patterns: (i) started at the same time with nine minute duration, (ii) started at 30 second intervals, all finishing at the nine minute mark, and (iii) started and stopped at the same time every 20 seconds.

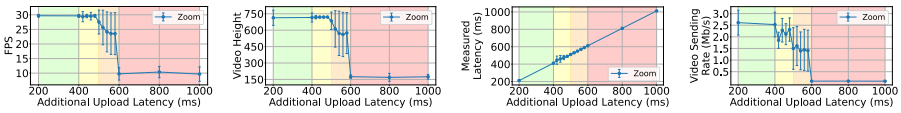
Figure A.1 shows the results of these measurements. Figures A.1(a), A.1(b), and A.1(c) all show that Jitsi immediately gives up practically all of its bandwidth in the presence of TCP. In particular, Figure A.1(b) shows that this occurs with only a single TCP flow sharing the link at the 30 second mark. This behavior is likely caused by GCC's sensitivity to changes in delay; once GCC gives up some bandwidth, TCP takes it, compounding the response. Conversely, Figures A.1(d), A.1(e), and A.1(f) show that Zoom sending rate actually increases in TCP's presence. This is likely a consequence of how it decides to add FEC as described in Section 4.3, the general insensitivity that Zoom has to packet loss, and the high threshold for responding to latency.

B Video Subsampling Methods

As mentioned in Section 1, the SFU is able to adjust the quality of forwarded video streams via subsampling. The exact method used for this differs between the considered VCAs, as described below, and summarized in Table 2.

Zoom uses a custom implementation of H.264 Scalable Video Coding (SVC) [2,23,36] to encode base and enhancement video layers. The SFU can then add/drop layers before forwarding to clients. Because H.264 SVC is used, subsampling of both the resolution and the frame rate is available to the SFU in Zoom as an orthogonal technique to adjust video bit rate [21,37,44,45].

Jitsi and BlueJeans use video *simulcast*, in which the VCA client sends multiple independent video streams at different resolutions. Jitsi uses simulcast when the VP8 encoder is used. In particular, the Jitsi clients make use of three simulcast streams, with resolution scaling factors of 1, 2, and 4. At 1280x720 native video resolution, this means the two other streams will be 640x360 and 320x180. BlueJeans also uses the VP8 encoder, implying use of simulcast, though there are no means of measuring the individual video streams. In a VCA with simulcast, the SFU chooses which one of the received video streams to forward, which determines the resolution. Additionally, it may choose to adjust the frame rate



(a) FPS (b) Resolution (c) Measured latency (d) Upload rate

Fig. A.2: Client sent video quality metrics in peer-to-peer mode, when adjusting the upload latency. The colors indicate the distinct modes of operation.

without re-encoding. Note that resolution changes without re-encoding are not possible with the VP8 codec [5,41], demanding the use of simulcast.

C Zoom Video Rate Control under Different Latency Conditions

Figure A.2 presents the video metrics for Zoom as a function of the additional latency on the upload. Overall, the sent video metrics begin to see an impact at 400 ms additional upload latency, and beyond 600 ms additional latency, there are no further impacts.

Frame Rate and Resolution. We group the consideration of both of these metrics as they share a very similar response to the additional upload latency. The most notable feature is the evolution of high variance in the measurement results for added latencies above 500 ms. We observe that much of the measured variance is not due to fluctuation in the frame rate or resolution; instead it is caused by Zoom returning to the maximum resolution and frame rate some time after the experiment starts. The time at which this occurs is a function of the latency; the higher the latency, the longer Zoom takes to begin recovering its sending rate. At higher latencies beyond 600 ms, the recorded variance is low. This is either because the time taken for Zoom to begin recovering is longer than the measurement duration, or because Zoom keeps the low sending rate indefinitely if the latency is beyond this value.

Measured video sending rate. The measured video sending rate corresponds well to the frame rate and resolution trends. Specifically, there appear to be two intermediate levels of video sending rate; the first averaging around 2.25 Mb/s between 400 and 500 ms added latency, and the second averaging around 1.5 Mb/s between 500 and 600 ms. Beyond 600 ms, a flat 100 kb/s sending rate is used, which corresponds to what is observed in Figure 4(f).

QP. The method for obtaining Zoom's QP outlined in Section 3.2 is incompatible with latency measurements. This is because starting a screen recording in the Zoom application prompts a sudden, very large spike in latency. Once this spike dissipates, the Zoom application begins sending at the maximum rate, even when the steady-state latency is above 1,000 ms. We are uncertain if this is intended behavior to maintain maximum quality for recording purposes, or if this is a bug. In any case, we are unable to ascertain the Zoom QP for these measurements.