Instant Messaging Meets Video Conferencing: Studying the Performance of IM Video Calls

Laurenz Grote*, Ike Kunze*, Constantin Sander*, Klaus Wehrle
Communication and Distributed Systems
RWTH Aachen University
Aachen, Germany
{grote, kunze, sander, wehrle}@comsys.rwth-aachen.de

***Abstract*—Video conferencing applications typically use UDP and often implement their own congestion control. Research studying these custom algorithms generally finds that they do react to congestion and can hold their own against competing TCP flows. However, these works focus on applications *specializing* in video conferencing, neglecting those that only offer video conferencing as one of many features, such as many instant messengers. While these instant messaging-based video call applications (IMVCAs) may opt to use standardized frameworks, such as WebRTC, their actual implementations and behaviors are wildly unknown. In this paper, we thus set out to study the behavior of three popular IMVCAs, analyzing their interplay with TCP and the impact on QoE.**

We find that the surveyed IMVCAs (Signal, Telegram, WhatsApp) are TCP-friendly, i.e., they do not choke TCP. However, their per-app behavior differs significantly and no app equals the other: Signal and Telegram, e.g., take TCP-friendliness too far, yielding up to $90\,\%$ of their bandwidth. This results in severe QoE detriments in the form of drastically reduced sending rates and visual quality. As Signal is known to use WebRTC, this finding suggests that the current variant might be too conservative for coexisting with TCP. In contrast, WhatsApp counters congestion by filling queues to avoid losing bandwidth. Overall, IMVCAs do not keep up with the performance of specialized applications.

I. Introduction

Video conferencing is an established way of telecommunication. Due to inherent real-time requirements, applications providing video conferencing services generally use UDP and opt to implement their own congestion control (CC). As the characteristics of these implementations are often unknown and as these applications see a growing traffic share [1], there is general interest in better understanding the exact behavior of such video conferencing applications.

Research has thus studied the CC performance of prominent applications [2]–[4], such as Google Meet, Microsoft Teams, Webex, and Zoom, finding that they do indeed react to congestion. Yet, researchers have also noted significant differences in behavior as, e.g., Zoom was quick to capture an unfair share of the overall bandwidth [2], [3]. Going beyond CC, researchers have also studied the video conference quality [5], [6] as well as the behavior of users [5] to judge the interplay of bandwidth use and quality of experience (QoE). While these studies provide a multi-dimensional view on current video conferencing practices, they focus on applications that *specialize* in video conferencing. However, many casual interactions are facilitated by instant messaging applications which increasingly provide quick video calls as an *add-on*.

Owing to their multi-purpose nature, several instant messaging-based video call applications (IMVCAs) use (standardized) open-source algorithms and libraries, such as Google Congestion Control and WebRTC, for providing the video call functionality [7]. The performance of these frameworks has already been studied by related works [8], [9], but critically depends on the specific parameterization. In contrast, other applications use custom implementations similar to those seen for specialized video conferencing applications. Consequently, as IMVCAs have not yet been in the focus of research, the actual behavior in this realm is largely unknown. For instance, it is unknown how IMVCAs react on congestion and bandwidth competition and whether they starve or take away over-proportional amounts of bandwidth. Additionally, it is unclear whether the network, e.g., in the form of active queue management (AQM), can help to alleviate potential problems.

In this paper, we thus set out to study the performance of IMVCAs in more detail. Focusing on Signal, Telegram, and WhatsApp in a controlled testbed, we find that these apps behave in stark contrast to previous findings on video conferencing applications. In particular, Signal and Telegram are overly TCP-friendly and yield vast amounts of bandwidth. When interacting with either BBRv2 or Cubic as the TCP CC, they give up their bandwidth to the point where they severely disadvantage themselves. WhatsApp in contrast yields less bandwidth. Overall, we contribute and find the following:

- We automate Signal, Telegram, and WhatsApp video calls to study their behavior in a controlled testbed.
- Characterizing their standalone performance, we identify significantly different transmission rates.
- Competing against TCP, Signal and Telegram yield bandwidth, while WhatsApp protects itself by filling queues.
- Call QoE changes immensely during flow competition.
- AQM can help to improve QoE, but requires to be applied together with fair-queuing.
- Our code and data is available under [10].

Structure Sec. II gives a short overview on video conferencing, CC, and previous work comparing both w.r.t. bandwidth and QoE performance. Sec. III describes our methodology

*Equal Contribution

978-3-903176-58-4 ©2023 IFIP

and testbed setup to allow for a controlled analysis of the selected IMVCAs. Sec. IV then presents the CC behavior of the apps, focusing on baseline numbers and reactions during different scenarios. Sec. V uses these CC findings to drill down into QoE impacts and performance. Finally, we discuss the implications of our findings for work on video conferencing in general in Sec. VI before Sec. VII concludes our paper.

II. Background and Related Work

Due to inherent real-time requirements, applications providing video conferencing services typically rely on UDP to avoid waiting for retransmissions or using kernel-defined congestion control (CC). Instead, they deploy their own CC algorithms (CCAs), striving for reduced delays while maintaining high throughput to optimize the quality of experience (QoE) of their users. In this context, the IETF RMCAT working group [11] works on fair and standardized multi-media CC for real-time applications. For example, RFC 8836 [12] specifies guidelines and requirements for CC of interactive real-time media.

Video conferencing best practices. Today, there are several standardized protocols that are used throughout the industry. Google Congestion Control (GCC) [8], e.g., is a congestion controller that is specifically designed for multi-media applications, using both delay and loss as rate adjustment signals. While analyses find an aggressive behavior in certain cases, the algorithm is generally considered to be TCP-friendly [8], [9]. Furthermore, it is part of the Web Real Time Communications (WebRTC) framework which provides a standardized interface for video conferencing via a browser API and a portable C++ implementation which may be compiled to Desktop, Android, and iOS platforms. Despite these efforts toward a standardized use of video conferencing, applications often use their own CC implementations. As these algorithms are mostly closed-source, their behavior is a frequent focus of research.

Related work on video conferencing. Driven by potential risks for the Internet's stability, researchers frequently analyze and assess the CC behavior of video conferencing applications. While studies generally find that common applications, such as Google Meet, Microsoft Teams, Skype, and Zoom, do indeed adapt their bandwidth to congestion [2], [3], [13], researchers note that they do not achieve consistent performance across the board. Zoom, e.g., uses bandwidth shares far off 50% [2], [3], which is often considered an unfair behavior. Besides the congestion responsiveness, the perceived performance of the applications, i.e., the QoE for their users, is also a subject of research and has been studied for Meet, Teams, Webex, and Zoom [2]–[4], [6]. The researchers find nuanced differences between the applications, e.g., that they provide different video qualities at the same bandwidth, and that the QoE depends on many more factors than CC alone. Hence, for these applications *specializing* in video conferencing, there does not seem to be a one-size-fits-all rule.

Instant messaging-based video call applications (IMVCAs). In recent years, a growing number of instant messengers has started to provide video conferencing as an *add-on*. In contrast to the special-purpose video conferencing applications, these video calls are typically embedded into existing chat applications and designed for a drastically different use case, e.g., likely optimizing for minimal cost to the app provider as these apps are mostly operated in a free-to-the-user model. Some of these applications are known to use the mentioned standardized methods and protocols as, e.g., Signal has publicly stated to use WebRTC and an implementation of GCC [7]. Analyzing the Telegram codebase [14] further reveals that after considering the use of SCReAM [15], [16], Telegram seems to have opted for WebRTC and GCC, too. However, the actual performance critically depends on the parameterization of these frameworks. Additionally, for other applications, such as WhatsApp, the concrete algorithms in use are unknown. Given that IMVCAs have not been studied in recent years with the most recent work dating back to 2014 [17], the actual behavior of today's IMVCAs is thus largely unknown.

Takeaway. *Research has provided a good overview on the current state of video conferencing. However, the detailed recent works focus on applications that* specialize *in providing video conferencing services. In contrast, the state of today's IMVCAs, such as Signal, Telegram, and WhatsApp, is largely unknown. Hence, we focus on these applications and study their CC behavior as well as the QoE provided to their users.*

III. Methodology

There is a large collection of IMVCAs and a number of different scenarios that can be used for studying them. For our work, we select three popular IMVCAs and opt to study them in a controlled testbed setup.

IMVCAs. In our study, we focus on Signal, Telegram, and WhatsApp. These apps show high popularity [18] on the one hand, but also vary in what is known about their video call implementations: Signal has announced [7] its use of WebRTC and GCC and Telegram's official clients [14] seemingly also use WebRTC and GCC, while there are no statements or hints on the CC used in WhatsApp. The selected IMVCAs thus allow us to study the performance of standardized protocols and their real-world parameterization while also discovering the behavior of unknown algorithms.

General approach. We install the official IMVCA clients on virtualized Android smartphones, let them start video calls while enforcing competition against TCP flows governed by different CCAs. We then derive network and QoE metrics to provide a multi-view assessment of their behavior.

A. Testbed setup

Research on CC is typically conducted in isolated and controlled testbeds. This methodology is also generally applicable for our work as the IMVCA clients use peer-to-peer connections for their video calls. However, they need to connect to their vendor backends for signaling, authentication, and the actual call establishment. We thus devise a testbed which permits Internet connectivity for signaling, while still allowing for shaping and parameterizing local video call traffic.

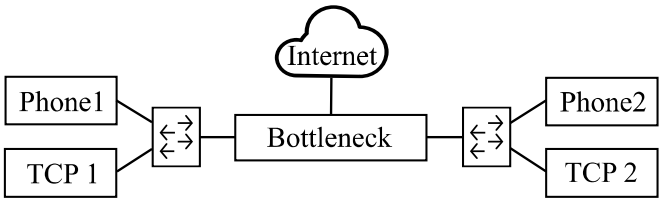

Fig. 1: Testbed representing a classic dumbbell topology connected to the Internet for backend communication/signaling.

Topology. We setup our testbed in a dumbbell topology as shown in Fig. 1. We place one emulated phone and one TCP endpoint on each side of a dedicated bottleneck machine, posing as an L2 switch and emulating the desired network conditions. The phones create the actual video call while the co-located TCP instances create cross-traffic. We further connect the bottleneck to the Internet so that the phones can signal/establish the video calls via their vendor backends.

Phone virtualization. For our study, we require replayable video sources and a reliable network connection. While wired connections to smartphones are possible via USB, multiplexing a video source together with a connection for charging and automation is prone to errors. Thus, similar to related work [3], we instead use virtualized phones leveraging Google's Android Emulator 30.9.5 running an x86 build of *Android 12*. We configure them to use TAP interfaces which directly map to dedicated Ethernet NICs of our testbed. We further install the official client software for each of our IMVCAs[1] and automate call establishment and the actual video call via Android's Monkeyrunner control framework [19]. Moreover, we map a virtual v4l2 loopback camera to the emulator to inject a looped, uncompressed test video[2] as well as test audio[3] into the emulated Android phone. With these features in place, we can automatically establish emulated video calls.

Crosstraffic and network characteristics. For crosstraffic, we deploy the TCP senders on dedicated Linux machines running an out-of-tree Linux 5.13.12 kernel [20] which provides the latest version of BBRv2. This crosstraffic then competes at our bottleneck which we configure using Linux's TC with its Token Bucket Filter (TBF) and *netem* module to shape different network characteristics. The TBF is configured to allow burst sizes of only one MTU (1500 Byte) on the *egress* queues of the bottleneck to allow for near-isochronous traffic shaping and its queue is sized via a bfifo queue. We shape round-trip times (RTTs) on the *ingress* queues using the netem module via intermediate functional blocks to avoid interference with the TBF. For detailed queuing and packet-based statistics, we deploy a custom eBPF probe which keeps track of the packets and their queuing through the bottleneck.

[1]Signal: v. 6.11.7, Telegram: v. 9.4.8, WhatsApp: v. 2.23.4.77
[2]https://media.xiph.org/video/derf/y4m/KristenAndSara_1280x720_60.y4m
[3]https://www2.cs.uic.edu/~i101/SoundFiles/preamble.wav

B. Study Targets and Parameterization

We design our study to investigate two distinct aspects of IMVCAs: their CC behavior and the resulting QoE provided to their users. For both aspects, we draw inspiration from related works discussed in Sec. II to define investigation scenarios which we outline in more detail in the following.

Target 1 - CC behavior. To assess the CC behavior of the IMVCAs, we let them compete against known TCP traffic, characterizing the performance based on the achieved throughput. We present the results of this part of our study in Sec. IV.

Target 2 - Impact on QoE. For video conferencing, the throughput might not always be the most critical metric. We argue that perceived quality is as important as the achievable bandwidth. We thus also study the resulting QoE as provided by the different IMVCAs. We present these results in Sec. V.

Parameterization 1 - Studied CCAs. For both study targets, we investigate the impact of several configuration parameters. For the CC assessment, we choose Cubic as it is the default CCA for most operating systems. We also use BBRv2 as it represents the newest class of CCAs which factor in delay while not succumbing to loss-based CC.

Parameterization 2 - Impact of active queue management (AQM). Previous studies [2] also investigate the effect of AQM mechanisms as there is an increased push toward deploying them. We thus also study the impact of AQM at the example of Controlled Delay (CoDel) and its fair-queuing variant (FQ_CoDel). In these settings, we put the AQM in front of the TBF, replacing the drop-tail queue.

IV. CC Behavior Evaluation

We start our analysis by studying the CC behavior of the three selected IMVCAs. For this, we first conduct baseline experiments to find out (i) how much bandwidth the IMVCAs claim when there is no competition, and (ii) how they react when provided with less than these limits (Sec. IV-A). We then study the interaction of the IMVCAs with TCP (Sec. IV-B).

Note that all surveyed apps use peer-to-peer communication on top of UDP for the actual call, while the backend communication is used for signaling and as a fallback if the peer-to-peer connection cannot be established. Hence, to allow for a controlled evaluation, we ensure that the peer-to-peer connections are established and omit any Internet-bound traffic from analysis, solely targeting the peer-to-peer traffic as it is contained in our testbed and, thus, fully controlled.

A. Baseline experiments

Our first series of baseline experiments studies two-party video calls without any crosstraffic in order to characterize basic properties of the IMVCAs. Furthermore, we validated our testbed configuration by verifying that all TCP variants can utilize the testbed to its capacity. In the following, we first determine the bandwidth demand of every IMVCA to then assess their behavior when falling below these limits.

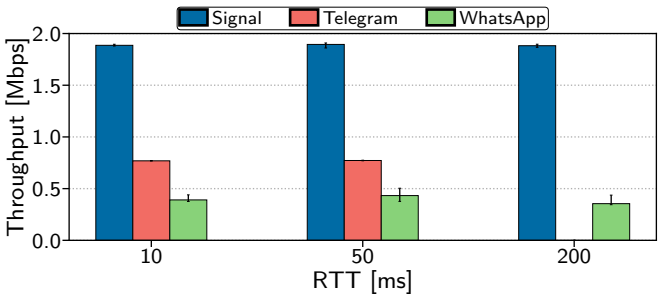

Fig. 2: Median bandwidth demand of the IMVCAs with lower/upper quartiles at different RTTs on a 5 Mbps link with a $2 \times BDP$ queue. Telegram did not establish calls at 200 ms.

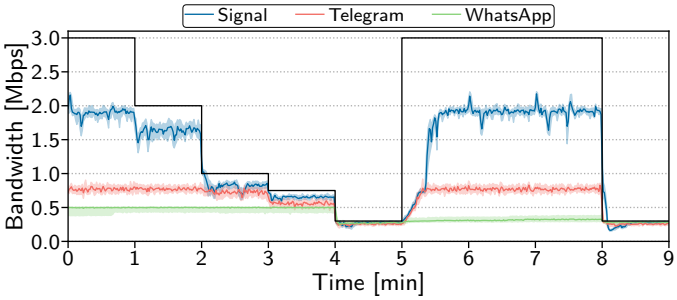

Fig. 3: Median *bandwidth profiles* of the IMVCAs for an RTT of 50 ms and a queue size of 10 kB with upper/lower quartiles as error bands.

1) Determining Bandwidth Demand: To first get a rough overview of the tested apps in isolation, we start by determining their bandwidth demands. For this, we set the bottleneck bandwidth to 5 Mbps, and configure RTTs of 10, 50, and 200 ms and a queue size of $2 \times BDP$. While the actually selected bandwidth is irrelevant, as it is chosen high enough to let all IMVCAs fully satisfy their needs, we found that under- or oversizing queues can cause call establishments to fail. In each scenario, we then run a video call for 2 min and measure the peer-to-peer throughput after a warmup phase of 1 min. We average this throughput over the remaining 1 min and repeat each scenario 30 times.

Results. Fig. 2 shows the median bandwidth demands of the three IMVCAs across the 30 iterations together with the lower and upper quartiles. All demands are relatively independent of the RTT. We only note that Telegram fails to establish the video calls at 200 ms RTT and WhatsApp's throughput varies slightly between the evaluated RTTs. For simplicity, we will focus on 50 ms in the following. Our results show that each IMVCA has distinct bandwidth requirements: Whatsapp uses ∼0.45 Mbps, Telegram ∼0.75 Mbps, and Signal ∼1.9 Mbps in the median case. Given these demands, the CC of the IMVCAs will only have to react in scenarios with a bottleneck bandwidth below their demands. We investigate the IMVCA behavior in such settings next.

2) Reaction to Changing Bandwidths: To determine the reaction of the apps to changing bandwidths, we start a video call and change the available bandwidth in 1 min intervals from as high as 3 Mbps to as low as 0.3 Mbps. We fix the RTT at 50 ms and set the buffer size to 10 kB, i.e., $\frac{1}{2} \times BDP$ for the highest and $5 \times BDP$ for the smallest bandwidth, both being close to our prior $2 \times BDP$ configuration. The applications manage to attain their demand bandwidth with this buffer configuration. Again, we repeat the experiments 30 times and determine the *bandwidth profiles* for the IMVCAs by first temporally aligning the throughput measurements of the individual iterations. We then separate the time curves into 1 s intervals for which we compute the median and quartiles over all iterations.

Results. Fig. 3 shows the *bandwidth profiles* for the three IMVCAs over the entire measurement duration, illustrating the median by the solid lines while we indicate the lower/upper quartiles using the lighter error bands. The throughput in the initial phase with a bandwidth of 3 Mbps visually confirms the demands observed in Sec. IV-A1: Signal has the highest demand while WhatsApp has the lowest. At the same time, WhatsApp offers the most stable behavior while Signal tends to fluctuate more than the other two IMVCAs.

When slowly decreasing the available bandwidth, we can observe corresponding reactions by the apps as they adjust their sending rates as soon as the bandwidth has fallen below their individual demands. For Signal, although the average is at about 1.9 Mbps, we can already see a reaction at 2 Mbps due to smaller variations and peaks in between on which Signal reacts directly. It seems to integrate a small buffer margin. However, it seeks a slightly higher bandwidth than Telegram for a bottleneck bandwidth of 0.75 Mbps (3 min to 4 min) while WhatsApp achieves a higher throughput than the other two applications at the lowest bandwidth of 0.3 Mbps (4 min to 5 min). We also notice that 25% of WhatsApp calls experience more than 1% packet loss at 0.3 Mbps (not shown). The other apps manage to maintain any rate without incurring any persistent loss.

Upon abruptly increasing the available bandwidth back to the initial value of 3 Mbps, Signal and Telegram quickly reclaim their demands. Interestingly, their ramp-up phases are nearly equal, hinting at the joint usage of GCC. In contrast, WhatsApp is very careful and only slowly increases its rate, failing to reach its demand in the allotted time frame of 3 min. WhatsApp recovers faster (within one minute) if the buffer is sized at $10 \times BDP$ for the highest bandwidth (not shown).

In a final step, we drastically decrease the bandwidth to the minimum value of 0.3 Mbps. WhatsApp shows a rather smooth transition while Signal significantly undershoots the target rate, although it recovers in a time similar to the previously seen ramp-up phase.

Takeaway. *All studied IMVCAs have distinct median bandwidths and unique reactions to bandwidth changes although some should use the same CC. Overall, however, all IMVCAs show suitable reactions to changing bandwidths.*

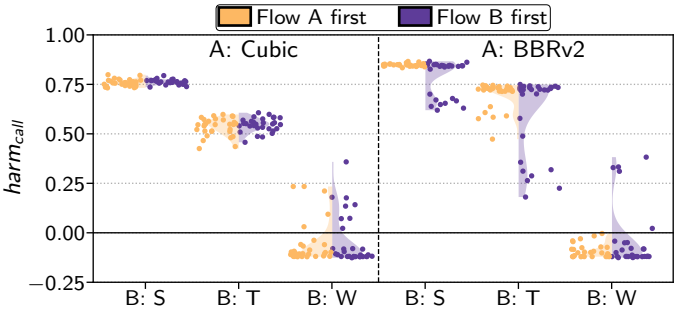

Fig. 4: Bandwidth harm for IMVCA flows competing at a $2 \times BDP$ queue against TCP for an RTT of 50 ms. Bandwidth limited to 5 Mbps

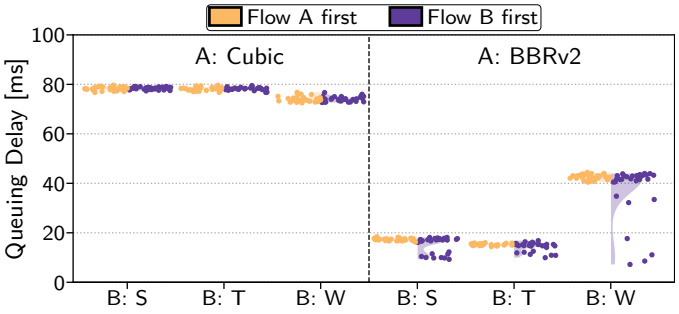

Fig. 5: Queuing Delay for IMVCA flows competing at a 2BDP queue against TCP for an RTT of 50 ms. Bandwidth limited to 5 Mbps

B. IMVCA vs. TCP

The previous Sec. IV-A has shown (i) that and (ii) how the studied IMVCAs adjust their sending rates solely based on the available bandwidth. However, these observations are made in isolated settings. To study the IMVCAs in a more realistic environment, we next assess their interaction with a TCP flow across different scenarios. In the following, we first describe our assessment metrics. We then start with testing the reactions of the IMVCAs to a TCP flow added to a link with sufficient bandwidth for both streams to coexist. Thereafter, we switch to a scenario with strong competition for which we configure the link to barely support the demand of the IMVCAs alone.

Assessing interaction. Assessing the CC interaction of multiple flows is often done using the notion of flow-rate fairness, i.e., two flows behave fairly if they share the bandwidth equally. Yet, whether such a metric is sensible is a frequent focus of discussions and research, as it ignores the actual characteristics of novel services. As a solution, Ware et al. [21] propose to focus on harm instead of the pure bandwidth share, taking the actual demands of the apps into account, e.g., w.r.t. bandwidth or delay. We follow this proposal and apply harm for assessing the interaction, leveraging the identified standalone rates of our IMVCAs (cf. Sec. IV-A1) and the fact that TCP generally strives for a maximum rate.

Harm. Harm [21] denotes the applied changes for bandwidth, delay, or jitter w.r.t. the demands of an application. For example, gaining bandwidth beyond equal shares is acceptable if the demands of other flows are retained. Harm inflicted on a flow b by a flow a is defined as

$$harm_b = \begin{cases} \frac{demand_b - performance_b(a)}{demand_b} & \text{higher perf. better} \\ \frac{performance_b(a) - demand_b}{performance_b(a)} & \text{lower perf. better} \end{cases}$$

where $demand_b$ reflects the demand of flow b and $performance_b(a)$ yields the performance metric of flow b subject to interaction with flow b. Note that $harm_b$ lies in the interval $[0, 1]$ where 0 indicates no harm and 1 represents that the entire demand of flow b is suppressed.

1) Coexistence Scenario: In a first setting, we investigate the interaction between the IMVCAs and the two TCP CCAs Cubic and BBRv2 on a 50 ms RTT, 5 Mbps link, i.e., a link exceeding the IMVCA demands by more than 100 % such that both streams could easily coexist. We compare the behavior using the harm metric for the calls for which we set the demand to the previously investigated median demands. As such, the values are in relation to the previous demands.

Impacted coexistence at a drop-tail queue. Fig. 4 shows the bandwidth harm inflicted by the TCP CCAs on our apps (S: Signal, T: Telegram, W: WhatsApp) when sharing a $2 \times BDP$ drop-tail queue. For each scenario, we further study two versions: one, where the TCP flow (flow A) starts first, and one, where the IMVCA flow (flow B) starts first. We can see that although the 5 Mbps link allows for more than 3 Mbps to be used by TCP, still the flow-rates of Signal and Telegram are impacted. Signal yields ∼75 % of its 1.9 Mbps bandwidth demand to Cubic and ∼85 % to BBRv2 if TCP started first. If Signal started first competing against BBRv2 it is able to sometimes achieve a harm of ∼65 %. Telegram, starting with a lower median flow-rate of 0.75 Mbps, sometimes achieves harm values near 25 %, but mostly yields 55 % to 70 % of its bandwidth. We attribute this effect to loss and rising queuing delays, which cause Signal and Telegram to throttle their rates. Both apps fall on a sustained flow-rate below that of WhatsApp's demand. In contrast, WhatsApp does not yield bandwidth although it also has a much lower demand to defend. It even reaches a negative harm value, as its median bandwidth can be exceeded.

With regards to delay, Fig. 5 shows the queuing delays seen at the queues during our test. Interestingly, Signal and Telegram manage to keep the queuing delay low when coexisting with BBRv2. This suggests that the delay-based CC components in BBRv2 and the IMVCAs play together particularly well. In contrast, for WhatsApp, we can see extensive delay in coexistence with BBRv2. For Cubic, the well-known filled queue behavior is noticeable for all three IMVCAs although the queuing delay seems to be slightly smaller for WhatsApp.

Coexistence with AQM. Given the detrimental impact of TCP on the IMVCAs, which we partly attribute to the rising queuing delays, we next experiment with ways to counteract these problems. AQM is one commonly used mechanism to reduce the observed queuing delays which is why we apply the CoDel [22] AQM to investigate its effect on the IMVCAs.

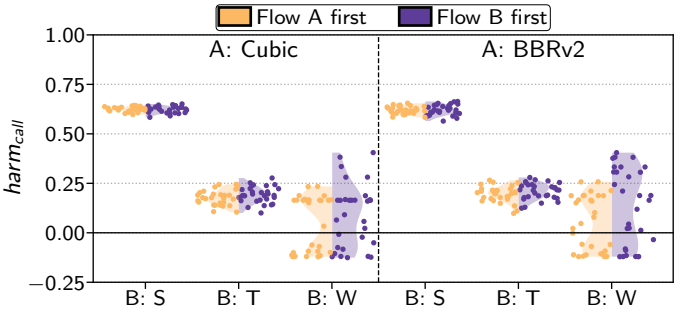

Fig. 6: Bandwidth harm for IMVCA flows competing at a CoDel-enabled queue against TCP for an RTT of 50 ms. Bandwidth limited to 5 Mbps

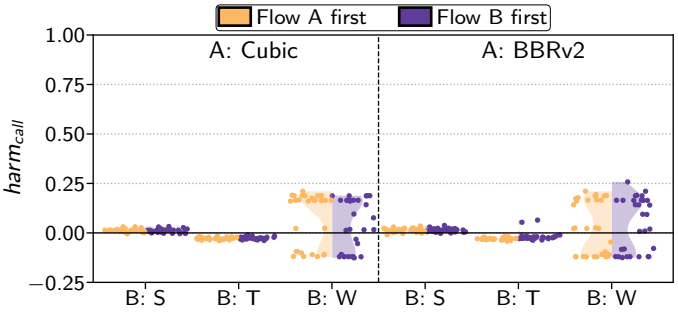

Fig. 7: Bandwidth harm for IMVCA flows competing at a FQ_CoDel-enabled queue against TCP for an RTT of 50 ms. Bandwidth limited to 5 Mbps

As expected, CoDel reduces the delay to less than 20 ms throughout all cases (thus not shown). However, as shown in Fig. 6, it cannot fully even out the bandwidth harm imposed by TCP upon the IMVCAs. For Signal and Telegram, we can see that they yield much less bandwidth to Cubic and BBRv2 compared to the drop-tail queue setting: instead of 75 % and 85 % harm for Signal, we can now observe around 62 % independent of the flow order and Telegram's harm is reduced to around 20 %-25 %. Looking at WhatsApp, the harm values now spread around 0 %, hinting at WhatsApp relaxing its bandwidth defense.

Coexistence with fair-queuing AQM. While CoDel improves the queuing delays as expected, it does not fully relieve the bandwidth harm for all IMVCAs. Hence, we also study the impact of the fair-queuing variant of CoDel. As can be seen in Fig. 7, FQ_CoDel helps Signal and Telegram to achieve their demands with harm values very close to 0. For WhatsApp, there is still a spread around 0 % as for pure CoDel. Yet, the spread is bimodal with both distributions being very sharp. In total, we can see that fair-queuing clearly enforces equal bandwidths and allows the apps to reachieve their demands.

Takeaway. *TCP inflicts serious bandwidth harm on the IMVCAs, even when the link leaves enough headroom for equal bandwidth sharing. In particular, Signal and Telegram yield bandwidth very quickly while WhatsApp is able to retain its lower bandwidth demand. Only using fair-queuing helps the IMVCAs in protecting their demands against TCP.*

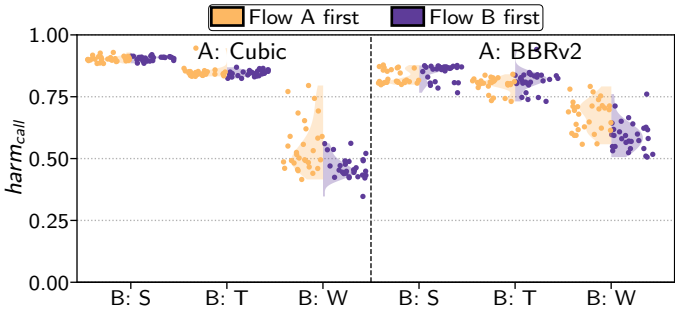

Fig. 8: Bandwidth harm for IMVCA flows competing at a $2 \times BDP$ queue against TCP for an RTT of 50 ms. Bandwidth per setting configured to the respective demand of the IMVCA.

2) Enforced Competition Scenario: The previous scenario was designed to provide sufficient bandwidth headroom for all IMVCAs so that they could, theoretically, coexist with TCP. Yet, there were very different reactions by the applications and WhatsApp showed a particularly interesting behavior as it did not yield bandwidth to TCP. However, WhatsApp also has the lowest demand, one possible reason for the observed behavior. Hence, to even the playing field and enforce competition, even for WhatsApp, we next scale the bottleneck bandwidth to the demands of the IMVCAs. This allows us to study the behavior and reactions of the apps under extreme conditions.

Drop-tail queue. We first study the behavior at a drop-tail queue with a queue size of $2 \times BDP$ and set the bottleneck bandwidth to the IMVCA demands, i.e., 1.9 Mbps for Signal, 0.75 Mbps for Telegram, and 0.45 Mbps for WhatsApp.

Fig. 8 shows the observed harm values. 0.5 denotes flow-rate equality, i.e., that TCP and IMVCA share the bandwidth equally. As such, values of 0.5 could be seen as desirable, although clearly the apps would benefit from higher bandwidths. We manually verified that each app is able to reduce its sending rate to half its demand, i.e., achieve flow-rate-equality.

A first observation is that the reactions of Signal and Telegram now only barely differ between Cubic and BBRv2 as TCP dominates the link in both cases. In particular, Cubic causes harm of 80 % and above, compared to 55-75 % in Fig. 4, while BBRv2 reaches similar levels, described by a partly bimodal distribution.

For WhatsApp, the reaction again looks different. Instead of harm values above 80 %, it reaches around flow-rate equality for Cubic. However, the variance of the results is significantly higher than for Signal or Telegram, beyond the difference in demands which of course also impacts scaling. For BBRv2, the harm is higher at about 60 % for WhatsApp being incumbent and at about 65 % with TCP being incumbent. For both scenarios, we see high loss rates of the WhatsApp call at about 9-12 % (not shown).

Takeaway. *Signal and Telegram back off further when reducing the link's headroom. WhatsApp, on the other hand, presents a much more diverse behavior defending its share.*

Impact of fair-queuing AQM. Given the dramatic reduction

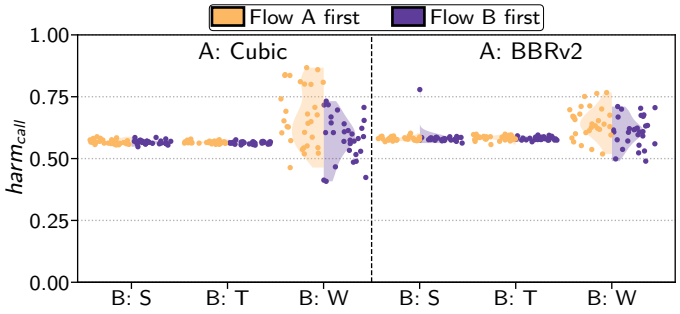

Fig. 9: Bandwidth harm for IMVCA flows competing at a FQ_CoDel-enabled queue against TCP for an RTT of 50 ms. Bandwidth per setting configured to the respective demand of the IMVCA.

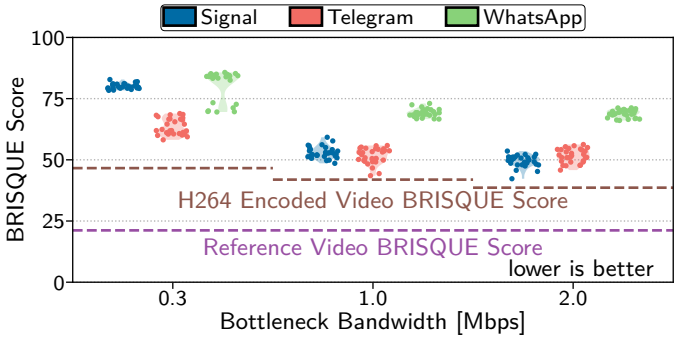

Fig. 10: BRISQUE scores of the investigated IMVCAs at different bottleneck bandwidths with a $2 \times BDP$ queue and 50 ms RTT. For reference, the input video's score and its score when encoded at the bottleneck bandwidth are added.

in sending rate for Signal and Telegram, and the unstable behavior of WhatsApp, we again study the effect of AQM. However, as seen in Sec. IV-B1, a pure CoDel only has minor effects on the bandwidth harm. We have confirmed these observations also for the enforced competition scenario. In the following, we thus directly focus on FQ_CoDel.

As can be seen in Fig. 9, using FQ_CoDel strongly improves harm as all harm values are now close to 0.5. Interestingly, all IMVCAs center slightly above the flow-rate equality mark. Also, WhatsApp still achieves a very scattered distribution of harm values. Even with fair-queuing AQM, its results are relatively indeterministic, often achieving harm values above those of Signal and Telegram, but also values below.

Takeaway. *In general, we can see that AQM is able to improve flow-rate stability also for links with no headroom. Yet, fair-queuing is required to significantly improve bandwidth equality for Signal and Telegram, while WhatsApp's results are very scattered. The impact on QoE is unclear.*

V. QoE Evaluation

The first part of our analysis focuses on the CC behavior of the IMVCAs and the corresponding impact of AQM. While we could discover significantly different bandwidth management behavior of the apps, the effects of these differences on the user's perception are still unclear. Thus, we next focus on shedding light on the QoE impact of our previous findings. For this, we repeat our measurements, this time also recording the screen of the virtual phone receiving the video stream to then assess the visual quality of the call.

A. QoE Assessment Methodology

In order to objectively gauge the QoE, we use an image quality assessment (IQA) model that exclusively focuses on the *video* quality. Prior works in video conferencing QoE either exploited debug outputs of the surveyed applications [3] or employed frame-by-frame analysis of screen-grabbed calls [4].

We assess the video QoE by losslessly screen-grabbing our virtual phones and extracting the last 10 frames of each video call. We then apply the no-reference (NR) IQA model BRISQUE [23], and calculate the mean of the resulting quality scores to obtain one value for each call.

As the screen capturing is resource-intensive, we separate these measurements from our prior measurements to rule out interference. However, we do not find any significant differences in the bandwidth sharing behavior compared to our previous results, indicating that the additional screen capture has a negligible effect.

Using an NR QoE model. We decide against a full-reference (FR) model, such as the structural similarity index measure (SSIM), as there are inherent spatial and temporal distortions between the webcam input and the output on the receiving phone. These distortions strongly affect FR models but are barely noticeable in subjective QoE. For example, the IMVCAs might re-encode the video at a different frame rate (25fps vs 30fps) than the reference. Furthermore, our setup of an emulated phone, emulated camera, and screen-grabber is not synchronized, which creates additional temporal misalignments. Capturing the exact reference frames subject to the video call encoding is thus infeasible. Yet, FR models expect to compare the very same frame such that rating the quality via such models would be biased in our setup.

Hence, we rely on an NR model: BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) [23]. BRISQUE measures the distortion of a digital image by comparing the distribution of luminance between *natural* images / images taken by a camera and its input image. These distributions are then fed into a machine learning model trained on human ratings to yield quality scores between 0 (best) and 100 (worst). Analyses showed that the scores are correlated with the human perception of image distortion and as statistically significant as the FR model SSIM [23]. Thus, our analyses only require the IMVCA output to rate and assess the influence of bandwidth and congestion on QoE.

B. Results

We begin by first measuring the baseline performance of the IMVCAs without any competition. When then add a

competing TCP flow and finally compare the performance when enabling CoDel and FQ_CoDel.

1) Baseline Results: Fig. 10 shows the BRISQUE scores of the IMVCAs for different bottleneck bandwidths (0.3 Mbps, 1 Mbps, and 2 Mbps) and a $2 \times BDP$ queue. CoDel, FQ_CoDel and a $10 \times BDP$ drop-tail queue yield similar scores (not shown). The plot further shows the BRISQUE scores of our reference input video (purple line) and those that can be achieved by OpenH264 using the bottleneck bandwidth as encoding rate (brown line). Note that lower BRISQUE scores are better.

Throughout all measurements, we can see that the theoretical BRISQUE minimum of the reference video is never achieved. This is expected as the video bandwidth is too high for the investigated bottleneck bandwidths and the video, thus, has to be encoded. However, the theoretical H264 BRISQUE scores are not achieved either. We attribute this observation to margins and FEC overheads that have to be taken into account, too. Yet, Signal and Telegram are surprisingly close, both achieving BRISQUE scores of around 50 for 1 Mbps and 2 Mbps, even though Signal uses more than twice the bandwidth of Telegram.

In contrast, the BRISQUE scores of WhatsApp are worse at around 68 as it uses a much smaller bandwidth. However, also for 0.3 Mbps, WhatsApp is not close to the H264 reference line. We conclude that WhatsApp encodes the video at a much lower rate internally and employs several overhead-adding measures, such as FEC. This can also be seen in the fact that, even at 0.3 Mbps, WhatsApp often achieves the same BRISQUE scores as before. In certain cases, however, it cannot keep up and reduces the quality, leading to a bimodal distribution of the BRISQUE scores. Interestingly, Signal, which previously achieved BRISQUE scores on-par with Telegram, now also reduces its quality significantly. Previously, we always saw a very similar reaction between Signal and Telegram but seemingly, at this point Signal, uses a different parameterization or implementation.

Takeaway. *WhatsApp seemingly uses lower video encoding rates than Signal and Telegram. Signal and Telegram further show a different behavior in the very low bandwidth region, a surprising finding given both rely on WebRTC and GCC.*

2) QoE when competing against TCP: The different IMVCAs use different baseline demands which equally reflect in their delivered video quality. However, we have previously seen that the IMVCAs also have a different behavior when competing with TCP. Hence, we next investigate the impact of this competition on the IMVCA video quality. For this, we reuse the configurations of Sec. IV-B2, i.e., we configure bottleneck bandwidths equal to the IMVCA demands and add Cubic and BBRv2 TCP flows to not only investigate the bandwidth harm, but now also the video quality interference.

Drop-tail queue. Fig. 11 shows the BRISQUE scores for a $2 \times BDP$ queue and a 50 ms RTT. Across the board, we can see a severe detriment in video quality. While our baseline scenario showed BRISQUE scores from 50 to 75 and slightly above, now the majority of measurements show values above

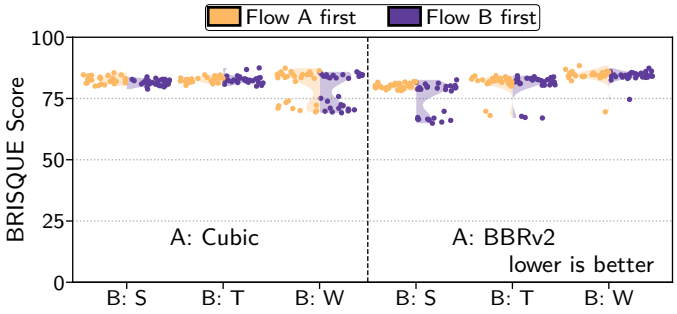

Fig. 11: Brisque score for IMVCA flows competing at a $2 \times BDP$ queue against TCP for an RTT of 50 ms. Bottleneck bandwidth per setting configured to the respective demand of the IMVCA.

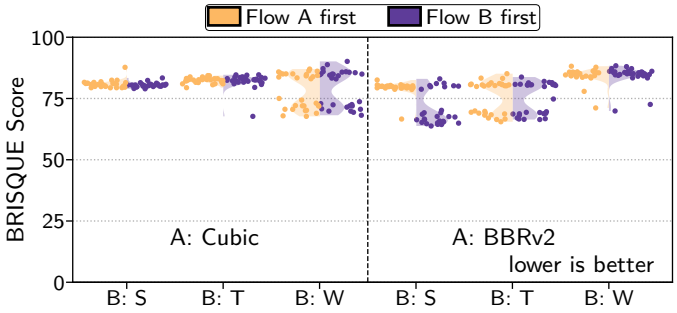

Fig. 12: Brisque score for IMVCA flows competing at a CoDel-enabled queue against TCP for an RTT of 50 ms. Bottleneck bandwidth per setting configured to the respective demand of the IMVCA.

75. The bimodal behavior of WhatsApp can again be seen when competing against Cubic, allowing it to yield a better BRISQUE score than the other apps in some cases. Comparing the observations for harm (cf. Sec. IV-B) to the BRISQUE values, we can see that WhatsApp's more aggressive defense seems to help. However, when a BBRv2 flow joins Signal and Telegram flows, we can also see the bimodal distribution and improvements which were also visible in harm. Overall, harm and video quality seem to correlate. With respect to the bimodality we notice a correlation to the bimodality in harm. A higher bandwidth attained is correlated with a better BRISQUE, even though there is a transient area in which both modes of BRISQUE are observed at the same throughput.

Impact of CoDel. The BRISQUE results for enabling CoDel can be seen in Fig. 12. The bimodal improvements for harm with BBRv2 now also reflect in the BRISQUE values, Telegram and Signal achieve better BRISQUE values more often. Yet, for Cubic, all apps barely benefit. As such, we also test the outcomes on QoE when enabling fair-queuing.

Impact of FQ_CoDel. Fig. 13 shows the BRISQUE scores when deploying FQ_CoDel. Only when adding fair-queuing we saw positive effects on harm and we now also see this effect for BRISQUE. Signal and Telegram now achieve BRISQUE scores of ∼60, improving the video quality significantly. How-

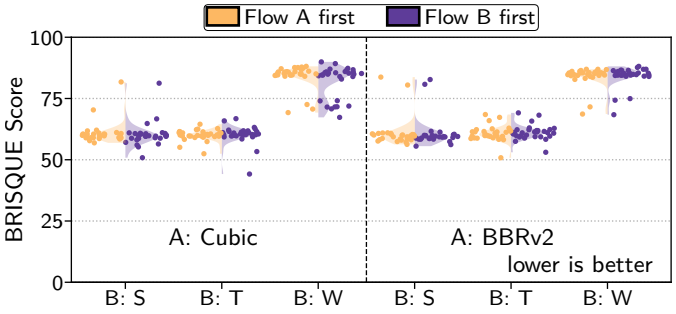

Fig. 13: Brisque score for IMVCA flows competing at a FQ_CoDel-enabled queue against TCP for an RTT of 50 ms. Bottleneck bandwidth per setting configured to the respective demand of the IMVCA.

ever, Signal does not achieve its theoretical baseline BRISQUE score of 50 for half of its demand. WhatsApp, in contrast, cannot improve.

Takeaway. *The QoE of the IMVCAs is severely impacted by up to more than 25%-points when competing against TCP. While CoDel reduced delay before, a visual improvement clearly requires FQ_CoDel.*

VI. Implications for Video Conferencing

Our findings in Sec. IV and Sec. V provide a first overview on the current state of instant messaging-based video conferencing. In the following, we discuss our main findings as well as areas and limitations which we identify as most relevant for extending the overview.

Harm and QoE correlate. Our two-fold assessment of the IMVCA behavior reveals that there is a correlation between harm inflicted by TCP and the corresponding QoE provided to the user. In particular, the decreased bandwidths directly translate to decreased QoE, with plots of the bandwidth harm and the BRISQUE scores showing very similar behavior. Thus, while related works point to a multi-dimensional dependency of video conferencing quality, our findings reinforce that bandwidth plays a major role in the achievable QoE.

WebRTC and GCC. Signal and Telegram both use the WebRTC framework. Consequently, they share many traits and show a similar behavior. However, we also note significant differences, especially in low bandwidth regions. Additionally, both IMVCAs excessively yield their bandwidth when competing against TCP. Hence, we identify the need for better understanding the behavior of WebRTC in low bandwidth scenarios and for improving its performance against TCP.

Robustness vs. performance. WhatsApp seemingly uses a robust encoding that ensures a consistent QoE, even for low bottleneck bandwidths. Additionally, it is quick to react to decreased bandwidths while it is slow to probe for new bandwidth. As such, WhatsApp stands in contrast to Signal and Telegram which seem to be better equipped for higher bandwidths at the cost of excessively yielding bandwidth. Consequently, WhatsApp might be better suited for harsher network conditions accepting loss and delay while the other IMVCAs could rather target modern networks. Thinking current standardization efforts further, deciding on how to weigh the trade-off of robustness and performance will be crucial.

A. Limitations and Possible Extensions

Extending our overview is possible into several dimensions due to inherent limits of our method.

Testbed setup. We use an Ethernet-based, fully controlled testbed to facilitate comparisons with related works, emulating the wireless access using the Android emulator. In reality, however, we expect different access topologies, such as 802.11 or 4G/5G, to influence the behavior of the IMVCAs, especially under congestion. Similarly, broadening the investigated network settings could provide additional input for the aforementioned trade-off considerations.

QoE assessment. Using BRISQUE allows for an effective, yet limited assessment of the call QoE. In particular, we focus on a sample of the individual frames making up the video call, but do not analyze the call audio nor the temporal aspects of video. Taking these into account might further widen the trade-off considerations.

Multi-party calls. Similar to related works, our investigation focuses on two-party calls. However, we expect group calls to show further competition issues. For example, group calls typically use a selective forwarding unit deployed at a central location as a video stream broker. This additional indirection could cause longer paths and might introduce additional bottlenecks, effects that we do not consider in our assessment.

VII. Conclusion

Research on video conferencing has mostly focused on applications that *specialize* in providing video conferencing services, painting a detailed picture of their congestion control behavior. However, these works neglect that many instant messengers now also provide video call functionality as an *add-on*. In this paper, we thus study the congestion control traits of and QoE provided by three popular instant messaging-based video call applications (IMVCAs): Signal, Telegram, and WhatsApp. Using a controlled testbed with emulated Android phones, we evaluate these applications in isolation and when competing with TCP in different network setups.

Our results indicate that the studied IMVCAs do adjust their bandwidth in times of congestion, i.e., they do not pose a threat to Internet stability. However, the IMVCAs are severely disadvantaged when competing against TCP, giving up more than half or even three quarters of their bandwidth in the case of Signal and Telegram. In contrast, WhatsApp maintains a lower baseline than the other IMVCAs and defends it more decidedly, although it also suffers against TCP. The bandwidth disadvantages further translate to decreased visual call quality. In particular, we find that the QoE, assessed using the visual quality indicator BRISQUE, correlates with the bandwidth harm inflicted by the TCP flows. Overall, we believe that the TCP-friendliness of the studied IMVCAs should not be reversed but tuned further to allow high-quality video calls even when competing against TCP.

Acknowledgments

This work has been funded by the German Research Foundation DFG under Grant No. WE 2935/20-1 (LEGATO). We thank the anonymous reviewers for their valuable comments.

References

[1] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic," in *Proceedings of the 2020 ACM Internet Measurement Conference (IMC)*, 2020. [Online]. Available: https://doi.org/10.1145/3419394.3423658

[2] C. Sander, I. Kunze, K. Wehrle, and J. Rüth, "Video Conferencing and Flow-Rate Fairness: A First Look at Zoom and the Impact of Flow-Queuing AQM," in *Proceedings of the 2021 International Conference on Passive and Active Network Measurement (PAM)*, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-72582-2_1

[3] K. MacMillan, T. Mangla, J. Saxon, and N. Feamster, "Measuring the Performance and Network Utilization of Popular Video Conferencing Applications," in *Proceedings of the 2021 ACM Internet Measurement Conference (IMC)*, 2021. [Online]. Available: https://doi.org/10.1145/3487552.3487842

[4] H. Chang, M. Varvello, F. Hao, and S. Mukherjee, "Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet," in *Proceedings of the 2021 ACM Internet Measurement Conference (IMC)*, 2021. [Online]. Available: https://doi.org/10.1145/3487552.3487847

[5] A. Choi, M. Karamollahi, C. Williamson, and M. Arlitt, "Zoom Session Quality: A Network-Level View," in *Proceedings of the 2022 International Conference on Passive and Active Network Measurement (PAM)*, 2022. [Online]. Available: https://doi.org/10.1007/978-3-030-98785-5_25

[6] M. Varvello, H. Chang, and Y. Zaki, "Performance Characterization of Videoconferencing in the Wild," in *Proceedings of the 2022 ACM Internet Measurement Conference (IMC)*, 2022. [Online]. Available: https://doi.org/10.1145/3517745.3561442

[7] P. Thatcher, "How to build large-scale end-to-end encrypted group video calls," 2021. [Online]. Available: https://signal.org/blog/how-to-build-encrypted-group-calls/

[8] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and Design of the Google Congestion Control for Web Real-time Communication (WebRTC)," in *Proceedings of the 2016 ACM International Conference on Multimedia Systems (MMSys)*, 2016. [Online]. Available: https://doi.org/10.1145/2910017.2910605

[9] B. Jansen, T. Goodwin, V. Gupta, F. Kuipers, and G. Zussman, "Performance Evaluation of WebRTC-based Video Conferencing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 3, pp. 56–68, 2018. [Online]. Available: https://doi.org/10.1145/3199524.3199534

[10] L. Grote, C. Sander, I. Kunze, and K. Wehrle, "Instant Messaging Meets Video Conferencing: Studying the Performance of IM Video Calls," 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8006901

[11] "RTP Media Congestion Avoidance Techniques, IETF Working Group," 2023. [Online]. Available: https://datatracker.ietf.org/wg/rmcat/

[12] R. Jesup and Z. Sarker, "Congestion Control Requirements for Interactive Real-Time Media," IETF, RFC 8836, 2021. [Online]. Available: https://doi.org/10.17487/RFC8836

[13] L. De Cicco, S. Mascolo, and V. Palmisano, "An Experimental Investigation of the Congestion Control Used by Skype VoIP," in *Proceedings of the 2007 International Conference on Wired/Wireless Internet Communications (WWIC)*, 2007. [Online]. Available: https://doi.org/10.1007/978-3-540-72697-5_13

[14] TelegramMessenger, "Telegram Codebase," 2023. [Online]. Available: https://github.com/TelegramMessenger/tgcalls/blob/d20de8e8bb2d605918e26ef697d4b8af1d6ea3ff/tgcalls/MediaManager.cpp#L813

[15] A. S. Jagmagji, H. D. Zubaydi, and S. Molnar, "Exploration and Evaluation of Self-Clocked Rate Adaptation for Multimedia (SCReAM) Congestion Control Algorithm in 5G Networks," in *Proceedings of the 2022 International Conference on Telecommunications and Signal Processing (TSP)*, 2022. [Online]. Available: https://doi.org/10.1109/TSP55681.2022.9851382

[16] I. Johansson and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia," IETF, RFC 8298, 2017. [Online]. Available: https://doi.org/10.17487/RFC8298

[17] C. Yu, Y. Xu, B. Liu, and Y. Liu, ""Can you SEE me now?" A Measurement Study of Mobile Video Calls," in *Proceedings of the 2014 IEEE Conference on Computer Communications (INFOCOM)*, 2014. [Online]. Available: https://doi.org/10.1109/INFOCOM.2014.6848080

[18] Statista, "Most popular social networks worldwide as of January 2023," 2023, accessed: 2023-03-28. [Online]. Available: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

[19] Android Open Source project, "Monkeyrunner," 2023. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner

[20] Google, "TCP BBR v2 Alpha/Preview Release," 2023. [Online]. Available: https://github.com/google/bbr/tree/v2alpha

[21] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Beyond Jain's Fairness Index: Setting the Bar For The Deployment of Congestion Control Algorithms," in *Proceedings of the 2019 ACM Workshop on Hot Topics in Networks (HotNets)*, 2019. [Online]. Available: https://doi.org/10.1145/3365609.3365855

[22] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," IETF, RFC 8289, 2018. [Online]. Available: https://doi.org/10.17487/RFC8289

[23] A. Mittal, A. K. Moorthy, and A. C. Bovik, "No-Reference Image Quality Assessment in the Spatial Domain," *IEEE Transactions on Image Processing*, vol. 21, no. 12, pp. 4695–4708, 2012. [Online]. Available: https://doi.org/10.1109/TIP.2012.2214050