An Empirical Evaluation of the Performance of Video Conferencing Systems

Richard Bieringa*
Vrije Universiteit Amsterdam
R.Bieringa@student.vu.nl

Abijith Radhakrishnan*
Vrije Universiteit Amsterdam
A.Radhakrishnan@student.vu.nl

Tavneet Singh*
Vrije Universiteit Amsterdam
T.S.Tavneet@student.vu.nl

Sophie Vos*
Vrije Universiteit Amsterdam
S.O.Vos@student.vu.nl

Jesse Donkervliet
Vrije Universiteit Amsterdam
J.J.R.Donkervliet@vu.nl

Alexandru Iosup
Vrije Universiteit Amsterdam
A.Iosup@vu.nl

ABSTRACT

The global COVID-19 pandemic forced society to shift to remote education and work. This shift relies on various video conference systems (VCSs) such as Zoom, Microsoft Teams, and Jitsi, consequently increasing pressure on their digital service infrastructure. Although understanding the performance of these essential cloud services could lead to better designs and improved service deployments, only limited research on this topic currently exists. Addressing this problem, in this work we propose an experimental method to analyze and compare VCSs. Our method is based on real-world experiments where the client-side is controlled, and focuses on VCS resource requirements and performance. We design and implement a tool to automatically conduct these real-world experiments, and use it to compare three platforms on the client side: Zoom, Microsoft Teams, and Jitsi. Our work exposes that there are significant differences between the systems tested in terms of resource usage and performance variability, and provides evidence for a suspected memory leak in Zoom, the system widely regarded as the industry market leader.

ACM Reference Format:
Richard Bieringa, Abijith Radhakrishnan, Tavneet Singh, Sophie Vos, Jesse Donkervliet, and Alexandru Iosup. 2021. An Empirical Evaluation of the Performance of Video Conferencing Systems. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3447545.3451186

1 INTRODUCTION

Due to the COVID-19 pandemic, many organizations, businesses, and educational institutions have adopted a new online work structure [13, 16]. Meetings occur through *video conference systems (VCSs)*, which operate as cloud services that allow hundreds of millions of users to communicate online with audio and video streams.

*These authors have an equal and leading contribution. The others have proposed the research problem, helped manage the project, and contributed to writing the article.

ICPE '21 Companion, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451186

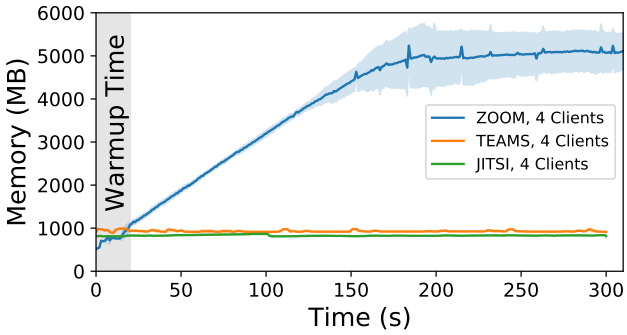

Figure 1: VCS memory-consumption over time.

However necessary, the performance of these systems is not well-understood. Existing research does not focus on today's popular systems used in professional and educational settings, or has a different focus than performance (e.g., studying societal impact [27]). Addressing these issues, in this work we aim to understand the performance of VCSs by designing a comprehensive set of *real-word experiments* and by using them to evaluate several popular VCSs.

Spurned by the Internet revolution, VCSs allow users to communicate with others online using audio and video streams, using their device's webcam and microphone. This provides support for a wide variety of use-cases, from personal calls between a handful of friends, to professional meetings with 4-10 people, to online academic conferences with hundreds of participants [20].

As many societies locked down or restricted activity during the COVID-19 pandemic, VCSs became indispensable for professional, but also for entertainment and social purposes. Their market is growing rapidly, as exemplified by the yearly increase of over 100 million users for market-leading VCS Zoom [4], which now has over 300 million daily users [26]; anecdotally, Zoom could offer superior performance and scalability [20].

Other rapidly growing services include (Microsoft) Teams [6] and the open-source system Jitsi.

It is important to understand the performance of VCS services. However, no common methods exist to evaluate the performance of VCSs. Studies of VCSs exist [19, 22, 27, 30], but we still lack a good understanding of the performance of today's popular systems. In contrast, Figure 1 depicts an exemplary result from our real-world experiments, comparing the client-side memory consumption over time for Zoom, Teams, and Jitsi. Notable in the figure is the large and increasing difference between Zoom and the other VCSs, which we identify as a performance bug and analyze in Section 5.2.2.

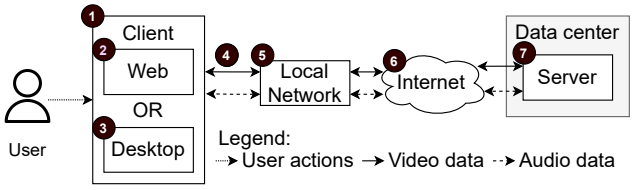

Figure 2: The VCS system model.

Obtaining meaningful performance results for VCSs is challenging. First, VCSs operate in *heterogeneous and unstable environments*. Users can interact with VCSs through various applications (e.g., Web-based and native applications) running on a variety of hardware configurations (e.g., PCs, laptops, phones), and connect to other users over the Internet, which only offers best-effort service. Second, these systems operate under *complex workloads*, which consist of a number of users, the configuration of enabled audio and video streams per user, and the quality of these streams provided by the user's webcam and microphone. Third, commercial VCSs are typically *complex yet opaque systems*: they are closed-source and do not expose their design or configuration parameters, and a large part of their operation occurs in cloud datacenters. To address these challenges, we focus in this work on the research question: *How do popular video conferencing systems perform?* This article provides a general overview of our research which is covered in more detail in the technical report of our work [15]. Our main contributions are:

(1) The design of an experimental method to evaluate and compare the performance of popular VCSs (Section 3). Our method includes a comprehensive set of *real-world* experiments, e.g., focusing on scalability and performance variability.
(2) The design and implementation of an automated tool for real-world experimentation with VCSs (Section 4). The tool implements our method, orchestrates real-world experiments, and simplifies repeating and reproducing experiments.
(3) A performance analysis of Zoom, Teams, and Jitsi using our experimental method (Section 5). We compare the scalability and performance variability of these systems on the client-side, and evaluate how these systems respond to changes in workload and deployment configuration.
(4) To foster the reproducibility our work, we release the complete dataset alongside a comprehensive technical report, on Zenodo [15].

2 VCS MODEL

Commercial VCSs are typically offered as cloud-based services and use a client-server architecture. In this architecture, users run client software which connects to a server controlled by the VCS operator.

We have created a model that depicts a typical architecture of a VCS (see Figure 2). The user starts a *client* (❶) on their device to start or join a video conference. The client is typically available as a *Web Application* (❷), running in a Web browser, or as a *Native Application* (❸) such as a desktop app. The client sends the user's audio and video input (❹) through a local connection (e.g., Wi-Fi, Ethernet) (❺) and a remote network (i.e., the Internet) (❻), to the VCS server (❼). Although the system is distributed, the user is not aware of the distinction between the clients and the servers, and perceives the video conferencing system as a single system. The server receives a continuous stream of audio and/or video data from each client, and forwards this to all other clients. As an optional step, the server can re-encode the data to reduce bandwidth requirements or provide better support for devices connected via unreliable networks.

3 DESIGN OF A METHOD TO COMPARE VIDEO CONFERENCING SYSTEMS

Motivated by the challenges introduced in Section 1, we design an experimental method aiming to evaluate VCS performance in real-world conditions. We first define a set of requirements for our method (in Section 3.1) and then design a method that addresses each requirement (in Section 3.2).

We followed the AtLarge design process for distributed systems [21]: we iterated and co-evolved the requirements and the method, testing each against practical and theoretical considerations, and later through a prototype (in Section 4) and in practice through actual experiments (in Section 5).

3.1 Definition of the Experiment Requirements

We define six main requirements (Rs):

R1 *Work with popular VCSs.* No VCS standard currently exists; instead, we need compatibility with a variety of VCSs.
R2 *Replicate easily.* For the experiments to be verifiable and reusable by the research community, they should be easy to replicate. A user-friendly interface could help.
R3 *Measure resource usage in scalability experiments.* We focus on weak scalability (i.e., workload varies proportionally to the amount of resources) [18] as opposed to strong scalability (i.e, workload stays fixed) [12] to understand how resource consumption scales under increasing load.
R4 *Measure performance variability.* VCS services are commonly deployed in clouds and connect to clients over the Internet. Both environments are unstable, which can lead to significant performance variability [28].
R5 *Compare VCS performance under audio±video workload.* In practice, VCSs do not have to manage audio-video workloads for each client, continuously. Users often mute their microphone and disable their camera, e.g., when listening during a presentation, when becoming active speakers. Thus, it is interesting to compare is the impact of audio, video, and combined workloads on VCS performance.
R6 *Compare VCS performance under web- and app-based workload.* Access to the cloud server can be web- and app-based. Thus, it is interesting to compare their performance impact.

3.2 Design of an Experimental Method

This section presents our experimental method, and how it meets the requirements discussed in Section 3.1. Our method defines a set of real-world experiments aimed to observe specific VCS behavior. An overview of these experiments is shown in Table 1.

To meet **R1**, we design our experiments using the model presented in Section 2. Our experiments collect the system-level metrics network bandwidth, CPU usage, and memory usage. These metrics are available for all VCSs, and are important to the user, who needs to run the VCS client on their own machine. Importantly,

the experiments definitions do not restrict the *system under test (SUT)*; the VCSs shown in Table 1 are those evaluated in Section 5.

To meet **R2**, our method defines all parameters it aims to control. It uses a fixed audio and video workload for each experiment. The video and audio are a webcam recording of a person talking to their computer screen; the technical details of this workload are provided in Table 2. This workload is provided as input to the VCS client, but the method does not limit the encoding used by the client before sending the data to the VCS-server. To limit the duration of each experiment, the method further limits the extent of each experiment (to 5 minutes) and the number of repetitions (here, 4 times). To fully meet **R2**, we need a specialized tool that can enforce these aspects automatically; we design it in Section 4.

To meet **R3** through **R6**, the method defines one real-world experiment for each. To meet **R3**, we define Experiment 1, focused on scalability. This experiment observes system behavior for 2, 4, and 6 clients in the same video conference. Our second (and last) step to meet **R4** is defining Experiment 2, focused on performance variability. This experiment observes VCS behavior over time, which allows studying its performance variability. To meet **R5** and study how audio and video streams affect VCS performance, we define Experiment 3, which compares system behavior when disabling audio or video in the workload. To meet **R6**, we define Experiment 4, which compares the behavior of the web-based and native applications of the same VCS.

Limitations: Our method proposes a limited set of actions toward addressing specific requirements. Beyond the scope of this work, future methods could further consider: beyond **R1**, standardizing a model for VCSs, focusing on both their main functions and their non-functional aspects; beyond **R2**, considering other aspects that facilitate reproducibility, from infrastructure for convenient experimentation [14, 17], to methodological aspects of reproducibility in the cloud [25, 28]; for experimentation, environment and configuration parameters, including variable resource performance and availability, different audio and video codecs, diverse user behavior, and more experience-related metrics [22]. During our experiments, we primarily focused on web clients due to its reproducibility. To include the performance of natives apps to realize **R6**, we investigated the performance of Zoom app while other popular native apps (e.g., Teams app) were not considered due to time constraints.

Table 1: Overview of experiments. *Scale* is the number of clients, *Chan.* is the use of *A*udio, *V*ideo, or Audio+Video (*A+V*). Systems under test (*SUT*) are *Z*oom, *J*itsi, and *T*eams.

ID / Section	Focus	Workload		SUT (§5.1)
		Scale	Chan.	
1/§5.2.1	Scalability	2, 4, 6	A+V	Z, J, T
2/§5.2.2	Variability	2, 4	A+V	Z, J, T
3/§5.3	Audio/Video	2, 4	A±V	J
4/[15]	Web/App	2, 4	A+V	Z, Z Desktop

Table 2: Experiment Workload.

Video		Audio	
Bitrate	570 KB	Bitrate	1.41 MB
Resolution	1280x720	Sample Rate	44100 Hz
Format	MJPEG	Format	WAV

4 TOOL FOR EXPERIMENTATION

In this section, we introduce the design of our experiment tool. The tool enables automatic, repeatable, real-world experiments which allows the user to evaluate the performance, scalability, and performance variability of VCSs. The experimental tool is tied to particular technology, but its operational principles are general and allow the tool to help meet **R2**. Section 4.1 discusses our high-level design. Sections 4.2 and 4.3 detail two core components.

4.1 Overview of the Tool Design

Figure 3 presents an overview of our experiment tool design. The tool follows a client-server architecture, using multiple *Automated Meeting Clients* (❶) and an *Orchestration Server* (❷). The clients connect to the orchestration server over the Internet. Once connected, the server can use the clients to run experiments, by instructing the clients to join a video conference. The clients start their VCS software, join the specified video conference over the Internet, and start capturing specific metrics. Post-experiment, clients send their measurement data back to the server, which stores it in the file-based *Results Storage* (❸) for later analysis.

Users can run large numbers of experiments by using the *Experiment Scheduler* (❹), which schedules sequentially batches of experiments submitted by users using file-based descriptions.

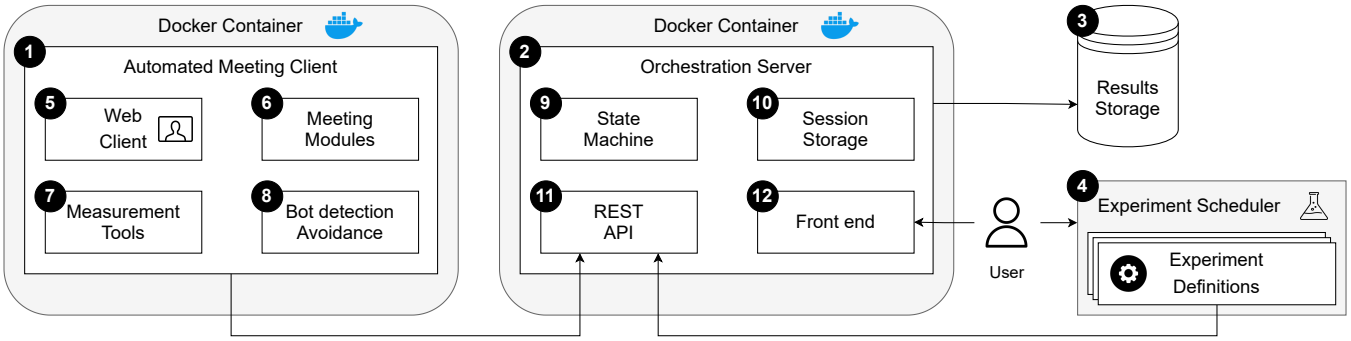

Figure 3: Design of the experiment tool.

The next two sections discuss the detailed designs of the Automated Meeting Client and the Orchestration Server. Our technical report [15] details the design of the Experiment Scheduler.

4.2 Automated Meeting Client

This section discusses the main components of the automated meeting client. This client (component ❶ in Figure 3) joins online conferences and measures the performance of the VCS. To simplify deployment and measurements, clients run in a containerized environment; we use the popular container environment `Docker` [2].

The meeting client contains a *Web Client* (❺) to support Web-based VCS applications. The Web Client connects to the video conference URL used in the experiment. Internally, the Web Client uses a headless Google Chrome browser (63% market share [1]). When performing experiments using a desktop application, the application replaces the Web Client (see our technical report [15]).

The Web Client interacts with the VCS using the *Meeting Modules* (❻). These modules use `NodeJS` [8] and `Puppeteer` [9] to perform scripted actions. The tool can be extended to support different VCSs by using custom meeting modules for each VCS, which implement actions to set-up a user, join the video conference, and enable the audio and video feeds.

Once the client is connected to an online video conference, it measures performance using a set of *Measurement Tools* (❼): `tcpdump` [11], `dpkt` [7], and `scapy` [10] to capture network traffic and calculate network bandwidth usage, and Linux' `ps` (process status) command to measure CPU and memory usage.

VCSs often use *bot detection* mechanisms to prevent malicious clients. Examples of such mechanisms are *captchas*, which require the user to complete a small puzzle before they are allowed to use the service. To prevent the detection of our automated clients and bypass captchas, our tool includes *Bot-Detection Avoidance* (❽). This consists of a set of tools and practices which is covered in more detail within the technical report [15].

4.3 Orchestration Server

This section discusses the components that make up the *Orchestration Server* (component ❷ in Figure 3), which creates and manages experiments conducted on various VCS platforms, keeps track of clients, and collects results after an experiment ends.

The *State Machine* (❾) controls the life-cycle of every experiment. It consists of a chain of states that correspond to the logical steps needed to run an experiment from start to finish. The chain progresses in lock-step with each client in the experiment.

The server keeps track of each client's state through an in-memory *Session Storage* (❿). When a client connects to the orchestration server, the server assigns it a unique identifier. This identifier is used to keep track of the client's state and allows the server to issue commands to individual clients.

The *REST API* (⓫) is used to handle communications between each client and the server. It also exposes endpoints which allow users to manually create, manage, and monitor experiments. Experiments are defined as JavaScript Object Notation (JSON) objects and contain details such as the platform the experiment is to be conducted on, the duration of the experiment, the URL of the conference room, and the password of the conference room (if present).

Finally, the orchestration server exposes a *Front End* (⓬), which is a web application allowing users to create, modify, and monitor experiments. At runtime, the user can see which clients are performing the experiment and the time remaining until completion.

5 REAL-WORLD EXPERIMENTS

This section presents our real-world experiments and their results. We start by explaining the experimental setup (Section 5.1), followed by an analysis of the results (Section 5.2).

For the experiments shown in Table 1, our **M**ain **F**indings are:

- **MF1** Different VCSs show large differences in resource usage, and respond differently to increasing numbers of clients. For example, Zoom uses on average up to 12× more bandwidth and 3× more memory than Teams and Jitsi.
- **MF2** The behavior over time of CPU, memory, and bandwidth consumption seems in general regular, but does not stabilize in all cases.
- **MF3** VCSs use more resources when both audio and video channels are enabled, but their effect on resource usage is complex. (See our technical report [15].)
- **MF4** Zoom app consumes significant less resources than Zoom Web. (See our technical report [15].)

5.1 Experiment Setup

This section describes our experimental setup. In our experiments, all automated clients run on separate physical machines. All measurements are obtained from a client running on a machine equipped with an 8th generation 4.2 GHz 8-core Intel i7 processor and 16 GiB RAM memory. The machine connects to the orchestration server and video conferencing sessions over the Internet, using a Wi-Fi 5 (IEEE 802.11ac) connection limited to 100 Mbps upload and download bandwidth.

Our experiments evaluate the Web-based applications of Zoom, Microsoft Teams, and Jitsi, and the Zoom desktop application; Table 1 summarizes their use per experiment.

5.2 Experiment Results

This section presents the results from our experimental results. An overview all experiments is available in Table 1. See our technical report [15] for detailed (and more) results.

*5.2.1 Performance and Scalability (leads to **MF1**).* Figure 4 shows the results of the scalability experiment. The results show large differences in the resource usage by different VCSs, and that VCSs respond differently when increasing the number of participants. The plots show the usage of system memory, CPU cores, and bandwidth on the horizontal axis, and different configurations of number of clients and VCSs on the vertical axis.

The results indicate that, on average, Zoom has higher resource usage and performance variability than Teams and Jitsi. Zoom's average memory usage ranges from 3,182 MB for 6 clients to 3,794 MB for 2 clients, which is more than 3× the average memory usage of Teams and Jitsi, which do not exceed an average memory usage of 921 MB in all cases. Zoom's average bandwidth usage ranges from 306 KB/s for 6 clients to 332 KB/s for 2 clients. This is an increase of over 54% compared to Jitsi, which uses on average 216 KB/s for

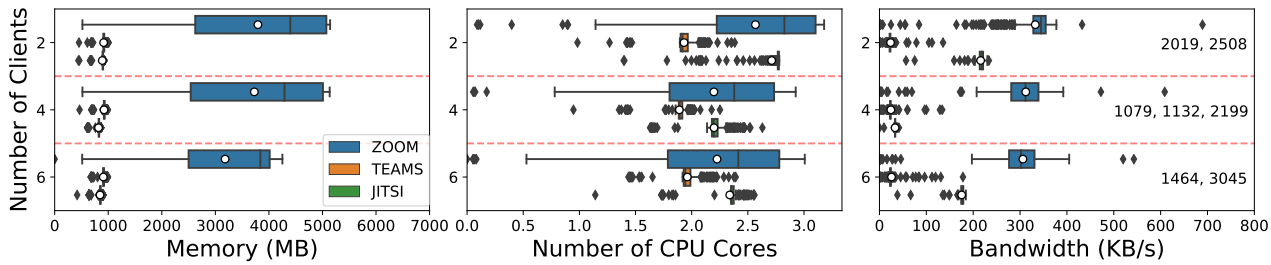

Figure 4: Effect of the number of clients on VCS client resource usage: memory (left), CPU (middle), and bandwidth (right). White dots show the arithmetic mean. Text labels in the right-most plot indicate outliers above 800 KB/s for Zoom.

2 clients. Teams' average bandwidth usage does not exceed 26 KB/s in all cases, which is only 8% of Zoom's peak average bandwidth usage of 332 KB/s.

Surprisingly, not all systems show increased resource usage when increasing the number of clients, for example, Zoom's resource usage goes down for all three metrics. Because its average CPU and bandwidth usage remains stable for more than 2 clients, we conjecture that Zoom switches to a more efficient mode of operation for calls with more than 2 participants. In contrast, the resource usage of Teams is similar for 2, 4, and 6 clients. We conjecture this is caused by Teams' only showing clients the video from the current speaker. Jitsi's CPU and bandwidth usage decreases when moving from 2 to 4 clients, but increases again when scaling to 6 clients. We were unable to find the cause of this behavior.

*5.2.2 Variability (**MF2**).* Because performance variability does not happen uniformly over the duration of the system's operation, we use Experiment 2 to capture when and how variability occurs. The results of this experiment appear in Figure 5 and (partially) Figure 1, and explain the large performance variability observed in Figure 4.

Before plotting, we clean the data by removing zero values, which improves plot readability. Because all clients use as input the continuous audio and video streams detailed in Table 2, zero values are likely caused by measurement errors or unstable network conditions. This operation removes about 2% of the data points.

Overall, the figures show that VCS resource consumption becomes regular after a warmup phase, but does not stabilize in all cases. The plots in the figure show time on the horizontal axis and usage of different resources on the vertical axis. The curves and shaded areas show the mean and interquartile range respectively. The experiment evaluates Zoom, Teams, and Jitsi, for 2 and 4 clients.

Figure 1 shows the memory usage is stable for both Microsoft Teams and Jitsi, but increases up to 5 GB before stabilizing for Zoom. We explain the linear memory increase as a memory leak in Zoom web; reports in public forums support our conjecture [3, 5]. (The stabilization to 5 GB is a memory limit for a single tab in Google Chrome.) This suggests users with a memory-critical system should reconsider using Zoom web. This result explains the large variability in memory usage observed for Zoom in Figure 4 (previous page). The left-side whisker (lower values) corresponds to Zoom's memory usage at the start of the experiment, which is comparable to that of Teams and Jitsi. The right-side whisker (higher values) corresponds to Zoom's peak memory usage. The quantiles and mean are skewed towards the higher values because Zoom spends roughly 150 seconds (half the experiment duration) at 5 GB memory usage.

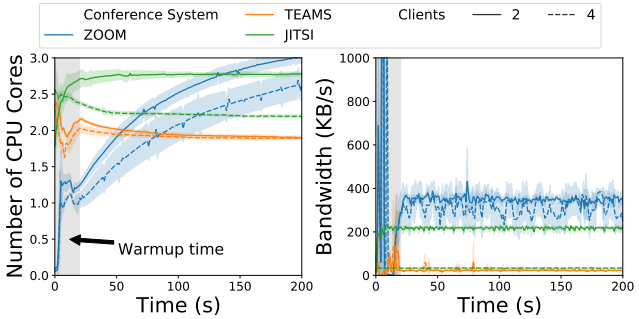

Figure 5: Behavior over time for different numbers of clients and VCSs. Plots show CPU (left) and network bandwidth (right). The curves show the arithmetic mean. The shaded areas show the standard deviation.

The left-most plot in Figure 5 shows that CPU usage is high, but stablizes over time for all VCSs. Microsoft Teams uses almost two full CPU-cores; this represent the lowest CPU usage among the systems we evaluate. Zoom's CPU usage is lower initially, but increases over time and takes almost 5 minutes (the full experiment duration) to stabilize. This increasing trend explains the large variability in CPU-usage observed for Zoom in Figure 4. The left-side whisker (lower values) corresponds to Zoom's initially low CPU usage. The right-side whisker (higher values) corresponds to Zoom's high CPU usage at the end of the experiment. Because Zoom's CPU usage increases sub-linearly, the quantiles are skewed towards the higher values, and the mean is lower than the median.

The right-most plot in Figure 5 shows that bandwidth usage is stable for all three VCSs after the initial setup time. This result confirms the range of values observed in the right-most plot in Figure 4. Unlike memory and CPU usage, bandwidth usage stabilizes for all systems after the initial warm up time. Zoom's bandwidth usage shows higher variability than Teams and Jitsi, but the overal trend shows a constant bandwidth usage.

5.3 Other Findings (MF3, MF4)

Here we present our results for Experiments 3 and 4, which are discussed in detail in our technical report [15].

Experiment 3 compares VCS behavior when using audio and video, only audio, and only video. The results show that using audio and video channels simultaneously requires more resources than when using only one, and that their effect on resource usage is complex. For each of the three resources, using audio and video with four clients uses on average more resources compared to using

only audio or only video. This matches our intuition that both require resources to be processed. The results show two additional surprising behaviors. First, using only audio results in higher CPU usage than using only video. We conjecture that this is caused by the bitrates of both streams. The audio bitrate is higher, and therefore generates more data that needs to be processed. Second, the sum of the bandwidth usage of only audio and only video is less than the bandwidth usage when using audio and video. We conjecture that this is caused either by additional bandwidth required to keep the two streams synchronized, or that Jitsi identifies clients with their audio enabled as potential speakers, improving the quality (and bandwidth requirement) of the video for this reason. Finally, the differences in memory and CPU usage are relatively small, which means the decision to enable or disable audio and video streams should be made primarily based on the quality of the user's network.

Experiment 4 compares the behavior of Zoom's Web-based app with its native desktop client. The results show that Zoom's desktop client performs significantly better than its Web-based client, using significantly less CPU, memory, and network bandwidth. First, it does not suffer from the memory leak observed in Experiments 1 and 2, reducing memory usage from 5 GB to an average of roughly 300 MB. A decrease of 94%. Second, while Zoom's Web client uses an average of 2.5 CPU cores for a 2-client conference, Zoom's desktop client does not use more than 0.5 cores. A decrease of 80%. Finally, in a 4-client setting, Zoom's Web client uses on average more than 300 KB/s, while its desktop client uses on average approximately 50 KB/s (a decrease of 83%).

6 THREATS TO VALIDITY

We consider our work to be an important first step in gaining a better understanding of the behavior of VCSs and discuss here limitations of our experimental method and real-world experiments.

We identify a main limitation in our experimental method: Because commercial VCSs are typically closed-source, our method takes a black-box approach in evaluating their behavior. This limits the types of metrics that we can collect (e.g., frame processing time) and how we can interpret them (i.e., lacking a detailed model).

We identify four limitations of our real-world experiments. First, we conduct our experiments over the Internet using Wi-Fi networks, and do not consider other popular network types such as Ethernet and 4G. Second, we conduct our measurements on a personal computer, which can run additional programs that can interfere with our measurements. Third, although VCSs are used in large settings (e.g., online conferences), we do not investigate the behavior of these systems for more than 6 clients. Lastly, we have conducted our experiments on a Linux based operating system, and did not consider other popular operating systems such as Windows and OSX.

7 RELATED WORK

We survey here work related to our study. Overall, ours is the first experimental study focusing on VCS performance for clients running in stable networks, where also memory and CPU resource usage are important. Ours is also the first study to compare VCSs popular in 2020, i.e., Zoom, Microsoft Teams, and Jitsi.

Closest to our work, Hortelano et al. [19] introduce a framework to evaluate the performance of video calls. Focusing on ad-hoc networks, the authors measure the throughput and inter-packet delay (jitter) of video calls using this framework. They conclude that as the number of hops increases, the chance of high delay times and packet losses increases as well. In contrast, we focus on stable networks, which are more commonly used in business, conference, and education settings.

Zhang et al. [30] study the video quality of Skype video calls. To analyze performance, the authors measured the Packet Loss Rate (PLR), the impact of the available network bandwidth, and the impact of propagation delay. They conclude that Skype is robust against mild packet losses and propagation delay, and Skype efficiently utilizes the available network bandwidth. Jansen et al. [22] study WebRTC-based conferencing. Complementing this work, we focus on the current market leaders, such as Zoom and Teams, and conduct different kinds of experiments.

The ICPE 2020 organizers give a sample performance result—the bandwidth consumption of Zoom during a single conference-session [20, Fig.2]. Our work extends and deepens this early measurement, proposing a method and conducting comprehensive experiments, and focusing on multiple VCSs.

Townsend et al. [27] study the attitudes of users towards video conferencing systems. They state that video conferencing systems and virtual collaboration have a major impact on social-work experience. Similar findings appear in other studies, e.g., [29].

Similar research has been conducted in related domains, such as video streaming. For instance, Hyunwoo et al. [23] introduced a tool called `YouSlow` to monitor buffer staling events while clients watch YouTube videos on Chrome browsers. Furthermore, Nguyen et al. [24] studied transport protocols to coordinate simultaneous transmissions of videos from multiple senders. To understand the behavior, they considered the sending rate, packet loss, and the probability of packets arriving late. In contrast, our study reveals the characteristic behavior of VCSs.

8 CONCLUSION AND FUTURE WORK

VCSs experienced a surge in popularity during the 2020 COVID-19 pandemic. Understanding their performance is desirable, but currently both methodological and practical results do not exist. In this work, we addressed the research question: *How do popular video conference systems (VCSs) perform, relative to each other?*

We proposed an experimental method to compare VCSs, designed an automated tool that implements the method, and used it to compare Zoom, Microsoft Teams, and Jitsi with real-world experiments. Our experimental results produced four main findings. First, there are significant differences in the resource usage of VCSs, and they respond differently to an increasing number of users. Second, the resource consumption of VCSs seems in general regular, but not all systems stabilize. Third, clients use significantly more resources when enabling both audio and video channels, but their effect on resource usage is complex. Fourth, the Zoom desktop app requires significantly less resources than the Zoom Web app.

Further research could cover more VCSs, including more diverse metrics related to security and QoS (e.g, number of interruptions, video and image quality), and more detailed metrics (e.g., application

specific metrics through profilers rather than system level metrics). Furthermore, the impact on the performance of a larger number of clients should be investigated.

DATA AND SOFTWARE AVAILABILITY

All the results used in this work are available, accompanied by a detailed technical report, on Zenodo:

https://doi.org/10.5281/zenodo.4463845

REFERENCES

[1] 2020. Browser Market Share Worldwide. https://gs.statcounter.com/browser-market-share [Online; accessed 22. Jan. 2021].

[2] 2020. Docker. https://www.docker.com/resources/what-container [Online; accessed 22. Jan. 2021].

[3] 2020. Memory leak in Zoom website meeting hosting? - Client Web SDK - Zoom Developer Forum. https://devforum.zoom.us/t/memory-leak-in-zoom-website-meeting-hosting/14503 [Online; accessed 24. Jan. 2021].

[4] 2020. Video conferencing becomes a Zoom boom. https://www.relocatemagazine.com/news/technology-video-conferencing-becomes-a-zoom-boom-coronavirus-0520-dsapsted [Online; accessed 22. Jan. 2021].

[5] 2021. Apparent memory leak using Zoom web app on Chromium (Ubuntu 20.04). https://superuser.com/questions/1589202/apparent-memory-leak-using-zoom-web-app-on-chromium-ubuntu-20-04 [Online; accessed 24. Jan. 2021].

[6] 2021. Best video conferencing software in 2021. https://www.techradar.com/best/best-video-conferencing-software [Online; accessed 24. Jan. 2021].

[7] 2021. dpkt. https://dpkt.readthedocs.io/en/latest/ [Online; accessed 22. Jan. 2021].

[8] 2021. NodeJS. https://nodejs.org/en/ [Online; accessed 22. Jan. 2021].

[9] 2021. Puppeteer. https://pptr.dev/ [Online; accessed 22. Jan. 2021].

[10] 2021. scapy. https://scapy.net/ [Online; accessed 22. Jan. 2021].

[11] 2021. TCPDump. https://www.tcpdump.org/ [Online; accessed 22. Jan. 2021].

[12] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '67 Spring Joint Computer Conference, April 18-20, 1967, Atlantic City, New Jersey, USA (AFIPS Conference Proceedings, Vol. 30)*. AFIPS / ACM / Thomson Book Company, Washington D.C., 483–485. https://doi.org/10.1145/1465482.1465560

[13] Aleksander Aristovnik, Damijana Keržič, Dejan Ravšelj, Nina Tomaževič, and Lan Umek. 2020. Impacts of the COVID-19 pandemic on life of higher education students: A global perspective. *Sustainability* 12, 20 (2020), 8438.

[14] Bal et al. 2016. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *Computer* 49, 5 (2016), 54–63.

[15] Richard Bieringa, Abijith Radhakrishnan, Tavneet Singh, Sophie Vos, Jesse Donkervliet, and Alexandru Iosup. 2021. An Empirical Evaluation of the Performance of Video Conferencing Systems. https://doi.org/10.5281/zenodo.4463845

[16] Erik Brynjolfsson, John J Horton, Adam Ozimek, Daniel Rock, Garima Sharma, and Hong-Yi TuYe. 2020. *COVID-19 and remote work: an early look at US data*. Technical Report. National Bureau of Economic Research. https://www.nber.org/papers/w27344

[17] Duplyakin et al. 2019. The Design and Operation of CloudLab. In *ATC*, Dahlia Malkhi and Dan Tsafrir (Eds.). 1–14.

[18] John L. Gustafson. 1988. Reevaluating Amdahl's Law. *Commun. ACM* 31, 5 (1988), 532–533. https://doi.org/10.1145/42411.42415

[19] Jorge Hortelano, Juan-Carlos Cano, Carlos T Calafate, and Pietro Manzoni. 2008. Evaluating the performance of real time videoconferencing in ad hoc networks through emulation. In *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 119–126.

[20] Alexandru Iosup, Catia Trubiani, Anne Koziolek, José Nelson Amaral, Andre B. Bondi, and Andreas Brunnert. 2020. Flexibility Is Key in Organizing a Global Professional Conference Online: The ICPE 2020 Experience in the COVID-19 Era. *CoRR* abs/2005.09085 (2020). arXiv:2005.09085 https://arxiv.org/abs/2005.09085

[21] Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent van Beek, Giulia Frascaria, Ahmed Musaafir, and Sacheendra Talluri. 2019. The AtLarge Vision on the Design of Distributed Systems and Ecosystems. In *ICDCS*. 1765–1776.

[22] Bart Jansen, Timothy Goodwin, Varun Gupta, Fernando A. Kuipers, and Gil Zussman. 2017. Performance Evaluation of WebRTC-based Video Conferencing. *SIGMETRICS Perform. Evaluation Rev.* 45, 3 (2017), 56–68.

[23] Hyunwoo Nam, Kyung-Hwa Kim, Doru Calin, and Henning Schulzrinne. 2014. YouSlow: A Performance Analysis Tool for Adaptive Bitrate Video Streaming. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 111–112.

[24] Thinh P Nguyen and Avideh Zakhor. 2001. Distributed video streaming over Internet. In *Multimedia Computing and Networking 2002*, Vol. 4673. International Society for Optics and Photonics, 186–195.

[25] Papadopoulos et al. 2019. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Trans. on Sw.Eng.* (2019), 1–1.

[26] TechRepublic. 2020. Watch out Zoom! https://www.techrepublic.com/article/watch-out-zoom-microsoft-teams-now-has-more-than-115-million-daily-users/ Article reports 300 million users for Zoom and over 100 million for Microsoft Teams in Oct 2020.

[27] Anthony M Townsend, Samuel M Demarie, and Anthony R Hendrickson. 2001. Desktop video conferencing in virtual workgroups: anticipation, system evaluation and performance. *Information Systems Journal* 11, 3 (2001), 213–227.

[28] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan S. Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is Big Data Performance Reproducible in Modern Cloud Networks?. In *NSDI*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 513–527.

[29] Caiping Xiong, Jun Ge, Qiyun Wang, and Xuejun Wang. 2017. Design and evaluation of a real-time video conferencing environment for support teaching: an attempt to promote equality of K-12 education in China. *Interact. Learn. Environ.* 25, 5 (2017), 596–609.

[30] Xinggong Zhang, Yang Xu, Hao Hu, Yong Liu, Zongming Guo, and Yao Wang. 2012. Profiling skype video calls: Rate control and video quality. In *2012 Proceedings IEEE INFOCOM*. IEEE, 621–629.