Distributed Systems Lab Report

An Empirical Evaluation of the Performance of Video Conferencing Systems

January 4, 2021

Richard Bieringa *
r.bieringa@student.vu.nl
2544975

Abijith Radhakrishnan*
mail@abijith.net
2667575

Tavneet Singh*
t.s.tavneet@student.vu.nl
2668222

Sophie Vos*
s.o.vos@student.vu.nl
2551583

Jesse Donkervliet
j.donkervliet@atlarge-research.com
Supervisor

Alexandru Iosup
a.iosup@atlarge-research.com
Course instructor

* *These authors contributed equally.*

Abstract - Since the global COVID-19 pandemic, a shift has occurred to remote education and work. This is enabled by various video conferencing systems like Zoom, Google Meet, Microsoft Teams which has consequently increased the pressure on digital infrastructure. Surprisingly, no prior research has been undertaken to understand the performance of various video conferencing systems.

In this study, we aim to evaluate and compare the performance of video conferencing systems in the context of distributed systems. To achieve this objective, we designed experiments to compare the performance and scalability of Zoom, Microsoft Teams, and Jitsi. Furthermore, we designed and implemented an experimentation tool that automatically hosts and joins video conferences, and measures the network bandwidth, CPU usage, and memory usage.

We observed that the video conferencing tools have a stable performance after the initial warm-up time for up to 6 clients. Except for the high memory usage of Zoom web, which grows linearly in time until a certain upper bound. Hence, we do not recommend Zoom web for a small number of clients.

Keywords - Distributed Systems, Automated Experimentation Tool, Empirical Performance Analysis, Video Conferencing Systems, Zoom, Jitsi, Microsoft Teams

1 Introduction

Due to the COVID-19 pandemic many organizations, businesses, and educational institutions have adopted a new online work structure [1, 2]. This is possible due to various video conferencing systems, such as Zoom and Jitsi, that replace face-to-face meetings. A consequence of this rapid switch to video conferencing systems is the increased pressure on the digital infrastructure. As our society relies heavily on video conferencing systems and their underlying infrastructure, it is essential to understand the performance of the various video conferencing systems.

Early studies of video conferencing systems exist, but do not yet capture a performance analysis and comparison of popular video conferencing systems. Townsend et al. [8] studied the attitudes of users towards video conferencing systems. They stated that video conferencing systems and virtual collaboration have a major impact on the social work experience.

Hortelano et al. [4] introduce a framework that allows evaluating the performance of video calls. The authors measured the throughput and inter-packet delay (jitter) of video calls using this framework and focused on ad hoc networks. They concluded that as the number of hops increases, the chance of high delay times and packet losses increases as well.

Zhang et al. [10] studied the video quality of Skype video calls. To measure the performance, the authors measured the Packet Loss Rate (PLR), the impact of the available network bandwidth, and the impact of propagation delay. They concluded that Skype is robust against mild packet losses and propagation delay, and Skype efficiently utilizes the available network bandwidth.

Currently, Skype is losing its market share as other video conferencing systems such as Zoom are dominating the field[1]. The currently most popular free video conferencing systems are Microsoft Teams, Google Meets, and Zoom[2]. No research has been conducted to evaluate and compare the performance of these video conferencing systems.

Similar research has been conducted in related domains such as video streaming. For instance, Hyunwoo et al. [6] introduced a tool called `YouSlow` to monitor buffer staling events while clients watch YouTube videos on Chrome browsers. Furthermore, Nguyen et al. [7] 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 this study, we aim to evaluate and compare video conferencing systems to understand their performance. The central research question is: *"How do popular video conferencing systems perform relative to each other?"*. To understand the performance of video conferencing systems, we design and perform experiments to evaluate and compare the video conferencing systems. The main contributions of this study are:

1. The design of experiments to evaluate and compare the performance of popular video conferencing systems (Section 4). The experiments currently presented in public literature do not seem to be based on a systematic approach. Hence, an experiment design based on clear, comprehensive requirements is still needed. Section 3 provides an overview of the requirements that form the foundation for our experiments.

2. The conceptual design and implementation of a tool that automatically hosts and measures the performance of popular video conferencing systems (Section 5). Experiments in the field are often performed manually. This has a major disadvantage that the experiments cannot be validated by the research community and the experiment setup cannot be replicated for similar experiments. We aim to fill this gap by introducing an open-source, automated tool that allows researchers to perform experiments regarding the performance of video conferencing systems through an intuitive Graphical User Interface (GUI).

3. A performance analysis of Zoom, Microsoft Teams, and Jitsi (Section 6). In the experiments, we measure the network bandwidth, CPU usage, and memory usage to evaluate and compare the performance. Furthermore, we analyze the scalability by comparing the performance for 2, 4, and 6 clients. By comparing the web-based and app-based Zoom versions, we aim to understand the performance difference of web- versus app-based video conferencing systems. Finally, we test the difference in performance of video and audio workloads in an in-depth study of Jitsi.

2 Background

In this section, we explain general architectural concepts that video conferencing systems have in common (Section 2.1). Furthermore, we describe and provide background information on the architecture and design decisions of the video conferencing systems that are analyzed in our study (Sections 2.2 - 2.4).

[1]https://www.datanyze.com/market-share/web-conferencing
[2]https://www.techradar.com/best/best-video-conferencing-software

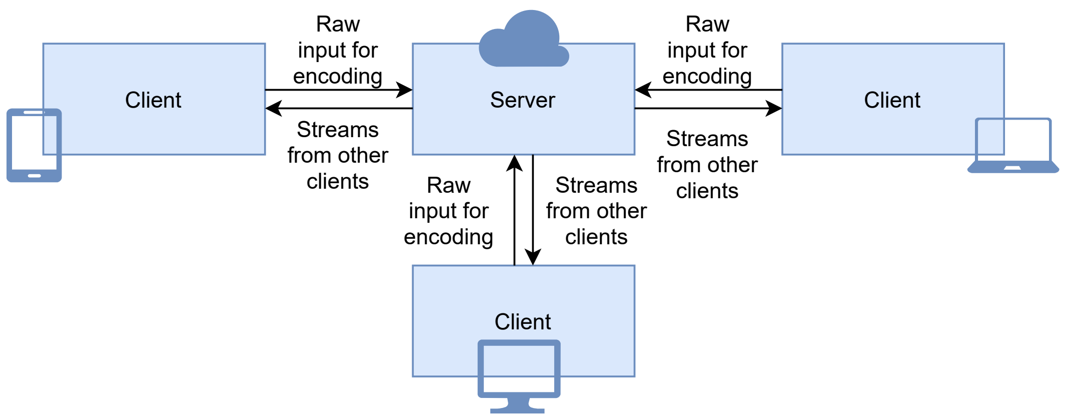


Figure 1: A general overview of the client-server architecture.

2.1 Client-Server Architecture

There are multiple architectures possible to design video conferencing systems (e.g. client-server, peer-to-peer). The most common architecture is the client-server architecture. All video conferencing tools that we analyze in this study follow the client-server architecture. Hence, we introduce this concept briefly.

Figure 1 presents an overview of the client-server architecture. The client-server architecture divides the system in two main components, namely, the client(s) and the server(s). The clients send requests to the server which processes the request and sends a response.

In the context of video conferencing systems, servers are often cloud-based. In practice, there are multiple servers distributed over geographically distinct locations. Furthermore, different servers can serve different tasks. To the clients, this architecture should be hidden. Hence, the cloud server is abstracted as if it is one coherent entity. The server receives raw video and audio input from different clients for encoding. The encoding process includes converting the input to different video resolutions and bitrates for other clients with diverse network bandwidth.

Clients do not necessarily refer to users. Clients are machines making the requests, however, these requests can be initiated by users. In the context of video conferencing systems, each client is a user that participates in the video conference through a device (e.g. laptop, smartphone). However, in our experiments, we simulate these users by setting up automated clients that use a pre-recorded video stream.

The clients and servers are connected through a network. This could be any network depending on the scale of the applications and system. Popular video conferencing systems are connected through the internet. Hence, the requests and responses are sent over the internet.

Even though the system is distributed, to the user it seems like they are interacting with one system. Accordingly, the user is not aware of the distinction between the clients and the servers, and perceive the video conferencing system as a uniform system.

2.2 Zoom

Zoom[3] provides a cloud platform for video, voice, content sharing, and chat runs across mobile devices, desktops, telephones, and room systems. Zoom is founded in 2011 and currently dominates the video conferencing market share[4]. Zoom has experienced a rapid growth since the COVID-19 pandemic. In April 2020, they experience a 50% increase of its daily meeting participants from 200 million at the start of April to 300 million at the end of April[5].

[3]https://zoom.us
[4]https://www.datanyze.com/market-share/web-conferencing
[5]https://www.bizjournals.com/sanjose/news/2020/04/30/zoom-user-numbers-zm.html

Unfortunately, most of the design decisions and technical architecture of Zoom is not publicly available. We refer to this as a *black-box architecture*. According to their blog[6], Zoom is cloud-native and optimized for video performance from the start rather than being built on top of legacy applications. Furthermore, Zoom's two most important technologies providing their video conferencing service are the cloud network and video architecture.

The cloud network contains 13 data centers that are connected through private connections. Since the COVID-19 pandemic (that led to an increasing demand for Zoom), Zoom moved large quantities of real-time video-conferencing traffic to Amazon Web Services (AWS)[7].

The architecture is optimized to handle video-specific requirements. This is realized through a distributed architecture that allows users to join the meetings via the nearest data center. Zoom uses multimedia routing, this delivers multiple video streams from other users to a user's device. Accordingly, the audio and video of every participant in the video call are directly streamed to the user without any preprocessing. This improves the user experience, as this enables a rapid switch between speakers and low latency.

Moreover, Zoom performs multi-bitrate encoding. This means that each stream can change to varying resolutions. This ensures that the video quality adapts to the maximum device and network capacity. Zoom allows users to join calls through both web and app, this allows us to compare the performance between the web and the app service.

2.3 Microsoft Teams

Microsoft Teams[8] is a proprietary business communication tool that allows workspace chat and video conferencing, file storage, and application integration. Similar to Zoom, Microsoft Teams has both a web and app version.

Since the worldwide launch of Microsoft Teams on 14 March 2017, the majority of the platform's growth happened in the year 2020 due to the COVID-19 pandemic. As of October 2020, Microsoft Teams has collected 115 million daily active users.

Microsoft Teams is developed based on the implementation of Microsoft 365 groups, Microsoft Graph along with other Microsoft products. Microsoft Teams leverages identities stored in the Azure Active Directory (Azure AD). The chat function is implemented using a micro-service in Azure. Furthermore, it uses a combination of OneDrive for Business, SharePoint, and Microsoft Exchange to store individual chats, images, and files. The video streams are generated using a media service in Azure which utilizes Microsoft Stream. Microsoft Teams utilizes several Microsoft-specific protocols along with others such as MNP24 and SILK.

2.4 Jitsi

Jitsi[9] is a set of open-source projects that allow building and deploying video conferencing solutions. The video conferencing solutions are freely accessible to anyone on the internet. Jitsi is flexible as it is fully open-source. Jitsi is funded by *8x8* which in turn uses the open-source contributions to improve their own products. The main services that allow the video conferencing are Jitsi Videobridge[10] and Jitsi Meet[11].

Jitsi Videobridge is a Selective Forwarding Unit (SFU) designed to run many video streams from a single server. Jitsi Videobridges requires good network bandwidth rather than CPU power.

Jitsi Meet is built on top of Jitsi Videobridge. Jitsi Meet allows video conferences through a browser without the need to install software on the client-side. Additional features are a chat function, screen sharing, conference recording, together with various other features.

[6]https://blog.zoom.us/zoom-can-provide-increase-industry-leading-video-capacity
[7]https://www.datacenterdynamics.com/en/news/most-zoom-runs-aws-not-oracle-says-aws
[8]https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/online-meetings
[9]https://jitsi.org
[10]https://github.com/jitsi/jitsi-videobridge
[11]https://github.com/jitsi/jitsi-meet

3 Requirements

In this study, we evaluate and compare the performance of video conferencing systems. To enable these experiments, we implemented an experimentation tool that allows automated hosting, joining, and performance measurements of popular video conferencing systems. In this section, we list and explain the rationale behind the requirements of the experiments. The experimentation tool should adhere to these requirements too in the sense that each functionality specified by the requirements should be enabled by the tool. The requirements are prioritized according to the *MoSCow* method [3]. Therefore, the requirements are classified as *Must have*, *Should have*, *Could have*, and *Won't have*. The requirements are derived from iterative group discussions following the AtLarge Design Cycle [5]. Furthermore, the requirements are grouped into functional requirements (Section 3.1) and quality requirements (Section 3.2).

3.1 Functional Requirements

The functional requirements (FRs) together with their rationale are presented in the following list:

FR1 *The experiments must compare multiple distinct video conferencing systems.* The main objective of this study is to evaluate and compare the performance of video conferencing systems. Hence, multiple video conferencing tools need to be analyzed.

FR2 *The experiments must measure appropriate system-level metrics for video conferencing systems.* In this study, we are interested in measuring the performance of the video conferencing systems. To do so, concrete metrics need to be selected that reflect the performance of the various systems.

FR3 *The experiments must measure the scalability of the different video conferencing systems.* Scalability is an important property for distributed systems as distributed systems consist of a network of autonomous computing components. Hence, it is interesting to understand how the systems scale for a larger number of clients.

FR4 *The experiments must counter the variability of the measurements in the experiment environment.* Experiments that are performed in a real-world setting experience a certain level of variability. The experiments cannot be performed in complete isolation, hence, replicating the same experiment might lead to different results based on the influence of outside variables. This is an undesirable outcome, therefore, appropriate measures need to be taken to mitigate the effects of variability within the environment on the measurements.

FR5 *The experiments should compare the performance of web- and app-based video conferencing.* Some video conferencing tools are accessible through the browser while others require the user to download an application. This affects the user experience. Thus, it is valuable to understand the difference in performance of web- and app-based video conferencing systems.

FR6 *The experiments should compare the performance of audio and video workload separately.* Another interesting distinction to further assess is the difference in performance of audio and video workload. This is relevant as users often mute their microphone and disable their camera when solely listing to a presentation. We are interested in the effect of disabling these streams on the performance.

FR7 *The experiments could measure appropriate application-level metrics for video conferencing systems.* If the time allows it, the experiments can be extended by including more application-level metrics. These metrics are interesting to assess the performance, however, they are more advanced to measure and hence are an alternative extension to the study.

Scaling the experiment for a large number of clients is currently out of scope. We aim to focus on correctly implementing and executing the experiments for a relatively small number of clients (2 - 6). Once these experiments adhere to the above mentioned FRs and the experiment framework is well-tested, future research can be conducted to extend the experiments by measuring the impact on the performance when scaling up the video conferencing systems by adding a larger number of clients.

Another limitation of the experiments is the lack of testing the effect of the experiment environment on the performance measurements. To illustrate, the performance of the video conferencing systems could be measured using different operating systems, network conditions, and devices. This would make the relation of the video conferencing system performance and its environment apparent. To minimize the effect of these environment variables on the results, we kept the environment stable throughout the performed experiments.

3.2 Quality Requirements

The quality requirements (QRs) together with their rationale are presented in the following list:

QR1 *The experiments should be performed in a real-world environment.* We aim to advise users about the performance of popular video conferencing systems. Hence, the experiments should be performed in an environment that is comparable to the real-world user environment. This allows us to draw conclusions that hold in the real-world.

QR2 *The experiments should be easy to replicate.* For the experiments to be verifiable and re-usable by the research community, the experiments should be easy to replicate.

QR3 *The experiments could be user-friendly to conduct.* The replicability and accessibility of the experiments are amplified by a user-friendly interface.

Possible extensions to the QRs are to perform the experiments fast and efficiently. Currently, this is out of scope as we initially aim to have a working framework to perform the experiments. Once this is realized, the framework can be optimized.

4 Experiment Design

This section describes the high-level experiment design. We start by defining the objective, research questions, and metrics of the overall experiment (Section 4.1). Afterward, we reflect on the relation between the experiment design and requirements (4.2). Lastly, the experiment designs for each concrete set of experiments, grouped by the research questions, are listed (Section 4.3).

4.1 Experiment Definition

To define our experiment, we applied the Goal-Questions-Metrics (GQM) method [9].

The **goal** of this experiment is: *to evaluate and compare the performance of video conferencing tools in the context of distributed systems.*

The **main research question** is: *how do popular video conferencing tools perform relative to each other?*

This is a broad research question that requires research in several disciplines. To scope down our research, we focus on the following aspects (based on the requirements listed in Section 3):

Q1 *How does the performance of video conferencing systems behave over time?* This question is required to tune the *time* parameter in the experiment (i.e., how long should an experiment last to capture the essential information).

Q2 *What is the variability of the measurements in our experiments?* This question implements **FR4**. Measuring the variability is required to understand the stability of the research environment and, hence, the quality of the results. Furthermore, we are interested to understand how many repetitions are required in the remaining experiments.

Q3 *What is the performance of Zoom, Microsoft Teams, and Jitsi relative to each other?* This question implements **FR1**. We chose to analyze three video conferencing systems as this allows a proper comparison feasible within the available time. Zoom, Microsoft Teams, and Jitsi are selected as systems under test within this experiment. We selected Zoom as this is currently the most popular video conferencing tool. Microsoft Teams is relatively new which makes it interesting to compare its performance against other players. Lastly, we chose Jitsi as this is an open-source tool and hence allows us to analyze application-level metrics, latency, and jitter.

Q4 *How scalable are Zoom, Microsoft Teams, and Jitsi?* This question implements **FR3**. The scalability of the video conferencing systems can be evaluated by comparing the performance for different numbers of clients. We chose to repeat these experiments for 2, 4, and 6 clients. These numbers are based on the maximum available resources within our research team to run the clients on.

Q5 *What is the difference in performance and scalability between web- and app-based systems?* This question implements **FR5**. We choose to analyze this question by comparing Zoom web vs. app as this ensures that we compare the same video conferencing system through a different means.

Q6 *What is the difference in performance and scalability between audio and video workload?* This question implements **FR6**. We are interested in this question to understand the difference in performance when users mute their microphone and/or disable their camera.

In this study, we evaluate the performance of video conferencing systems. Performance is a broad notion that can be measured by many different metrics (from subjective metrics such as user satisfaction to objective metrics such as memory usage). We focus on system-level metrics as these are in the scope of distributed systems research.

The **metrics** are: *Network Bandwidth*, *CPU Usage*, *Memory Usage*

Selecting these metrics implements **FR2**. We selected these metrics as they represent the system and network usage of the various video conferencing tools. Moreover, these metrics are important for the user experience and relatively easy to measure. In each video conference, we measured the metrics for each client separately. However, in the experiment results, we used the measurements of the same client to ensure that we can accurately compare the different experiments. Note that this entails that within an experiment of 6 clients, the results are the measurements from one client that participates in a video conference together with 5 other clients.

4.2 Relation to Requirements

FR1 - FR6 are implemented by **Q1 - Q6** together with the selection of the performance metrics in the previous subsection. The detailed explanation of the realization of these requirements is described in the previous subsection. The only FR that is not implemented by the experiments is **FR7**. Application-level metrics are also important to understand the user experience, however, we omitted these metrics due to time constraints. Future research is required to include these metrics to have a more complete understanding of the performance of the video conferencing systems.

The QRs (**QR1 - QR3**) are addressed in the design of the experimentation tool (Section 5).

4.3 Experiment Design

We distinguish four sets of experiments that realize all research questions (**Q1 - Q6**). The first set of experiments aims to understand the variability in the measurements (Section 4.3.1). The second set of experiments are experiments in which we compare and evaluate the performance and scalability of Zoom web, Microsoft Teams, and Jitsi (Section 4.3.2). Next, the third set of experiments focuses on comparing Zoom web and app (Section 4.3.3). The last set of experiments is performed using Jitsi and evaluates the performance of video and audio workload separately (Section 4.3.4).

4.3.1 Variability

To mitigate the effect of external events and processes on the performance results, each measurement needs to be repeated and aggregated. This ensures that outliers do not drastically affect the results. The first set of experiments aims to understand the variability of the performance and answers **Q1** and **Q2**. This allows us to understand the time a video conferencing system should last in the remaining experiments. Furthermore, we measure the variability in the metrics of the experiments. This provides feedback on the stability of the research environment and provides insights into the number of repetitions of each experiment.

The experiment parameters are listed in Table 1. We chose the initial parameter 5 minutes for the time variable as we did not expect the performance of a video call to change significantly after the system and the video conference is set-up. Moreover, as these are experiments to tune our parameters, we did not want to spend too much time on the experiments.

This is also the reason we chose 4 repetitions, partly because we did want to understand the variance in the metrics but also as we had a limited time window to perform the exploratory experiments in.

Variability				
Video Conference Tool	Clients	Time (min)	Repetitions	Metrics
Zoom Web	2, 4	5	4	network bandwidth, CPU usage, memory usage
Jitsi	2, 4	5	4	network bandwidth, CPU usage, memory usage
Teams	2, 4	5	4	network bandwidth, CPU usage, memory usage

Table 1: Exploratory experiments to tune the time and repetitions parameters.

4.3.2 Scalability

The second set of experiments focuses on answering **Q3** and **Q4**. An overview of the performed experiments is provided in Table 2. In this set of experiments, we consider the video conferencing tools: Zoom Web, Jitsi, and Microsoft Teams. We measure the network bandwidth, CPU usage, and memory usage.

We decided that the duration of each video conferencing experiment remains 5 minutes (this applies to Sections 4.3.3 and 4.3.4 as well). This is a compromise between needing a high duration to make the experiment useful and having time restrictions imposed due to the short time window in which we conducted the experiments. Furthermore, the outcome of the variability study showed not much variance after the start-up of the video conferencing systems has finished. We do not expect that having the call lasts longer significantly affects the performance. In this experiment, the time is kept at a single value to ensure that its impact on the measurements is minimal.

Moreover, the repetitions are kept at 4 as the results did not show much variability (this applies to Sections 4.3.3 and 4.3.4 as well). Every video conferencing tool is analyzed for 2, 4, and, 6 clients. We chose to change the number of clients in order to assess the scalability of the video conferencing tools.

Scalability				
Video Conference Tool	Clients	Time (min)	Repetitions	Metrics
Zoom Web	2, 4, 6	5	4	network bandwidth, CPU usage, memory usage
Jitsi	2, 4, 6	5	4	network bandwidth, CPU usage, memory usage
Teams	2, 4, 6	5	4	network bandwidth, CPU usage, memory usage

Table 2: Experiments to evaluate and compare the performance and scalability of Zoom Web, Jitsi, and Microsoft Teams.

4.3.3 Web vs. App

The third set of experiments allows us to compare the performance and scalability of web- versus app-based video conferencing system Zoom. This experiment aims to answer **Q5** and the experiment parameters are listed in Table 3. In order to compare the performance and scalability of web- versus app-based video conferencing tools, we performed experiments using Zoom as this tool allows joining a video conference both through a browser as through a pre-installed app. As both versions are from the same platform, other variables that potentially affect the performance are as stable as possible.

Web vs. App				
Video Conference Tool	Clients	Time (min)	Repetitions	Metrics
Zoom App	2, 4	5	4	network bandwidth, CPU usage, memory usage
Zoom Web	2, 4	5	4	network bandwidth, CPU usage, memory usage

Table 3: Experiments to compare web- vs. app-based video conferencing in Zoom.

4.3.4 Audio vs. Video

To obtain in-depth information regarding the difference in performance and scalability of video and audio workload we performed several experiments using Jitsi in which we separated the video and audio workload. The focused experiments aim to answer **Q6** and are listed in Table 4.

Audio vs. Video				
Video vs. Audio	Clients	Time (min)	Repetitions	Metrics
video	2, 4	5	4	network bandwidth, CPU usage, memory usage
audio	2, 4	5	4	network bandwidth, CPU usage, memory usage
video and audio	2, 4	5	4	network bandwidth, CPU usage, memory usage

Table 4: Experiments to compare audio vs. video workload in Jitsi.

4.4 Experiment Workload

To ensure input data consistency between various experiments and for a compelling comparison between different video conferencing systems, the same set of audio and video files were streamed throughout the experiments. The technical details of the workload are provided in Table 5. The workload was chosen in a way so that the content is similar to a real-world video stream.

Experiment Workload		
Video	Bitrate	570 KB
	Resolution	1280x720
	Format	MJPEG
	Size	5.9 MB
Audio	Sample Rate	44100 Hz
	Precision	16 bit
	Bitrate	1.41 MB
	Format	WAV
	Size	51 MB

Table 5: The workload used to perform the experiments.

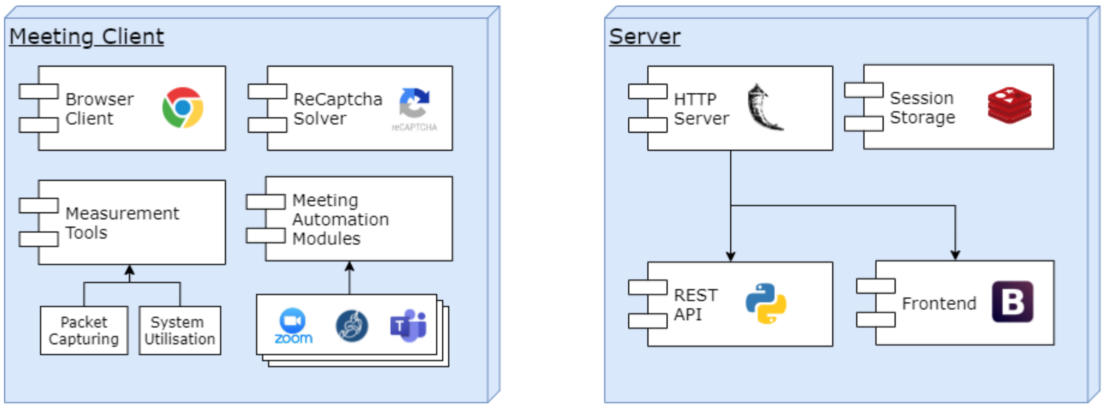


Figure 2: The client-server architecture of the experiment framework.

5 Experimentation Tool

We evaluated the video conference systems through a series of experiments using multiple experiment clients. This section describes the experimentation tool architecture together with the various systems used to get consistent executions in a highly varying environment (Section 5.1). The core experiment set-up consists of the orchestration server (Section 5.2) and the experiment client (Section 5.3). In addition, we have created a scheduler that allows automatic experiment creation and allows the clients to run in a batch processing mode (Section 5.4).

5.1 Framework Architecture

The experiment framework adheres to a classical client-server architecture. Within this architecture, each client acts as an experiment conductor, which sole purpose is to emulate a person using a conferencing system while gathering the required measurements to answer our research questions as defined in Section 4.1. The server has the job to orchestrate these clients and to gather the data that is captured by them. A general top-down overview of the client-server architecture can be seen in Figure 2, where both the client and the server are depicted with the multiple modules that they consist of. The design decision for client-server architecture stems from our goal to emulate a real-world environment as specified in requirement **QR1**. We use this architecture to connect multiple clients over the internet rather than a cluster to emulate a generic web conference. The design and implementation of these components are discussed in more detail in the following sections while a flowchart depicting the flow of actions can be found in Figure 16 located in the Appendix.

5.2 Experiment Server

The experiment server provides a user-interface and API to orchestrate the experiments we have conducted. The server is hosted on an `AWS EC2`[12] instance and is accessible through the `HTTP` protocol. If you access the server through a web browser, the user is presented with a user-interface to manage and create experiments. The server also exposes a REST API to directly manage the experiments through generic HTTP requests. In this section, we discuss how the server manages experiments and the stages each experiment can reside in.

First of all, the server allows us to create experiments. If a user creates an experiment through the web interface as seen in Figure 12 located in the Appendix. The user-interface provides a user-friendly manner to create, modify, and monitor experiments to satisfy **QR3**. This experiment configuration interface enables the user to customize every aspect of the experiment. For instance,

[12]https://aws.amazon.com/ec2

Experiment Stage	Description
START	This is the default state prior to when an experiment is created.
WAIT	This is after an experiment has been created but not enough clients have connected to perform it.
SETUP	Experiment setup is used to sync all clients up until the point in time that they are ready to join the actual meeting.
RUN	This stage indicates that the clients are currently conducting the experiment and are gathering data.
DATA_COLLECTION	This state occurs after the experiment has been conducted, during this stage the clients are uploading their collected metrics.
DATA_VIZ	This state occurs when all clients have successfully uploaded their data, the server can now visualize and serve the results.

Table 6: The different stages of an experiment within the framework.

the user can select any of the supported video conference platforms (Jitsi, Zoom Web, Zoom App, or Microsoft Teams). In addition, the user can tune other experiment settings such as enabling or disabling audio and video, setting the experiment duration, and defining the room URL and password.

The server also grants the ability to monitor the experiment status. This allows checking which clients are connected through which IP addresses and in which stage the experiment is currently in. An example of this interface can be seen in Figure 13 located in the Appendix where it shows that one client is connected and the experiment is in the `WAIT` stage since the experiment settings show that this specific experiment requires 10 connected clients.

Lastly, it allows us to see the experiment results as seen in Figure 15 located in the Appendix. This interface is presented when an experiment ends or when we browse through the previously ended experiments. From here we can manually download the data gathered from the clients and see the settings for a particular experiment that has ended.

There different stages into which an experiment can reside in and how it is used within the experiment framework. At each point in time, the server has a status associated which reflects the stage an experiment is in, all of the stages can be found in Table 6. Prior to launching any experiment, the server will be in the `START` phase and during the different stages of an experiment the status will update accordingly.

We have implemented this system in order to monitor the status of an experiment and to synchronize the clients when they reach certain stages. Once a client is connected to the server, it must first wait until enough clients are connected before advancing to the next stage. The same mechanism of these different stages is used throughout the various phases of the experiment and allows us to have fine-grained control of the clients in order to perform consistent experiments.

5.3 Experiment Client

The experiment client consists of a containerized set of tools that allows to automate the process of joining video conference rooms and gather performance metrics of the video conferencing services that the framework supports. This section discusses the various components that the client consists of as well as the rationale behind these implementation details.

5.3.1 Headless Browser Automation

Most of the experiments that we perform are done using the browser-based variants of the meeting clients that we use for this project. In order to run these meeting clients, we chose to use `Google Chrome`. In order to automate this browser, we have opted for using `Puppeteer`[13]. Puppeteer is a

[13]https://developers.google.com/web/tools/puppeteer

high-level `NodeJS`[14] library created by the Google Chrome team itself. Puppeteer provides an API that allows automating tasks within a Google Chrome browser.

Moreover, the library provides several features that enable the replication of the experiments consistently to satisfy requirement **QR2**. First of all, Puppeteer allows to run a `headless` Chrome Browser. This means that we do not require an actual screen to render the browser and, thus, can encapsulate the client in a `Docker` Container[15] which we can scale indefinitely. Second of all, we can emulate the webcam and microphone attached to the browser by using video and audio files respectively. By doing this, we have minimal variance in the actual data that we transmit during the experiments per client. The final feature that aims to match the real-world environment as specified in requirement **QR1**, is that the experiments use an actual Google Chrome browser. This allows measuring real-world performance since it is the most common tool to access the web with a market share of more than 63%[16].

5.3.2 Measurement Tools

In order to retrieve the measurements from the client, we utilize two tools. The first tool that we use for our experiments is `tcpdump`. This allows us to capture the network traffic for the conference systems where we can deduct our metrics such as network bandwidth from. By including this tool in our containerized application, we only capture the data that is specific to the conference system. This allows us to have very specific data regarding the system that we are using since most of these systems do not document the network usage. Furthermore, we used Python libraries `dpkt` and `scapy` to analyze and collect bandwidth information from the saved tcpdump packet files.

The second tool that we use is a `Python` script that gathers the system statistics of the client that is running the experiment. By using this tool, we can record the memory and CPU usage of the video conference system. The Python script internally uses Linux's `Process Status` tools to measure CPU and memory usage.

5.3.3 Anti-bot Detection

Most of the online systems do not tolerate the usage of automated web clients using their services. In order to circumvent this, we have two subsystems within our meeting clients. Since the client is already using an actual version of the Chrome browser this gives us an advantage, but detection systems are in place to detect these systems as well. More specifically the Zoom web client detects bots and also implements `ReCaptcha`[17], an advanced risk analysis engine that detects bots and prohibits them from using their services.

The first subsystem that we use is a plugin for the puppeteer browser framework [18]. This provides various mechanisms to mask the fact that we are using a headless Chrome browser to access various services. The second subsystem that we use is a ReCaptcha solver. This allows us to send our ReCaptcha token to an external provider which solves our ReCaptcha and sends us a valid token back. These systems overcome the obstacles that we encountered when developing the automated testing system.

5.3.4 Batch Mode

The clients in this framework run a single experiment before shutting down as default behavior. Since the goal of our research is to conduct many experiments, we have included an additional batch script for these clients to solve this problem. This script runs the containerized application and executes the entire workflow. After the client finishes and the measurements are recorded successfully, the script will restart and connects to the server again to wait until a new experiment is created.

[14]https://nodejs.org/en
[15]https://www.docker.com/resources/what-container
[16]https://gs.statcounter.com/browser-market-share
[17]https://www.google.com/recaptcha/about
[18]https://github.com/berstend/puppeteer-extra/tree/master/packages/puppeteer-extra-plugin-stealth

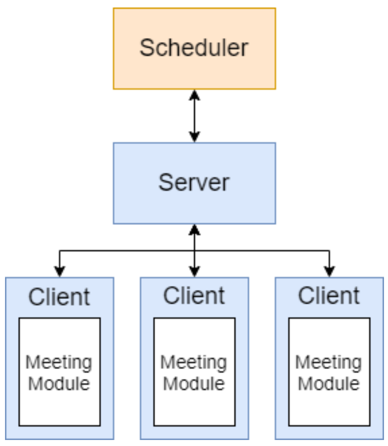


Figure 3: Top down view of the experiment scheduler within the experiment framework.

We have included a two-minute timeout before each experiment to ensure that the new experiment will not encounter any leftover clients from the previous experiments. The batch script allows us to run an infinite number of experiments per client without having to manually start them again.

5.4 Experiment Scheduler

The experiment scheduler is a `Python` based script that automatically creates and runs our experiments. Although we can manually create our experiments it can take a lot of time to do so for the amount of experiments that we planned on running. This script assumes that there are clients running in batch mode as discussed in Section 5.3.4 where the clients keep trying to join and experiment until it successfully does so. The scheduler is configured to match our experiment requirements and iterates over all the different experiment configurations that we have defined in Section 4. After each experiment, the scheduler waits for a predetermined amount of time to let the clients gracefully shut down and allows time for the clients to leave the meeting rooms. Afterward, the scheduler repeats this process until all different experiment combinations have been conducted.

6 Results

This section presents the results of the conducted experiments. We start by explaining the experiment execution environment (Section 6.1). After which the general metrics used to conduct the data analysis are defined (Section 4.1). Lastly, we present the results of the performed experiments (Section 6.2).

6.1 Experiment Execution

In this section, we explain the specifications of the machine from which we reported the client measurements. The specification are from one common machine which is used across all experiments for an accurate comparison. The specifications of the common machine used for the experiments are provided in Table 7.

For scaling up to 4 and 6 clients, virtual machines from the Google Cloud platform were utilised besides individual laptops. The experiments were manually run through a batch script on each individual client which joins the server and the statistics are collected. The detailed process is explained in Section 5. The bandwidth measurements are aggregated based on seconds, this causes some outliers as seen in the following sections where the results of the experiments are discussed.

Experiment Machine Specification	
Operating System	Elementary OS 5.1.7 (Linux Distro based on Ubuntu LTS 18.04)
CPU	Intel i7 (8th gen, 8 cores, and 4.2 GHz max clock speed)
RAM	16 GB
Docker Version	20.10.0
Docker Compose Version	1.17.1

Table 7: The setup used to perform the experiments

6.2 Experiment Results

This section describes the results obtained from our experiments. We classified the experiments in four classes, namely, variability (Section 6.2.1), performance and scalability (Section 6.3), web- vs. app-based video conferences (Section 6.3.1), and video vs. audio workload (Section 6.3.2).

Before the data visualization, value entries that recorded 0 for each metric are removed as part of data cleaning to improve plots readability. These values can be caused by measurement errors or unstable connectivity. For the variability experiments, this reduces the rows from 7350 to 7220, for performance and scalability experiments from 11024 to 10824, for app vs. web experiments from 5186 to 4660, and for audio vs. video workload from 7221 to 6929.

6.2.1 Variability

To answer **Q1** on time taken for system metrics to become stable, the three conferencing systems were run for 5 minutes and the corresponding CPU percentage over time is shown in Figure 5, memory usage in Figure 6, and bandwidth in Figure 4. From Figure 6, it can be observed that the memory for Zoom web increases linearly up to 5 GB before stabilizing while Jitsi and Teams memory usage is stable around 1GB during the entire run. We conjecture the linear memory increase due to a potential memory leak in Zoom web which has been reported in discussion forumns[19] [20] and the stabilization of 5GB being an application memory limit for a single tab in Google Chrome.

From Figure 5, it can be seen that CPU usage is stable over time with Zoom web having the highest CPU utilization. For bandwidth, it is observed in Figure 4 that initially both Zoom web and Teams take some warm-up time of 20 seconds where the amount of bandwidth consumed is variable before stabilizing whereas Jitsi has a stable bandwidth from the start. We conjecture that the initial warm-up time is taken by the applications to adjust the video and audio bit rate across connections before reaching a steady state while Jitsi does not follow a similar cold start model.

To answer **Q2**, the variability across 4 iterations was observed to finalize the parameters for the remaining experiments. As seen in Figure 7, from the inter-quartile range and error bars that represent the min/max values, the performance of Jitsi for all three metrics is stable across the 4 iterations. A high number of outliers are observed for the CPU percentage. A similar trend was observed for Teams but in Zoom as shown in Figure 8, the variability of CPU usage and memory usage was high but the median was similar across iterations. Even though the variability in Zoom was higher than the other systems, the number of iterations was fixed to 4 and repetitions to 5 due to time constraints.

6.3 Performance and Scalability

To answer **Q4** and **Q3**, experiments were run across multiple number of clients, namely, 2, 4 and 6. As seen in Figure 9, the memory consumption and bandwidth of Zoom web are higher than Jitsi and Teams across all clients. The variation of the median across the iterations is also low. The bandwidth in Teams is extremely low in comparison to the other two systems but this is due to

[19]https://superuser.com/questions/1589202/apparent-memory-leak-using-zoom-web-app-on-chromium-ubuntu-20-04

[20]https://devforum.zoom.us/t/memory-leak-in-zoom-website-meeting-hosting/14503

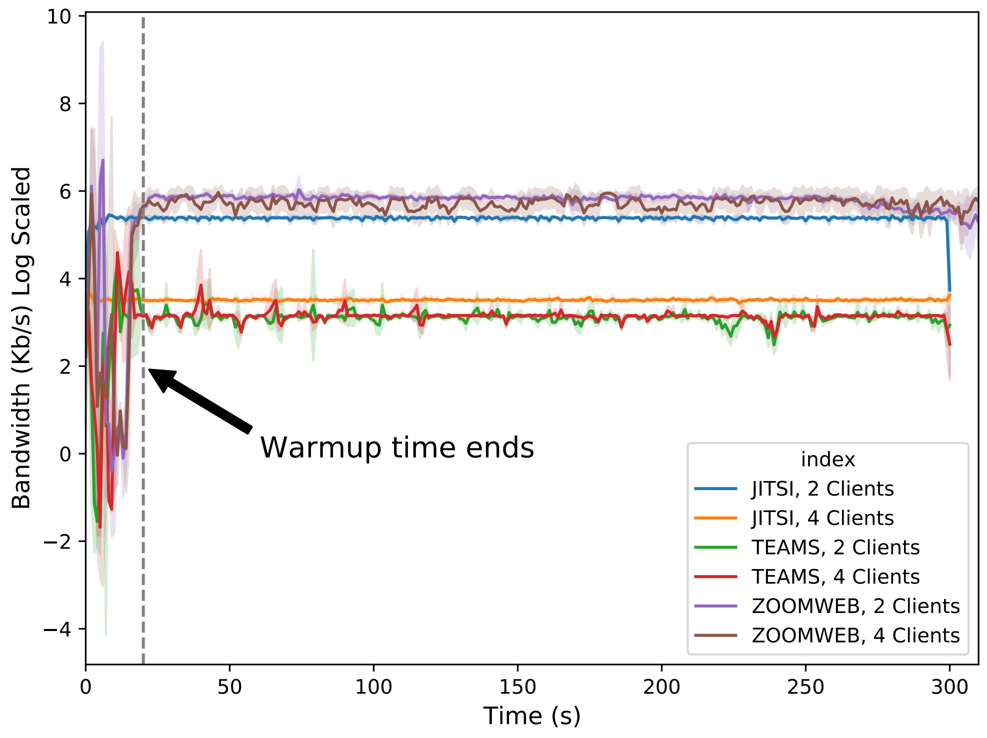


Figure 4: Bandwidth over time.

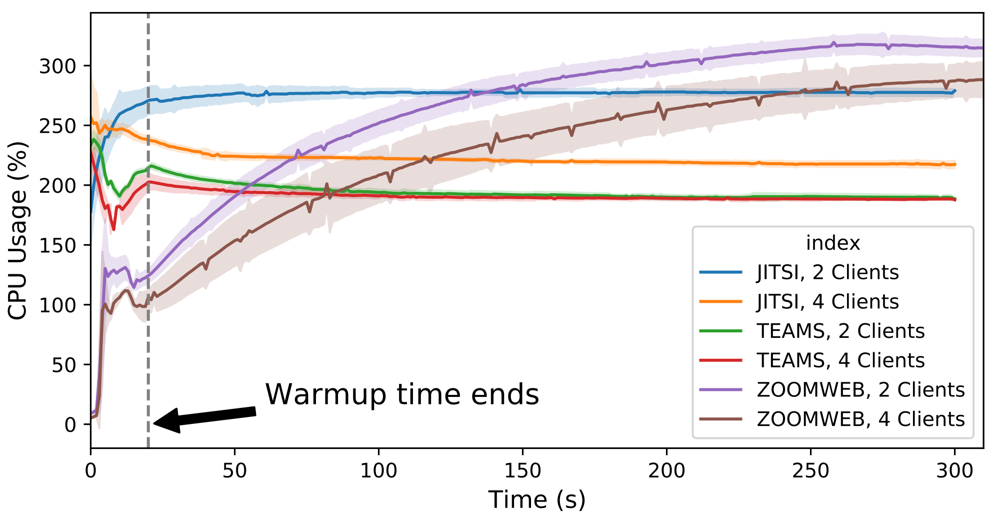


Figure 5: CPU over time.

an architectural choice where only the current speaker video is displayed on all clients leading to a lower bandwidth requirement. Jitsi requires the highest CPU percentage across the systems. This

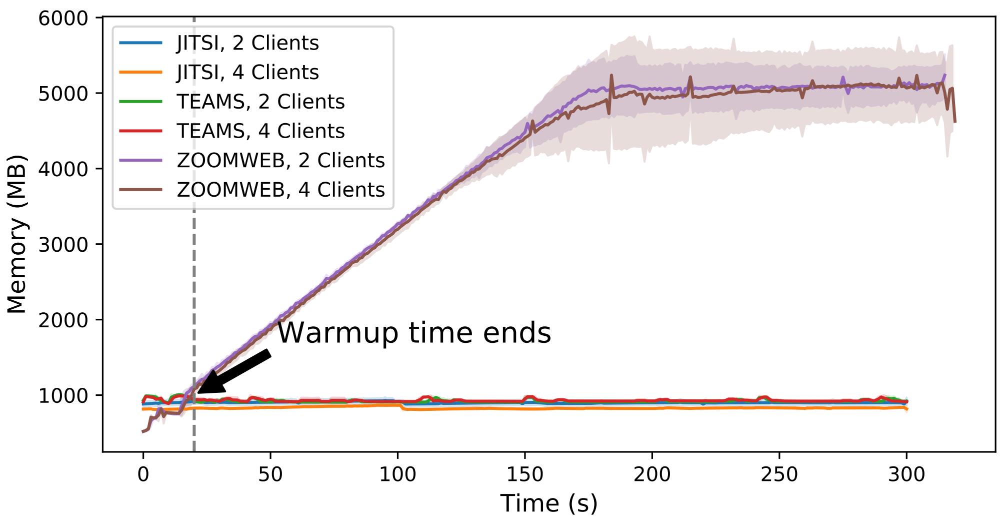


Figure 6: Memory over time.

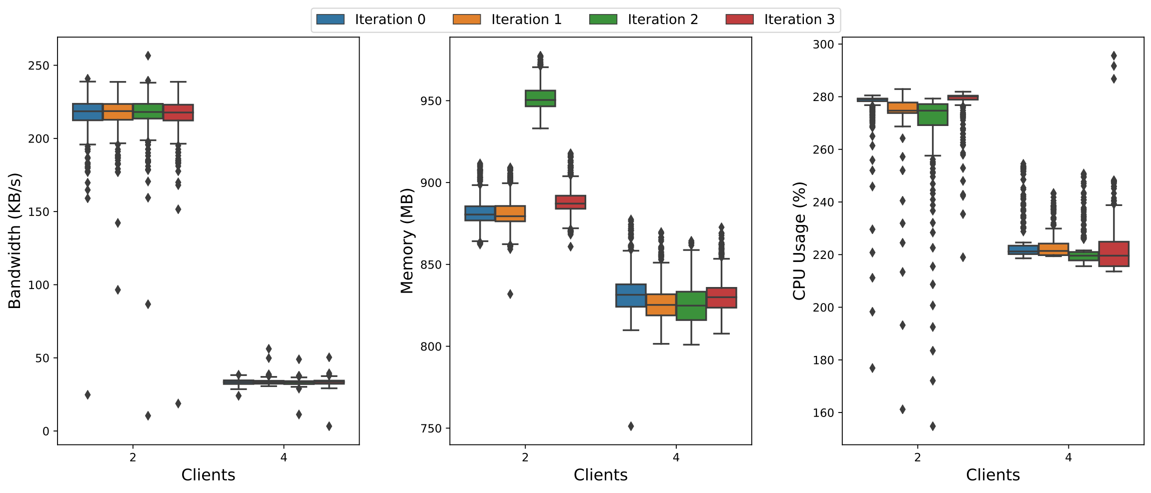


Figure 7: Jitsi execution across 4 iterations.

is a known issue of Jitsi Meet being CPU heavy [21] with forum posts disabling hardware acceleration and other parameters on Chrome to lower CPU utilization. But overall the CPU usage percentage for all three systems make the systems compute heavy.

A surprising observation was the gradual decrease in all three metrics across all three conferencing systems with an increase in the number of clients. A possible conjecture is that a possible decrease in video quality leads to lower values but it is difficult to reason due to the black-box property architectures of Zoom and Teams.

6.3.1 Web vs. App

To answer **Q5**, Zoom web and app were compared across 2 and 4 clients. The number of clients was limited to 4 due to a lack of physical machines available. As seen in Figure 10, Zoom app outperforms Zoom web for all three metrics. To illustrate, CPU utilization and memory consumption are multiple

[21]https://community.jitsi.org/t/high-cpu-utilization-on-client-end/25764/39

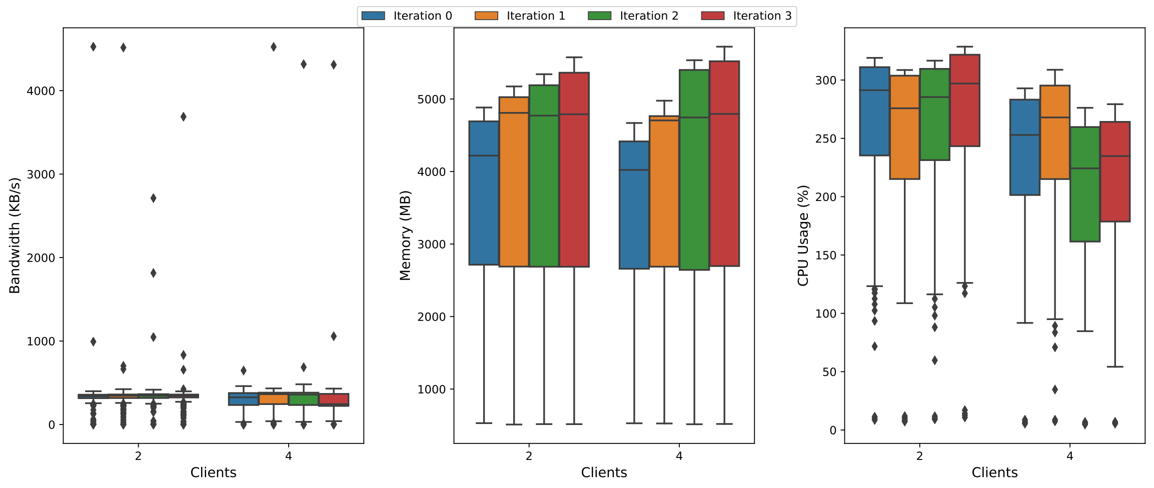


Figure 8: Zoom execution across 4 iterations.

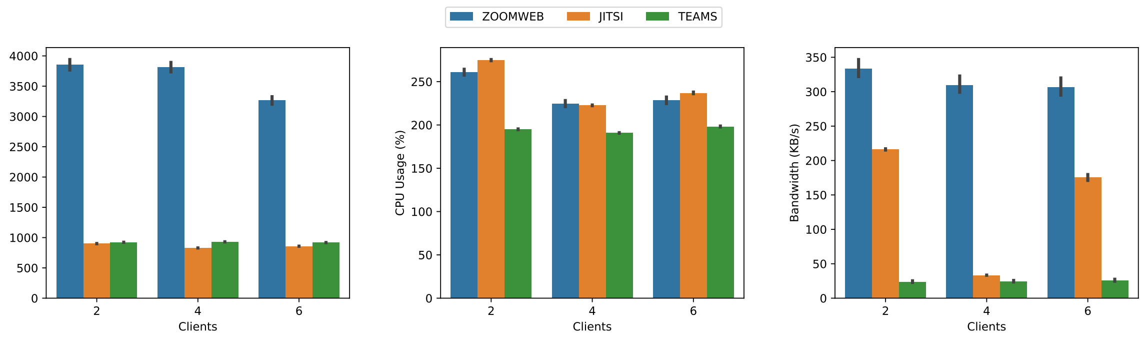


Figure 9: The impact of the number of clients on the memory usage, CPU usage, and network bandwidth.

factors lower in the app. This shows the company has focused on the desktop app which has more features [22], is more popular, and can be developed by directly interacting with system resources instead of serving the content through a browser. Another observable trend is the reduction in metric values with the increase in clients for both the Zoom app and web.

6.3.2 Audio vs. Video

To answer **Q6**, experiments were run on 2 and 4 clients for Jitsi with audio only, video only, audio and video combined workloads. As seen in Figure 11, as expected the video and audio combined workload consumes more bandwidth, CPU percentage, and memory than only audio and only video separately. The audio workload separately consumed the least amount of resources for bandwidth and memory usage which is expected because the data size to be transferred is the least for only audio. The surprising result is the CPU percentage for audio only workload being higher than video only workload.

Another surprising observation is that the difference in CPU utilization and memory utilization across the three workloads is quite less as compared to bandwidth where the difference is in orders of magnitude between audio and audio-video workload, so from the application's perspective, the system requirements/usage remain similar irrespective of the workload.

[22] https://support.zoom.us/hc/en-us/articles/360027397692-Desktop-client-mobile-app-and-web-client-comparison

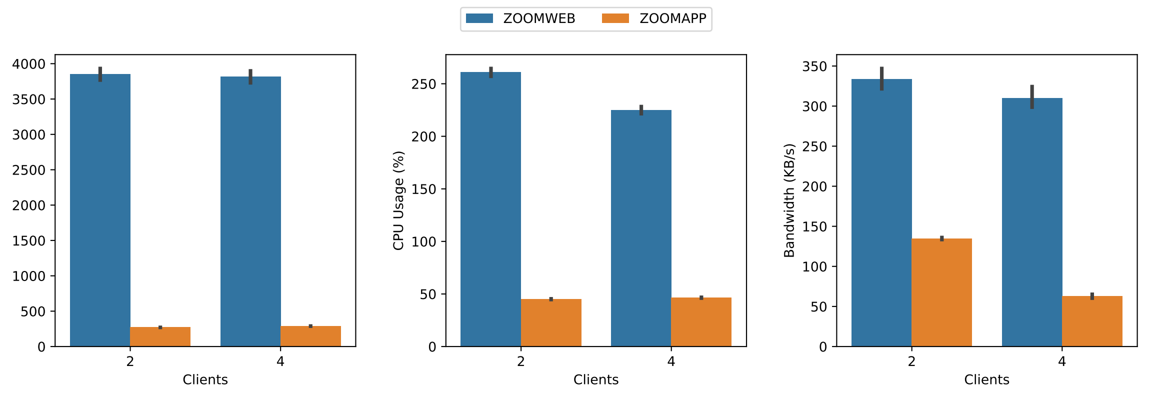


Figure 10: The impact of using Zoom web and app on the memory usage, CPU usage, and network bandwidth.

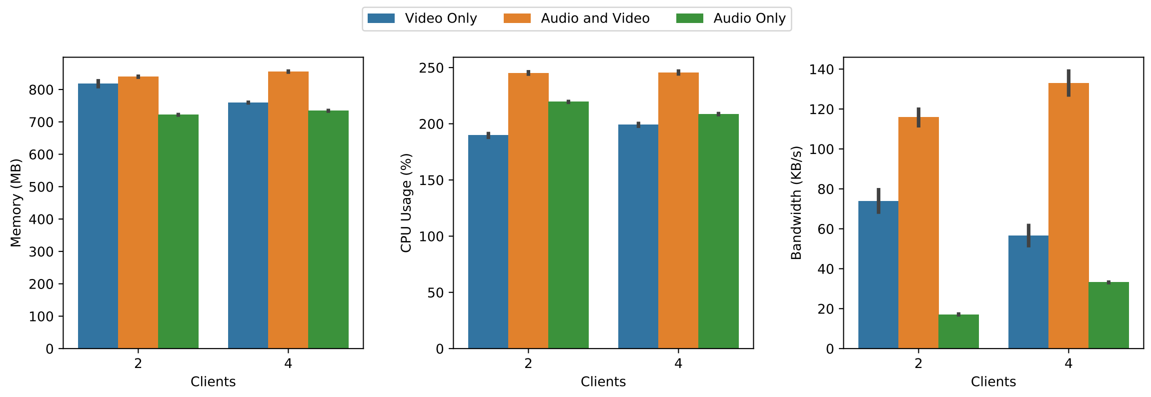


Figure 11: The impact of using audio and video workloads on the memory usage, CPU usage, and network bandwidth.

7 Discussion

In this study, we aimed to answer the research question: *how do popular video conferencing tools perform relative to each other?* To answer this question, we performed experiments using Zoom (web and app), Microsoft Teams, and Jitsi. In these experiments, we measured the bandwidth, CPU usage, and memory usage. Furthermore, we assessed the scalability by repeating the experiments for 2, 4, and 6 clients.

From our experiment results, we observed that after an initial warm-up time (typically between 20 - 30 seconds after joining the conference) the performance of the various video conferencing systems becomes relatively stable. From this, we learned that relatively short experiments (≤ 5 minutes) are adequate to measure the performance of video conferencing systems.

An exception to the stable performance after the warm-up time is the memory usage of Zoom web over time. We showed that after the warm-up time, the memory usage of Zoom web grew linearly up until a certain upper bound. We conjecture that this shows a memory leak and that the upper bound is due to browser-specific restrictions. From this, we recommend users with a memory critical system to refrain from using Zoom web.

A major trend across all experiments was the reduction in bandwidth, CPU usage, and memory usage as the number of clients increases. We did not expect this trend initially as we expected that a higher number of clients would increase the pressure on the network and system. We reason that this is due to the system reducing the quality of the audio and video streams as more clients are added to ensure a stable connection and performance. An exception to this trend is Microsoft Teams as the performance remains stable when increasing the number of clients. So for a stable performance, we would recommend Teams.

When comparing Zoom app and Zoom web, we verified our previous observation that Zoom web performs relatively poorly. This is a striking observation as Zoom is such a large player in the field. By comparing the performance of Zoom app to Zoom web, we learned that the performance of Zoom app is significantly better in all areas. From this, we expect that Zoom mostly focuses on optimizing the application rather than the web-based version. Hence, we recommend users to download the application of Zoom when using this video conferencing system.

When comparing different workloads within the video conference systems such as audio-only, video-only, and both audio and video we had some unexpected results at first. Our initial results did not align well with what we initially expected. After rerunning the experiments, the results aligned more with our expectations where the audio and video workload combined consumes most resources within the system. From the experiment, we recommend users in lower inter-connectivity areas should use audio-only workloads. The initial results showed once again that the internet is an unreliable network and that many external factors can influence the results of the experiments.

Finally, we discuss some of the threats to validity encountered in our experiments and describe them in the following list:

- Lack of interleaving of experiments could cause the results to be variable across web-app where network congestion at a particular time could affect the results of these applications. Our experiment suffers from the threat as the iterations of an experiment were run in succession.

- The WiFi-connectivity being used affect the bandwidth and other values. As video and audio experiments trends changed when connecting to a new WiFi connection. Internet connectivity in an uncontrolled environment. To mitigate the threat, the experiments are repeated.

- The experiments were only run on laptops and VMs having Ubuntu 18.04 OS. The application performance can vary for Windows and Mac users (so the results cannot be generalized).

- Due to a low number of resources, the experiments are only run for a few clients and it is possible that for higher clients the trends can change or some conference clients cannot scale past a certain number.

- The results for only one common machine used for all experiments are plotted, so system-level metrics like CPU utilization and memory usage can be highly dependent on the system's hardware.

8 Conclusion

In this study, we aimed to evaluate and compare the performance of video conferencing tools in the context of distributed systems. To achieve this objective, we designed experiments to compare the performance and scalability of Zoom, Microsoft Teams, and Jitsi. Furthermore, we designed and implemented an experimentation tool that automatically hosts and joins video conferences, and measures the network bandwidth, CPU usage, and memory usage.

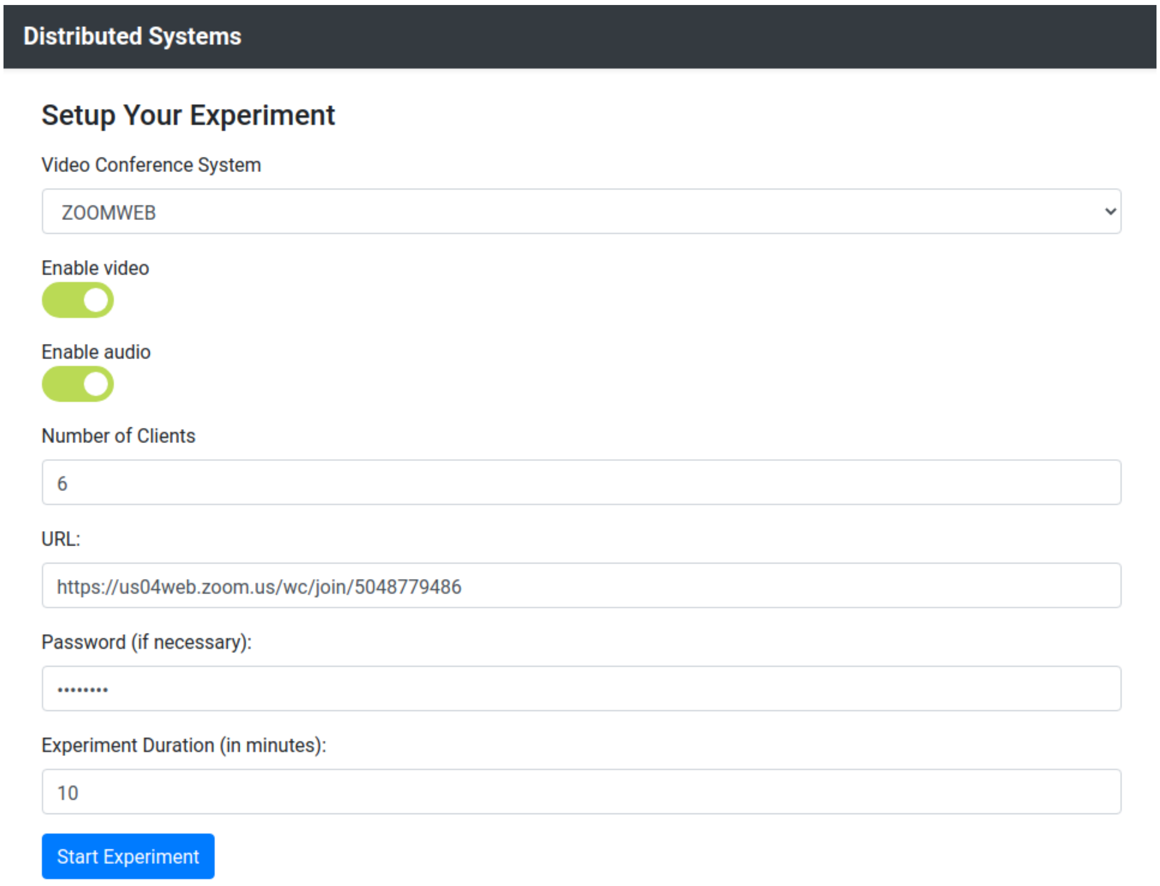
The design and implementation of the experiment tool contributes to the research community as it allows the reproduction of performed experiments. The tool can be extended with other features such as the support of other video conferencing systems (e.g. Google Hangouts) and the inclusion of other metrics to understand the behavior of the video conferencing systems (e.g. jitter).

Based on the performed experiments, we concluded that the performance of the various video conferencing systems is irregular during the warm-up time. After approximately 20 - 30 seconds, the warm-up time has finished and the performance of the video conferencing systems becomes relatively stable. This trend has one major exception regarding the memory usage of Zoom web, this increases linearly over time until a certain upper bound has been reached. Hence, we advise users of video conferencing systems not to use Zoom web on a memory critical device. Another interesting trend that we observed is that as the number of clients increases, the network and system utilization decreases. This contrasted to our initial intuition and we expect that this is due to video conferences reducing the quality of the audio and video streams as the number of clients increases to ensure stable performance.

As this is the first study to evaluate and compare the performance of popular video conferencing systems, many gaps remain. Future work is required to understand the performance when scaling up the video conferences to a larger number of clients (10+). Furthermore, application-level metrics are required to further understand the performance of the various video conferencing systems. Lastly, the relation between the performance of the video conferencing systems and their environment (OS, browser, etc.) needs to be studied to further understand their behavior.

APPENDIX

A Figures



Distributed Systems

Setup Your Experiment

Video Conference System: ZOOMWEB

Enable video: on

Enable audio: on

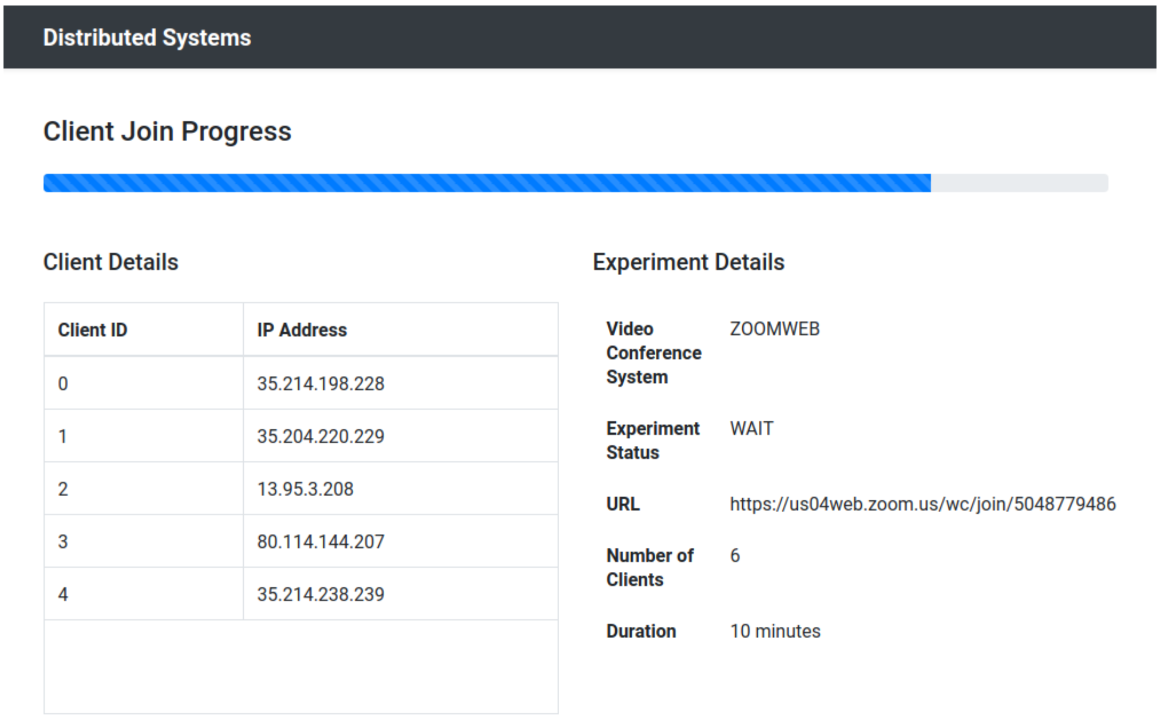
Number of Clients: 6

URL: https://us04web.zoom.us/wc/join/5048779486

Password (if necessary): ••••••••

Experiment Duration (in minutes): 10

Figure 12: Server - Creation of an experiment.



Distributed Systems

Client Join Progress

Client Details

Client ID	IP Address
0	35.214.198.228
1	35.204.220.229
2	13.95.3.208
3	80.114.144.207
4	35.214.238.239

Experiment Details

Video Conference System	ZOOMWEB
Experiment Status	WAIT
URL	https://us04web.zoom.us/wc/join/5048779486
Number of Clients	6
Duration	10 minutes

Figure 13: Server - Status of an experiment.

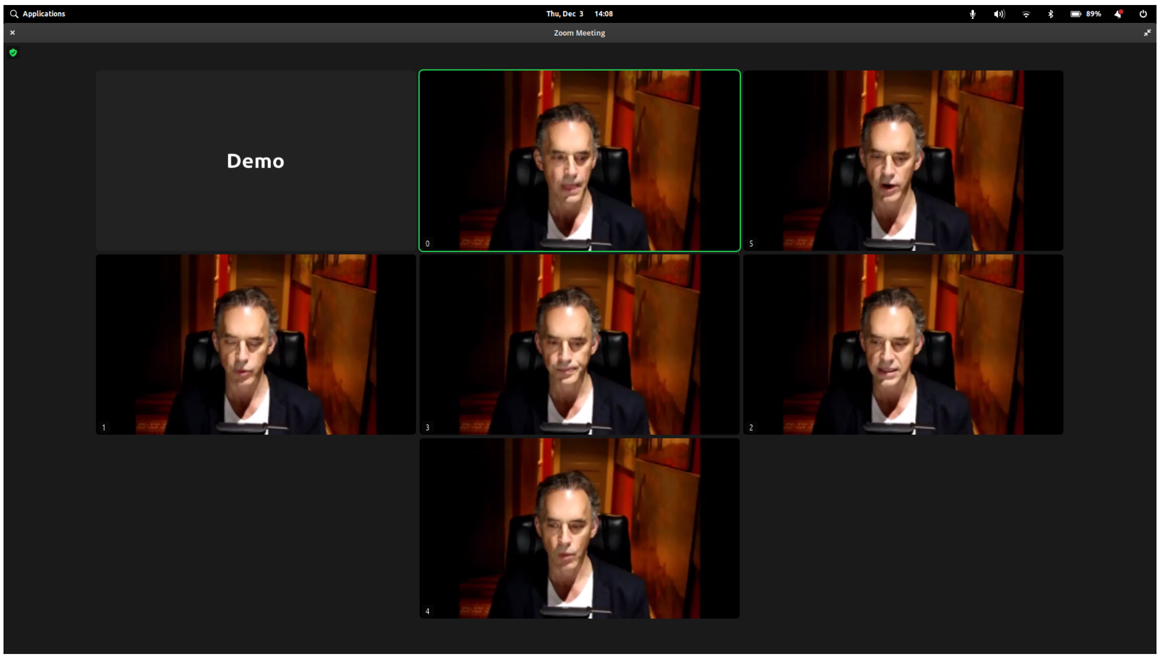
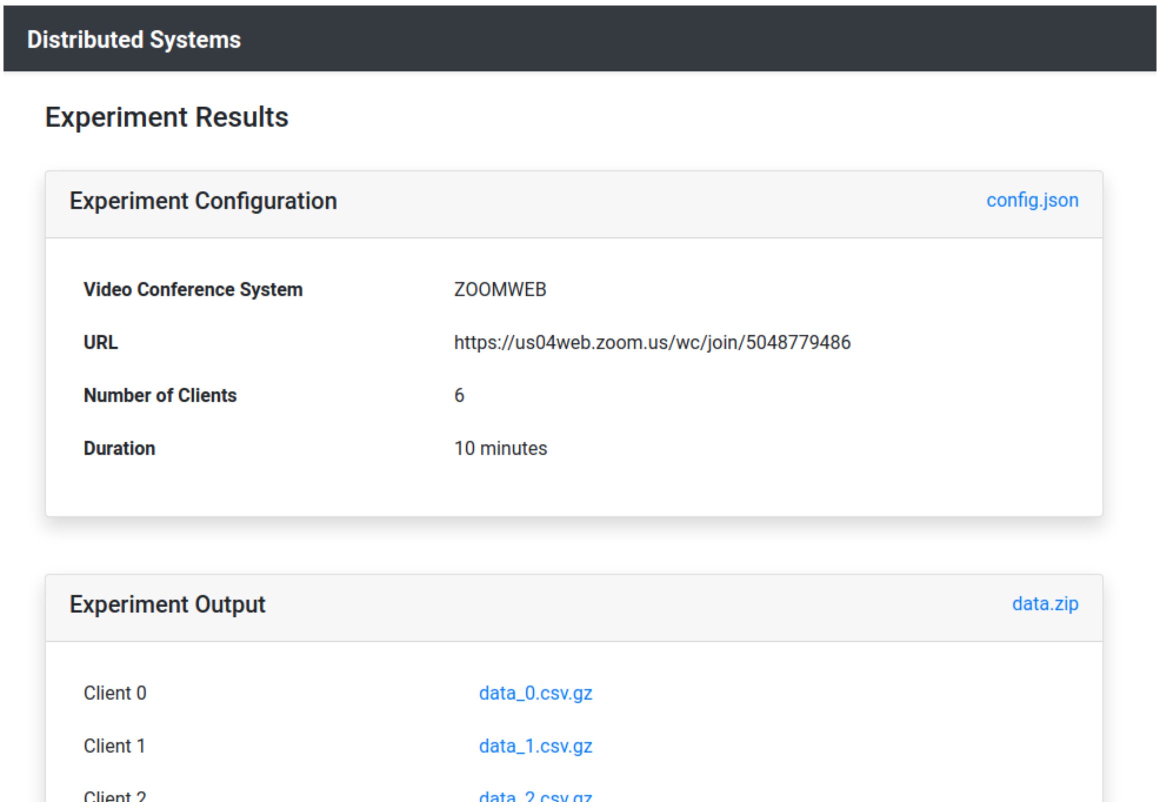


Figure 14: Preview of a standard experiment as an external observer.
Disclaimer: The video was used as we wanted a speaker talking directly to a webcam. No political statement was intended by the authors. In hindsight, we should have changed the experiment workload using less controversial material.



Distributed Systems

Experiment Results

Experiment Configuration config.json

Video Conference System	ZOOMWEB
URL	https://us04web.zoom.us/wc/join/5048779486
Number of Clients	6
Duration	10 minutes

Experiment Output data.zip

Client 0	data_0.csv.gz
Client 1	data_1.csv.gz
Client 2	data_2.csv.gz

Figure 15: Server - Results of an experiment.

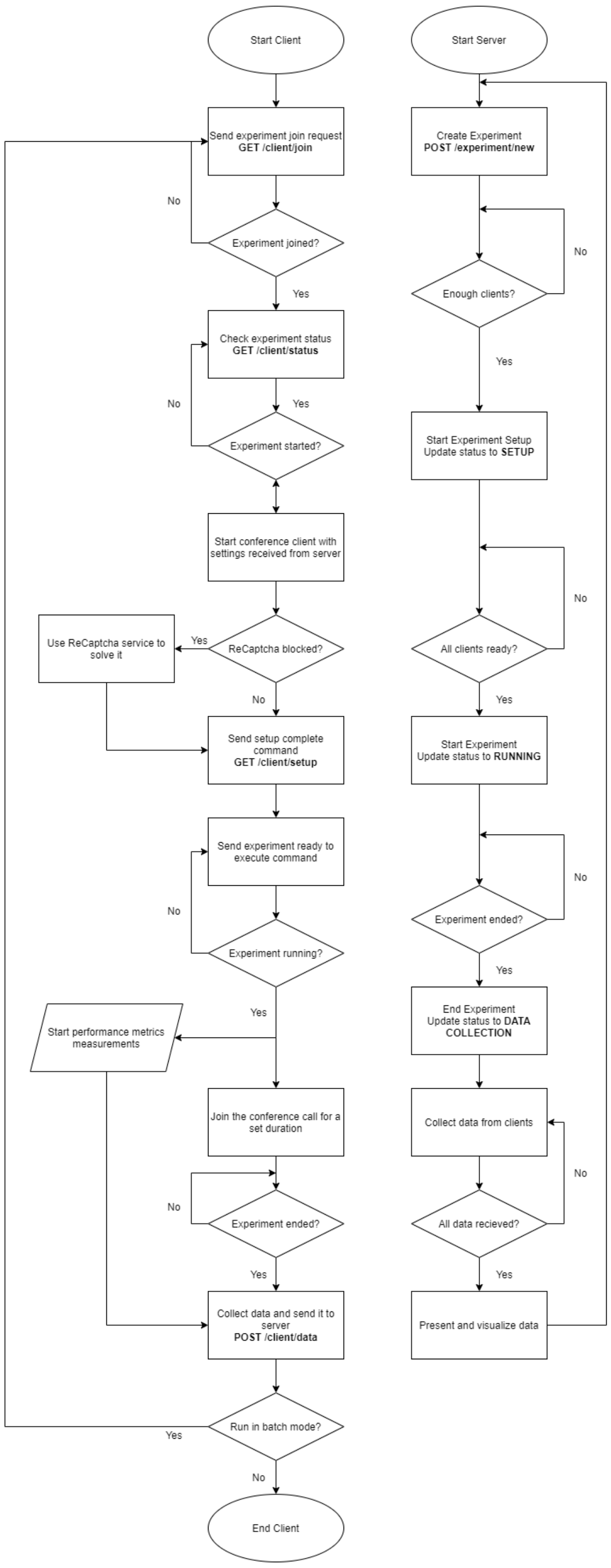

Start Client
Start Server
Send experiment join request GET /client/join
Create Experiment POST /experiment/new
No
Experiment joined?
Yes
No
Enough clients?
Check experiment status GET /client/status
Yes
No
Yes
Experiment started?
Start Experiment Setup Update status to SETUP
Start conference client with settings received from server
No
Use ReCaptcha service to solve it
Yes
ReCaptcha blocked?
All clients ready?
No
Yes
Send setup complete command GET /client/setup
Start Experiment Update status to RUNNING
Send experiment ready to execute command
No
No
Experiment running?
Experiment ended?
Yes
Yes
Start performance metrics measurements
End Experiment Update status to DATA COLLECTION
Join the conference call for a set duration
Collect data from clients
No
No
Experiment ended?
All data recieved?
Yes
Yes
Collect data and send it to server POST /client/data
Present and visualize data
Run in batch mode?
Yes
No
End Client

Figure 16: Flow-chart of the experiment framework.

B Time Sheet

Time sheets	
Activity	Time Spent (hours)
Meetings	
Group meetings	20
Meetings with supervisor	10
Development	
Framework design	20
Framework development	180
Experiments	
Experiment Execution	20
Analysis	
Discuss outcomes	15
Make figures	15
Report	
Set-up document structure	10
Write report	50
Report review	5
Wasted Time	
Rerunning experiments	5
Total time spent	**350**

Table 8: Time spent as group effort per section of the project.

References

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

[2] Erik Brynjolfsson, John J Horton, Adam Ozimek, Daniel Rock, Garima Sharma, and Hong-Yi TuYe. Covid-19 and remote work: an early look at us data. Technical report, National Bureau of Economic Research, 2020.

[3] Dai Clegg and Richard Barker. *Case Method Fast-Track: A Rad Approach.* Addison-Wesley Longman Publishing Co., Inc., USA, 1994.

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

[5] Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent Van Beek, Giulia Frascaria, Ahmed Musaafir, and Sacheendra Talluri. The atlarge vision on the design of distributed systems and ecosystems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1765–1776. IEEE, 2019.

[6] Hyunwoo Nam, Kyung-Hwa Kim, Doru Calin, and Henning Schulzrinne. Youslow: A performance analysis tool for adaptive bitrate video streaming. *SIGCOMM Comput. Commun. Rev.*, 44(4):111–112, August 2014.

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

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

[9] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. Experimentation in software engineering. Springer US, 2012.

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