A Measurement Tool for Videoconferencing User Experience

by

Caroline Jin

B.S., Computer Science and Engineering, Massachusetts Institute of Technology (2023)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by: Caroline Jin
Department of Electrical Engineering and Computer Science
May 12, 2023

Certified by: Mohammad Alizadeh
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

A Measurement Tool for Videoconferencing User Experience

by

Caroline Jin

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The COVID-19 pandemic forced people to work remotely and use videoconferencing software like Zoom in their daily lives. While people are returning to their pre-pandemic lifestyle, many still depend on videoconferencing software. As a result, application developers need to regularly monitor user experience in terms of video quality, stalls, and network conditions, and identify areas of potential improvement. Companies and academic researchers focus user experience analysis on dual-endpoint, controlled conditions that do not reflect everyday user calls. Gathering data on a large scale without knowing the network structure and getting permission for traffic analysis takes time and effort. Such large-scale experiments often use lengthy procedures to obtain the right permissions and deploy monitoring infrastructure in the middle of the campus network.

In contrast to existing approaches, an ideal measurement application would merely run on users' devices without cooperation from the other endpoint that they're conversing with. Such an application enables researchers to collect network statistics across a wide range of Internet conditions at a fine-grained level without significant overheads. This thesis proposes the Single Endpoint Zoom Measurement Application (SEZMA) that computes and logs network and video metrics when a user is on a Zoom call and sends metric logs to a centralized server. In addition to providing insights for users and researchers, the application aims to be explanatory, usable, lightweight, and privacy-preserving.

Thesis Supervisor: Mohammad Alizadeh
Title: Associate Professor

Acknowledgments

I would have been unable to complete my thesis work without the help of the following people. Thank you to my advisor Professor Mohammad Alizadeh for providing guidance through this whole process and steering me toward my project goals. Thank you to PhD student Vibhaalakshmi (Vibhaa) Sivaraman for meeting with me every week to lend mentoring support and to help me overcome the challenges I faced during this project and thesis process. I am tremendously grateful for their passion and time for my thesis work.

Finally, I would like to thank my family and friends for helping me test my measurement application. Their support throughout this process has been invaluable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The COVID-19 pandemic changed people's everyday lives, forcing people to rely on videoconferencing software like Zoom, FaceTime, Facebook Messenger, and Google Meet to communicate with colleagues, peers, and relatives. From December 2019 to May 2020, the number of Zoom daily meeting participants grew at a rate of 1900% [7]. As of 2022, there are 300 million Zoom participants every day [7]. Even though companies are transitioning towards more in-person meetings in the last year, industries like marketing & advertising, travel, and technology still have a higher number of meetings per person in the post-outbreak world compared to pre-pandemic [11]. Video calls for personal use have also grown. As of March 2020, 47.6% of US adults use FaceTime to talk to family and friends, followed by 44.1% for Facebook Messenger and 31.5% for Zoom [5]. Videoconferencing software may not replace all forms of in-person communication, but it is still an important tool for users.

As video calls have become more widely integrated into firms and people's personal lives, users want these calls to have nearly the same quality as in-person interactions. Video calls fall short of the user's expectations. One reason for this is that videoconferencing software oftentimes have bad video quality. In a survey of 1,755 users conducted by Dialpad, the top issue is poor audio quality. When users encounter audio issues like choppiness, they turn off the video, which solves these audio issues. This shows that audio problems may be caused by video problems. The survey also shows that video degradation is one of the top three issues causing videoconferenc-

ing fatigue [11]. Video degradation refers to blurry or blocky video frames, frozen screens, and choppy video (i.e., movements in video are irregular). When such situations occur, it is hard to diagnose the root cause or even capture the frequency of these situations. For example, Zoom provides a support page to diagnose video degradation, but it does not pinpoint specific network problems [6]. Resolving issues with poor video quality is challenging due to the lack of data on real-time network events that cause quality degradation. However, if there is a way to identify these events in the real world with large-scale experiments on videoconferencing software like Zoom, it would help improve such software and make the network infrastructure that supports video calling more robust.

Large-scale measurements on videoconferencing software would provide insight into network events that cause stalls and poor video quality. Current studies focus on comparing the performance, such as video quality, between different videoconferencing software like Zoom and Google Meet. However, these studies use dual-endpoint video calls where both endpoints are users who must cooperate with each other to allow for performance measurements of different videoconferencing software. Dual-endpoint cooperation includes forcing the sender and receiver of the calls to send to each other packets from the same video [10] or actively extracting packets on both ends in test Zoom calls [19]. Other methods have attempted to observe all endpoints in a call using an edge router where all traffic passes through [12, 19]. These methods operate well on a small scope, where researchers can work with a few videoconferencing clients or have access to a critical device in the computer network. However, scaling these approaches incurs significant overheads to gather data from all endpoints in a call or deploy in-network measurement devices and obtain permissions for access to sensitive traffic data. An ideal measurement application would collect and evaluate data on only one endpoint, using the storage and computing resources at that endpoint, and only require the permissions of a single user.

In this thesis, we design a measurement application focused on Zoom with the following design goals: explanatory, usable, lightweight, and privacy-preserving. The application should be explanatory: it should inform users of what network events

caused bad video quality during a Zoom call. The application aims to be usable: it must be easy to install and set up, and require only one endpoint to run in a Zoom call. The application should be lightweight: it should minimize storage and computing resources. The application should respect user's privacy: no identifying user information in the data is left from the user's device.

With these design goals, we develop the Single Endpoint Zoom Measurement Application (SEZMA), an application to run streamlined processes that computes metrics as data is collected and send metrics to a centralized server where researchers can analyze these metrics. SEZMA has one process for analyzing network data collected from the Zoom call and another process for analyzing video data. The network data is the packet data sent between users of the Zoom call. The video data is the video shown on the display screen of the user's device. As the data is collected, both the network and video processes compute and record metrics, such as the size of the packet for the network data and the video frame quality for the video data, into metric logs. Once the Zoom call ends, both processes stop running and the metric logs are sent to the centralized server for determining network events that impact video quality.

For our initial iteration, SEZMA runs as a command-line application on MacBooks. We evaluate whether SEZMA achieves the design goals by running it with Zoom calls and measuring CPU and memory usage. We observe that SEZMA can distinguish between Zoom calls with good and bad perceived video quality using video frame quality metrics like Laplacian and identifying video issues like stalling using frozen frame metric. In simulated network conditions, SEZMA is able to reflect patterns induced by the network (*e.g.,* a decrease in bandwidth correlates with a decrease in the values of network metrics like the amount of network data received per second). SEZMA also incurs only reasonable overheads when running on a user's device: it's CPU Core usage is similar to that of Zoom, and its memory consumption is within a factor of 2–3 of Zoom. SEZMA has been open-sourced at https://github.com/cjin2019/SEZMA.

Chapter 2

Relevant Work

Although researchers have studied video quality [13, 15] and videoconferencing software [10, 18, 12, 19], no previous work has developed a single-endpoint application that allows users to identify network conditions affecting their everyday video call quality. Video quality research focuses on the user's video experience and examines to what extent poor video quality decreases user engagement [13, 15]. Studies on videoconferencing software primarily use dual endpoints to benchmark network performance measurements like latency [10, 18, 12] or monitor network vantage points that all traffic passes through to reveal traffic patterns on campus [12, 19]. This work provides a single endpoint measurement application that analyzes network effects on video quality in conferencing settings, extending the existing measurement methodologies like Michel et al. [19] to compute metrics for network and video quality.

Video Quality and User Engagement. Previous research on video quality shows that video degradation can negatively impact user engagement when watching a video. Dobrien et al. [13] quantify quality metrics such as buffer ratio, the fraction of the total video length spent in buffering, and user engagement measurements like playtime, the duration of viewing one video. They find that a 1% increase in buffering ratio can reduce user engagement by more than 3 minutes in a 90-minute live video. Karthikeyan et al. [15] extend this research by investigating the effects of video quality on user engagement across different streaming TV apps and other TV-related platforms. Their results show that an increasing number of buffering interruptions

causes viewers to watch fewer TV programs over time rather than stopping immediately. While this research focuses on TV, their results reveal the importance of good video quality for user experience in a wider range of video applications, including videoconferencing software and the need to pinpoint problems in degradation.

Simulated Video Call Experiments. When analyzing videoconferencing software, a common approach for evaluating performance uses a measurement testbed with an emulated network environment. Such approaches enable a high degree of reproducibility. For instance, Chang et al. [10] use virtual video conferencing clients where the meeting host sends a video with a blank screen that flashes an image every two seconds to the recipient. They compute latency by observing when the host sends the first big packets corresponding to the first burst and when the recipient receives those packets. Chang et al. [10] also analyze the video quality of each videoconferencing software, comparing the video output on the call with the original reference video. Once all frames are processed, they measure the video call quality difference with the original video using metrics like peak signal-to-noise ratio (PSNR) and structural similarity (SSIM). MacMillan et al. [18] set their testbed using a pre-defined network environment setup and having one participant send pre-recorded talking-head video to the other. They study the effects of constraining network parameters, like downlink capacity, on video metrics, like video resolution, by measuring the video width in pixels on the receiver side.

However, a common downside of such controlled approaches is that coordinated clients run under simulated network conditions where there is low variance in packet loss and latency. Specifically, the above approaches only include test calls where the host sends a one-way pre-recorded video with known patterns like periodic flashes [10] under network environments with controlled and consistent packet loss and latency [18]. Furthermore, these methodologies involve recording the entire video, storing it, and then processing the video to compute video quality metrics for analysis. This is impractical for everyday video calls due to the huge storage needed on the user's devices and the privacy concerns of storing the video. Hence, an ideal measurement application should not require coordination among participants, must run in any network

environment, and computes video quality metrics immediately after some video data is collected.

Zoom Analysis. Few video conferencing studies focus on analyzing Zoom to understand how it handles traffic and system and its underlying design. Choi et al. [12] investigate their college campus's traffic data to glean information on network paths and traffic patterns. Using campus logs from an edge router, they discover that the Zoom infrastructure switches between server mode (all clients retrieve data from a centralized server) and peer-to-peer (P2P), and that Zoom uses separate channels for audio, video, and screen sharing. Michel et al. [19] provide further analysis and extract Zoom packet headers, such as media type and video frame. In SEZMA, we leverage their parsing semantics to implement the application to infer network events that cause poor video quality. The drawback of both works is that they actively monitor packets from the sender and the receiver, and observe all traffic passing through a border router. Such an approach necessitates identifying edge routers and configuring them to capture packet information. Specifically, this approach requires deploying a router, which is feasible on a large campus or corporate network that is over-provisioned and can handle high volumes of network data. Poor network conditions like high fluctuations in bandwidth and large packet loss do not occur frequently on such large campus networks. Running measurements on these networks does not provide insights into understanding video call quality for all types of networks. This approach also constrains analysis to video calls where one participant is using the campus network. In contrast, SEZMA relies on only one endpoint and does not require in-network support for analysis, providing more flexibility to the user and the application developer.

Chapter 3

Single Endpoint Zoom Measurement Application (SEZMA)

For this thesis, we designed and developed SEZMA that provides a timeline of network events explaining the user's Zoom quality during each video call that the user makes. The goals are for the application to be:

1. Explanatory: The richness of data captured should enable meaningful interpretation of it for the user to identify network events causing poor Zoom call quality.

2. Usable: Users should be able to install and run SEZMA with minimal overheads and interventions on their end. SEZMA should only use one endpoint and not need coordination from the other(s).

3. Lightweight: SEZMA should not consume too much storage, CPU, or memory on the user's device.

4. Privacy-preserving: Identifiable user data should not be sent off from the user's device.

To achieve these goals, SEZMA runs on the user's device, conducting analysis in real-time as the network and video data are received during a Zoom call. SEZMA

runs the collection and computation of network and video data in separate processes, maximizing the amount of data that can be captured and analyzed. Once the Zoom call ends, computation results from the network and video data are sent to a centralized server, where they are then independently post-processed to glean insights. With this design, SEZMA will enable researchers, engineers, and users to study daily Zoom calls easily since participants can run the SEZMA during their Zoom calls, and researchers can retrieve and analyze the metric logs from the centralized server.

We describe the application design choices and implementation by outlining the application architecture (§3.1). From there, we will dive deeper into the main components of SEZMA: the Network Analysis Module (§3.2), the Video Analysis Module (§3.3), and the centralized server (§3.4). We also describe how users can install and set up SEZMA on their device in (§3.5). Additionally, we evaluate the design goals of SEZMA by running it with Zoom calls and conducting microbenchmark results (§4).

3.1 SEZMA System Architecture

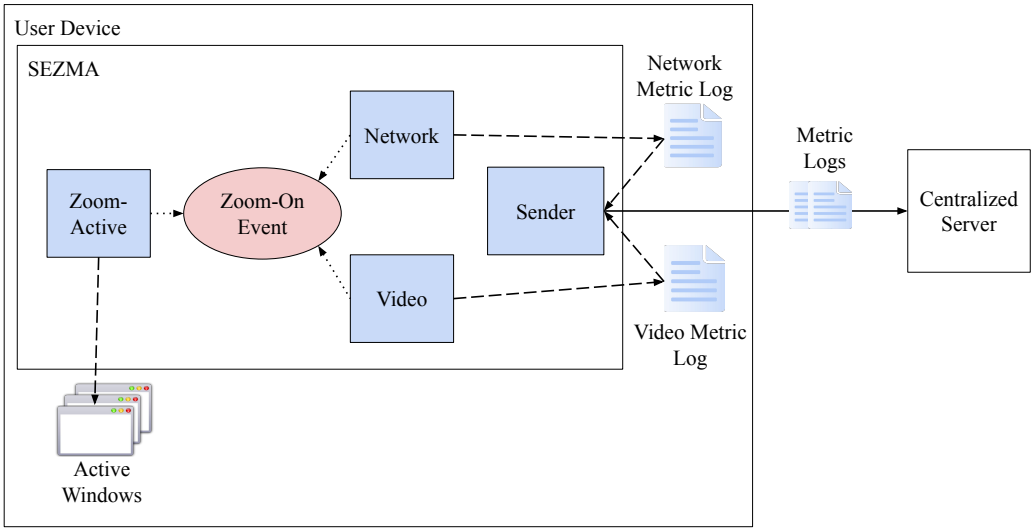

Figure 3-1: SEZMA System Architecture.

We created SEZMA to run end-to-end as shown in Fig. 3-1. The user opens and starts SEZMA on their device. SEZMA starts with three modules: Zoom-Active,

Network, and Video. These three modules share an event called Zoom-On, which captures whether a Zoom call is running. The Network and Video modules wait on the Zoom-On event, while the Zoom-Active module checks for the presence of a Zoom Meeting window. If a Zoom Meeting window exists, it turns the Zoom-On event from off to on, which initiates the Network and Video modules to start collecting data, analyzing, and recording metrics to metric logs. When the Zoom call ends, the Zoom-Active module turns the Zoom-On event off. Once this event is turned off, the Zoom-Active module finishes running, and the Network and Video modules complete their work and stop. SEZMA then initiates the Sender module to send the metric logs to the centralized server. After the metric logs are sent, SEZMA stops.

Zoom-Active Module. The first module that SEZMA runs is the Zoom-Active module, which verifies if a Zoom Meeting window exists. This module lists all available active windows on the user's device and checks whether the Zoom Meeting window is inside this list. If the Zoom call has not started, the Zoom-Active module repeatedly checks every 3 seconds. Once the call has begun, the Zoom-Active module sets the Zoom-On event from off to on. During the call, the Zoom-Active process continues to check every 3 seconds to determine whether the call ended. When it does not see the Zoom Meeting window in the list, it waits 10 seconds to check again. If there is still no window found, the Zoom-Active module sets the Zoom-On event to off. This module enables the user to start SEZMA whenever they want without needing to begin a call.

The Zoom-Active module uses Python's `Quartz` package to get the list of active windows. The Python `Quartz` package is a wrapper around Quartz-related frameworks on macOS, such as Core Graphics. We use `Quartz`'s version of a Core Graphics function called `CGWindowListCopyWindowInfo` that enables us to get a list of active windows containing fields with information, such as the window ID and the window dimensions. For each active window in the list, we check whether the field "Window-Name" is "Zoom Meeting". If a window has this, the Zoom Meeting window exists. Otherwise, it does not.

Analysis Modules. The Network (§3.2) and Video (§3.3) modules in SEZMA run in

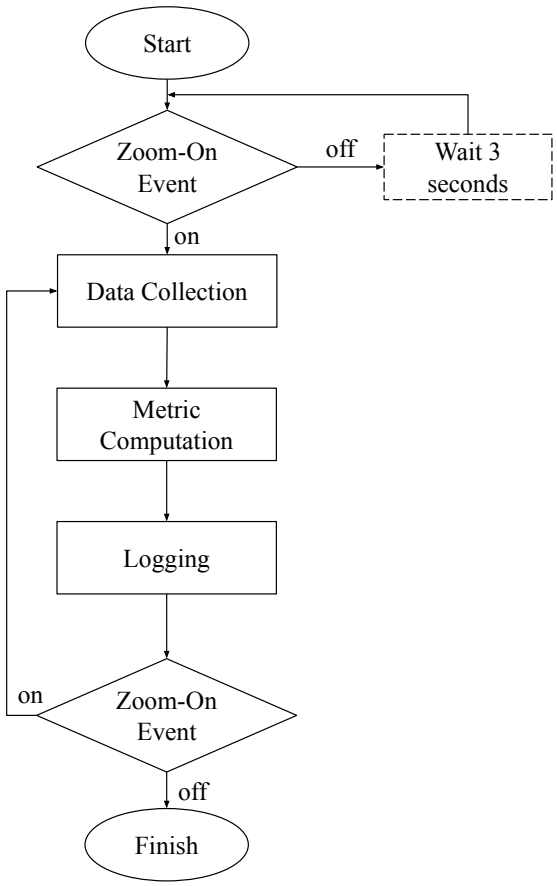

Figure 3-2: Network and Video Modules Workflow.

parallel. The Network module processes one packet at a time, while the Video module processes one frame at a time. Specifically, each received packet or frame undergoes the submodules of Data Collection, Metric Computation, and Logging before the next received packet or frame goes through the same submodules. For example, in the Video module, the Data Collection submodule gets the video frame and sends it to the Metric Computation submodule. The Metric Computation submodule computes video metrics and sends the metrics to the Logging submodule, which records these metrics into a log. The purpose of processing the information streams and logs in real-time and online is to keep the data only in memory and reduce the storage of the user's device to just metric logs. This also is better for privacy since there are no recorded files that need to later be protected from malicious actors. Both the video and network modules have a Logging submodule to write these metric logs, which are

CSV files saved on the user's device (until the call is done).

Aggregating Metrics. After the Network and Video modules finish running, SEZMA starts the Sender module (Fig. 3-1), which uploads the metric logs to the centralized server. The metric logs do not contain any identifiable information, but just a call ID to distinguish between calls and location information (IP address) that can be used for correlations later. The Sender module takes the metric logs and sends them through an HTTP POST request. In some cases, the metric logs may take more than 1 MB, which may cause sending the request to take a few seconds. To ensure that they do not take up too much space on the user's device, the Sender module deletes the metric logs after they are sent to the server. The centralized server aggregates all metric logs from the user so that researchers with access to the server can conduct an analysis of network events impacting Zoom video quality.

3.2 Network Analysis Module

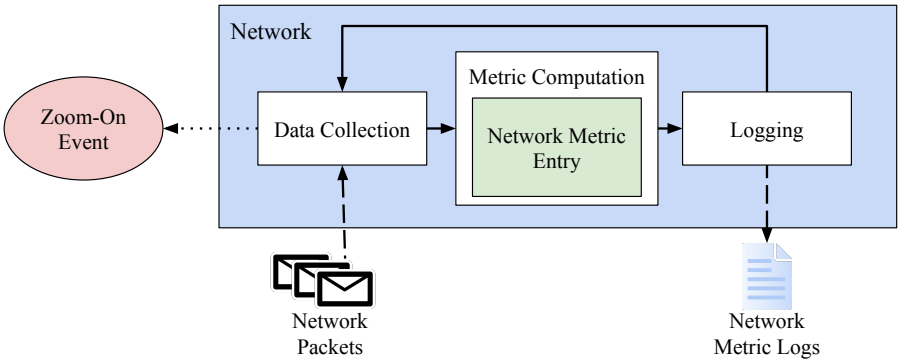

Figure 3-3: Network Module System Diagram.

The Network module captures and analyzes Zoom packets received by the user's device. Each packet goes through the submodules in Fig. 3-3, during which the packets are collected, and the network metrics are computed and logged. The Data Collection submodule extracts only the video packets that correspond to the Zoom call and gathers packet information useful for calculating network metrics. Once all necessary packet fields are parsed, the Metric Computation submodule takes the

packet information and calculates metrics related to network events, such as packet loss. Finally, the Logging submodule (§3.1) records these network metrics into a CSV file that's later sent to the centralized server. This section will focus on only the Data Collection and Metric Computation submodules.

Data Collection. Obtaining network packets involves a filtering process where we use packet headers to ensure that we are retrieving relevant packets sent by other participants in the Zoom call. The filtering process uses two filters. Using one filter is not enough since filters, like Berkley Packet Filters (BPF), only remove packets based on layers defined in the Internet Protocol, like Transmission Control Protocol (TCP), and are not fine-grained enough to detect only Zoom video packets.

The first filter restricts the collected packets to only User Datagram Protocol (UDP) data packets with a destination corresponding to the user's device. Video-conferencing software like Zoom uses UDP as the standard protocol for video traffic, where fast transmission is prioritized over reliable transmission. The first filter implementation uses a Python package `Scapy` [3] to get the UDP packets that have a destination IP address of the user's device, which is obtained using `Scapy`'s `conf.iface`. While there are many available tools such as `tcpdump` and Wireshark [4], those tools require running a separate process in SEZMA, which may incur additional overheads. We pick `Scapy` since it enables easy packet parsing layer-by-layer and integrates the Data Collection submodule easily with the rest of SEZMA.

All UDP packets passing through the first filter are parsed in a second filter to determine whether they are Zoom video packets. The second filter also simultaneously records some parsed data for later network metric computations. Michel et al. [19] reveal that Zoom sends its video data in a specific packet structure. Through systematic controlled experiments, they discover that Zoom fragments its video frames over several packets and adds unencrypted headers to each packet in addition to the headers specified in Request For Comment (RFC) specifications 3550 [21]. We use their schema to parse the payload of the packets received from the first filter.

In the parsing procedure, we parse the header from the outermost to the innermost layer following the Zoom header schema, as shown in Tab. 3.1. If we cannot fully

Layer Name	Features
UDP Header	• Time of Packet Received • Packet Source IP Address • Packet Destination IP Address • Packet Source Port Number
Zoom Media Header	• Frame Sequence Number • Number of Packets Per Frame • Media Type: RTP Video, RTP Audio, RTP Screen Share, RTCP Sender Report, Keep Alive
RTP Header	• Payload Type: Video, FEC

Table 3.1: Packet information used to filter out relevant packets. The top row of the table corresponds to the outermost layer of the packet, and each row below corresponds to the next layer in the packet.

parse each layer, we assume that the packet is not a Zoom video packet and continue to process the next packet that enters the Data Collection submodule. The parsing procedure begins with the UDP header, which contains the time of the packet received, the source IP address, and the source port number. The source IP address and source port number determine how to parse the subsequent layers. Zoom structures its inner layers differently based on how data is sent among participants. Specifically, Zoom sends traffic among participants in a call either via an external Zoom server (server mode) or using Peer-to-Peer (P2P) transmissions [12, 19]. When it uses the server mode, it adds a special header [19]. We identify the mode based on the source port in the UDP header. If the source port is 8801, Zoom is operating in server mode. Otherwise, the two Zoom clients send UDP packets directly to each other.

For server mode, we disregard the special header and continue to parse the packet's Zoom Media header. In P2P mode, the only additional header the packet contains is the Zoom Media header. This header contains the media type, which differentiates video packets from other media sources that Zoom outputs such as audio, as well as other information like the frame sequence number and the number of packets sent of that frame. The next parsed layer is the RTP header, which contains a payload type to determine if the packet received is the original Zoom packet or a Forward Error Correction (FEC) packet [21]. Zoom has a patent for its FEC procedure and notes that FEC packets are redundant packets to deal with lost video frame packets. As higher packet loss is identified, an increasing number of FEC packets are sent to the receiver of the call [16]. The next layers are the Network Abstraction Layer (NAL) [14] and Fragment Unit A (FU-A) [22]. Neither layer provide new information, so we ignore those headers and stop the parsing process.

Throughout the parsing process, SEZMA stores in memory the relevant packet information in a Packet Info Entry, as described in Tab. 3.2. The Packet Info Entry is passed directly to the Metric Computation submodule. We find that the entries in Tab 3.2 are useful because they enable the Metric Computation submodule to group packets based on frames using "Frame sequence number" and time using "Time the packet is received." They also provide insights into network events. For instance, the difference between the "Expected number of packets received" and the actual packets received and "Is an FEC" may help reveal packet loss.

Packet Info Entry
Frame sequence number
Time the packet is received
Packet size (bytes)
Expected number of packets received
Is an FEC type (boolean)

Table 3.2: Useful packet information for the Metric Computation submodule of the Network module.

Metric Computation. The Metric Computation submodule computes each network metric using the packet info entries retrieved from the Data Collection submodule.

The network metrics are aggregated under three categories (Tab. 3.3): packet (*e.g.,* packet size), frame (*e.g.,* the number of FEC packets), and short time-scale (*e.g.,* the amount of network data captured per second). Considering these different categories makes it easier to correlate the network metrics with the video metrics (§3.3) during the graphical analysis. The following paragraphs describe the metrics we decide to compute.

We use *packet metrics* to identify network events like packet loss or reduced bandwidth during the call. The metrics we consider are the packet time (*i.e.,* the time the packet is received by the user's device) and the duration between packets (*i.e.,* the difference between packet times). In good network conditions with minimal packet loss and delays, we observe that the duration between packets on average is 0.01 seconds. A duration between packets greater than 0.01 seconds reveals potential packet loss or delay in the network. We also compute the packet size or the total number of bytes of the Zoom video packet received. The packet size may vary based on the network condition. For instance, decreasing packet size over time may reveal when the network bandwidth is constrained.

We use *frame metrics* to get a summary of the network environment. We group packets by frames using frame sequence numbers to compute the frame-level metrics: the fraction of packets missing and the number of FEC packets. The fraction of packets missing is the ratio of the number of packets missing over the expected number of packets. The expected number of packets is retrieved from Packet Info Entry (Tab. 3.2). We get the number of packets missing by computing the difference between the expected number of packets of a frame and the actual number of packets received for that frame (as measured during the call) that are not FEC packets. The fraction of packets missing shows whether packet loss occurred. A fraction of 0 corresponds to all packets received for a video frame, and a non-zero value indicates that some frame packets have been lost. As the fraction of packets missing increases, the metric shows that the network is experiencing packet loss. Another metric is the number of FEC packets. We count the number of packets where the "Is an FEC type" value is true for a given frame (Tab. 3.2). An increase in the number of FEC packets over

time also indicates an increase in packet loss levels. Some of these loss events can be recovered from if the FEC packets are received, while some might create an extended glitch or frozen video experience.

We also want to consider overall network trends in a Zoom call by computing *short time-scale metrics* over a one-second period. These metrics are the amount of network data (Kbps), the number of packets per second, the number of frames per second, and the amount of FEC data (Kbps) averaged over a time interval. Changes in these measurements may capture trends like a decrease in network bandwidth and an increase in packet loss over the duration of the Zoom call, but show them over longer time scales than individual packets or frames. For example, a decrease in the amount of network data and an increase in the amount of FEC data over many seconds may indicate that there is a prolonged increase in packet loss.

3.3 Video Analysis Module

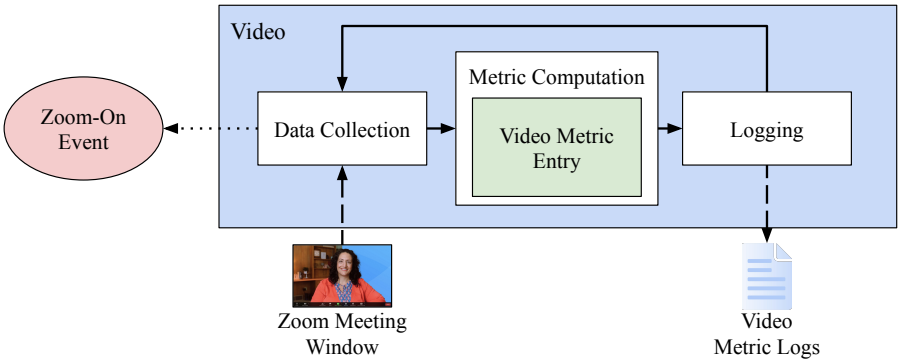

Figure 3-4: Video Module System Architecture.

The Video module captures and analyzes the video of the Zoom call. Every image of the Zoom call is immediately analyzed to record video metrics into a metric log (Fig. 3-4). The Data Collection submodule takes an image of the Zoom call at a constant rate and sends the image to the Metric Computation submodule. The Metric Computation submodule preprocesses the image and calculates metrics that show the quality of the image (*i.e.,* blockiness and blurriness) and video (*i.e.,* stalling). The

Network Metric Type	Metrics
Packet	• Packet Time • Duration Between Packets (s) • Packet Size (B)
Frame	• Fraction of Packets Missing • Number of FEC Packets
Short Time-Scale	• Amount of Network Data (Kbps) • Number of Packets Per Second • Number of Frames Per Second • Amount of FEC Data (Kbps) • Fraction of Packets Missing Per Second

Table 3.3: Network Metrics Computed.

Logging submodule records these metrics. This section describes in detail the Data Collection and Metric Computation submodules.

Data Collection. The Data Collection submodule takes images of the Zoom call by capturing screenshots of the Zoom Meeting window. This submodule uses Python's `Quartz` package to select the Zoom Meeting window. With this package, the Data Collection submodule can get the image of the Zoom Meeting window without having it be full-screen and in front. The screenshot image is then converted into a 2D array that can be processed by the Metric Computation submodule. Since the entire Zoom Meeting window is captured, the Data Collection submodule is suited to collect frames for only two-party calls. We discuss this limitation and potential workarounds for multi-party calls in §5.

The Data Collection submodule must capture screenshots of the Zoom Meeting window at high enough constant frames per second (fps) to get enough data showing video degradation occurrences detectable by the eye. The frame capture rate of each window can be set by updating SEZMA's configuration file (§3.5) but is upper-bounded by how long the metric computation takes. The metrics with more computations, like those based on a trained model, involve a slower capture rate. Section §4.3 reveals the fastest fps per different metrics and their corresponding CPU usage. We observe that we can capture up to 30 fps on MacBook M1 and up to 5 fps on older MacBooks like MacBook Pro 2019.

Metric Computation. As the Data Collection submodule for video data sends each frame captured, the Metric Computation submodule computes metrics (Tab. 3.6) relating to the video frame (*e.g.,* blurriness of the video frame) or the overall video (*e.g.,* stalling). The following discusses the metrics SEZMA computes.

To measure video frame quality, we compute video frame metrics by applying no-reference image quality metrics to each video frame that seek to capture how natural each received frame is to the human eye. We experiment with Natural Image Quality Evaluator (NIQE) [20], Perception-based Image Quality Evaluator (PIQE) [9], and Laplacian. We chose these metrics instead of standard distance-based visual metrics such as peak signal-to-noise ratio (PSNR), structural similarity index measure (SSIM), and learned perceptual image patch similarity (LPIPS) because we run measurements using a single endpoint (the receiver) and do not have access to ground-truth images from the sender to compare each received video frame with.

Each metric we use differs in how "natural" is defined and the setup it requires. Laplacian computes over the entire image to determine how blurry an image is. A low Laplacian score indicates blurrier pictures. The PIQE algorithm divides the images into 16-by-16-sized blocks and computes a distortion score to measure how much distortion (*i.e.,* specks of pixels and blockiness) is in each block. The PIQE algorithm uses a threshold to classify which blocks should be considered distorted and computes the mean of the distortion score over all the distorted blocks to get

the PIQE score. Higher PIQE scores correspond to images of poorer quality [9]. NIQE uses trained models that capture the naturalness of a scene [20]. High scores correspond to lower-quality images, and quality is learned based on the dataset it is trained on.

Our preliminary analysis suggests that the video frame metrics do not always reflect image quality. For instance, Fig. 3-5 compares a video frame of perceived good quality with a frame of perceived poor quality and Tab. 3.4 lists the exact scores for each image and each metric. As expected, the poor perceived image scores higher than the good quality image for NIQE and lower for Laplacian. However, the poor image had a lower PIQE score than the better image. This may be because PIQE scores only account for blockiness and noise, not blur. Thus, PIQE is unable to capture the quality of images that have a constant blur. NIQE also sometimes produces very low scores for images of poor quality, as shown in Fig. 3-6 and Tab. 3.5. The Laplacian score for the bad image is only slightly lower compared to that of the good image, and the PIQE score follows as expected. The bad image contains blockiness, which NIQE and Laplacian scores may not quantify as well as PIQE.

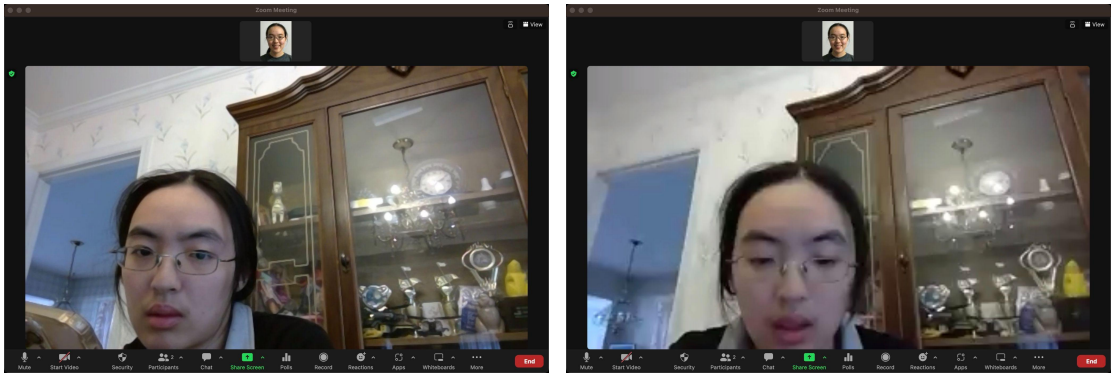

Figure 3-5: The video frame scores where PIQE is incorrect. The left image is an image of good perceived quality, and the right is an image of poor perceived quality.

Since each metric captures different features of a video frame, we decide to have SEZMA configurable so that the user can decide which set of video frame metrics they want to use and which video frame quality they want to detect (§3.5). NIQE, which is a model-based metric, may take longer computing time compared to Laplacian, which

Metric Type	Score for Good Image	Score for Poor Image
NIQE	11.32	13.10
PIQE	44.15	39.78
Laplacian	158.88	157.55

Table 3.4: Video frame metric scores for good and poor images in Fig. 3-5.

is a non-model-based metric. This longer computation time causes frame capture to be lower when using NIQE than when using Laplacian (§4.3).

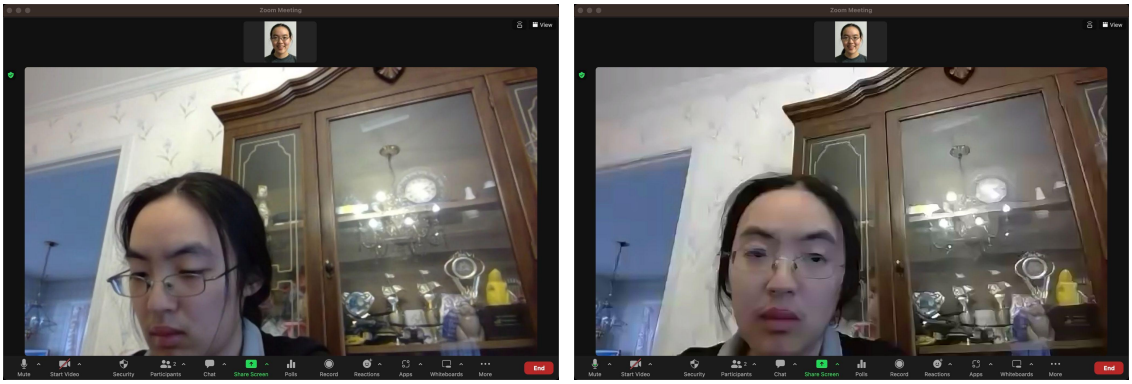

Figure 3-6: The video frame scores where NIQE is incorrect. The left image is an image of good perceived quality, and the right image is an image of poor perceived quality.

Metric Type	Score for Good Image	Score for Poor Image
NIQE	10.75	11.34
PIQE	47.83	44.36
Laplacian	159.99	159.88

Table 3.5: Video frame scores for good and poor images in Fig. 3-6.

We also want to compute a metric that considers the video as a whole. Specifically, we focus on analyzing video stalling using the frozen frame metric. To determine whether stalling occurs, SEZMA keeps track of the previous and current frames and compares pixel-by-pixel whether the two are the same. If the two frames are the same, stalling occurred and the frozen frame metric is 1. Otherwise, the value is 0. We face issues where the Zoom Meeting window contains a small view of the user's self, causing a pixel-by-pixel comparison to fail even if stalling occurs. To solve this issue, SEZMA is restricted currently to only two Zoom participants and requires users

to toggle their view modes on Zoom to "Speaker" and "Hide Self View" during a Zoom call. In the future, SEZMA could crop the speaker video screen of the window (§5).

Video Metric Type	Metrics
Per-Frame Non-Model-Based	• Laplacian • PIQE
Per-Frame Model-Based	• NIQE
Overall Video	• Frozen Frame

Table 3.6: Video Metrics Computed.

3.4 Centralized Server

We maintain a centralized server that hosts different versions of SEZMA to enable efficient distribution of SEZMA among users. Users download SEZMA via a webpage. The webpage sends an HTTP GET request specifying the application filename containing the version they desire. The HTTP server process takes the GET request, retrieves the corresponding file from the SEZMA file repository, and sends the SEZMA file to the user. Currently, we have SEZMA available for Mac with Arm and Intel. We scope it to be a command-line tool on MacBook that users can run on Terminal. We develop SEZMA in Python due to the availability of data processing and metric evaluation libraries for SEZMA's Network and Video modules. Since SEZMA is built using Python, we use Python's `Nuitka` library [1] to convert the Python codebase into a macOS application. The `Nuitka` library compiles Python codebase to C source code and applies optimizations in compile time that enables SEZMA to run faster than running it through Python.

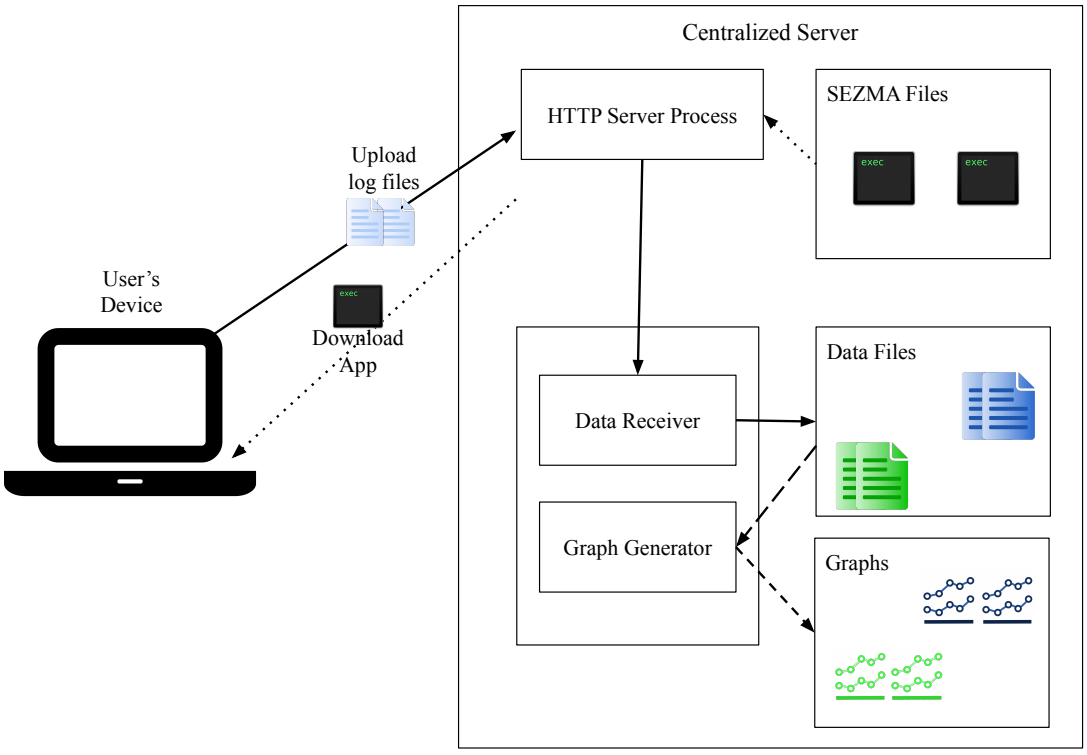

Figure 3-7: Centralized Server System Diagram.

The centralized server also stores all metrics received from users. It runs an HTTP server process that allows users to send POST requests to upload the network and video metric logs (Fig. 3-7). The HTTP server process takes the metric logs and sends them to the data receiver, which stores them into a data files repository. The data files repository accumulates all metric logs that researchers can later access to conduct analysis.

Periodically, the centralized server runs a graph generator on the received metric logs to create graphs of these metrics and stores these graphs in the graph repository. These graphs identify specific network events and trends that may impact the video quality of a call. We decide to put the graph generator on the server to enable researchers to customize how they want to analyze the metric logs. Further, this avoids the memory and computation overhead of creating the graphs on SEZMA running on the users' devices.

Command	Request Type	Request Fields	Response
`/upload`	POST	Metric Logs	
`/download`	GET	Application Filename	SEZMA and Config Files

Table 3.7: HTTP requests sent to the centralized server.

3.5 Application Installation and Set Up

The users visit the centralized server on `http://sezma.csail.mit.edu/` where we list links of the zip files that the user can download. The zip file contains SEZMA and an application configuration file, `config.json`. Each zip file name corresponds to the SEZMA version available, where each filename specifies the macOS version and the processor architecture (*i.e.,* Arm or Intel). After the users download the zip file, they must update the configuration file given in `config.json` where they can set the parameters of how SEZMA will run on their device. The following shows the parameters that users can configure:

```
{
  'OutputDirectory': '/full/path/to/output/directory',
  'FrameRate': 5,
  'VideoFrameMetricsUsed': ['LAPLACIAN', 'NIQE', 'PIQE'],
}
```

1. OutputDirectory: This parameter specifies the file path of the directory to store the metric logs outputted from the Network and Video modules. These metric logs are stored temporarily in the directory specified by OutputDirectory and immediately deleted once sent to the centralized server.

2. FrameRate: This value specifies the frame capture rate for the Video module. The default rate may be too high, since the rate is set to be the fastest possible capture rate for the given video frame metrics, specified in VideoFrameMetricsUsed.

3. VideoFrameMetricsUsed: This setting has users supply video frame metrics

they want to use as a list. The default `config.json` file uses all metrics that SEZMA can compute.

Once the users have updated the configuration file, they can start SEZMA on the Terminal of their devices.

Chapter 4

Results

We evaluate SEZMA by running test Zoom calls showing particular network events that may cause video degradation. In addition, we also run several Zoom calls to measure the memory and CPU impact of this application. We run SEZMA on a 2022 MacBook with an M1 processing chip in two-person Zoom calls that last 4–8 minutes for all experiments. Sections §4.1 and §4.2 show that SEZMA is explanatory in real network conditions and simulated conditions respectively, and Section §4.3 shows SEZMA is lightweight.

4.1 Real Network Conditions

This section compares a good Zoom call with a bad Zoom call when run in the wild without any control over the network bandwidth or loss pattern, and illustrates that SEZMA is able to distinguish between the two. We define a bad Zoom call as having many occurrences of video degradation, like frames that are frozen and have blockiness or blur. A good Zoom call will consist of a few of the aforementioned video degradation. When doing our initial analysis, we focus on the network metrics (*i.e.,* *packet metrics* like the packet time, *frame metrics* like the fraction of packets missing and the number of FEC packets, *short time-scale metrics* like the number of frames per second, the amount of network data (Kbps), the amount of FEC data (Kbps), and the number of packets per second), following the video quality trends. For video

quality, we focus on the frozen frame metric and video frame metrics like Laplacian and NIQE because they provide useful insights for our analysis. A good video frame quality has a high Laplacian score and a low NIQE score, and a bad video frame quality has a low Laplacian and a high NIQE score.

Good Zoom Call. We expect a good Zoom call to have overall higher frame quality during the call, compared to a bad Zoom call where video degradation occurred. Fig. 4-1 shows higher Laplacian scores and lower NIQE scores on average for a good Zoom call compared to a bad call.

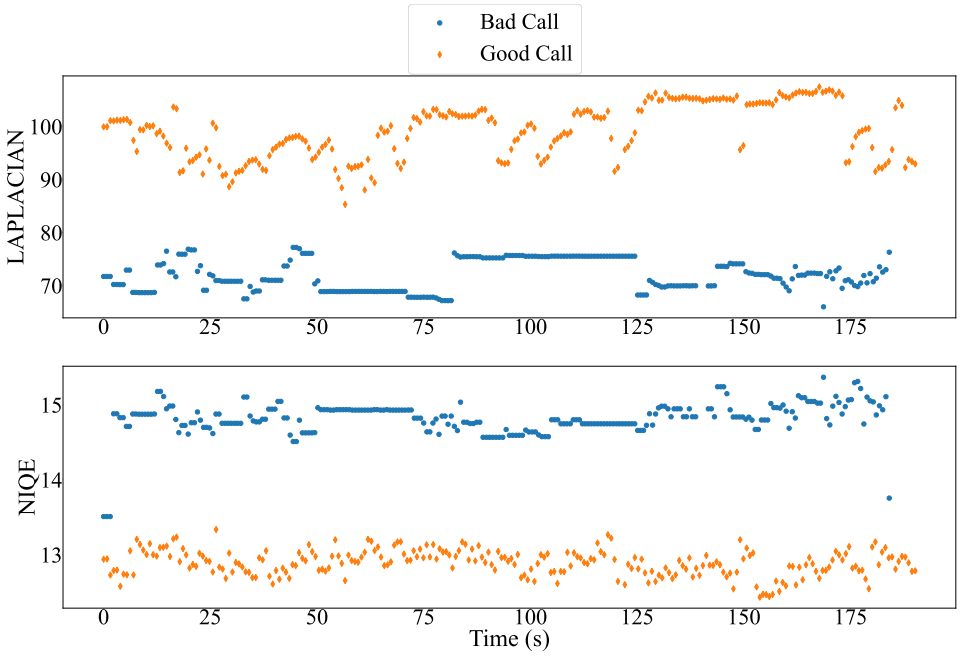

Figure 4-1: A timeline of video frame quality between the good and bad Zoom call.

Under stable network conditions with high bandwidth and minimal packet loss, we expect the Zoom call to be good. The *short time-scale metrics* should stay relatively horizontal over time. Fig. 4-2 shows that the number of frames per second varies between 23 and 31, the amount of network data varies between 1337 Kbps to 1778 Kbps, the amount of FEC data stays at 0 throughout the call, and the number of packets per second varies between 144 and 188. The amount of network data sent is relatively constant at an average value of 1605 Kbps. To understand packet loss characteristics, we consider frame metrics like the fraction of packets missing and the

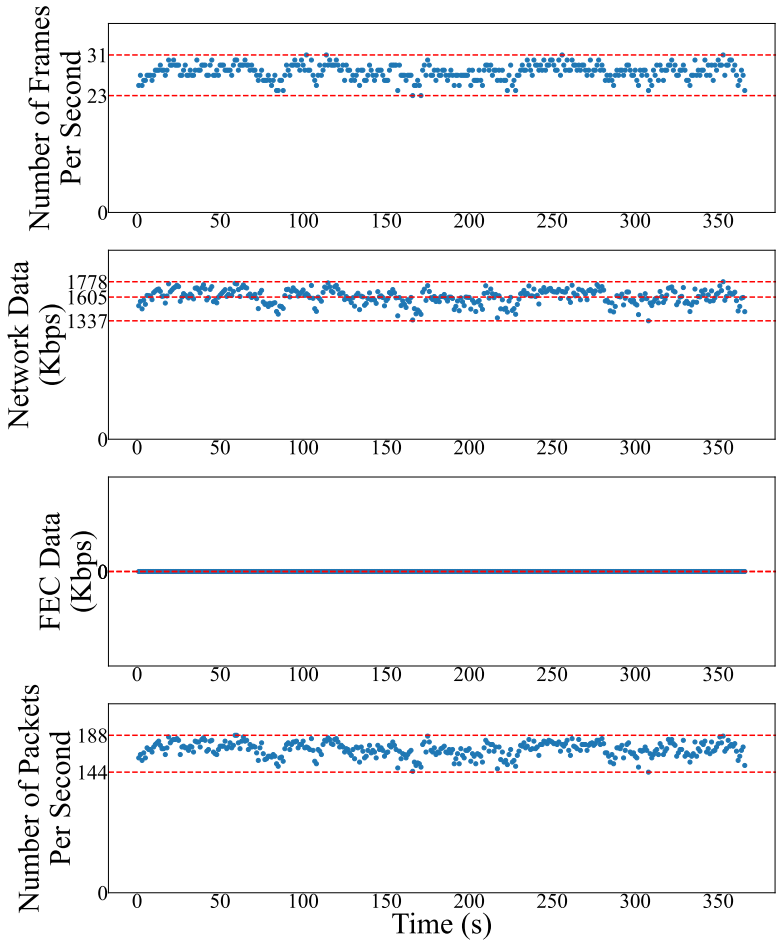

Figure 4-2: A timeline of *short time-scale metrics* during the good Zoom call.

number of FEC packets. In Fig. 4-3(1), the good call has only 2% of frames that have any packets missing, and all other frames are completely received. Fig. 4-3(3) shows that no FEC packets are received. These metrics indicate that there is low packet loss

Bad Zoom Call. The network metrics of the bad Zoom call show particularly high packet loss. Fig. 4-3(2) shows that 19% of frames are partially or completely lost during the call, and Fig. 4-3(4) shows that 40% of the frames received have one or more FEC packets. The gaps between packet time received during the call identify

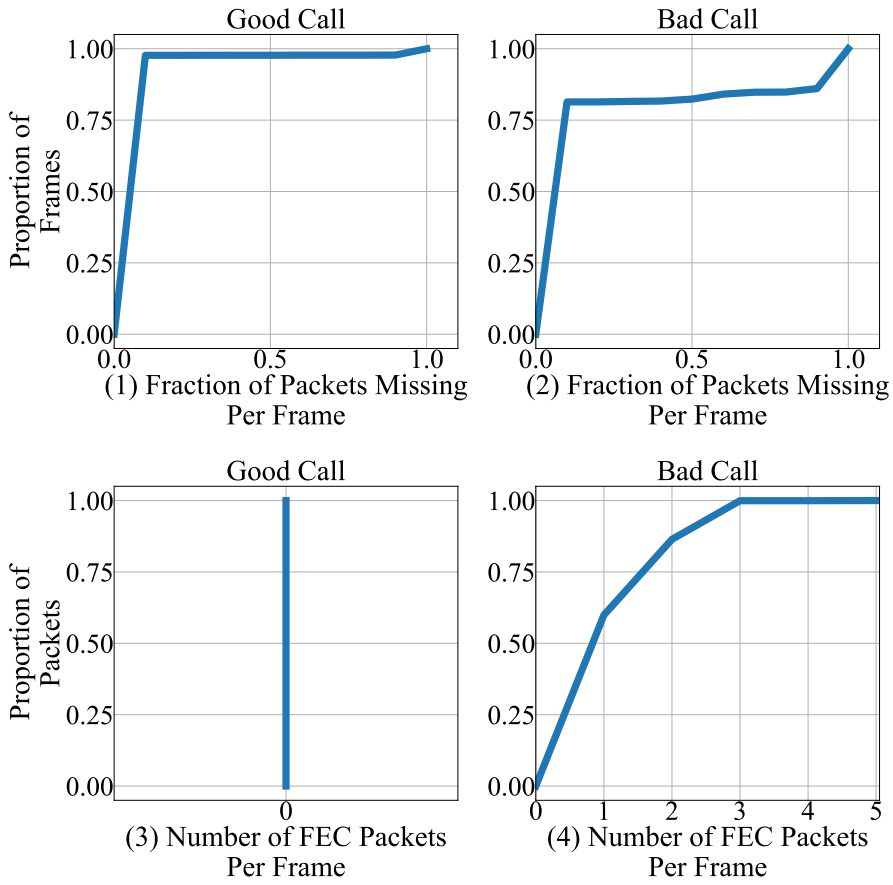

Figure 4-3: A histogram comparing the proportion of frames for a given frame or packet metric between a good and Zoom bad call.

when packet loss occurs in Fig. 4-4. Gaps start from $t = 25$s to $t = 200$s.

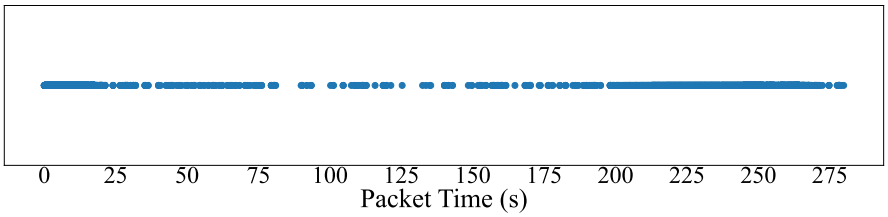

Figure 4-4: A timeline of all packets received by SEZMA during the bad Zoom call. Each dot represents a received packet.

During this call, we observe many periods of stalling (*i.e.,* frame freezing) and the quality of the video frame diminished. Fig. 4-5 shows snippet of time wherein the blue dots represent received packets (as inferred by the Network module), while

the orange dots represent frozen frames (as inferred from the Video module). We observe that the frame freeze occurs within the gaps where no Zoom packets are received, indicating that packet drop affects (and causes) stalling in this call. No frame freezing occurs when many packets are received, as shown after $t = 200$s.

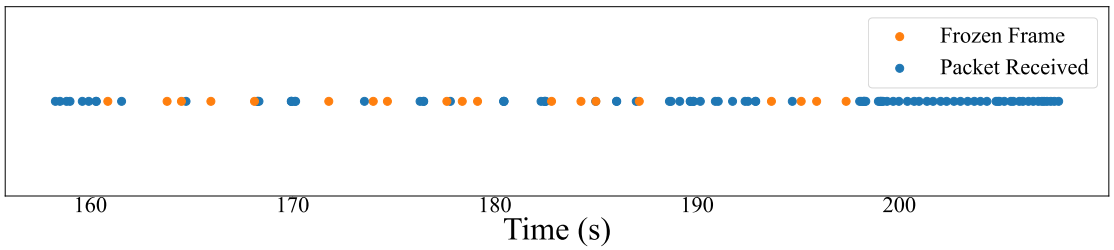

Figure 4-5: A snippet of the timeline of packets received by SEZMA and occurrences of frozen frames during a bad Zoom call. The frozen frames occur within the gaps without received packets.

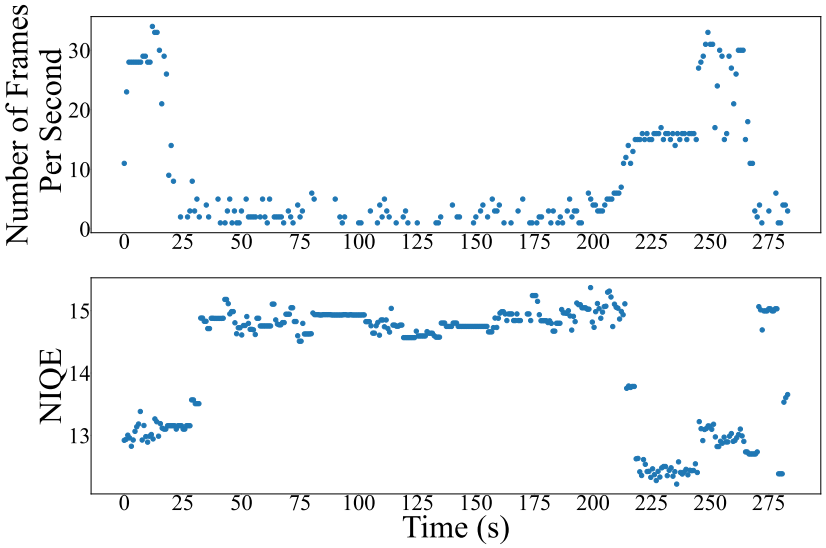

Figure 4-6: A timeline of the number of frames per second over time vs. NIQE score during the bad Zoom call.

Fig. 4-6 shows a timeline of the number of frames per second and NIQE during this bad Zoom call. As expected, whenever the number of frames per second is high and the frames are updated very quickly to reflect any small changes in frame content, the NIQE score is low, meaning the quality of the video frame is good. Conversely, for the low number of frames per second, the NIQE score is high, meaning the quality

of the video frame is degraded. The decrease in the number of frames per second corresponds to high packet loss from $t = 25$s to $t = 200$s.

4.2 Simulated Network Conditions

To explore other network conditions, we use the Network Link Conditioner provided by Apple's Xcode tools to create specific network conditions to test SEZMA's functionality. SEZMA runs on one endpoint, and the Network Link Conditioner runs on the other endpoint of the Zoom call. The Network Link Conditioner enables us to define a profile that sets network conditions, such as the bandwidth, the delay through the network when sending packets ("out delay"), and the packet loss ("out packet loss").

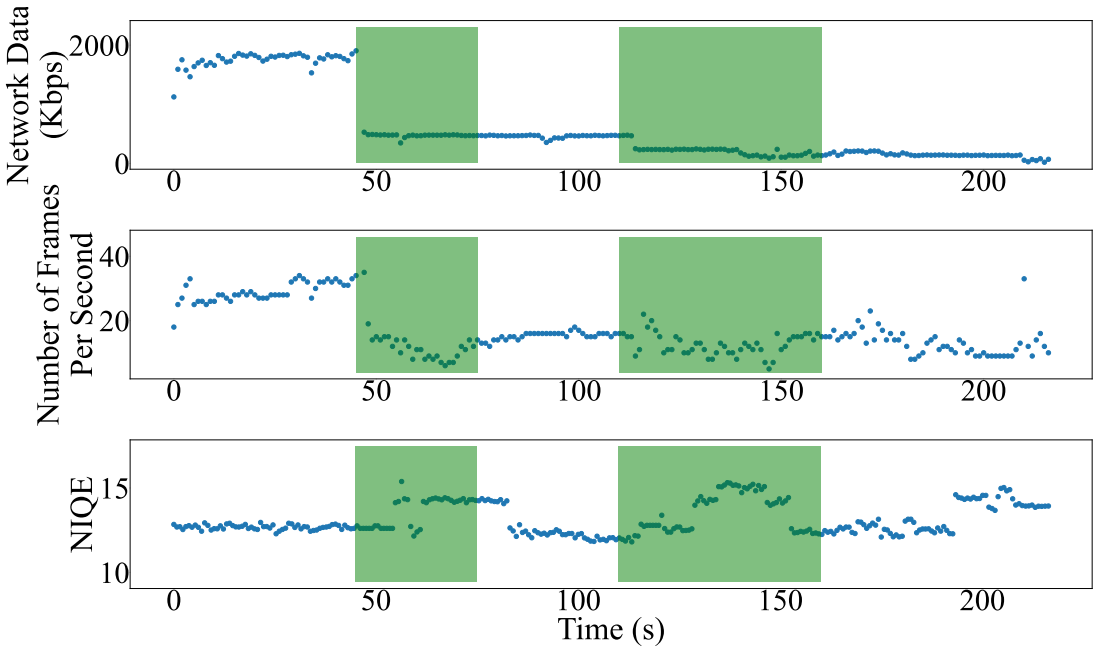

Figure 4-7: A timeline of the amount of network data, the number of frames per second, and NIQE score during a Zoom call where the Network Link Conditioner decreases bandwidth over time.

Decrease in Bandwidth. We use the Network Link Conditioner to constrain the bandwidth of the network and analyze the bandwidth impact on the video quality. The Network Link Conditioner is initially supplied with a high bandwidth around 40

Mbps (0-50s), and then the supplied bandwidth is reduced to 5 Mbps (50-75s), and finally reduced further to 2.5 Mbps (105-160s), during the Zoom call. The top two graphs in Fig. 4-7 show the amount of network data in Kilobits-per-second of Zoom video traffic actually received and the number of frames received per second. The bottom graph is the NIQE metric over time.

The green box indicates when the bandwidth has dropped. For instance, the receiver starts out receiving 1800 Kbps of data, which first drops to 480 Kbps, and then drops again to 230 Kbps. During each bandwidth drop period, the amount of network data and the number of frames per second also decreases. At the same time, the NIQE score increases, indicating that the quality of video frames worsens. There is a slight delay in the Zoom application between when packets are received and when the corresponding decoded video frame is shown on the Zoom screen. Once the bandwidth change settles, the number of frames per second increases after each green box. This increase corresponds to a decrease in the NIQE score, pointing to an increase in video frame quality. The increased number of frames per second may show that Zoom is figuring out how many frames it should send to the receiver while maintaining good video frame quality.

Increase in Delay. We also experiment with increasing the "out delay" field in the Network Link Conditioner that controls the delay through the network for packets sent. The Network Link Conditioner begins with the "out delay" or propagation delay at approximately 0 seconds and then increases to 3 seconds. The top-left graph of Fig. 4-8 shows the average duration between packets during three-time periods of the Zoom call: (0-10s) the propagation delay is approximately 0 seconds, (10-15s) the propagation delay increases to 3 seconds, and (15-40s) the propagation delay is 3 seconds. Since the bandwidth is not changed during the Zoom call (40 Mbps), multiple packets can be sent per second, so the duration between packets is much lower than 3 seconds. This only affects the end-to-end latency of frames and increases the latency to 3 seconds. The bottom-left and bottom-right graphs show the number of frames per second and the number of packets per second decreases as the propagation delay increases.

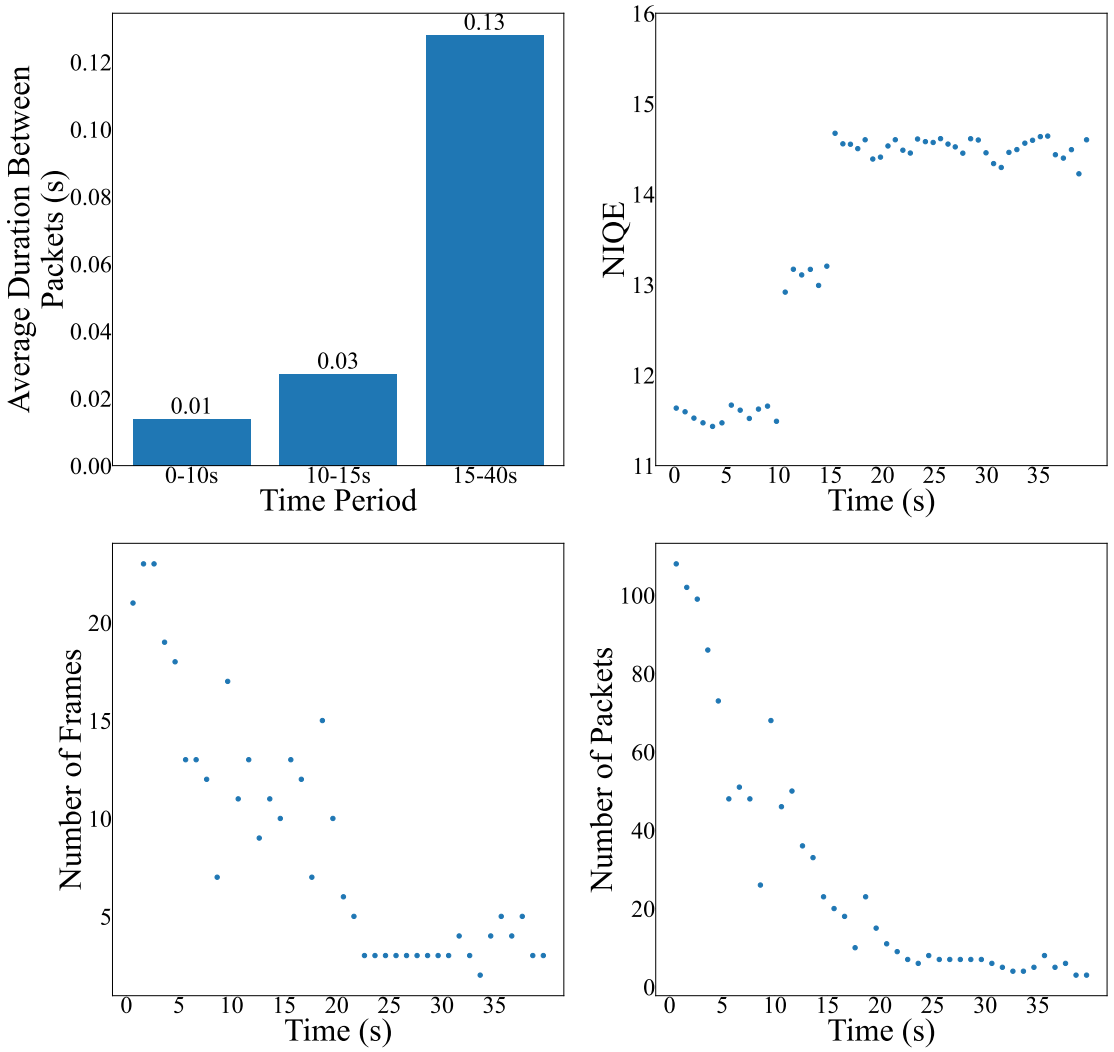

Figure 4-8: The top-left graph shows the average duration between packets at time periods corresponding to the times in the remaining graphs. The top-right graph shows NIQE scores, the bottom-left graph shows the number of frames per second, and the bottom-right graph shows the number of packets per second during a Zoom call where the Network Link Conditioner increases the out delay.

The top-right graph of Fig. 4-8 shows the NIQE scores during the Zoom call. The increase in the average duration between packets, the decrease in the number of frames per second, and the decrease in the number of packets per second correlate with the increase in NIQE scores per time period. We find that the fraction of packets missing per frame is 0 throughout, so packet loss does not explain the decrease in frame quality. Instead, Zoom may discover that frames take longer to arrive when it monitors end-to-end delay. It assumes queue buildup or congestion in the network, and reduces

its bitrate in response. This is common in many congestion control algorithms for video applications, including Google Congestion Control [8] that operates within the WebRTC framework (that Zoom is built on top of). This decrease in bitrate manifests as a decrease in the number of frames per second and the number of packets per second, as well as reduced visual quality. As a result, these frames appear to be blurry or blocky.

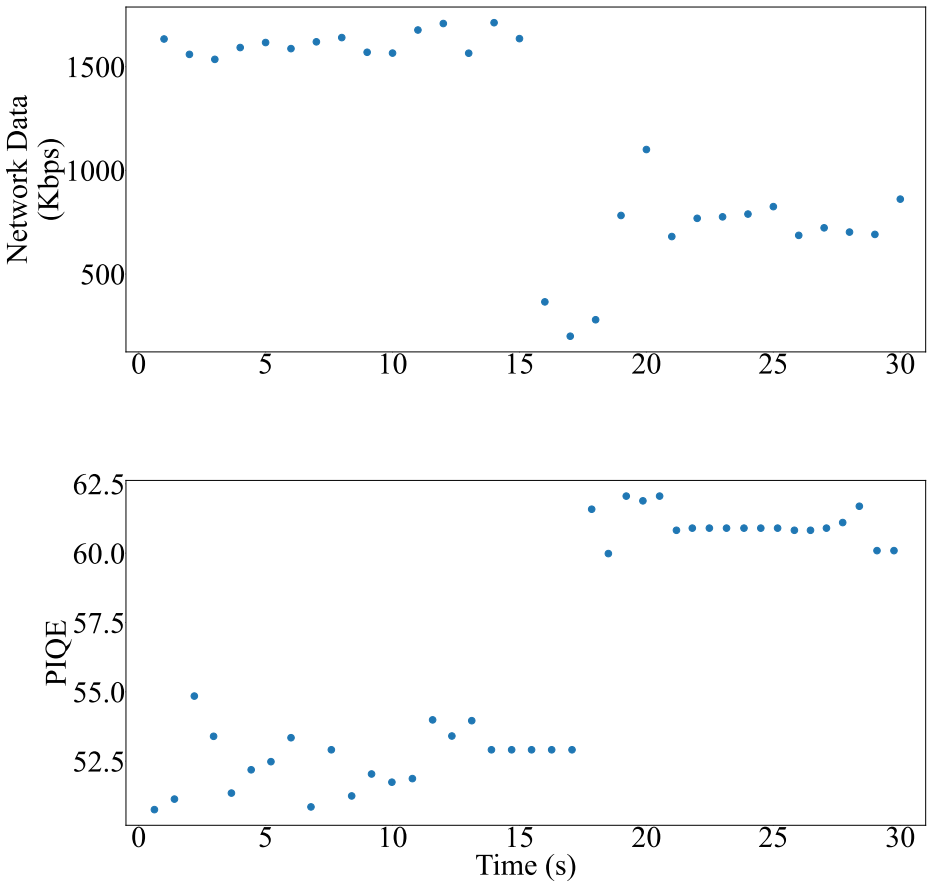

Figure 4-9: The graph shows the amount of network data and the PIQE score of a Zoom call where the Network Link Conditioner increases out packet loss.

Lossy Conditions. We test SEZMA with an increased "out packet loss" field in the Network Link Conditioner that controls the percent of packets that are lost. The Network Link Conditioner begins with an "out packet loss" of 0 and then increased to

an "out packet loss" of 30%. Fig. 4-9 shows that the amount of network data decrease around $t = 15$s, indicating that the packet loss has increased to 30%. The PIQE scores increase at $t = 17$s, showing that the packet loss subsequently causes a decrease in the video frame quality. Like Fig. 4-7, there is a slight delay in the Zoom application between when packets are received and when the corresponding decoded video frame is shown on the Zoom screen. The PIQE score increase, therefore, corresponds with the decrease in the amount of network data. Like the delay scenario above, Zoom may discover the packet loss and reduces its bitrate in response (potentially to allow for more FEC packets instead), which leads to lower video quality. Some frames may also be decoded incorrectly or partially leading to stalls due to corrupt decoder state. For instance, Fig. 4-10 shows a snippet of the occurrence of frozen frames and packets received during the transition between $t = 15$s to $t = 17$s where packet loss occurred. Gaps between packets received map the occurrence of frozen frames, indicating the packet loss may cause video to freeze.

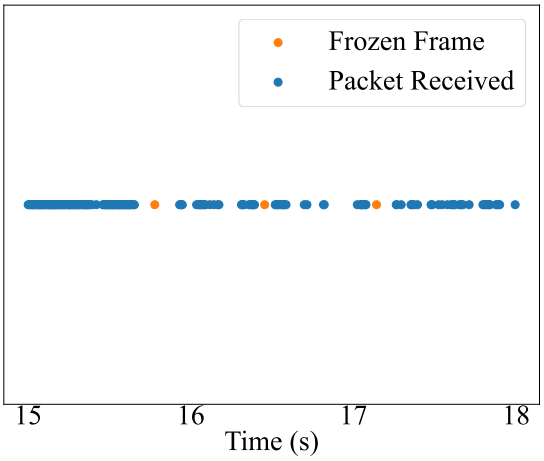

Figure 4-10: The graph shows a snippet of the Zoom call where the Network Link Conditioner increases.

4.3 SEZMA CPU and Memory Performance

We investigate the CPU and memory performance of SEZMA on a 2022 MacBook with an M1 processing chip. We compare SEZMA with Zoom to check whether the memory and CPU usage of SEZMA is within a reasonable range. We also evaluate SEZMA's CPU and memory usage under different Zoom call settings (*e.g.,* whether the microphone is on) and different application settings (*e.g.,* what video frame quality metrics are collected).

We use the command-line tool, `top`, to sample the CPU and memory usage of Zoom and all sub-processes of SEZMA, including the main application process, the Network process, the Video process, and the Zoom-Active process (which periodically checks for an active Zoom window), every second during a Zoom call. We aggregate the CPU and memory usage of all processes of SEZMA, and also calculate the average CPU and memory usage of each process over the entire Zoom call.

Zoom vs. SEZMA. Using the above data collection method, we ran one Zoom call to compare the average CPU and memory usage of SEZMA to Zoom. During this call, SEZMA is configured to run only one video frame metric, Laplacian, at a frame rate of 10 fps.

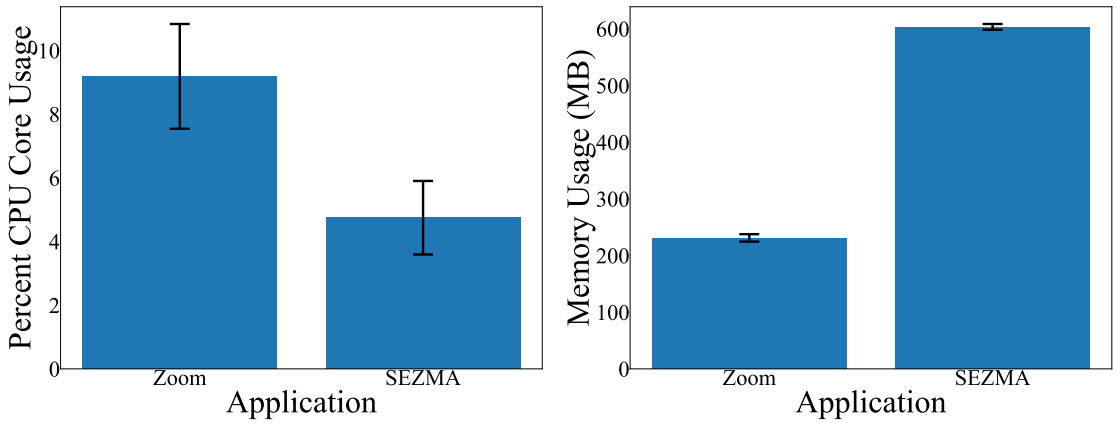

Figure 4-11: The average CPU and memory usage comparison between Zoom and SEZMA for a Zoom call.

Fig. 4-11 shows average values of CPU Core and Memory Usage for both Zoom and SEZMA. During the Zoom call, SEZMA uses an average of 4.74% CPU core

compared to Zoom's 9.19% while SEZMA uses an average memory usage of 603 MB compared to Zoom's 231 MB. SEZMA's increased memory usage comes from holding video frames and captured packets in memory before processing metrics for them.

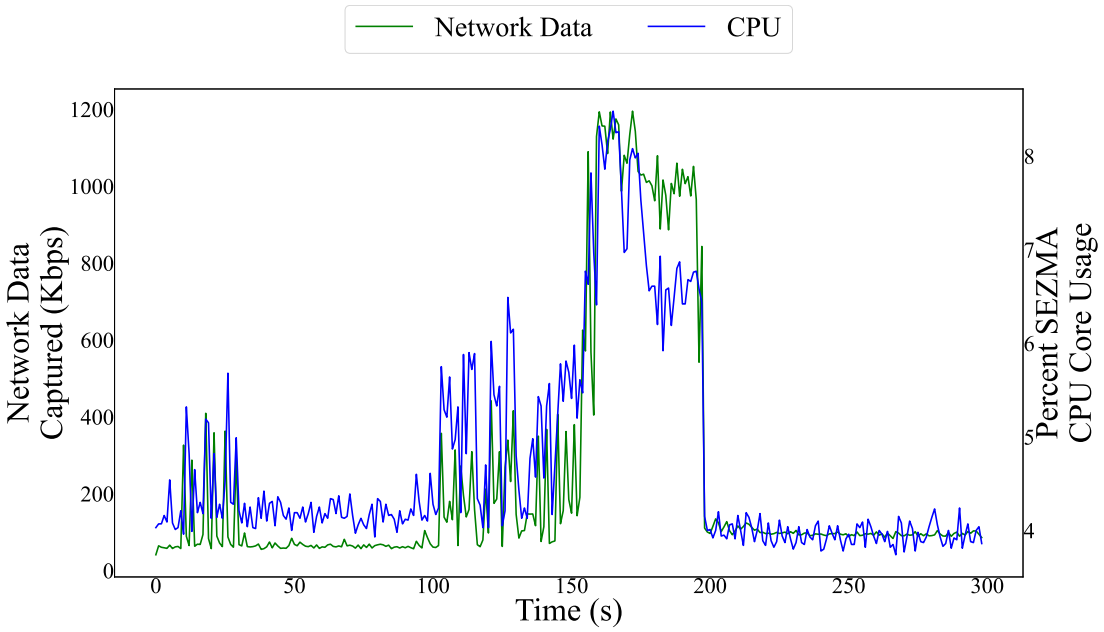

Figure 4-12: A timeline of SEZMA's CPU usage and the amount of network data collected over a Zoom call.

Fig. 4-12 shows that the CPU usage of SEZMA is correlated with the network data that is captured by SEZMA over time. When collecting the network data, we run the `tcpdump` command with the same network data filter as the Network module of SEZMA to get a rough estimate of how much data is being captured by SEZMA. The filtered network data is mostly Zoom and no more than 0.6 Kbps are from background processes running on the MacBook. When there is a change in the other participant's Zoom settings (*i.e.,* microphone is on, a person leaves or enters the camera view of the Zoom call), these changes correspond to a spike in the amount of network data received and the CPU usage of SEZMA. This correlation suggests that SEZMA's CPU core usage depends on the amount of network data that needs to be processed.

Zoom vs. SEZMA vs. YouTube. We also run an experiment with a Zoom call running with a Google Chrome window containing a YouTube video stream with an application configuration of computing only Laplacian at 10 fps. The purpose of this

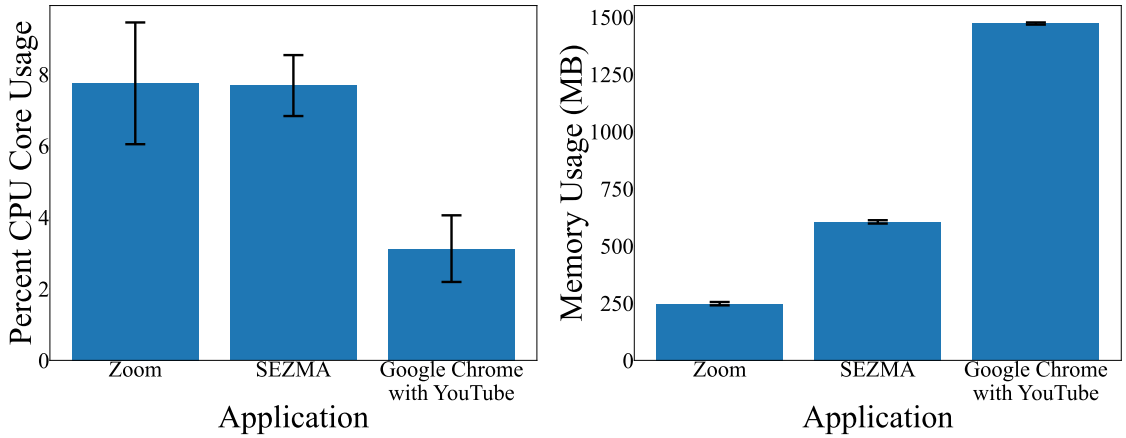

Figure 4-13: The average CPU and memory usage comparison between Zoom, SEZMA, and Google Chrome with YouTube running for a sample Zoom call.

analysis is to determine the impact of other processes with high CPU or memory usage, particularly video streaming services, on this application. Fig. 4-13 shows that Google Chrome running YouTube took the highest memory usage, using on average 1472 MB during the call whereas Zoom uses 248 MB and SEZMA uses 605 MB. The CPU usage of Google Chrome with YouTube is the lowest, using on average 3.13% of the CPU core. SEZMA uses 7.68% of CPU core on average, and Zoom uses 7.75%. Fig. 4-14 shows that the CPU usage of SEZMA is correlated with the amount of network data captured, including the network data from YouTube video stream and Zoom. Large gaps in the amount of network data captured from both YouTube and Zoom correspond to drops in the CPU core usage of SEZMA. At $t = 79$s, the CPU Core usage dropped from 7.26% to 5.31%. This drop corresponds to no YouTube data captured and Zoom data being captured at a maximum rate of 800 Kbps between $t = 68$s and $t = 78$s. At $t = 236$s, the CPU Core usage dropped from 7.72% to 6.68%. This drop corresponds to no YouTube data captured and Zoom data being captured at a maximum rate of 500 Kbps This shows that SEZMA's CPU core usage may be dependent on how much video-related network data, not just Zoom data, is received.

Microbenchmarks. We run larger experiments to observe how our Network and Video modules' CPU and memory usage varies under different Zoom call scenarios and application configuration settings. All calls are 4–5 minutes long and between two people. For each call, we compute the averages of the CPU Core Usage, memory

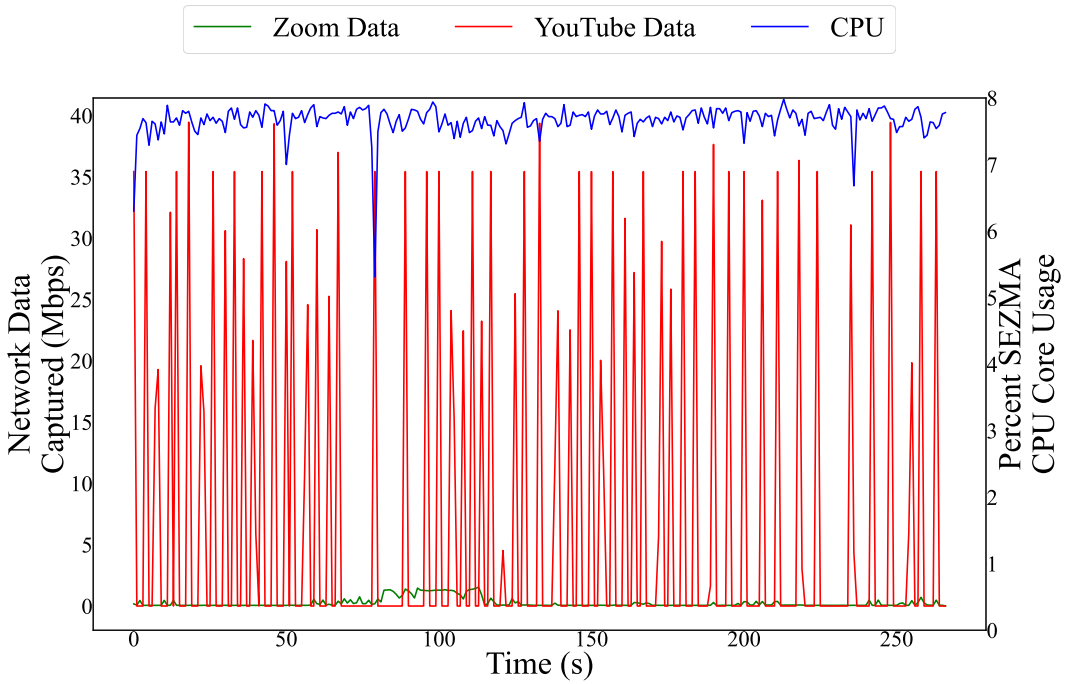

Figure 4-14: A timeline of SEZMA's CPU and the amount of Zoom network data and YouTube network data collected over a Zoom call.

and network data, as well as the Video module's frame capture rate collected over the entire call.

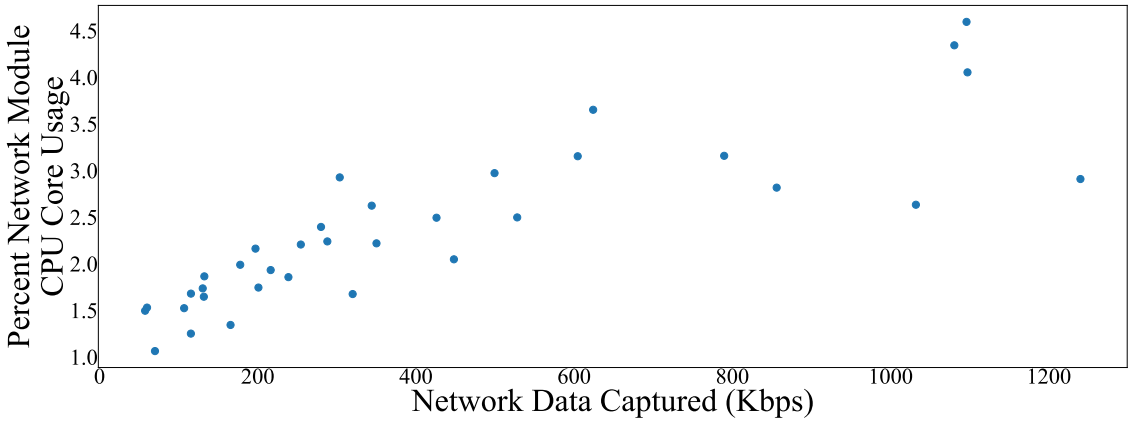

Figure 4-15: The percent CPU core usage of the Network module and the amount of network data captured per Zoom call.

We run SEZMA on many Zoom calls to determine whether there is a correlation between the amount of network data captured and the CPU Core usage of the Network module. Zoom calls may vary in the amount of network data depending on the device

the other participant is using (*i.e.,* laptop, phone), whether the other participant is mute or not, or whether the other participant has high movement in their video. Regardless of these different Zoom call scenarios, Fig. 4-15 shows how a Zoom call with more network data captured per second correlates to the increase in the Network module's CPU core usage.

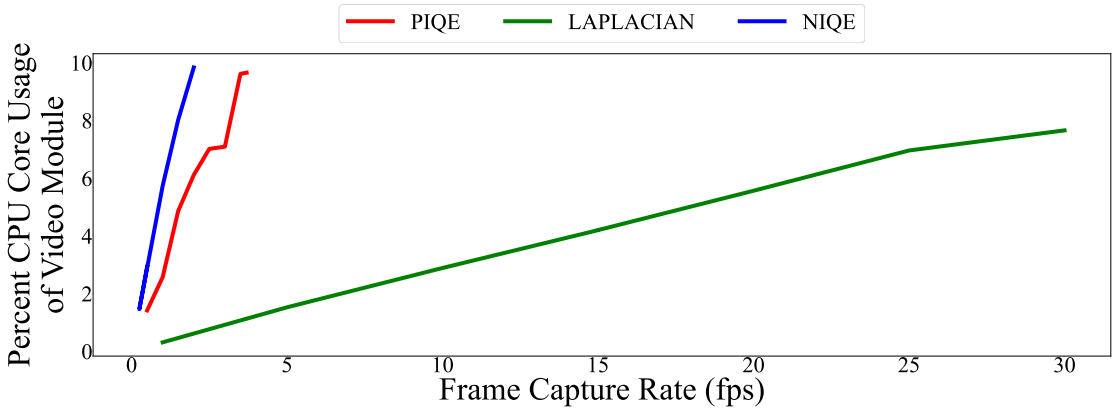

Figure 4-16: The relationship between frame rate and metric application setting on the Video module's CPU usage.

The Video module's CPU usage is dependent on what video frame metrics and frame capture rate we configure in the application configuration (§3.5). For this analysis, the Video module uses one video frame metric per call from the non-model-based metrics, Laplacian and PIQE, and the model-based metric, NIQE. Fig. 4-16 shows the CPU core usage of the Video module versus the frame capture rate for different video frame metrics. For all video frame metrics, the CPU core usage of the Video module increases as the frame rate at which frames are captured increases. The Video module running the Laplacian metric can have a higher frame capture rate than the Video module running either PIQE or NIQE at the same CPU core usage. This is unsurprising: running a neural network for the model-based metric NIQE is more computationally expensive than capturing non-model-based metrics. Fig. 4-17 shows the same information in a different format by computing the fastest frame capture rate supported by the Video module running each video frame metric. We find that the Laplacian supports the highest frame capture rate of 30 fps. Further, we find that the Video module computing Laplacian at its highest possible frame capture rate of

30 fps has a CPU core usage lower than using NIQE at its highest possible frame capture rate of 2 fps. PIQE falls in between these two, with a somewhat high CPU usage and low fastest frame capture rate. PIQE is more computationally expensive than Laplacian due to the need to divide and classify 16-by-16-sized blocks in an image as distorted or not (§3.3).

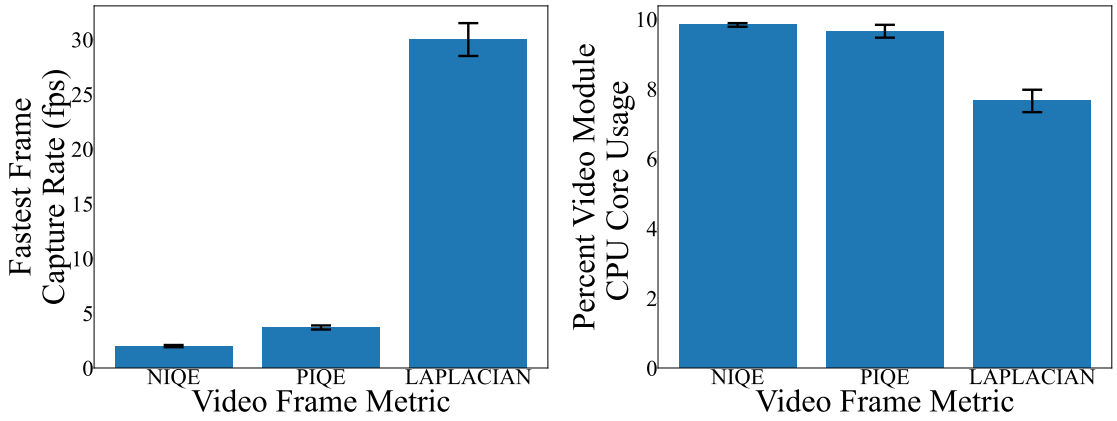

Figure 4-17: The fastest frame capture rate the Video module can support for each video metric, and the percentage of CPU core usage by the Video module to capture video frame metrics at its fastest frame rate.

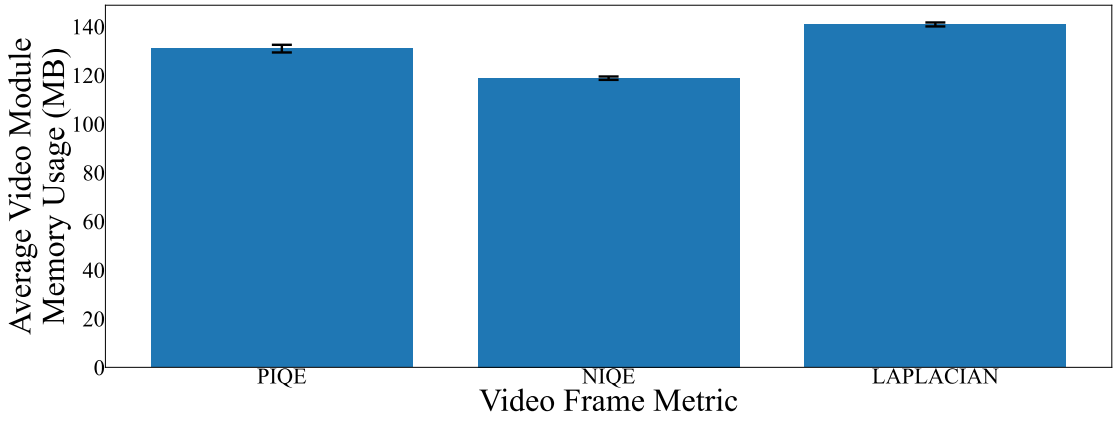

Figure 4-18: The memory usage for Video module per video frame metric.

We find that the Network and Video modules do not exhibit significant variance in memory usage when changing different parameters like the amount of network data captured per second or frame capture rate. The only exception is the average memory usage of the Video module, which varies across different video frame metrics, as shown

in Fig. 4-18. The Laplacian has the highest memory usage of approximately 141 MB, PIQE with 130 MB, and NIQE with 118 MB at each of their highest supported frame rate. Since the memory usage of NIQE is not that high, this suggests that if we can change the Video module to batch and run NIQE with multiple frames at a time, SEZMA with NIQE can support a higher frame rate. We leave this to future work.

Chapter 5

Limitations and Future Work

SEZMA has three main limitations in its current implementation for MacOS Zoom Users. Firstly, aligning the packets for a given frame from the Network module with the video frame recorded from the Video module is challenging. There is a lag between when the frame is received from the network and when the frame is displayed on Zoom. As a result, SEZMA cannot use the same timestamp to map each video frame with each packet corresponding to a frame. Secondly, automating the correct video frame metrics to quantify video frame quality is hard since each video frame metric focuses on a specific feature of an image, such as blurriness (§3.3). Thirdly, scaling SEZMA to run on Zoom calls with more than two people requires a more careful design. In a call with more than two participants, capturing screenshots per participant video screen is challenging because Zoom provides multiple view mode options (*e.g.,* "Gallery" and "Speaker"). In "Gallery" mode, cropping each participant's video screen in a Zoom Meeting window requires getting the boundaries of each video screen. Selecting these boundaries in real-time during a Zoom call is difficult without knowing the video screen positions. In "Speaker" mode, the Zoom Meeting window only shows the camera of the participant speaking and not all participants.

We leave a deeper investigation of these three limitations to future work and provide high-level descriptions of potential solutions to explore. Correlating packets per frame and the frame between the Network and Video modules can be solved by profiling and computing the duration it takes Zoom to process packets received

for a video frame and to display it on screen. The timestamps of the video frame can be shifted by the duration to get the timestamps of the packets per frame. The automation of the correct video frame metrics can be fixed by using a combined metric for Laplacian, NIQE, and PIQE. This combined metric can quantify each video frame based on all the quality features that all the video frame metrics compute. Scaling SEZMA to run on Zoom call with more than two people involves considering whether the view mode is "Gallery" or "Speaker". For 'Gallery", the Video module's Data Collection submodule can be improved to select video screens within the Zoom call using edge detection methods, such as EdgeFlow [17]. These edge detection methods may allow the application to run on Zoom calls with more than two participants. For "Speaker", the Video module's Data Collection submodule needs to identify each frame to a particular user. Solutions for these limitations help SEZMA become deployable in all types of Zoom calls and improve its explanatory power.

In addition to the above limitations, SEZMA needs to be distributed more widely and deployed in different network settings. For this thesis, the outputs of this application are on a small set of Zoom calls constrained under good network conditions (*e.g.,* connected to a network with high bandwidth), which only scratches the surface of different types of observable network events. Conducting larger-scale user tests may reveal this application's true explanatory power.

SEZMA also only supports MacOS users right now. However, the application uses a modular design that provides a framework to add new features or extend existing ones on the application. Because of this, SEZMA can also be extended to run on other operating systems like Linux or Windows. Each module is not tied to the implementation and libraries that are only available for Mac. For instance, the Video module's Data Collection submodule can be configured to capture Zoom Meeting windows using libraries, such as `PyGetWindow` [2] for Windows. New metrics can be added to the Metric Computation submodules for the Network and Video modules to improve the application's explanatory power. For example, the Network module's Metric Computation submodule can include a metric that shows packet retransmission. The overall CPU Core usage of SEZMA can be reduced by improving

the CPU Core usage of each individual module. For instance, we can reduce the CPU Core usage of the Network module by improving the data collection to sample one packet every five packets received, rather than capturing all the packets. With these additions, this application could provide a more thorough analysis of network conditions impacting Zoom video quality.

Chapter 6

Conclusion

This thesis provides an application, SEZMA, that runs on a user's device to analyze their Zoom call. SEZMA achieves the following design goals:

1. Explanatory: The results show that SEZMA can pinpoint network events (*e.g.,* packet loss and bandwidth constraints) that may cause video degradation (*e.g.,* blurry or frozen video frames).

2. Usable: SEZMA requires only a single endpoint to collect and analyze the Zoom call. A webpage makes it easy for users to download SEZMA. Users can install and run SEZMA with minimal effort.

3. Lightweight: SEZMA conducts metric analysis as the Zoom call is ongoing rather than after the call, reducing the storage of the user's device to store the metric logs. Benchmark comparisons with Zoom show that SEZMA does not place a high CPU and memory burden on the user's device.

4. Privacy-preserving: SEZMA only sends metric logs to the centralized server. These metric logs do not contain any identifying user information.

In future work, we aim to improve SEZMA's explanatory ability and scalability. SEZMA's explanatory ability can be improved by mapping each video frame from the Video module to the packets for a frame from the Network module, a combined video

frame metric, and a network metric measuring retransmissions of packets. These additions can allow more thorough explanations of what network events cause bad video quality. SEZMA can be more scalable by running it with Zoom calls with more than two people, using techniques like edge detection methods to obtain a video screen per participant in the call. In addition, SEZMA can also be deployed on other operating systems by changing the implementation to consider these systems.

Bibliography

[1] Nuitka. https://nuitka.net.

[2] Pygetwindow. https://github.com/asweigart/pygetwindow.

[3] Scapy. https://scapy.net.

[4] Wireshark. https://www.wireshark.org.

[5] Most popular videoconferencing services used by adults in the united states to chat with family and friends during the coronavirus pandemic as of march 2020. https://www.statista.com/statistics/1119981/videoconferencing-services-us-coronavirus-pandemic, May 2020.

[6] Zoom support:wireless (wifi) connection issues. https://support.zoom.us/hc/en-us/articles/201362463-Wireless-WiFi-connection-issues, Jun 2022.

[7] Sky Ariella. 31 trending zoom meeting statistics [2023]: How many people use zoom? https://www.zippia.com/advice/zoom-meeting-statistics, Jan 2023.

[8] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[9] R.W. Chan and P.B. Goldsmith. A psychovisually-based image quality evaluator for jpeg images. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, volume 2, pages 1541–1546 vol.2, 2000.

[10] Hyunseok Chang, Matteo Varvello, Fang Hao, and Sarit Mukherjee. Can you see me now? a measurement study of zoom, webex, and meet. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 216–228, New York, NY, USA, 2021. Association for Computing Machinery.

[11] Debbie Chew and Mahsa Azizi. The state of video conferencing 2022. https://www.dialpad.com/blog/video-conferencing-report.

[12] Albert Choi, Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. Zoom session quality: A network-level view. In *Passive and Active Measurement: 23rd International Conference, PAM 2022, Virtual Event, March 28–30, 2022, Proceedings*, page 555–572, Berlin, Heidelberg, 2022. Springer-Verlag.

[13] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 362–373, New York, NY, USA, 2011. Association for Computing Machinery.

[14] Randell Jesup, Tom Kristensen, Yekui Wang, and Roni Even. RTP Payload Format for H.264 Video. RFC 6184, May 2011.

[15] Vidhyalakshmi Karthikeyan, Brahim Allan, Detlef D. Nauck, and Miguel Rio. Benchmarking video service quality: Quantifying the viewer impact of loss-related impairments. *IEEE Transactions on Network and Service Management*, 17(3):1640–1652, 2020.

[16] Qiyong Liu, Zhaofeng Jia, Kai Jin, Jing Wu, and Huipin Zhang. Error resilience for interactive real-time multimedia application, Feb 2018.

[17] Wei-Ying Ma and Bangalore S Manjunath. Edgeflow: a technique for boundary detection and image segmentation. *IEEE transactions on image processing*, 9(8):1375–1388, 2000.

[18] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 229–244, New York, NY, USA, 2021. Association for Computing Machinery.

[19] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. Enabling passive measurement of zoom performance in production networks. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 244–260, New York, NY, USA, 2022. Association for Computing Machinery.

[20] Anish Mittal, Rajiv Soundararajan, and Alan C. Bovik. Making a "completely blind" image quality analyzer. *IEEE Signal Processing Letters*, 20(3):209–212, 2013.

[21] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[22] Thomas Stockhammer, Magnus Westerlund, Miska M. Hannuksela, David Singer, and Stephan Wenger. RTP Payload Format for H.264 Video. RFC 3984, February 2005.