Enabling Passive Measurement of Zoom Performance in Production Networks

Oliver Michel
Princeton University
Princeton, USA
omichel@cs.princeton.edu

Satadal Sengupta
Princeton University
Princeton, USA
satadal.sengupta@cs.princeton.edu

Hyojoon Kim
Princeton University
Princeton, USA
hyojoonk@cs.princeton.edu

Ravi Netravali
Princeton University
Princeton, USA
rnetravali@cs.princeton.edu

Jennifer Rexford
Princeton University
Princeton, USA
jrex@cs.princeton.edu

ABSTRACT

Video-conferencing applications impose high loads and stringent performance requirements on the network. To better understand and manage these applications, we need effective ways to measure performance in the wild. For example, these measurements would help network operators in capacity planning, troubleshooting, and setting QoS policies. Unfortunately, large-scale measurements of production networks cannot rely on end-host cooperation, and an in-depth analysis of packet traces requires knowledge of the header formats. Zoom is one of the most sophisticated and popular applications, but it uses a proprietary network protocol. In this paper, we demystify how Zoom works at the packet level, and design techniques for analyzing Zoom performance from packet traces. We conduct systematic controlled experiments to discover the relevant unencrypted fields in Zoom packets, as well as how to group streams into meetings and how to identify peer-to-peer meetings. We show how to use the header fields to compute metrics like media bit rates, frame sizes and rates, and latency and jitter, and demonstrate the value of these fine-grained metrics on a 12-hour trace of Zoom traffic on our campus network.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; **Network measurement**; Network architectures.

KEYWORDS

Video Conferencing, Zoom, Measurement, Network Performance, Protocol Analysis, Reverse Engineering

ACM Reference Format:
Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. 2022. Enabling Passive Measurement of Zoom Performance in Production Networks. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22), October 25–27, 2022, Nice, France.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3517745.3561414

IMC '22, October 25–27, 2022, Nice, France

ACM ISBN 978-1-4503-9259-4/22/10.
https://doi.org/10.1145/3517745.3561414

1 INTRODUCTION

Video-conferencing applications have seen an unprecedented surge in popularity over the past few years [10, 16]. Zoom has been at the forefront of this phenomenon, with adoption by many organizations to foster teaching, meetings, and presentations during the COVID-19 pandemic [12, 15, 35].

To keep pace with ever-stringent user performance expectations [18, 51] and increasing resource contention, practitioners and researchers need the ability to measure (and improve) Zoom performance in the wild, *without requiring cooperation from end hosts*. For instance, granting this capability to network operators would enable more targeted capacity planning, problem troubleshooting, and traffic-prioritization policies. Realizing this requires the ability to extract metrics such as media bit rates, delay, frame rates, and frame-level jitter solely via analysis of packet captures of Zoom sessions. These insights, in turn, grant a clear understanding of the inner-workings of a Zoom meeting, and the performance and quality experienced by each of its participants. Taken together, we require *fine-grained* measurements and performance insights for Zoom derived from analyzing passively-collected network traffic *in the wild*.

Unfortunately, existing measurement approaches all fall short of at least one of these goals. Some researchers instrument end hosts to run controlled experiments that study Zoom's rate adaptation and performance [7, 10, 25, 27]. However, controlled experiments are labor-intensive, limited in scope, and do not reveal Zoom performance in the wild. Other researchers conduct measurement studies on production networks [12, 35]. Due to Zoom's proprietary network protocol, these studies collect only coarse-grained statistics such as byte and packet rates, which are insufficient for the use cases outlined above. Lastly, while some performance metrics are available to operators through Zoom's API, this data is also coarse-grained (i.e., not packet level) and measured far from the operator's network (i.e., in Zoom's data centers); consequently, this API data is insufficient for rapid adaptation at on-premise network devices.

In this paper, we address this void, and enable direct (and systematic) measurements of Zoom by (1) demystifying how Zoom works at the packet level and (2) designing tools and techniques for analyzing Zoom performance from packet traces. The key challenges are twofold. First, Zoom uses a proprietary packet format, encrypted control and media traffic, and closed-source client software [12, 25, 29, 35]. As a result, Zoom cannot be analyzed easily

using diagnostic tools for WebRTC [4] or packet-analysis tools like Wireshark [49]. Second, Zoom employs complex control logic whereby both user behavior (e.g., muting audio/video, changing the display configuration, sharing the screen, enabling recording) and network behavior (e.g., packet loss, delay, and throughput) can result in similar changes in application behavior.

To grow our understanding and overcome these hurdles, we first use controlled experiments (solely) to decipher Zoom's protocols, discovering (1) the relevant unencrypted fields in the Zoom packet format, (2) how to group streams into meetings, and (3) how to identify peer-to-peer meetings. We make our packet-parsing logic available as a Wireshark plugin for the community to use. Perhaps surprisingly, we find that valuable information can be gleaned from the unencrypted parts of Zoom traffic. We expect these unencrypted parts of Zoom packets to remain in the clear, since Zoom itself requires these for scalable forwarding of media traffic.

Armed with this knowledge, we show how to estimate Zoom meeting performance from in-network measurements at fine granularity, supporting metrics such as media bit rates, packet latency, jitter, retransmission, and more. To validate our methodology and ability to collect fine-grained performance insights without end-host modifications, we apply our passive measurement techniques to Zoom traffic collected on our campus network. Our study used a scalable Zoom traffic capture system we built in the data plane (using P4 [6] on Intel Tofino [1, 20] switches) to filter and anonymize Zoom packets for analysis on servers.

While our analysis and study focuses on Zoom, many of our techniques should generalize across video-conferencing applications that use the Real-Time Protocol (RTP) which is the vast majority of such applications, including Google Meet and Microsoft Teams [35]. We describe in detail how we deciphered Zoom's protocols and provide a blueprint for future studies on proprietary network protocols. Finally, the core of our work is the analysis of Zoom and the development of measurement techniques; as such, the measurements reported only serve to demonstrate and validate the accuracy and granularity of our techniques. We leave a full measurement study of Zoom performance in the wild as future work.

Taken together, our paper makes the following contributions:

(1) We demystify the most relevant parts of Zoom's proprietary protocols and shed new light on how Zoom delivers media streams over the network.
(2) We demonstrate how understanding Zoom's protocol enables us to extract a wider range of metrics from Zoom network traffic, including packet loss, latency, jitter, media bit rates, frame rate, and frame sizes.
(3) We make our artifacts — the Wireshark plugin, the P4 traffic-capture program, and the software-based analysis tools — available to the community [31].

2 VIDEO CONFERENCING BACKGROUND

Video-conferencing applications (VCAs) are complex distributed systems with many components. Design and implementation choices that must be made when building a VCA fall into three main categories which we will now outline.

Network Protocols and Mechanisms. Most major conferencing applications use the Real Time Protocol (RTP) [40] and its counterpart, the Real Time Control Protocol (RTCP) [40], to transport media. An RTP stream is identified by a unique identifier, the Synchronization Source Identifier (SSRC), which can be used to multiplex several media streams (e.g., audio and video) over a single UDP flow. Each stream can contain several sub-streams to, for example, carry forward error correction (FEC) data. While RTP headers are usually transmitted in cleartext to allow conferencing servers to forward or modify streams without having to decrypt them, RTP payload is mostly encrypted (see SRTP [36]). Before media can flow between participants, clients negotiate through a signaling mechanism what codecs to use, how to encapsulate media, and where to send the respective streams. The Session Initiation Protocol (SIP) [38] and the Session Description Protocol (SDP) [37] are commonly used for this.

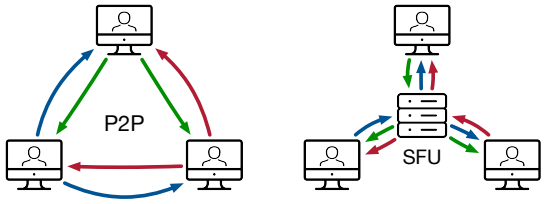


Figure 1: Video Conferencing Architectures: Peer-To-Peer (P2P) vs. Selective Forwarding Unit (SFU). Each Color Represents a Participant's Video Stream.

Media Encoding and Rate Adaptation. Over the past years, audio and video codecs (e.g., Opus [46] and H.264/AVC [48]) have become increasingly efficient; to achieve high quality at low bandwidth consumption, video codecs leverage complex intra- and inter-frame prediction schemes [43, 44]. A feature of modern video codecs particularly relevant to video conferencing is Scalable Video Coding (SVC). A scalable video stream contains several layers corresponding to different quality levels where higher-quality layers build upon lower-quality layers. SVC allows the conferencing server to remove higher-quality parts of a video bit stream to rapidly adapt to, for example, varying network conditions or device capabilities, for each participant individually [41].

System Architecture and Topology. Video-conferencing applications may choose different architectures for interconnecting meeting participants. Direct peer-to-peer (P2P) connections between meeting participants result in the lowest latency between two given clients but can be challenging to realize in the presence of firewalls and network address translators (NAT) [3, 17, 23]. Additionally, P2P meetings with many participants become infeasible due to the quadratically growing number of streams between the clients. As a result, most VCAs use P2P connections only for two-party calls (e.g., Zoom) or not at all (e.g., Google Meet) [35]. The alternative is to use an intermediate conferencing server that either transcodes incoming streams to a single outgoing stream per participant or one that selectively replicates and forwards media streams among clients. The server in the latter approach is called a Selective Forwarding Unit (SFU). SFUs enable the use of SVC and are the de-facto standard in video conferencing today. Figure 1 illustrates the difference between a P2P meeting and one using an SFU.

Finally, as the space of possible implementation choices is vast, WebRTC provides a common framework for implementing real-time collaboration and communication systems [4]. WebRTC provides several APIs, means to coordinate the use of the various protocols, a standard congestion control algorithm (Google CC [9]), and a common set of media codecs.

3 WHAT IS (NOT) KNOWN ABOUT ZOOM

Prior studies have reported on the flow-level structure of Zoom meetings [10, 12, 25, 35] and shed light on Zoom's rate-adaptation algorithm through controlled experiments [10, 25]. Additionally, Zoom provides high-level documentation about some internals of their system [52, 56, 59]. In contrast to prior results, we are interested in extracting finer-grained, continuous performance metrics, such as frame rates and frame-level jitter, from large-scale passive packet captures. We will now give a brief overview of prior findings, and outline why they fall short in enabling our target use cases for operators and researchers outlined in Section 1.

Server Traffic. Zoom publishes the list of IP subnets they use [59] to aid operators with firewall configuration. Prior works have used this list to filter traffic to and from Zoom servers and analyzed the resulting traffic from a client's perspective. The network traffic observable during a Zoom call consists of several TCP connections to various Zoom servers and between one and three UDP flows to a single Zoom server. The TCP connections use server port 443 and carry TLS-encrypted data [12] which is presumably control traffic. The UDP flows carry the actual media traffic; there is always one flow per media type in use (audio, video, or screen sharing), regardless of the number of participants in the meeting. Prior works have confirmed this by enabling and disabling audio, video, and screen sharing during a meeting and observing the respective flows appear or disappear in their network trace [10, 12, 25, 35]. The media flows use ephemeral port numbers at the client and port 8801 at the server [10, 12].

P2P Traffic. On top of connections to Zoom servers, prior studies have observed that Zoom media traffic switches to a direct, peer-to-peer (P2P) connection between participants for meetings with exactly two participants. In this case, all three media types are sent over the same UDP flow which uses ephemeral port numbers at both peers. When the meeting media traffic switches from being sent through a server to P2P, the new single UDP media flow starts with a new port number [10, 12, 25, 35]. Meetings with a single participant (before any others have joined) start transmitting to a server. Once the second participant joins, the Zoom client then sometimes establishes the direct P2P connection within tens of seconds. As soon as a third participant joins, the meeting reverts back to using a server where it then stays even if the number of participants goes back to two [12]. No prior work has been able to deterministically detect and filter these P2P connections due to the unknown peer IP addresses and ephemeral port numbers, as opposed to the public IP and port information published for Zoom servers.

Header Format. Only Nisticò et al. [35] went beyond flow-level characteristics of Zoom's network traffic by reporting that Zoom uses RTP embedded in a custom, undocumented four-byte header. They observed two different values in this header. Their study does not go further than this and their publicly available tool to extract RTP headers from Zoom traffic [33] (from May 2020) was not able to extract any headers from our traces collected during 2021 and 2022. We further discuss this in Section 4.2.

Rate Adaptation. Finally, prior works on Zoom's session quality and rate adaptation [10, 25] used controlled experiments where they injected a video stream at one side of a two-party meeting and compared a capture at the receiver with the original video. Lee et al. [25] studied Zoom's rate-adaptation algorithm by monitoring its bandwidth use and frame rate (using a timestamp in the source video) while injecting cross-traffic. They find that Zoom media streams adapt to network congestion primarily by adjusting the sender's bit- and frame rate, as opposed to adjusting the stream at the SFU. Furthermore, the paper reports that Zoom uses jitter as opposed to absolute delay for rate adaptation. Chang et al. go further by also comparing the Structural Similarity Index (SSIM) [47] of the sent and received videos to quantify picture quality degradation. The authors find that Zoom's video encoding is sensitive to the type of video with high-motion videos significantly reducing the received video quality. It has been reported that Zoom uses SVC over AVC for rate adaptation and scaling of video streams [19].

Taken together, all prior studies on Zoom report operational details at a *flow level*, e.g., overall packet and data rates of Zoom traffic. Unfortunately, this level of detail is not sufficient to obtain deeper understandings of the performance of a Zoom meeting. For example, previous approaches are unable to deterministically differentiate audio from video packets, quantify packet loss, or infer frame rates or latency from Zoom traffic. Attributes and metrics like these are (at a minimum) required to estimate if an ongoing meeting suffers from poor quality as experienced through, for instance, low frame rates, poor video resolution, or audio lag. In Section 4, we outline our approach to understand Zoom's network protocols in more detail and extract such *packet-level* metrics.

4 DEMYSTIFYING ZOOM'S PROTOCOLS

We identified three main questions that must be answered to understand Zoom's network protocols in detail and to estimate Zoom performance:

(1) How do we reliably detect all Zoom traffic (including P2P connections) in a network? *(Section 4.1)*
(2) What is Zoom's header format and what information can be extracted from individual packets? *(Section 4.2)*
(3) How do we group packets belonging to the same meeting together to make meeting-wide inferences? *(Section 4.3)*

We developed a set of methodologies to dissect and analyze the captured traffic and use these methods on data gathered from controlled experiments to answer the questions above step-by-step. In contrast to earlier work, we do not use our controlled experiments to draw conclusions about Zoom performance or rate adaptation specifics, rather, we use them to understand enough about Zoom's protocols to enable large-scale passive measurement studies in the future. Even though only applied to Zoom traffic in this paper, we believe that our approach is applicable to studying other proprietary, black-box protocols (cf. Section 8).

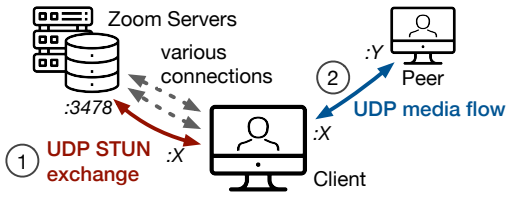


Figure 2: Connection Establishment in a P2P Meeting.

4.1 P2P Connection Detection

While previous work has reported that Zoom uses P2P connections for two-participant meetings [10, 12, 25], no prior work has been able to deterministically detect this traffic due to the use of ephemeral port numbers at both ends and the fact that the IP addresses belong to the clients and are not publicly known (in contrast to Zoom's server addresses).

We observed that before any P2P connection is established, each client exchanges *Session Traversal Utilities for NAT (STUN)* [30] packets with a Zoom server. This exchange determines if a P2P connection is possible and always uses UDP port 3478 (the well-known port for STUN) on the server side and the ephemeral port which is later used for the P2P connection on the client side. The packets are a series of STUN binding requests and are transmitted in cleartext. Figure 2 illustrates this process with *:X* denoting the UDP port number that is later used for the media flow.

These observations about the P2P connection establishment process allow us to reliably capture not only server-based, but also P2P Zoom traffic within and leaving our campus. To do so, we observe STUN packet exchanges with Zoom servers, store the IP address and ephemeral port number used at the client together with the time of the STUN exchange. If the same client then uses this port number within a configurable timeout again to communicate with another IP address, we treat this traffic as a Zoom P2P media flow. While this method, depending on the timeout used, can lead to false positives due to port reuse, in all traffic that we collected, all resulting P2P flows did actually contain Zoom media traffic. Even if false positives are collected, they can easily be filtered out by inspecting the packet format (cf. Section 4.2).

4.2 Entropy-based Header Analysis

Previous work [35] has reported that Zoom uses RTP within a custom four-byte header but does not go further than this (cf. Section 3). The Zoom traffic that we captured in late 2021 and early 2022 uses a different packet format than the one described in previous work and, as a result, we could not reproduce the prior findings. We assume that Zoom changed the used header format since the publication of this particular work in 2020.

Our findings are also vulnerable to becoming (partially) invalid if Zoom's protocols change. Many of our techniques, however, are based off widely-used protocols (e.g., RTP, RTCP, and STUN) and we do not expect Zoom to dramatically change the set of standard protocols employed. Of course, the way Zoom uses and encapsulates them may change, which would require slight modifications to our application logic. For this reason, we share in detail our methodology of analyzing Zoom's header format; this methodology can be repeated if the format changes.

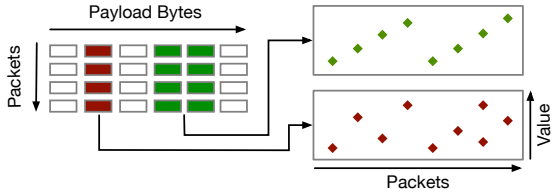


Figure 3: Entropy-based Packet Header Analysis.

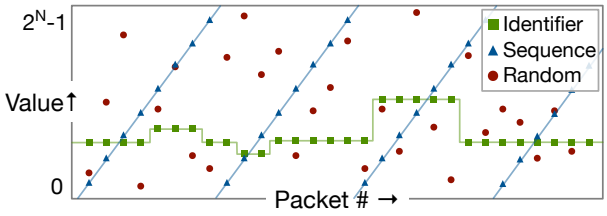
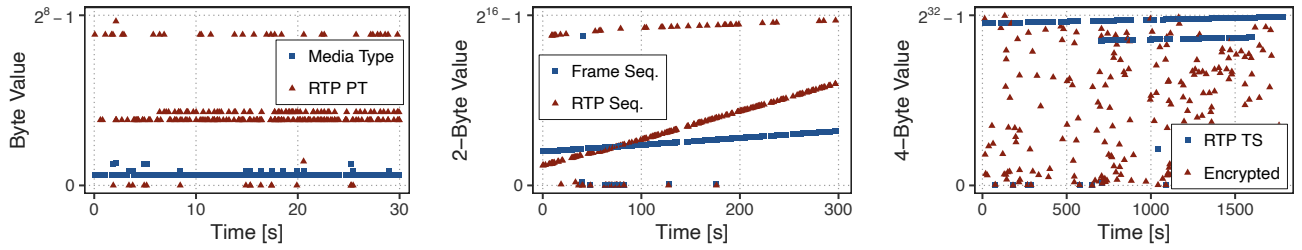


Figure 4: Patterns Observed in Packet Header Analysis.

4.2.1 Finding Unencrypted Header Fields in Zoom Traffic. While Zoom claims that all media traffic is encrypted [55], it does not specify at what granularity encryption is applied and what information may be transmitted in the clear. To find out whether all UDP payload is encrypted or not, we wrote a program that extracts the (binary) values of 8, 16, and 32-bit blocks at various offsets from the beginning of the UDP payload across all packets within a UDP flow. For example, for four bytes of payload, this results in four 8-bit, two 16-bit, and one 32-bit value sequences. We then plotted each such sequence with the packet index on the x-axis and the respective byte range's values on the y-axis to see if the values appear to be uniformly distributed and show maximum entropy, as expected in encrypted data. This approach, illustrated in Figure 3, allowed us to quickly and visually inspect Zoom's protocol. Oftentimes, after finding parts of the payload that does not seem to be random (or encrypted), we manually adjusted the block sizes and offsets of the value sequences extracted to further inspect the protocol fields.

We automatically generated hundreds of such plots; they show three different types of value distributions. For illustration, Figure 4 depicts (fabricated) examples of these distribution types: values are either entirely randomly distributed (red dots), follow horizontal lines (green squares), or follow angled lines (blue triangles). Several such lines, often with different slopes, usually overlap at the level of a UDP flow; angled lines commonly wrap around. We suspect that randomly distributed points, especially when covering the entire value space, indeed belong to encrypted portions of the header. Data points following horizontal lines are likely either identifiers (e.g., a stream identifier) or a bitmask (e.g., as in TCP flags). Data points following angled lines are probably either sequence numbers, timestamps, or counters, depending on the range covered and whether the values are monotonically increasing or not.

Figure 5 shows actual examples of such plots from Zoom packets that we collected during controlled experiments. Each plot is labeled with the two different header field types that we inferred. Note that not necessarily every data point is the inferred type as other, non-RTP/RTCP packets are typically interleaved with media packets. Figure 5a shows two different 1-byte-wide value sequences of a single UDP stream over 30 seconds (250 randomly sampled points);



(a) 1-byte Variables (b) 2-byte Variables (c) 4-byte Variables

Figure 5: Examples of Extracted 1, 2, and 4-Byte Ranges From a Single Zoom UDP Flow and Their Inferred Variable Type.

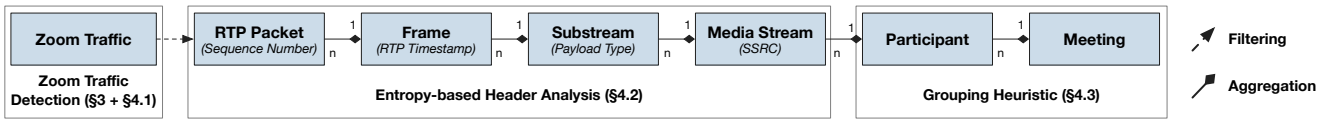


Figure 6: Aggregation Levels Within Zoom Meetings With Used Methodology.

the sequences correspond to a superset of bits that contain the Zoom media type and the RTP payload type, respectively. Figure 5b shows the same type of plot with two 2-byte-wide fields, corresponding to the frame sequence number and the RTP (packet) sequence number. Lastly, Figure 5c shows sequences corresponding to the RTP timestamp and to four bytes of encrypted payload.

As we were expecting to find RTP headers, we started looking for the most discernible pattern within the RTP header, which is a two-byte sequence-type field (RTP sequence number), followed by a four-byte sequence-type field (RTP timestamp), followed by a four-byte identifier-type field (SSRC). Using this methodology, we are able to detect RTP headers at various offsets in most packets within our trace. We could confirm the presence of RTP headers also by checking the other header values for compliance with the protocol specification [40]; for example, the first two bits of the RTP header, the version field, must contain the value `10`.

Finally, suspecting that Zoom also uses RTP's counterpart, the RTP Control Protocol (RTCP), we searched all remaining payloads (where we did not see RTP headers) for the set of SSRC values seen in RTP packets. This is based on the insight that RTCP packets always refer to one or more specific SSRCs and that these values are also carried in the RTCP header. Using this method, we were indeed able to find RTCP *sender reports (SR)* containing timestamps and packet counters being emitted from each sender for each media stream at every second. We did not find any RTCP *receiver reports (RR)* that would contain performance-related information, such as jitter and lost packets. Later on, we will show how these metrics can also be calculated from analyzing the RTP packets alone.

4.2.2 Identifying Different Types of Zoom Media Packets. After finding RTP and RTCP headers at different offsets, we needed a recipe to determine where the headers for a given packet start. To do so, we analyzed the payload before the RTP or RTCP header. We took a group of packets with the same RTP header offset and compared them with groups of packets with a different offset. This method allows us to see if there are any header fields before the RTP header whose values are consistently the same within one group but differ between groups. Such a field could be an identifier for the type of packet.

Field Name	Byte Range	Comment
Zoom SFU Encapsulation		
– Type	0	`0x05` for 98.4% of packets
– Sequence #	1–2	
– Direction	7	`0x00`/`0x04` – to/from SFU
Zoom Media Encapsulation		
– Type	0	media type or RTCP
– Sequence #	9–10	
– Timestamp	11–14	
– Frame seq. #	21–22	only in video packets
– # Packets/frame	23	only in video packets

Table 1: Select Header Fields in Cleartext

We used this method on the eight different groups of RTP/RTCP header offsets in our traces. Using this methodology, we determined that there is a variable-length header before the respective RTP or RTCP header where the first byte indicates the type of packet which also determines where the RTP/RTCP header starts; we will refer to this header as *Zoom Media Encapsulation*. While P2P traffic starts with this header directly (after the UDP header), server-based traffic first has another fixed-length, 8-byte header; we refer to this header as *Zoom SFU Encapsulation*. The overall structure of these headers is depicted in Figure 7. Both headers start with a one-byte identifier; we refer to them as *type* fields. The vast majority of SFU encapsulation headers (98.4% of Zoom server-based UDP packets in our trace) start with the type value 5. We found that this value indicates that a Zoom media encapsulation header is following. The remaining header fields we identified and their respective locations within the headers are listed in Table 1.

The vast majority (90.03%) of media encapsulation headers in our trace start with the values 13, 15, 16, 33, and 34. Types 13, 15, and 16 are used for RTP media packets and indicate the media type while packets of types 33 and 34 contain RTCP headers. Table 2 shows the mapping between type values, corresponding payload, and offset (from end of the UDP header) where the encapsulated payload starts.

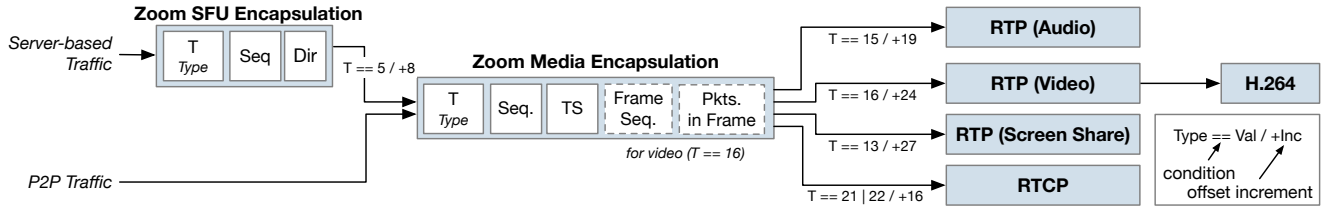


Figure 7: Simplified Structure of Zoom's Custom Headers.

Value	Packet Type	Offset	% Pkts.	% Bytes
16	RTP: Video	24	59.48	80.67
15	RTP: Audio	19	24.77	8.89
13	RTP: Screen Share	27	4.62	4.90
34	RTCP: SR + SDES	16	0.89	0.09
33	RTCP: SR	16	0.27	0.02
		Sum:	**90.03**	**94.57**

Table 2: Zoom Media Encapsulation Type Values

Media Type	RTP PT	Description	% Pkts.	% Bytes
Video (16)	98	main stream	62.00	79.27
Audio (15)	112	speaking mode	22.04	7.92
Video (16)	110	*FEC*	6.14	7.47
Screen Share (13)	99	main stream	3.59	3.72
Audio (15)	113	mode unknown	2.96	0.89
Audio (15)	99	silent mode	2.60	0.56
Audio (15)	110	*FEC*	0.62	0.13
		Sum:	**99.98**	**99.99**

Table 3: RTP Payload Types Values in Trace.

The table also includes the percentage of packets or bytes that contain the specific type in our trace (see more in Section 6); we were able to decode over 90% of all Zoom packets as media-carrying packets (94.5% of bytes). We conjecture that the remaining less than 10% of packets carry other control information, e.g., congestion control packets. While we did see some sequence numbers in such packets, we did not further analyze their payload.

4.2.3 How Zoom Uses RTP and RTCP. RTP follows a structure of aggregation levels to map a packet to a media stream (cf. Section 2). The levels are depicted in the center part of Figure 6 and Zoom uses them in the following way:

- To identify **media streams** (i.e., a participant's audio or video), Zoom uses a limited set of *synchronization source identifiers (SSRC)*. These are unique within a meeting, yet neither globally unique nor appear to be randomly sampled as specified in the relevant RFC [40].
- Each Zoom media stream carries between one and three **sub-streams** identified by an RTP *payload type (PT)*. For audio and video streams, we see the combination of PTs 99, 110, 112, and 113, and 98 and 110, respectively. The sub-streams with PT 110 are not always present and generally make up the minority of packets in a stream. If present, they use the same timestamps but different sequence numbers than the other sub-stream. We suspect that these streams carry forward error correction data (FEC). In audio streams, we either see packets of types 99 and 112 interleaved or type 113 exclusively. Through controlled experiments, we found that sub-stream 112 carries audio packets when the respective participant is talking (or emitting any significant sound). During periods of silence or only background noise, Zoom uses fixed-size (40B of RTP payload) packets of type 99. This enables us to quantify how much and when a participant actually talks during a meeting when not muted. When type 113 is used, we cannot tell if the participant talks or not; we saw type 113 used when joining a Zoom meeting from the mobile app. Screen sharing always uses type 99. Our trace occasionally contained packets with other payload types where we do not understand their meaning; however,these packets made up less than 0.02% of the over 1.5 billion media packets in our trace. Table 3 shows the RTP payload types we understand and their relative frequency.
- Each sub-stream carries **frames** identified by a *timestamp* corresponding to the time when the media was sampled. These timestamps are not expressed in wall-clock time but depend on the sampling rate (see [40]).
- Each frame can be spread over several **packets**. Each packet within a sub-stream is uniquely identified by a *sequence number*. The last packet of a frame generally has RTP's *marker* bit set.

Further Observations. The *contributing source count (CSRC count)* in Zoom RTP packets is always zero indicating that every RTP packet only carries a single media source. This suggests that Zoom indeed uses an SFU architecture and not a multipoint control unit (MCU) where media is transcoded and resulting frames effectively carry several signals (sources).

The RTP header in media packets includes RTP extensions and is followed by a *H.264 fragmentation unit (FU) network abstraction layer (NAL)* header in case of video packets. After this header, the remaining payload appears to be encrypted. We did not further investigate the payload of audio packets.

The RTCP packets that we see are only *sender reports (SR)*. RTCP sender reports accompany a media stream and are used to periodically synchronize wall-clock time with RTP timestamps by carrying an NTP timestamp. This mechanism ensures that receivers play media at the right speed and that different streams from the same source (e.g., audio and video) are synchronized. Some SRs also include a *source description (SDES)* which is, however, always empty.
Wireshark Plugin. We wrote a Wireshark dissector plugin incorporating all our findings. Using this plugin, which we make available to the community, it is easily possible to analyze Zoom traffic. More details on the plugin can be found in Appendix C.

4.3 Grouping Streams into Meetings

Finally, we developed a heuristic to group media streams belonging to a single meeting together. This enables us to (1) calculate round-trip-time (RTT) between our monitor and the SFU by comparing copies of the same stream while going to and coming from the

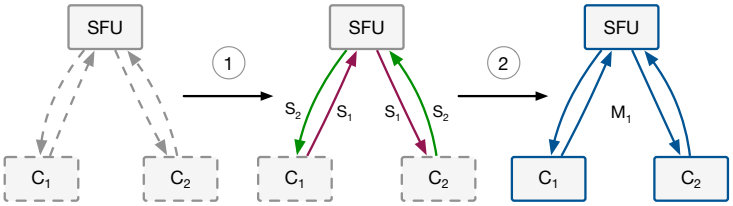


Figure 8: Process for Grouping Streams Into Meetings.

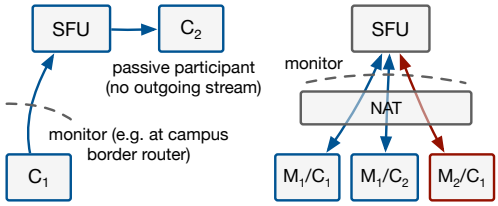


Figure 9: Limitations of Grouping Heuristic.

SFU and (2) judge whether only a single participant is affected by poor meeting performance or if the meeting in general suffers from problems. During our experiments and campus-level traffic analysis, we did not find a meeting identifier in Zoom's packet headers, and as a result, we need to rely on other flow properties and header fields including IP addresses, port numbers, SSRC, RTP timestamps, and sequence numbers to group streams.

4.3.1 Challenges Associated with Grouping Streams. Developing such a heuristic, however, is challenging for several reasons: (1) Locally used port numbers and server/peer IP addresses and port numbers change when a meeting switches between SFU and P2P modes. As a result, a heuristic cannot easily associate an IP address with a meeting and must account for changes in the meeting mode. (2) SSRCs used by Zoom are unique within a meeting but not globally unique nor randomly chosen (as specified in the RTP RFC), making it difficult to detect duplicates of a particular meeting stream (i.e., after stream replication by the SFU) solely through its SSRC. (3) A participant that mutes their microphone and has their camera turned off also does not emit any media streams despite being a participant in the meeting. As network-level heuristics for this task rely on observing media streams, depending on the vantage point, such passive participants may be invisible. This point, in particular, compromises the accuracy of any such heuristic and makes it difficult to validate their accuracy. Fortunately, to estimate performance, we are only interested in active participants as there is no stream performance to measure when there is no media stream.

While the above-mentioned limitations can compromise the accuracy of the participant count and meeting duration, one of the main purposes of this heuristic for our study is to enable fine-grained RTT estimation. For this, we look at *copies* of a media stream observable from our vantage point. This occurs when an on-campus participant sends a media stream which is then replicated by the SFU and sent "back", through our monitor, to another on-campus participant. We explain this method in more detail in Section 5.3. Detecting stream copies, which is the only part of the heuristic required for RTT estimation, is based on four different features (time, SSRC, RTP sequence number, and RTP timestamp) that all need to match, making it relatively robust.

4.3.2 Heuristically Grouping Streams into Meetings. Despite these limitations, we developed a heuristic to provide an estimate of the number of meetings and participants in a meeting. As an example, consider an SFU-based, audio-only Zoom meeting with two participants (C_1 and C_2) as illustrated in Figure 8. Each participant sends their audio stream (S_1 and S_2) to the SFU which forwards it to the other participant. Our heuristic consists of two steps.

Step 1: Finding Duplicate Streams. The first step reads RTP packet records and groups them into media streams identified by IP 5-tuple and SSRC. Whenever a new stream is created (i.e., when the stream key does not exist yet), it checks if there is an existing stream with the same SSRC (but different 5-tuple) where the most recently seen RTP timestamp is within a small range of the first RTP timestamp of the new stream. This is based on the insight that during a transition between P2P and SFU modes the IP 5-tuple of the flow changes, but not any of the RTP-level information (e.g., SSRC). Also, Zoom's SFU does not translate timestamps or sequence numbers and, as a result, an outgoing media stream that is sent back to a different client within a campus (i.e., within the monitor's vantage) will have the same RTP-level information but appear slightly later (RTT to SFU plus processing time). After this step, every stream that carries the same media (e.g., a single participant's audio or video) is assigned a unique identifier (here S_1 and S_2).

Step 2: Assigning Streams to Meetings. The second step operates on stream records, including SSRC, IP 5-tuple, stream start and end time, and, most importantly, the unique identifier from step one. This identifier (based on RTP header values) greatly increases the accuracy of the following algorithm. The algorithm starts by assigning the first stream to a new meeting. For every subsequent record, it then decides whether to assign the stream to an existing meeting or to create a new meeting for the stream. The heuristic maintains mappings from the unique stream id, the client's IP, and client's IP and port combination to meeting identifiers. If a lookup in this data returns at least one match, the stream is assigned to the respective meeting. If there are several matches with different meeting ids, the matched meetings are merged. If there is no match, a new meeting is created.

This heuristic works well in most cases but its efficacy is constrained, as mentioned above, by the vantage point where packets are captured. The two key issues of our approach are depicted in Figure 9. The left side of the figure shows the above mentioned case of a passive meeting participant whose media stream is not visible at the monitor. The right side shows the issue that arises when NAT is used within the campus (e.g., by connecting a personal hotspot) or when the monitor is placed outside of a large-scale NAT. Here, meetings M_1 and M_2 could be merged as they appear to the monitor as using the same IP address.

5 ESTIMATING PERFORMANCE METRICS

Understanding parts of Zoom's packet headers, enables us to extract and analyze a wide range of protocol fields from media traffic. However, extracting the fields does not, by itself, provide useful insights into meeting performance. In this section, we discuss our methods of deriving various network-level and performance metrics

Metric	Requires Headers	Available in Z. Client	Validated
Overall Bit Rate (§5.1)			
Media Bit Rate (§5.1)	•		
Frame Rate (§5.2)	•	•	• (Fig. 10a)
Frame Size (§5.2)	•		
Latency (§5.3)	•	•	• (Fig. 10b)
Jitter (§5.4)	•	•	• (Fig. 10c)

Table 4: Key Zoom Performance and Quality Metrics

from Zoom traffic and show how we validate our methods using statistics provided by the Zoom client application. Table 4 provides an overview of the most important metrics discussed below.

Validation of Metrics. Zoom provides session performance statistics to users and operators in three different ways: (1) in the Zoom client application's *Statistics* window, (2) through a dashboard and REST API for operators [57], and (3) programmatically through an SDK [58]. The available metrics are latency, jitter, packet loss, audio frequency, video resolution, and video frame rate. Each metric is available for each of the three media types. Both the Zoom client and the SDK seem to update their data roughly once per second while the REST API provides updated data once per minute [54].

Using the GUI would require us to automatically take frequent screenshots of different tabs in the Settings window and process the images later. The API's update rate of once per minute is too coarse-grained. Consequently, we decided to use the SDK to validate our methods and compare the accuracy of our metrics to the ground truth provided by Zoom. We wrote a custom Zoom client application using the macOS SDK in Objective-C based on the example code provided by Zoom [58]. The application opens a standard Zoom meeting window and session. The traffic generated by this client looks exactly like the traffic generated by the public Zoom client. We instrumented the code to log all available performance metrics for both audio and video once per second (the finest granularity that the API supports).

Using this setup, we performed a series of controlled experiments where we collected both the Zoom-provided data and all network traffic for later analysis using our tools. We ran several 5–6 minute-long two-person meetings where we introduced cross-traffic twice during each call by running a network bandwidth test for 10–20 seconds each.

5.1 Overall and Per-Media Bit Rates

Flow-level Bit Rate. The overall data rate of Zoom flows (per IP 5-tuple) can be easily measured from network traffic captures and does not require parsing any Zoom headers. Data rate has been used to (1) characterize Zoom's bandwidth requirements, (2) estimate session quality, and (3) differentiate between audio and video streams (via the relative difference in data rate between flows) [12, 25, 27, 35].

While suitable for quantifying bandwidth requirements, the overall data rate does not correctly characterize session performance or quality. Zoom's rate-adaptation algorithm adjusts the sending picture quality and frame rate in response to network conditions, user interactions, and device capabilities. Recall from Section 4 that the following user interactions and circumstances affect the overall bit rate:

- Moving a little or a lot while on camera causes the video data rate to change rapidly.
- Resizing the video display (e.g., to thumbnails during screen sharing or using "speaker-only" view) at the receiver, can reduce the frame rate by half.
- In Zoom, mobile devices use different aspect ratios and video resolutions than desktop computers, making comparisons between overall data rates difficult.

Furthermore, a single UDP flow may carry several, individual media streams. Thus, overall flow-level bit rate is insufficient to estimate meeting quality as low bit rates can be caused by other factors. Moreover, differentiating between media types based on relative bit rate of a flow is inaccurate as a stream with low frame rate and low resolution (as possible in thumbnail mode or sometimes during screen sharing) can have a similarly low overall bit rate as an audio stream (more in Section 6). Lastly, in our trace, roughly 10% of Zoom UDP packets carry no media data (cf. Table 2), causing inferences about media bit rate from overall flow bit rate to be inherently inaccurate.

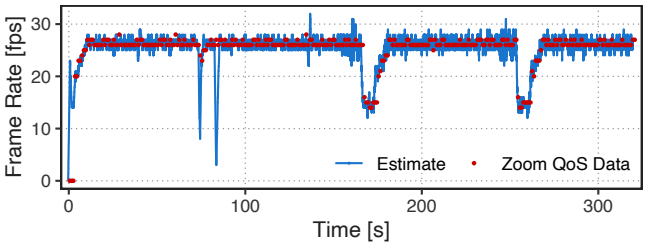
Per-media Bit Rate. Our deeper understanding of Zoom's protocol, including the ability to determine the media type and sender (via SSRC) for each packet, allows us to put packet and bit rates into context. Further, knowing exactly which packets carry media and where the media payload starts, enables us to compute the actual media bit rate.

5.2 Frame Rate and Frame Size

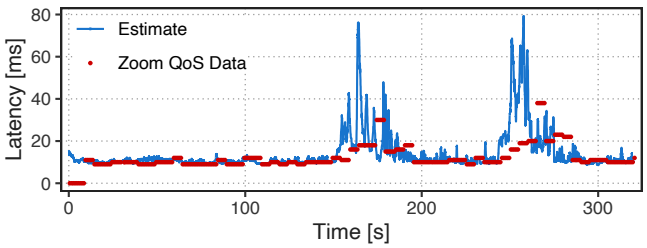
Frame Rate. The frame rate of a Zoom video or screen-sharing stream can precisely be calculated in two ways. As each frame is identified by an RTP timestamp for the decoder to know when to play back the frame, frame rate can be estimated simply by counting the number of unique timestamps seen in a 1-second period. The second approach is to derive the encoder's frame rate from the increments of the RTP timestamp in conjunction with the stream's sampling rate. The results of this approach can be different from the first method in the presence of network congestion; we explain the difference in more detail after describing both methods.

Method 1: To compute the frame rate, we use a circular buffer to store all frames that were *completely* delivered within the last second. We discovered a field in the Zoom Media Encapsulation header that contains the number N of packets in a given frame (cf. Section 4.2). We consider a frame complete when we see N distinct (per sequence number) RTP packets with the same RTP timestamp. Whenever a new frame completes, we check if the head of the buffer still falls within a 1-second interval from the current time and remove it if not. The current frame rate is then simply the occupancy of this buffer and can be computed at any time (e.g., for each frame). Using this method, we can also calculate how long it takes to deliver a given frame which can then be compared to the time the frame covers in the media stream, the *packetization time* [40] (more in Section 5.5).

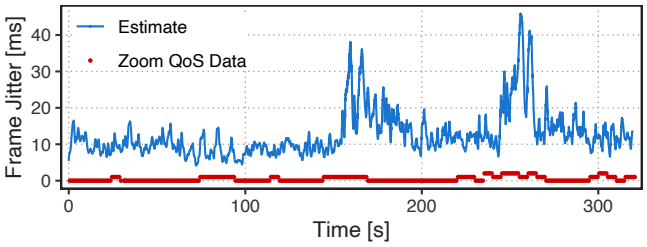
Method 2: A second approach to calculate frame rate leverages the RTP timestamp to exactly calculate the encoder's (i.e., the "intended") frame rate but requires knowledge of the stream's sampling rate. Through a simple parameter sweep and comparing the result with data obtained through the method above, we found that



(a) Frame Rate Estimation Accuracy.



(b) Latency Estimation Accuracy.



(c) Frame-level Jitter Estimation Accuracy.

Figure 10: Estimation Accuracies From Single Experiment.

Zoom's video streams use a sampling rate of 90 kHz, which also happens to be the recommended value for sending conferencing video over RTP [39]. The frame rate FR at sampling rate SR for each frame can then be calculated as $FR = SR/\Delta RTP$ where ΔRTP is the RTP timestamp increment from the last frame. The packetization time is then given by FR^{-1}. Note that the encoder's frame rate is not necessarily the rate of successfully delivered frames over the network as computed using the first method; it is rather the frame rate the encoder is currently sending at. In the presence of congestion, the two numbers can temporarily diverge before the encoder adjusts the frame rate, indicating a network problem.

We validated our frame rate estimation based on the first method using the statistics provided by the Zoom SDK. Figure 10a shows the results of one such experiment; our estimate closely matches the data provided by Zoom. The frame rate mostly fluctuated between 26 and 28 fps and dropped temporarily during the competing download. Our data also reveals the frame rate dropping twice before where only the first drop is reflected in Zoom's data due to its relatively low refresh rate and potentially a smoothing mechanism.

Frame Size. Finally, knowing which packets belong to a particular frame, how many packets are expected in a given frame, and where the RTP payload starts, allows us to exactly calculate the size (in bytes) of a media frame. Even though frame size does not account for user interaction (e.g., reduced frame size due to thumbnail mode), together with frame rate, it gives a more accurate estimate of the size, resolution, and quality of the currently displayed picture than the overall flow bit rate.

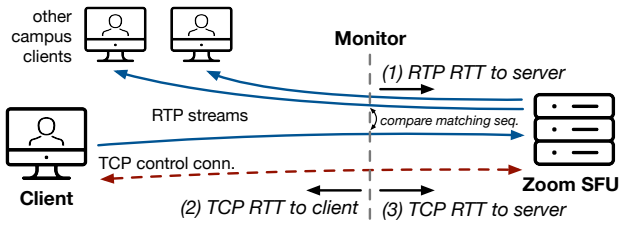


Figure 11: Methods for Measuring Session Latency.

5.3 Latency

End-to-end latency between participants is a key performance metric in video conferencing. The ITU found that users begin noticing difficulty having conversations when the mouth-to-ear latency (i.e., the time from recording an audio signal to playing it back at the receiver) exceeds 200 ms [22]. While we cannot accurately measure mouth-to-ear latency, we developed techniques to (1) estimate the end-to-end latency (through the SFU) between pairs of on-campus participants in the same meeting and (2) estimate the round-trip time between an individual participant and the SFU.

Method 1 (latency via RTP sequence numbers): The first approach to estimate session latency leverages the media stream itself. RTP packets carry sequence numbers and the SFU simply forwards these packets to all meeting participants. As a result, if we monitor a meeting with several participants and have RTP streams carrying the same media grouped together, the RTT between the point where packets are captured and the Zoom SFU can be measured by comparing the egress and ingress timestamps of RTP packets with matching sequence numbers (see blue, solid lines in Figure 11). Depending on media type and quality, each stream produces tens and up to hundreds of packets per second, making this method a way to obtain very frequent RTT probes.

Method 2 (latency via TCP as a proxy): Even if our monitor sees only one of the meeting participants communicating with the SFU, we can leverage the client's TCP control connection (cf. Section 3) to estimate the RTT from the monitor to the client and to the SFU. This is shown by the red dashed line in Figure 11. TCP RTTs can be measured by matching TCP sequence numbers of outgoing packets with acknowledgment numbers of incoming packets [11, 42]. That is, we use TCP RTTs as a *proxy* for the latency of real-time media [2, 32]. Despite providing fewer samples than the RTP-based method due to the comparatively lower packet rate of the control connection, it does help us measure the latency from our vantage point both to the Zoom SFU and the client. The difference between the two can be used to pinpoint whether congestion is located upstream or downstream from the measurement device, e.g., inside vs. outside our campus network.

Figure 10b shows the accuracy of our latency estimation based on the first method for the same experiment and video stream as in Figure 10a. We can see that Zoom only updates its latency estimate every five seconds. Without cross traffic, our estimate matches the data provided by Zoom but yields significantly more data points as

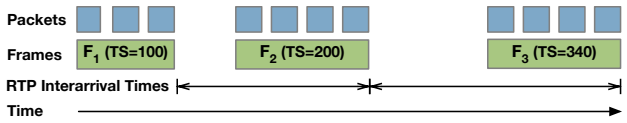


Figure 12: Frame-level Interarrival Time Calculation

we can calculate latency for every single RTP packet. As a result, our method more clearly highlights fluctuations in latency, especially during periods of rapid variation in network condition.

5.4 Jitter

Jitter, defined as the "statistical variance of the RTP data packet interarrival time", provides a short-term measure for network congestion and may indicate congestion before loss occurs [40]. Of the metrics outlined in Table 4, it is the most direct estimator for network quality and therefore important for reasoning whether, for example, low frame rate is caused by the network or by the user's behavior as explained above.

Simply computing the variance in interarrival time between packets is not useful in the context of RTP streams for two reasons. First, a UDP video-conferencing stream can carry several media streams which can then carry multiple sub-streams. The notion of packet order (via RTP sequence numbers) is only valid within each such sub-stream, requiring parsing the respective RTP headers. Second, RTP traffic at the packet level is bursty by nature, as each frame generally spans several packets which are transmitted back-to-back. As a result, we usually see short bursts of packets belonging to a frame, followed by a pause before the next burst, as illustrated in Figure 12. Moreover, the packetization time (i.e., the time a frame covers in the media signal) is not necessarily constant throughout the stream which requires correcting the jitter computation by what the interarrival time for any two frames *should* be. In fact, Zoom uses variable packetization intervals as indicated by variable increments between RTP timestamps in Zoom traffic. We use the formulas outlined in the RTP RFC [40] to compute this adjusted time difference and subsequently the frame-level jitter. Jitter can either be computed in terms of RTP time or as wall-clock time by first converting between them using the stream's sampling rate.

In our experiments, Zoom always reported very low jitter which never exceeded 2ms, even in the presence of congestion. As shown in Figure 10c, our jitter estimate does not match Zoom's estimate but appears more in line with the fluctuation in latency during the same experiment which was also reported by Zoom, especially during the two congestion events (Figure 10b) which even caused Zoom to adjust the video frame rate (Figure 10a). We are surprised by this result as the significant fluctuation in latency depicted in Figure 10b should also be reflected in the (resulting) jitter metric in Figure 10c. We hypothesize that Zoom computes jitter differently, perhaps taking forward error correction into account, or using a very long smoothing interval. We use the jitter computation method recommended for RTP by the corresponding RFC [40].

5.5 Other Metrics

Loss and Retransmissions. While computing the number of lost packets, retransmissions, and out-of-order deliveries for TCP connections is relatively straightforward using TCP sequence numbers, this is ordinarily not possible for UDP traffic. Our ability to parse RTP headers in Zoom traffic, however, allows us to estimate these metrics over an RTP stream. In our controlled experiments, we found that Zoom retransmits lost packets up to 2 times. As a result, we rarely see entirely lost packets in our trace but rather duplicates. Since we are unable to find any explicit loss signals in the traffic, we must rely on analysis of the seen sequence numbers to estimate reordering, loss, and retransmissions. This method, however, can be inaccurate as we are not able to differentiate between retransmissions and regular packets (apart from observing elevated jitter). Some observed reorderings might also be due to retransmissions. In conclusion, Zoom's use of retransmissions makes it fundamentally difficult to infer loss and packet reordering from in-network measurements and sequence numbers alone and we require other metrics (e.g., jitter) to assess network condition.

Frame Delay. Finally, we can measure the time between the first packet of a frame and the time the frame is fully delivered; we call this time *frame delay*. High frame delay (in comparison to other frames in the stream) indicates that retransmissions took place to fully deliver the frame. In those cases, we observed that the frame delay is elevated by at least the current RTT to the SFU plus a timeout that appears to be 100ms. As retransmissions in Zoom use the same RTP sequence numbers as the originally lost packet, it is not straightforward to detect lost packets if the packet was dropped before the vantage point. Observing a packet with suspiciously high delay (i.e., 100ms + RTT) delivered out-of-order, however, is a strong indicator that the respective packet was retransmitted in response to loss.

Also, we can compare a frame's packetization time with its delay. If the delay is larger than the packetization time over the course of several frames, the jitter buffer gets drained and the video will eventually stall. We leave the detection and deeper analysis of audio and video stalls based on this metric for future work.

6 ANALYZING ZOOM CAMPUS TRAFFIC

We now show how our methods can be applied to a large dataset by first describing our capture system and then discussing performance metrics computed over a 12-hour trace collected at our campus network. The trace contains 1.8 billion Zoom packets and 59,020 RTP media streams. More details on the trace are summarized in Appendix A. Our ethics statement can be found in Section 9.

6.1 Scalable P4-Based Zoom Traffic Capture

To study video-conferencing systems, we must first identify and capture their traffic. A commonly employed method is capturing a large volume of on-campus traffic using a tool like *tcpdump* and then extracting packets of interest—Zoom packets, in this case—in a post-processing step. However, this approach does not scale when the traffic rate ranges from several Gbps to tens of Gbps, as is the case on our campus. Bottlenecks exist at links from our packet broker system to our collection server where *tcpdump* runs, as well as packet and disk I/O at the server. Storage space also becomes an issue. Fortunately, our campus traffic capture setup is equipped with a high-speed programmable switch, namely the Intel Tofino switch [20]. The switch sits between the packet broker system and the server where *tcpdump* operates. This allows us to deploy a

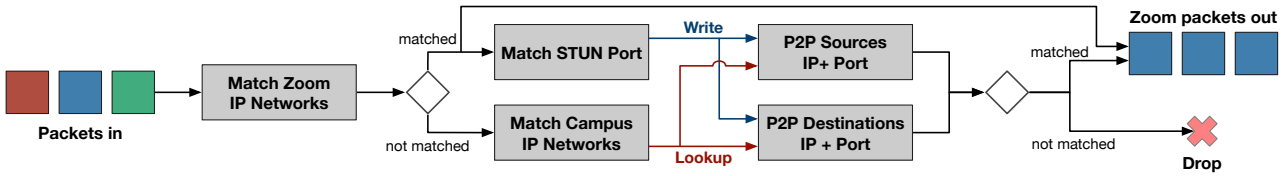


Figure 13: Zoom Packet Capture Program Implemented in P4 for the Intel Tofino Programmable Switch.

data-plane program (written in the P4 [6] language) that takes all campus packets as input and only allows Zoom packets to pass through to *tcpdump*.

Our campus is connected to the Internet via two separate gateways. At both gateways and in both directions each we have taps installed that passively capture all packets and forward them to our packet broker system. To manage overall capture volume, we do exclude some of our campus' internal subnets from the capture at the broker. These excluded subnets mostly belong to research computing facilities that run large bulk transfers to the Internet but are unlikely to contain significant amounts of Zoom traffic. Zoom clients in these unmonitored subnets look to us like external (off-campus) clients. For these clients, we see outgoing streams only if they are being forwarded to a client that is within one of the monitored subnets; we never see the SFU to client leg of incoming streams to those clients. For all other (monitored) subnets, we do see every Zoom packet to and from the Internet. As a result, every stream that we do see is complete but we may miss a limited number of streams entirely which can affect the number of meeting participants that we report.

Figure 13 illustrates the design of our Zoom packet filter system. For TCP and server-based UDP traffic, it suffices to check in a stateless manner whether one of the source or destination IP addresses matches the list of IPs published by Zoom [59]. Detecting P2P traffic, however, requires a more sophisticated stateful approach. As shown in the figure, whenever we *see* a Zoom STUN packet, we write the campus peer's address (IP address and port number) to these registers (cf. Section 4.1). Subsequently, for all future non-server UDP packets, we extract the campus-side address and look it up against our hash tables. The P4 program also performs anonymization for all outgoing packets using an existing system [24].

The resource consumption (by resource type and functional component) of our system on the Intel Tofino Programmable Switching ASIC [20] is shown in Table 5. The percentages in the table refer to the fraction required of the respective resource type available on the Tofino. The table shows that the most complex operation we perform is anonymization, which may be optional in certain production environments. We conclude that our program is lightweight as it uses less than 15% of most of the resource types available on our switch; it can therefore easily and practically be combined with other data-plane processing logic. Our capture system, while designed for Zoom, can be extended to support other applications with known *signatures*, e.g., other video-conferencing applications.

Resource Type	Zoom IP Match	P2P Detection	Anonymization [24]
Stages	2	7	11
TCAM	0.7%	1.0%	1.4%
SRAM	0.1%	10.9%	1.1%
Instructions	1.3%	3.4%	5.2%
Hash Units	0.0%	16.7%	8.3%

Table 5: Hardware Resource Usage of the Tofino-based Capture Program (Divided by Functional Component).

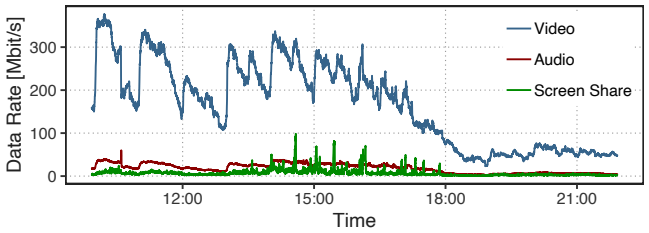


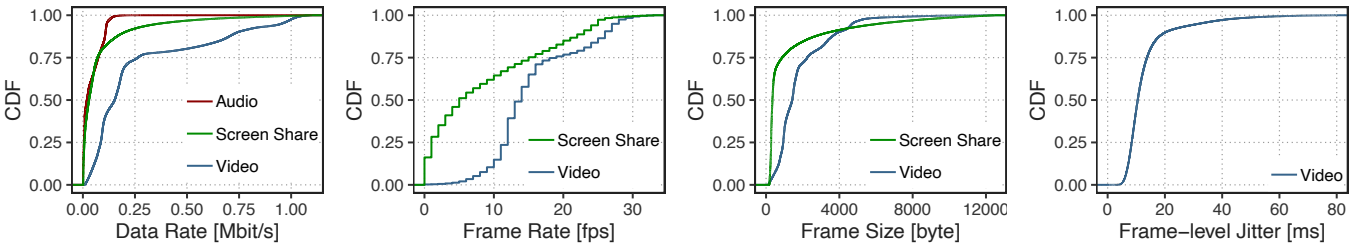
Figure 14: Data Rate per Media Type in Campus Trace

6.2 Zoom Performance Metrics in the Wild

We calculated various performance metrics for each video, screen share, and audio stream in 1-second bins over our entire trace, resulting in roughly 33 million data points for each metric.

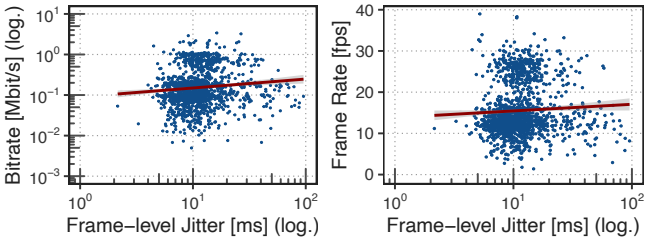
Media Bit Rate. Figure 14 shows the total media bit rate for all Zoom streams per media type over local time. Video traffic makes up the vast majority of data and we can clearly see spikes in bit rate at each full hour and (to a lesser extent) every 30 minutes as meetings presumably begin during those times. There is a dip during lunchtime and significantly less activity after the end of work day. The distribution of media bit rate per media type is depicted in Figure 15a. Interestingly, the bit rate distribution of screen sharing traffic is much closer to that of audio traffic as opposed to video traffic. This illustrates that it is inaccurate to differentiate different stream types based on relative bit rates.

Frame Rate. The distribution of frame rates between screen sharing and video traffic, depicted in Figure 15b, shows that Zoom uses a very fine-grained encoding scheme for screen-sharing traffic where no new frames are generated, presumably when the picture does not change frequently, as is often the case in presentation slides. In fact, roughly 15% of frame rate samples for screen sharing showed a frame rate of zero; approximately half of the samples had a frame rate of five or less with the remainder of the samples being relatively evenly distributed. In contrast, video frame rates are under ten frames per second (fps) only in 10% of the data points with a lot of the probability mass being centered around 11–14 fps. As mentioned earlier, in controlled experiments we observed that Zoom usually tries to achieve a frame rate around 28 fps and in cases



(a) Data Rate (b) Frame Rate (c) Frame Size (d) Frame-level Jitter

Figure 15: Distribution of Performance Metrics per Media Type in Campus Trace



(a) vs. Bit Rate (b) vs. Frame Rate

Figure 16: Lack of Correlation between Jitter and other Performance Metrics

of video thumbnails or massive network congestion reduces the rate abruptly to around 14 fps. Our data reflects this and shows that the majority of video streams have frame rates in this range, meaning that a lot of video is transmitted in this "reduced-fps mode". We assume that the samples in the range of 20–25 fps (approximately 10–15%) are due to short-term adaptations of frame rate or partially delivered frames in the presence of network events (cf. Section 5.2). This data suggest that low frame rates are usually caused by user interaction rather than network instability as there is a disparity between the frequency of low frame rate below 20 fps (almost 75% of cases) and high jitter of more than 20ms (less than 20%).

Frame Size. The frame size distribution in Figure 15c also shows differences between screen sharing and video streams. We see that over half of screen-sharing frames are smaller than 500 bytes but the distribution has a long tail. We assume that the character of the media (need for high resolution but little movement in the image) causes this distribution. Frames that contain a lot of new information (e.g., initial frames, changing slides, etc.) take more data than usual video frames and after that only small, incremental changes are required. In contrast, the majority of video frames are smaller than 2000 bytes and only few are larger than 5000 bytes.

Frame-level Jitter. Since we are not certain about the sampling rate of audio and screen-sharing streams, we only include the results of our frame-level jitter computation for video traffic, where we determined the sampling rate to be 90 kHz (cf. Section 5.4). Overall, most samples have a frame-level jitter below 20ms but the distribution's tail is long. Zoom recommends jitter below 40ms [53]; roughly 5% of samples in our trace show jitter greater than this.

Causes of Low Performance Metrics. Lastly, we illustrate how seemingly poor metrics are not necessarily a result of poor network performance. We mentioned before that low frame rate is often caused by user interactions and meeting characteristics. As an example, Figure 16 shows 1,500 randomly chosen samples from our data set of performance metrics in 1s-bins where we plot video bit rate and frame rate, respectively, on the y-axis and frame-level jitter on the x-axis. Jitter, here, serves as a metric that is mostly influenced by the network whereas the other two metrics are influenced by a variety of factors. We can see that there is no direct correlation between jitter and bit rate, or between jitter and frame rate, meaning that bit rate and frame rate adaptations are in many cases not a result of poor network conditions. Figure 16b also clearly shows the two aforementioned frame rate modes as clusters centered around 14 fps and 28 fps, respectively. As a result, relying on a single metric, like bit rate, alone is not sufficient to estimate the quality of an ongoing meeting; several fine-grained performance metrics must be evaluated in conjunction, which is what our work enables.

7 RELATED WORK

Demystifying Black-box Protocols. Research in reverse engineering of network protocols has a long history, with Skype P2P audio call analysis as one of the earliest works for real-time network applications [3]. Our methodology and techniques are built upon this large body work in protocol reverse engineering [5, 13, 26, 34, 45], including inferring protocol structure from network traces [8, 14, 50]. These works focus on determining boundaries between header fields but do not use entropy-based analysis to infer the semantics of those fields. Additionally, we are the first to use this systematic approach in the context of video conferencing.

Measuring Performance of VCAs. Nisticò et al. performed a controlled experiment on 13 popular Real-Time Communication (RTC) applications to understand how they operate and consume bandwidth [35]. MacMillan et al. analyzed Zoom, Google Meet, and MS Teams to measure their performance and network utilization [27]. Both works provide key insights about how VCAs operate and perform. Yet, they focus on bit rate and link utilization, and how VCAs compare with each other. Our work focuses on Zoom and dives deep into its protocol to extract performance metrics to better understand user experience. Chang et al. measured and analyzed Zoom, Webex, and Meet based on data from over 700 VCA sessions [10]. The video QoE metrics are measured at the Zoom clients, applying Peak Signal to Noise Ratio (PSNR) and structural similarity index (SSIM) analysis on actual video feeds. Our work is about *inferring* the performance and quality of Zoom calls using passive measurements collected in the network data plane.

Zoom Traffic Analysis. Several prior works [10, 12, 25, 27] studied Zoom among other VCAs to uncover implementation details and characterize its performance using controlled experiments. Their findings on Zoom characteristics are summarized in Section 3. Unlike these prior works, we go further by inferring performance metrics from packet traces that are passively collected from a large production network. A second line of work on Zoom focuses on security, instead of performance. Mahr et al. performed a detailed forensic analysis of Zoom, primarily using disk, network, and memory forensics [28]. It demonstrates that it is possible to find users' critical information, such as chat messages, names, email addresses, and passwords, in plain and/or encrypted text. Similarly, a Citizen-Lab blog post [29] provides insights into the privacy and general security properties of Zoom.

8 DISCUSSION

Generalizability. While our work focuses on Zoom, we believe our header and protocol analysis methods can be used to demystify other black-box systems. Also, if Zoom changes its protocol in the future, these techniques can be applied again. Our tools and techniques to study Zoom performance are also largely applicable to other video-conferencing systems that employ RTP for media transfer. As RTP is used in the vast majority of these systems as reported in Table 2 in [35], our techniques for estimating (among others) latency, frame rate, jitter, and bit rate are applicable to a wide range of applications, including Google Meet, Microsoft Teams, Cisco Webex, and Apple FaceTime. Of course, other aspects such as P2P connection detection, mapping of payload types, or other parts of Zoom's behavior reported in this paper are specific to Zoom. We, however, did share our methodology in detail to faciliate future similar studies on Zoom, if required, or on other proprietary protocols. Consequently, even if Zoom were to make their protocol and header format public in the future, our performance measurement techniques will continue to remain relevant and novel.

Limitations. Two of our techniques are fundamentally limited. First, for quantifying loss and retransmissions, we cannot disambiguate with certainty between fresh packets and their retransmitted copies. A heuristic to detect retransmissions could analyze frame delay (Section 5.5). If the delivery of a frame (normally consisting of packets sent back-to-back) takes longer than the connection's RTT, at least one retransmission likely happened within this frame. Second, due to our vantage point, we do not see media streams of off-campus participants that do not send any media (i.e., are completely passive). This is a problem for quantifying the number of overall meeting participants as their media streams are transported entirely outside of our campus and we do not have access to the respective packets. For this reason we are not able to measure the performance of the media streams sent to these participants.

In-Network Monitoring and Control. There will be extra benefits if our performance analysis can run in a high-speed programmable switch at line rate (e.g., Intel Tofino [1, 20]). In particular, the switch's proximity to the client would enable it to take immediate actions in response to degraded Zoom performance. This would benefit both the campus network operators and Zoom. We can already identify and parse Zoom headers in the data plane; the computations of our performance metrics can be implemented in a streaming fashion and are amenable to data-plane implementation. The space constraints of high-speed programmable switches may require approximate data structures limiting overall accuracy. Control actions that may be performed on a switch include selectively forwarding layers in an SVC stream or annotating packets (e.g., using DSCP) based on their type, relative importance, or dynamically in response to congestion. We leave this as future work.

Labeled Datasets for ML-based QoE Inference. While we report performance metrics that affect meeting quality, we do not use these to infer the quality-of-experience (QoE). Our metrics can, however, be used as *features* in a QoE ML inference model where labels can be created by collecting opinions from viewers. To this end, our system can help automatically generate large, feature-rich data sets from real-world traffic. While applied to on-demand video streaming as opposed to real-time conferencing, Bronzino et al. presented a system inferring QoE from network traffic using ML [7]; this approach could be accelerated using our methods.

9 ETHICS

The campus traces used in this study were anonymized with a one-way hash. All packet traces were inspected and sanitized by a network operator to remove all personal data before being accessed by researchers. The media payloads have been removed as well before researchers gained access to the packet traces. This study has been conducted with necessary approvals from Princeton University, including its Institutional Review Board (IRB).

10 CONCLUSION

Zoom is at the forefront of the recent unprecedented surge of video-conferencing traffic. Zoom's proprietary header format and encrypted traffic, however, make it hard for network operators and researchers to understand how Zoom actually operates and performs in the wild. To this end, we demystify Zoom far beyond existing studies by digging deep into its protocol and header format. We show how to extract metrics that closely relate to the quality of a Zoom call, such as media bit rates, frame rate, and frame-level jitter. Our method achieves this by solely inspecting passively collected packet traces, *without any coordination from Zoom clients or servers*. We also create open-source software artifacts to analyze Zoom packet traces, including a Wireshark Zoom plugin [31]. We believe our work paves the way for enabling network operators and researchers to conduct in-depth measurements of Zoom performance in production networks.

Acknowledgments. We thank our shepherd, Ajay Mahimkar, and the anonymous reviewers for their insightful feedback. We also thank Princeton University's Office of Information Technology (OIT), the Office of Institutional Research, and the Institutional Review Board for enabling us to use anonymized campus traffic for our study. We thank Bryon Coltrane from OIT for his help and insight about Zoom usage on campus. This work is supported by DARPA grant HR0011-20-C-0107 and NSF grants CNS-2147909 and CNS-1901510.

REFERENCES

[1] Anurag Agrawal and Changhoon Kim. 2020. Intel Tofino2: A 12.9 Tbps P4-Programmable Ethernet Switch. In *IEEE Hot Chips Symposium*. IEEE, New York, NY, USA, 1–32.

[2] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. 2021. Performance-Driven Internet Path Selection. In *ACM SIGCOMM Symposium on SDN Research*. ACM, New York, NY, USA, 41–53. https://doi.org/10.1145/3482898.3483366

[3] Salman A. Baset and Henning G. Schulzrinne. 2006. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *IEEE INFOCOM 2006*. IEEE, New York, NY, USA, 1–11. https://doi.org/10.1109/INFOCOM.2006.312

[4] Niklas Blum, Serge Lachapelle, and Harald Alvestrand. 2021. WebRTC: Real-Time Communication for the Open Web Platform. *Commun. ACM* 64, 8 (2021), 50–54. https://doi.org/10.1145/3453182

[5] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. 2007. Revealing Skype Traffic: When Randomness Plays with You. In *ACM SIGCOMM*. ACM, New York, NY, USA, 37–48. https://doi.org/10.1145/1282380.1282386

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[7] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2020. Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience. *SIGMETRICS Perform. Eval. Rev.* 48, 1 (2020), 27–28. https://doi.org/10.1145/3410048.3410064

[8] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. 2007. Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis. In *ACM Conference on Computer and Communications Security*. ACM, New York, NY, USA, 317–329. https://doi.org/10.1145/1315245.1315286

[9] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2017. Congestion Control for Web Real-Time Communication. *IEEE/ACM Transactions on Networking* 25, 5 (2017), 2629–2642. https://doi.org/10.1109/TNET.2017.2703615

[10] Hyunseok Chang, Matteo Varvello, Fang Hao, and Sarit Mukherjee. 2021. Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet. In *ACM Internet Measurement Conference*. ACM, New York, NY, USA, 216–228. https://doi.org/10.1145/3487552.3487847

[11] Xiaoqi Chen, Hyojoon Kim, Javed M Aman, Willie Chang, Mack Lee, and Jennifer Rexford. 2020. Measuring TCP round-trip time in the data plane. In *ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure*. ACM, New York, NY, USA, 35–41.

[12] Albert Choi, Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. 2022. Zoom Session Quality: A Network-Level View. In *Passive and Active Network Measurement*. Springer, Berlin, Germany, 555–572.

[13] Paolo Comparetti, Gilbert Wondracek, Christopher Krügel, and Engin Kirda. 2009. Prospex: Protocol Specification Extraction. In *IEEE Symposium on Security and Privacy*. IEEE, New York, NY, USA, 110–125. https://doi.org/10.1109/SP.2009.14

[14] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. 2007. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. In *USENIX Security Symposium*. USENIX Association, USA, Article 14, 14 pages.

[15] Augusto Espin and Christian Rojas. 2021. The Impact of the COVID-19 Pandemic on the Use of Remote Meeting Technologies. *SSRN Electronic Journal* (1 2021). https://doi.org/10.2139/ssrn.3766889

[16] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. 2020. The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic. In *ACM Internet Measurement Conference*. ACM, New York, NY, USA, 1–18. https://doi.org/10.1145/3419394.3423658

[17] Bryan Ford, Pyda Srisuresh, and Dan Kegel. 2005. Peer-to-Peer Communication across Network Address Translators. In *USENIX Annual Technical Conference*. USENIX Association, USA, 13.

[18] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *USENIX Networked Systems Design and Implementation*. USENIX Association, USA, 267–282.

[19] Todd Hoff. 2020. A Short On How Zoom Works. (2020). http://highscalability.com/blog/2020/5/14/a-short-on-how-zoom-works.html

[20] Intel Corp. 2022. Intel Tofino. (2022). Retrieved September 12, 2022 from https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[21] ipinfo.io. 2022. The Trusted Source for IP Address Data. (2022). Retrieved April 3, 2022 from https://www.ipinfo.io

[22] ITU-T. 2003. *One-way transmission time*. Recommendation I.371. International Telecommunication Union, Geneva, Switzerland.

[23] Ari Keränen, Christer Holmberg, and Jonathan Rosenberg. 2018. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. RFC 8445. (July 2018). https://doi.org/10.17487/RFC8445

[24] Hyojoon Kim and Arpit Gupta. 2019. ONTAS: Flexible and scalable online network traffic anonymization system. In *ACM SIGCOMM Workshop on Network Meets AI & ML*. ACM, New York, NY, USA, 15–21.

[25] Insoo Lee, Jinsung Lee, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2021. Demystifying Commercial Video Conferencing Applications. In *ACM International Conference on Multimedia*. ACM, New York, NY, USA, 3583–3591. https://doi.org/10.1145/3474085.3475523

[26] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. 2006. Unexpected Means of Protocol Inference. In *ACM SIGCOMM Conference on Internet Measurement*. ACM, New York, NY, USA, 313–326. https://doi.org/10.1145/1177080.1177123

[27] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. 2021. Measuring the Performance and Network Utilization of Popular Video Conferencing Applications. In *ACM Internet Measurement Conference*. ACM, New York, NY, USA, 229–244. https://doi.org/10.1145/3487552.3487842

[28] Andrew Mahr, Meghan Cichon, Sophia Mateo, Cinthya Grajeda, and Ibrahim Baggili. 2021. Zooming into the pandemic! A forensic analysis of the Zoom Application. *Forensic Science International: Digital Investigation* 36, Article 301107 (2021), 12 pages. https://doi.org/10.1016/j.fsidi.2021.301107

[29] Bill Marczak and John Scott-Railton. 2020. Move fast and roll your own crypto. (2020). Retrieved March 26, 2022 from https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/

[30] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. 2008. Session Traversal Utilities for NAT (STUN). RFC 5389. (Oct. 2008). https://doi.org/10.17487/RFC5389

[31] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. 2022. Zoom Analysis Source Code. (2022). https://github.com/princeton-cabernet/zoom-analysis

[32] Giovane C. M. Moura, John Heidemann, Wes Hardaker, Pithayuth Charnsethikul, Jeroen Bulten, João M. Ceron, and Cristian Hesselman. 2022. Old but Gold: Prospecting TCP to Engineer and Live Monitor DNS Anycast. In *Passive and Active Measurement*. Springer, Berlin, Germany, 264–292. https://doi.org/10.1007/978-3-030-98785-5_12

[33] Maurizio M. Munafo and Martino Trevisan. 2020. Zoom PCAP Cleaner. (2020). Retrieved April 14, 2022 from https://github.com/marty90/rtc_pcap_cleaners

[34] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. 2015. A Survey of Automatic Protocol Reverse Engineering Tools. *Comput. Surveys* 48, 3, Article 40 (dec 2015), 26 pages. https://doi.org/10.1145/2840724

[35] Antonio Nistico, Dena Markudova, Martino Trevisan, Michela Meo, and Giovanna Carofiglio. 2020. A comparative study of RTC applications. In *IEEE International Symposium on Multimedia*. IEEE, New York, NY, USA, 1–8.

[36] Karl Norrman, David McGrew, Mats Naslund, Elisabetta Carrara, and Mark Baugher. 2004. The Secure Real-time Transport Protocol (SRTP). RFC 3711. (March 2004). https://doi.org/10.17487/RFC3711

[37] Colin Perkins, Mark J. Handley, and Van Jacobson. 2006. SDP: Session Description Protocol. RFC 4566. (July 2006). https://doi.org/10.17487/RFC4566

[38] Eve Schooler, Jonathan Rosenberg, Henning Schulzrinne, Alan Johnston, Gonzalo Camarillo, Jon Peterson, Robert Sparks, and Mark J. Handley. 2002. SIP: Session Initiation Protocol. RFC 3261. (July 2002). https://doi.org/10.17487/RFC3261

[39] Henning Schulzrinne and Stephen L. Casner. 2003. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551. (July 2003). https://doi.org/10.17487/RFC3551

[40] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. 2003. RTP: A Transport Protocol for Real-Time Applications. RFC 3550. (July 2003). https://doi.org/10.17487/RFC3550

[41] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2007. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 17, 9 (2007), 1103–1120. https://doi.org/10.1109/TCSVT.2007.905532

[42] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. 2022. Continuous In-Network Round-Trip Time Monitoring. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 473–485. https://doi.org/10.1145/3544216.3544222

[43] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (2012), 1649–1668. https://doi.org/10.1109/TCSVT.2012.2221191

[44] Gary J. Sullivan and Thomas Wiegand. 2005. Video Compression - From Concepts to the H.264/AVC Standard. *Proc. IEEE* 93, 1 (2005), 18–31. https://doi.org/10.1109/JPROC.2004.839617

[45] Ismael Valenzuela and Douglas McKee. 2021. Hacking Proprietary Protocols with Sharks and Pandas. (2021). Retrieved March 28, 2022 from https://www.mcafee.com/blogs/enterprise/security-operations/hacking-proprietary-protocols-with-sharks-and-pandas/

[46] Jean-Marc Valin, Koen Vos, and Tim Terriberry. 2012. Definition of the Opus Audio Codec. RFC 6716. (Sept. 2012). https://doi.org/10.17487/RFC6716

[47] Abhinau K. Venkataramanan, Chengyang Wu, Alan C. Bovik, Ioannis Katsavounidis, and Zafar Shahid. 2021. A Hitchhiker's Guide to Structural Similarity. *IEEE*

Access 9 (2021), 28872–28896. https://doi.org/10.1109/ACCESS.2021.3056504

[48] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (2003), 560–576. https://doi.org/10.1109/TCSVT.2003.815165

[49] Wireshark. 2022. The Wireshark Network Protocol Analyzer. (2022). Retrieved March 30, 2022 from https://www.wireshark.org

[50] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. 2008. Automatic network protocol analysis. In *Network and Distributed System Security Symposium*. Internet Society, Reston, VA, USA, Article 13, 18 pages.

[51] Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, and Changhui Wu. 2021. Loki: Improving Long Tail Performance of Learning-Based Real-Time Video Adaptation by Fusing Rule-Based Models. In *ACM MobiCom*. Association for Computing Machinery, New York, NY, USA, 775–788. https://doi.org/10.1145/3447993.3483259

[52] Zoom Video Communications, Inc. 2020. Connection Process. (2020). Retrieved March 30, 2022 from https://explore.zoom.us/docs/doc/Zoom%20Connection%20Process%20Whitepaper.pdf

[53] Zoom Video Communications, Inc. 2021. Accessing meeting and phone statistics. (2021). Retrieved May 18, 2022 from https://support.zoom.us/hc/en-us/articles/202920719-Accessing-meeting-and-phone-statistics

[54] Zoom Video Communications, Inc. 2021. How QoS Metrics are determined in the Zoom API. (2021). Retrieved April 12, 2022 from https://devforum.zoom.us/t/how-qos-metrics-are-determined-in-the-zoom-api/47891

[55] Zoom Video Communications, Inc. 2021. Zoom Encryption. (2021). Retrieved April 14, 2022 from https://explore.zoom.us/docs/doc/Zoom%20Encryption%20Whitepaper.pdf

[56] Zoom Video Communications, Inc. 2022. End-to-end (E2EE) encryption for meetings. (2022). Retrieved April 5, 2022 from https://support.zoom.us/hc/en-us/articles/360048660871-End-to-end-E2EE-encryption-for-meetings

[57] Zoom Video Communications, Inc. 2022. Zoom Dashboard API. (2022). Retrieved April 12, 2022 from https://marketplace.zoom.us/docs/api-reference/zoom-api/methods/#tag/Dashboards

[58] Zoom Video Communications, Inc. 2022. Zoom Meeting SDKs. (2022). Retrieved May 10, 2022 from https://marketplace.zoom.us/docs/sdk/native-sdks/introduction

[59] Zoom Video Communications, Inc. 2022. Zoom network firewall or proxy server settings. (2022). Retrieved March 30, 2022 from https://support.zoom.us/hc/en-us/articles/201362683-Zoom-network-firewall-or-proxy-server-settings

A SUMMARY OF CAMPUS TRACE

The campus trace we use to report the frequency of different header types in Section 4.2 and for the study of performance metrics in the wild in Section 6.2 was collected at two border routers on our University campus on May 5th, 2022 over the course of 12 hours. The key statistics of the trace are summarized in Table 6. We also instrumented our P4-based capture system to log the total number of processed packets and the number of filtered (Zoom) packets. The resulting packet rates from these counters are depicted in Figure 17. Our Tofino switch processed an average of 626,069 packets per second, with an average of 43,733 per second being Zoom traffic and filtered out.

Capture duration	12h
Zoom packets	1,846 M (42,733/s)
Zoom flows	583,777
Zoom data	1,203 GB (222.9 Mbit/s)
RTP media streams	59,020

Table 6: Capture Summary

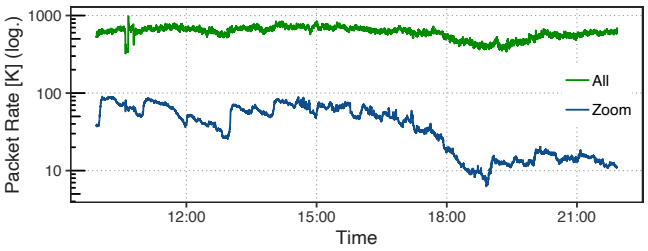


Figure 17: Packet Rate in Campus Trace

B ZOOM INFRASTRUCTURE

The two types of Zoom servers that are most relevant for the actual media transmission are *Multi-media routers (MMR)* which is Zoom's term for an SFU, and *Zone controllers (ZC)*. Apart from serving as a STUN server, we do not have any deeper insights into what ZCs exactly do. Zoom, however, provides some information regarding this in [52].

We analyzed the publicly available list of Zoom IP addresses [59] and performed a reverse DNS lookup for each of them to see which addresses resolve to names of the form *zoom<location><id><type>.<location>.zoom.us* where location is a two-letter identifier for the data center site, id is a simple number of the respective server, and type is either "mmr" or "zc" depending on the type of server. We furthermore performed a GeoIP lookup using ipinfo.io's [21] free service for each of these addresses to see where MMRs and ZCs are located.

Location	# MMRs	# ZCs
United States (all)	3,710	167
– California (multiple)	1,410	68
– New York (New York City)	1,280	62
– Colorado (Denver)	758	21
– Virginia (Washington D.C.)	166	4
– Washington (Seattle)	96	12
Netherlands (Amsterdam)	419	21
China (Hongkong)	274	8
Germany (Frankfurt)	214	2
Australia (Sydney, Melbourne)	210	20
India (Mumbai, Hyderabad)	196	10
Japan (Tokyo)	128	2
Brasil (São Paulo)	124	6
Canada (Toronto)	93	12
China (Mainland)	84	8
Total	**5,452**	**256**

Table 7: Locations of Zoom Servers

At the time of writing, Zoom's official list of IP addresses [59] contains 117 IPv4 networks ranging from /16 to /27 in size, totalling 427,168 IP addresses. Out of these addresses, 156,672 (36.7%) belong to Zoom's own autonomous system (AS30103) which connects to 15 ISPs (mostly Tier-1) and has 4 peerings. 169,152 (39.6%) addresses belong to Amazon Web Services and 99,456 (23.2%) addresses belong to Oracle Cloud. The remaining 0.5% of IP addresses are scattered across autonomous systems of various Chinese ISPs, Level3, and Equinix. 5,452 addresses resolve to names consistent with the MMR naming scheme while 256 resolve to names consistent with Zone Controllers; these are all part of Zoom's own AS. Table 7 shows the countries where MMRs and ZCs are located. It is worth noting that the servers listed for Frankfurt, Germany are located by our GeoIP service to this location but the the naming scheme is consistent with those in Denver, Colorado, USA.

C ZOOM WIRESHARK PLUGIN

To document our findings, aid the community in future research on Zoom, and also to simplify our own initial data analysis, we wrote a plugin for the Wireshark network protocol analyzer [49]. The plugin will automatically treat all UDP traffic to port 8801 as Zoom server-based traffic and can also be manually applied to a P2P flow; once activated, it will dissect all Zoom media traffic based on the findings outlined in Section 4.2. The plugin detects audio, video, and screen sharing packets, RTCP reports, and integrates with various Wireshark analysis capabilities, including RTP stream analysis. Figure 18 shows a screenshot of our plugin in action.

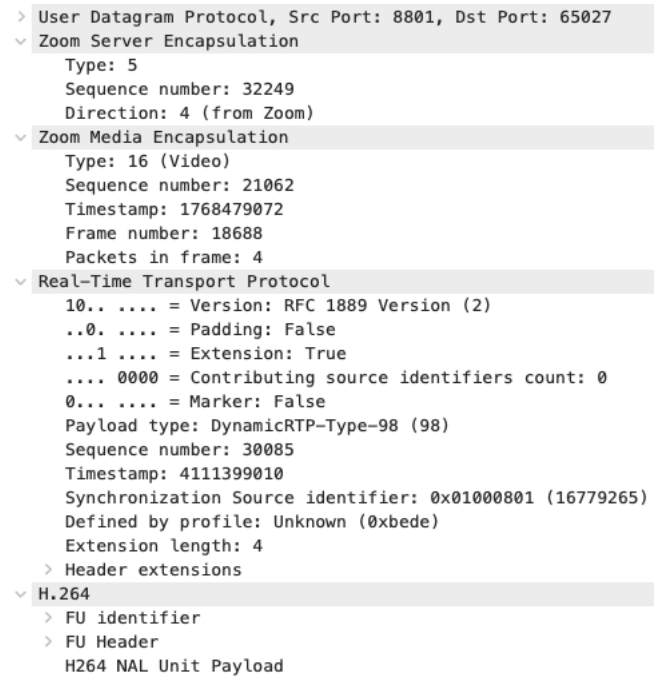


Figure 18: Screenshot of a Zoom video packet in Wireshark's *Packet Details* view using our plugin.