callstats.io

# WebRTC METRICS REPORT 2016/01

Usage metrics from real world WebRTC deployments.

A collection of WebRTC statistics from May 2016 to October 2016 brought to you by callstats.io, the realtime communications monitoring platform.

# A word from our CEO

Thank you for downloading this callstats.io industry report on WebRTC metrics. This year, WebRTC turned five years old and hundreds of products use it. The apps and services vary across several industries, for example, team communication apps, call centres, e-health, and online education.

In 2016, we grew our deployment from under 50 products using callstats.io to over 200. This report is the first one from the series that will be published every quarter by callstats.io, with the next one planned for March 2017. Since this is the first report, we are going through the last six months of sessions, May-October 2016. In this period, our install-base increased 20% month on month and the traffic grew by 9x. Our active customers are located world-wide, lead by U.S and followed by Europe, China, and Russia.

The 24-page report includes 16 charts and 5 tables that provide easy-to-digest insight into the growing WebRTC deployments. This report should help product managers and engineers design better real-time systems. Chiefly, being mindful of potential bottlenecks as they plan for their application service's growth. This report answers to questions like:

- How many participants per call?
- What operating system and browsers dominate WebRTC?
- How often do calls fail to set up and for what reasons?
- How often do participants drop out of a conference due to connectivity issues?

I hope you enjoy reading this report as much as we enjoyed writing it.

You can find the original page for the report at
www.callstats.io/industry-reports/webrtc-metrics-report-2016-01/

Regards,
Varun Singh

# Table of contents

## Executive Summary

Web Realtime Communications (WebRTC) is a technology that enables audio and video calls in the web browser without plugins. WebRTC is part of the HTML5 standard, which is defined at the World Wide Web Consortium (W3C).

WebRTC has a statistics API, which can be used to collect data from the peer connection between two participants. This report presents and discusses anonymised WebRTC metrics collected by callstats.io.

The report discusses metrics sampled across all apps and services using callstats.io. The metrics cover statistics about endpoints, participant-level metrics, setup failure metrics, transport metrics, setup times, and network and media metrics.

### TL;DR

- Microsoft Windows dominates the WebRTC desktop market share and Google Chrome dominates the WebRTC browser market share.
- About 50% of the sessions have two participants, and a growing number (up to 40%) have 3 participants.
- About 4% of the sessions fail to set up due to NATs and firewall traversal. Since February, the failures improved from 12% to 4% because more services deployed variants of TURN relay servers, e.g., TURN/TCP.
- 2% of the sessions use IPv6. The gateways, TURN servers, and media servers are often on infrastructures where IPv6 support is poor or not preferred.
- 23% of the sessions use a TURN server. Of the relayed sessions, 80% use TURN/UDP and rest uses either TURN/TCP or TURN/TLS.
- Improvements to ICE pacing, reduced the set up times for 20% of the sessions.
- 77% of the sessions have no packet loss.
- 80% of the sessions have less than 240ms RTT.

# 1. The WebRTC Media Pipeline

The metrics that are gathered via the WebRTC Statistics API, which corresponds to individual components across the media pipeline. The following section discusses the structure of the media pipeline and the metrics associated with the individual component.

## 1.1. Setup of the WebRTC pipeline

The WebRTC getStats() API exposes metrics from various components within the media pipeline (at the sending and receiving endpoint). Figure 1.1. shows an example media pipeline. Media is captured at regular intervals by a device (microphone, camera, or via screen capture). The raw media frame is then compressed by an encoder and further packetized into MTU (maximum transmission unit ~1450 bytes) before the packets are sent over the Internet. The endpoint on receiving these packets, concatenates these into frames, complete frames are then sent to the decoder and finally rendered. Some implementations may discard incomplete frames, or use concealment mechanisms to hide the missing parts of the frame, these may have varying impact on the quality of experience.

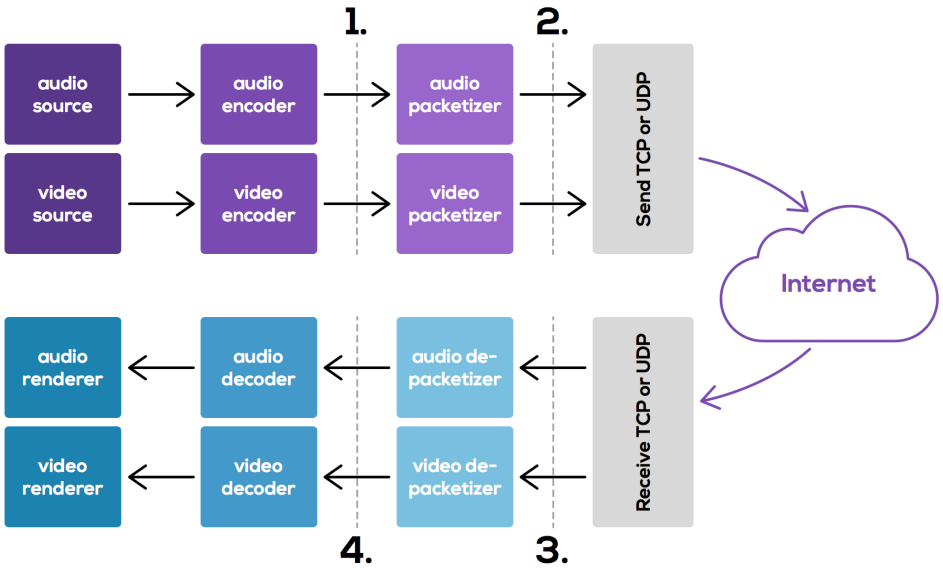



FIGURE 1.1. THE WEBRTC PIPELINE

The getStats() API is structured as follows:

1. Sender media capture statistics: corresponds to the media generation, typically frame rate, frame size, clock rate of the media source, the name of the codec, etc.

2. Sender RTP statistics: corresponds to the media sender, typically packets sent, bytes sent, Round Trip Time (RTT), etc.

3. Receiver RTP statistics: corresponds to the media receiver, typically packets received, bytes received, packets discarded, packets lost, jitter, etc

4. Receiver media render statistics: corresponds to the media rendering, typically frames lost, frames discarded, frames rendered, playout delay, etc.

5. Network-related metrics: corresponds to the bytes, packets, sent or received on that particular interface. Additionally, it may report the connectivity requests and responses.

## 1.2. Applying the WebRTC Statistics

When running a WebRTC service, the metrics exposed by the getStats() API are important for diagnosing the issues within a particular session, i.e. the link between two participants. However, to assess the quality of the overall service, the data from each session in a conference call needs to be summarised. Furthermore, the summarised data needs to be aggregated over period of hours and days to observe and predict the trends in the service usage.

We get a more comprehensive diagnosis when metrics are collected at not only at the endpoints, but at various points in the path, such as, media servers, TURN relay servers, and media routers.

# 2. Endpoint Statistics

As WebRTC is part of the HTML5 standard, it is supported by various platforms and browsers. Currently, WebRTC is supported by Chrome (and variants, like Opera), Firefox, and Edge. Albeit interoperability sometimes requires *adapter.js* to iron out the inconsistencies between different browser implementations. In the following section, we discuss the operating system and browser distribution metrics.

## 2.1. Operating Systems Distributions

**Windows is the dominant platform.**

Most of our customers use our javascript library (callstats.js) to send data to the callstats.io product. We parse the User Agent (UA) string to decipher the OS and Browser versions. Figure 2.1 shows the distribution of the various Operating Systems (OS).

While Windows is dominant as expected, the other platforms should not be ignored during the development process. The native corresponds to apps using the web-view on native clients or web applications rolled out as desktop apps using Electron, node-webkit, or similar frameworks.

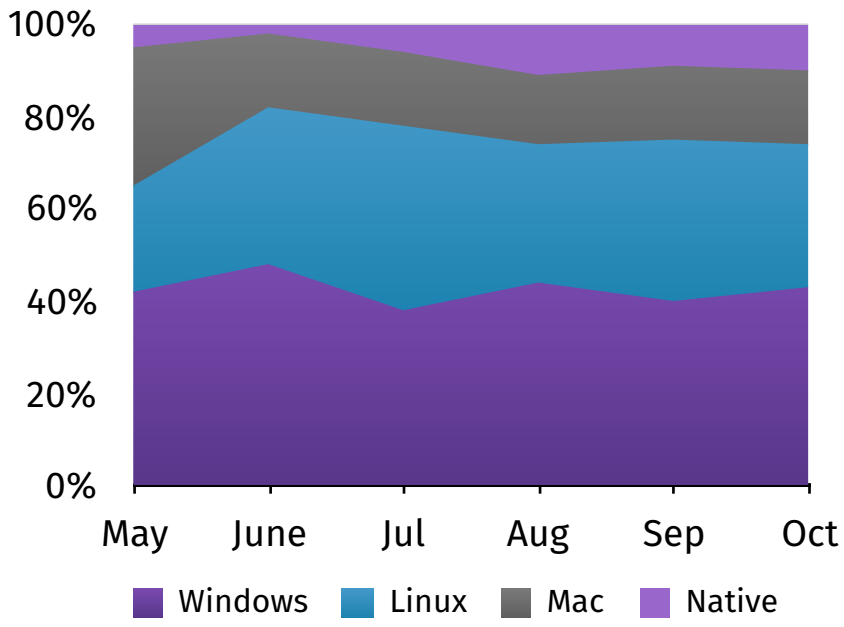
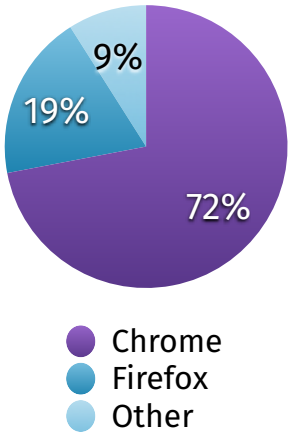



FIGURE 2.1. DISTRIBUTION OF OPERATING SYSTEMS




## 2.2. Browser Distributions

WebRTC-support in browsers has come a long way since 2012. Today, browsers update regularly, with a new stable version released every six weeks. The browsers also release beta versions at six week interval. The beta releases assist app and service developers to discover bugs and upcoming changes that might affect their app.

Some WebRTC apps cannot or don't want to keep up with the browser updates, and want to have more control over the expected user experience. These companies have started rolling out their WebRTC products as native desktop apps. The schedules for the browsers are available at:

- Google Chrome: https://www.chromium.org/developers/calendar
- Mozilla Firefox: https://wiki.mozilla.org/RapidRelease/Calendar
- Electron: https://github.com/electron/electron/releases?after=v1.4.1

## 2.3. Google Chrome

In the period from May to October 2016, the Chromium project released four stable versions: M51 in late-May and M54 in early-October. Figure 2.2 shows the monthly variation in the fraction of active sessions for a particular version of the Chrome browser.

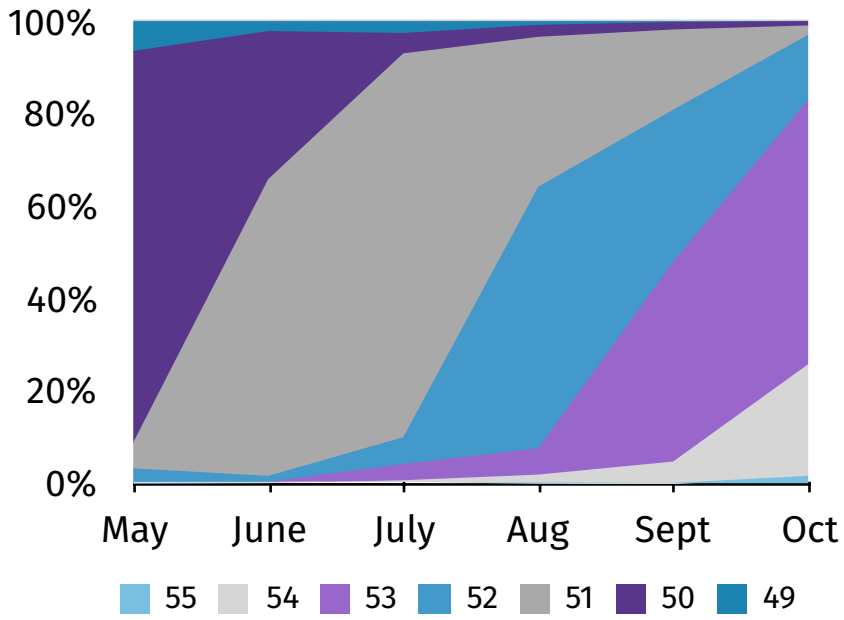



FIGURE 2.2. DISTRIBUTION OF GOOGLE CHROME VERSIONS

For example, we observe that the stable version of the Chrome M51 released at the end of May grew from ~5% to 60% usage share in the month of June. A similar pattern is observed with each new stable version. An interesting observation is that Electron v1.2.6, which was re-

leased in end of June and corresponds to Chrome 51, contributed to about 5% of the traffic in September.

## 2.4. Mozilla Firefox

Firefox (FF) released four stable browser versions, FF 46 at the end of April and FF 49 at the end of September. Figure 2.3 presents the monthly variation in the fraction of active sessions for a particular version of the Firefox browser. Firefox's new stable releases experienced similar uptake to Chrome. The release notes for Firefox's WebRTC features are available at:
https://wiki.mozilla.org/Media/WebRTC/ReleaseNotes/46.

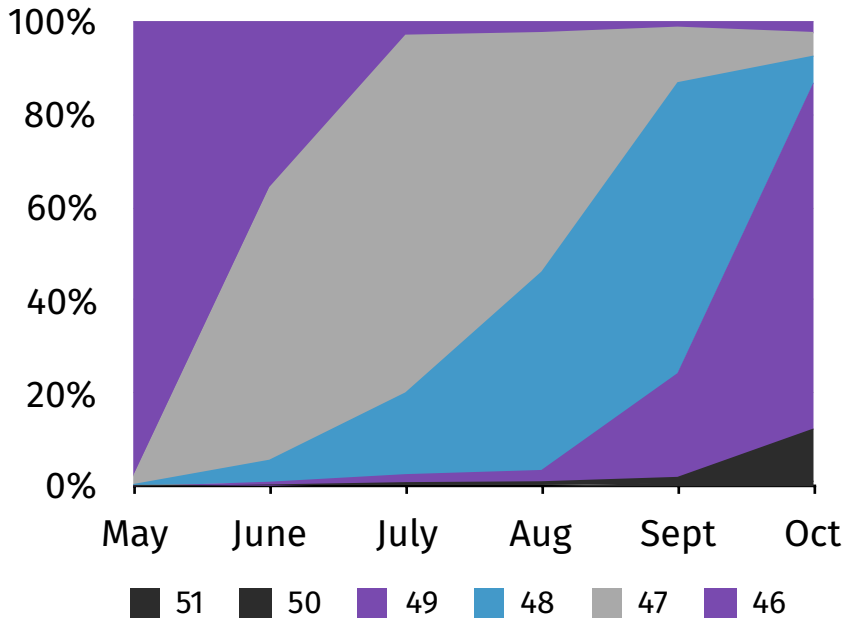



FIGURE 2.3. DISTRIBUTION OF FIREFOX VERSIONS

# 3. Participant-level metrics

Every WebRTC service, large or small, is hawkishly looking at their service- and participant-level metrics. Usually daily, weekly, and monthly variations in:

- Number of conference calls.
- Average duration of conference calls.
- Average number of participants.
- Average number of streams per participant.

Typically, product managers use these metrics to assess the usage trends and variations of their product. Consequently, these metrics also help in identifying potential bottlenecks. For example, if the number of participants per conference call is growing, the architecture needs to be evolved, be capable of handling larger conference sizes.

In the following sections, we discuss these service level metrics in more detail.

## 3.1. Participants per Conference

Six months ago, system-wide the average number of participants per conference was two (65% of the sessions). In October, about 50% of the sessions have two participants in a conference. The trend towards more participants in a conference is due to an increasing number of team communication apps on callstats.io. In the same period, the average conference size with three participants grew from 30% to 43%.

The largest conference size observed on callstats.io is also growing. In May the largest conference had 36 unique participants, and in October it was 194 unique participants. Moreover, in October we observed roughly 20 conferences with 100+ unique participants.

TABLE 3.1. LARGEST CONFERENCE BY UNIQUE PARTICIPANTS

| May | June | July | Aug | Sept | Oct |
|---|---|---|---|---|---|
| 36 | 98 | 97 | 110 | 99 | 194 |

In the coming months, we are curious about the “mega conferences” and how they evolve.

## 3.2. Active media tracks per participant

Active media tracks per participant represents the number of simultaneous tracks (audio, video, screen share, etc.) transmitted by a sender. For example, in call centres the agent responds to or makes telephone calls, typically, there will be just a single audio track per participant. Similarly, screen-sharing only apps, i.e. participants chat and one endpoint shares their screen. In this case, the session is video-only with just one participant and one active track.

**16%**

**audio-only sessions**

In the reporting period (May to October), audio-only sessions have remained fairly constant, at about 16%, and video-only sessions are marginal less than 1%. The bulk of the traffic, more than 70% of the sessions, have two media tracks (audio and video), and 10% of the sessions have three or more active media tracks, for example, mic, camera, and screen sharing.

## 3.3. Offers and Answers per Participant

A session can have a lot of variability: participants join and leave at different times, switching the media sources (e.g., changing from video to screen sharing, or switching from one camera to another), etc. In some of these cases, the sessions will need to be renegotiated. If the renegotiation fails, the changes will not be applied, i.e., the session will continue as before. For example, if an app expects to use one of the audio, video, and screen sharing streams, it may offer all three upfront when setting up the session in order to avoid renegotiating the particular stream when a participant switches between them.

Figure 3.1 shows that in more than 60% of the sessions the negotiation happens only once. In about 10-15% of the cases, the renegotiations happen more than ten times. This is typically due to conferences with a large number of participants because participants may join and leave at different times.

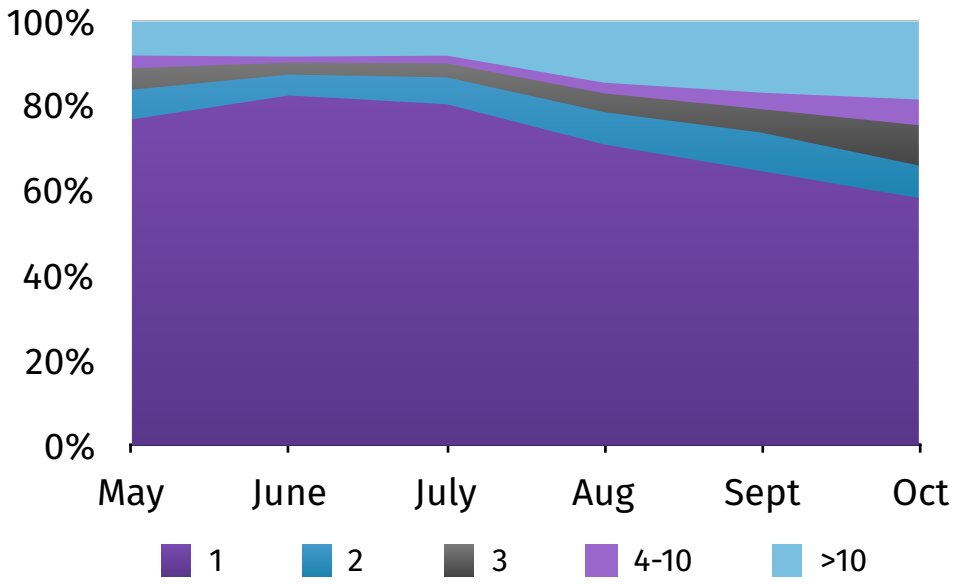



FIGURE 3.1. NEGOTIATIONS PER PARTICIPANT

A WebRTC service, which has multi-participant conferences can expect a slightly higher amount of failures from re-negotiations and should take measures to reduce the amount of failures.

## 4. Setup Failure Metrics

All WebRTC deployments fail to setup or establish connectivity between the participants for various reasons. Oftentimes, this is due to a NAT or firewall dropping media packets, which causes the conference to fail before setup is completed. Figure 4.1 shows the various errors that can occur while setting up a conference.

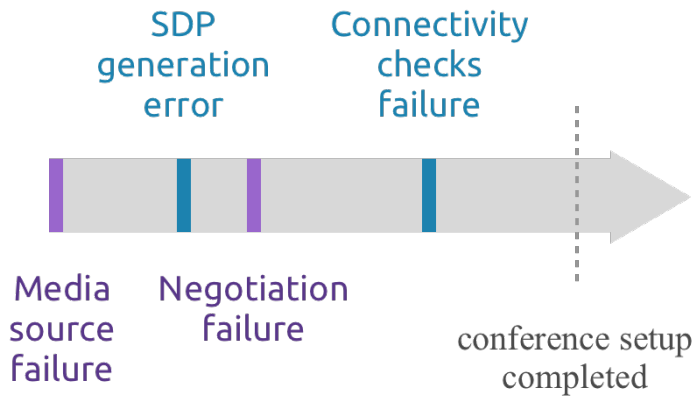



FIGURE 4.1. FAILURES BEFORE CONFERENCE SETUP IS COMPLETED

1. Access to the media source fails (mic, camera, screen capture) due to device hardware (e.g., DeviceNotFound), device configuration (e.g., UnsupportedFrameResolution), or device permission issues (e.g., PermissionDenied).

2. SDP generation fails during configuring the local media pipeline. For example, the WebRTC API (createOffer() and createAnswer()) exposes a callback when the multimedia pipeline changes .

3. Negotiation fails due to differing media and network configuration between endpoints. These errors are reported in the callbacks returned from setLocalDescription() and setRemoteDescription().

4. Connectivity checks fail when the endpoint is unable to establish connectivity between participants, as these mostly occur when the NAT/firewall traversal fails.

After the conference is set up and the session loses connectivity for any period of time, i.e., no media packets are received, the session is marked as disrupted. The session drops if the connectivity is not restored. The disruption period is the period from when the connectivity is lost to either when the connectivity is restored or totally lost. During the disruption period, the application may attempt to re-establish connectivity automatically. These re-attempts to establish connectivity might fail and are measured as Re-establishment failure. When a session drops, the user may have to manually re-establish connectivi-

ty, for example, by refreshing the webpage, this is measured as Churn. Figure 4.2 shows the failure events after the conference is setup successfully.

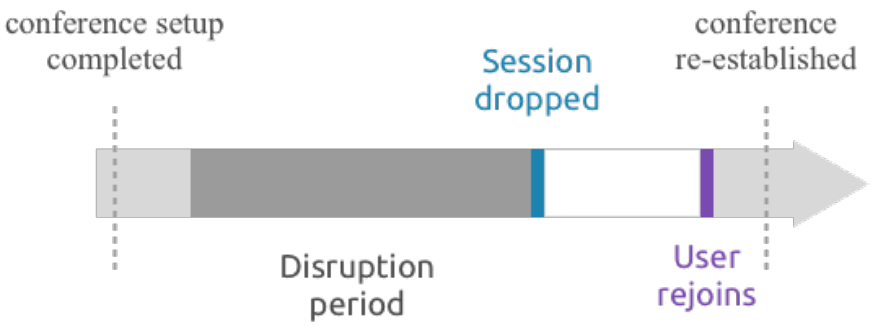



FIGURE 4.2. FAILURES AFTER CONFERENCE SETUP IS COMPLETED

In the following section, we discuss the observed failure statistics in more detail.

## 4.1. Media Configuration Errors

Media configuration errors cover the constraints-related issues. It excludes permission- and device-related errors. In the reporting period, the configuration errors reduced from 3% to 1%. The apps improved the failure fraction by adapting the constraint ranges to better fit the capabilities of their users. For example, an app increased the apps usage by 12% when it changed its minimum capture resolution from 480p to 360p. They did this by monitoring usage and constraints errors.

## 4.2. Network Connectivity Failure

**4%**

**of sessions fail due to network connectivity**

At Enterprise connect 2016, we reported observing about 12% of the sessions failing to set up, i.e., the total number of session failed to setup/total number of attempted sessions. For this report, we are revisiting the metric and we are happy to report that the observed number has dropped significantly (see Figure 4.3 for details).

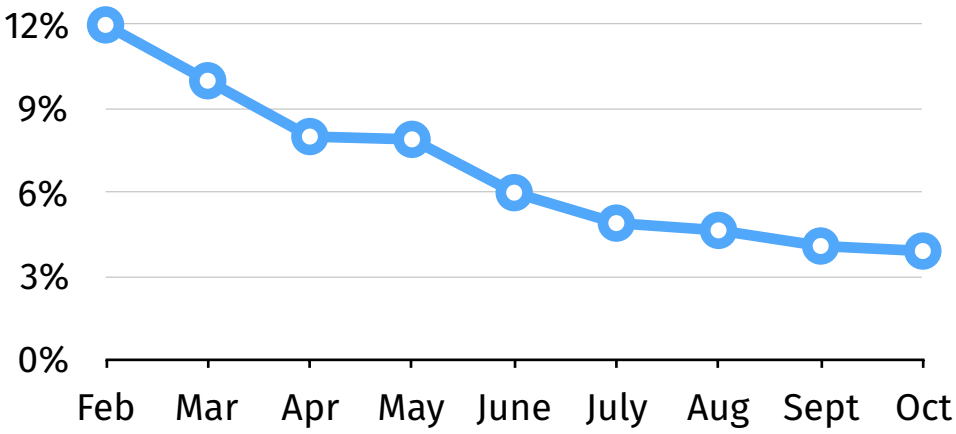



FIGURE 4.3. ICE FAILURE RATIO

In October the number of failed sessions due to NAT or firewalls had dropped to 4% from 12% in February. Meanwhile, the traffic on callstats.io grew 15 times since February. The main reason for the significant drop from 12% to 4% is deploying more relay servers. This is also observed in the next chapter, transport metrics, which shows the variation in usage of the various types of relay servers.

## 4.3. Sessions with Disruptions

**3%**

**of the sessions have disruptions**

Disruptions are periods when the connectivity between endpoints is poor, i.e., the end-to-end capacity is limited, packet losses are high, latencies are high, or the jitter buffer is dropping packets. The end-user experiences these disruptions as a choppy audio or black or pixelated frames of video. During the reporting period from May to October, the rate for session disruptions oscillated between 2-4%.

## 4.4. Re-establishment Failed

**5%**

**of sessions fail to re-establish**

Many apps try to re-establish connectivity when the endpoint loses connectivity to other participants. This may happen because the endpoint lost a device (for example, a camera was disconnected) and needs to gather new ICE candidates to re-establish connectivity. If the re-establishment fails to restore connectivity, the session is marked as re-establishment failed. During the reporting period, about 5% of the conferences had a session, which failed to re-establish connectivity for at least one participant.

## 4.5. Sessions with Churn

**18%**

**of sessions experience churn**

In a two-party call, churn usually results in a new session, i.e., the calls are redialled repeatedly. While in a multi-party call, it shows up as a user rejoining repeatedly. Churn usually occurs when one or more participant has a frail network: insufficient capacity, packet loss or RTT is high, causing them to drop out of the session and redial. In Figure 4.4 we observe that the most of the sessions exhibiting churn drop just once and renegotiating the sessions seems to solve the issue.

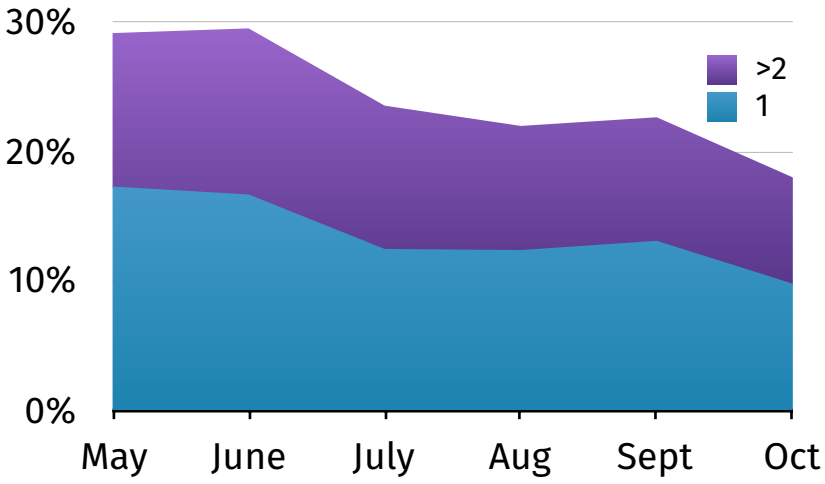



FIGURE 4.4. SHARE OF SESSIONS EXPERIENCING CHURN

Based on the observations in the first half of the year, we suggested to our customers the following improvement: renegotiate the session when the callstats.js library reports disruptions using the statsCallback method.

# 5. Transport Metrics

The Internet Connectivity Establishment (ICE) mechanism makes it easier for media packets carried in UDP to traverse the Internet. One key component in the mechanism is the use of a TURN relay server.

In this chapter, we look at usage metrics of various transport attributes: IP versions, TCP vs UDP, and various types of TURN relays.

## 5.1. IPv6 Usage

IPv4 exhaustion has been discussed at length over the past few years, and there has been a push towards deploying more IPv6. Most recently, Comcast reported transitioning to IPv6 completely on the subscriber end. While IPv6 is available on many routes, they are not preferred in many cases. Our records show that about 2% of the traffic is carried over IPv6.

TABLE 5.1. IPV6 USAGE

| May | June | July | Aug | Sept | Oct |
|---|---|---|---|---|---|
| 1,3% | 1,7% | 2,8% | 2,6% | 1,6% | 1,2% |

## 5.2. TCP Usage with a Media Server

Services that support large number of simultaneous participants, use a media server to route the media from each participant in a conference, instead of sender sending the media individually to each each participant. The media server essentially helps reduce the upstream bandwidth. These media servers are typically, called Selective Forwarding Units (SFUs), and the most common are: Jitsi Video Bridge, Kurento, Janus, etc. When a participants is behind a firewall or NAT that blocks UDP, the media servers work as media relays and carry the media over TCP. We observe that about 5% of the traffic uses TCP on services that incorporate a media bridge.

TABLE 5.2. TCP USAGE

| May | June | July | Aug | Sept | Oct |
|---|---|---|---|---|---|
| 7,04% | 9,13% | 2,67% | 2,17% | 4,34% | 3,85% |

**23%**

**of the sessions use a TURN relay server**

## 5.3. TURN Relay Usage

The relay server usage is an important metric for applications and services that want to use the least amount of infrastructure servers. Without any TURN relay servers, these sessions would fail to setup, therefore, deploying TURN servers is important to establish connectivity. We observe that on average 23% of the sessions use a TURN relay server.

TABLE 5.3. TURN USAGE

| May | June | July | Aug | Sept | Oct |
|---|---|---|---|---|---|
| 20,4% | 20,5% | 27,0% | 25,2% | 22,7% | 21,9% |

**80%**

**of the Relayed sessions use TURN/UDP**

## 5.4. Types of TURN

The relay server can relay over several protocols (UDP, TCP/TLS). About 80% of the relayed sessions use UDP, while about 20% need TCP or TLS, typically in enterprise environments, where all UDP traffic might be dropped and tunnelling over port 80 or 443 would be the only option to stream media in real-time.

TABLE 5.4. TRANSPORT PROTOCOL SHARE FOR TURN SERVERS

| | May | June | July | Aug | Sept | Oct |
|---|---|---|---|---|---|---|
| UDP | 81,0% | 76,8% | 80,3% | 89,8% | 74,5% | 75,7% |
| TCP | 19,0% | 23,2% | 18,9% | 18,5% | 19,2% | 19,5% |
| TLS | | | 0,8% | 6,1% | 6,3% | 4,8% |

# 6. Setup Time

In the last 10 years, Internet companies has spent enormous resources to reduce the page load times, and stream data as soon as it becomes available. Similarly, telecommunication companies have measured call set up times. In real-time communication, time to first media (TTFM) is the corollary to page load times. TTFM represents the time it takes for the participant to see the first media frame rendered after initiating or acknowledging to receive a multimedia session. We want the TTFM to be as low as possible.

Figure 5.1 shows the various steps the endpoint takes to set up the session with the remote participant. It should be noted, even though the figure shows the steps in linear order, some of these steps can take place in parallel. For example, the candidates can be exchanged and checks can occur while the codecs and session parameters are being negotiated.

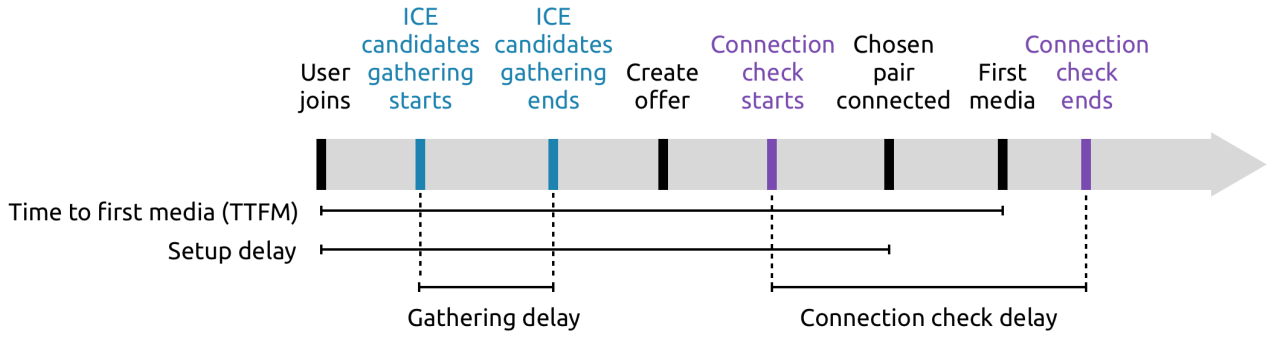



FIGURE 5.1. CONFERENCE SETUP PROCESS

## 6.1. ICE Candidate Gathering Delay

The ICE gathering delay represents the time taken for the endpoint to collect all the ICE candidates associated with the endpoint. These include the host, peer, and server reflexive candidates.

Figure 5.2 shows that about 80% of the sessions are able to gather these in 100ms or less time, while about 5% of the sessions take over one second. In several cases, the timing depends on how quickly the endpoint is able to get a port allocation from the TURN server. By placing the servers closer to the endpoints, apps can reduce the gathering delay.

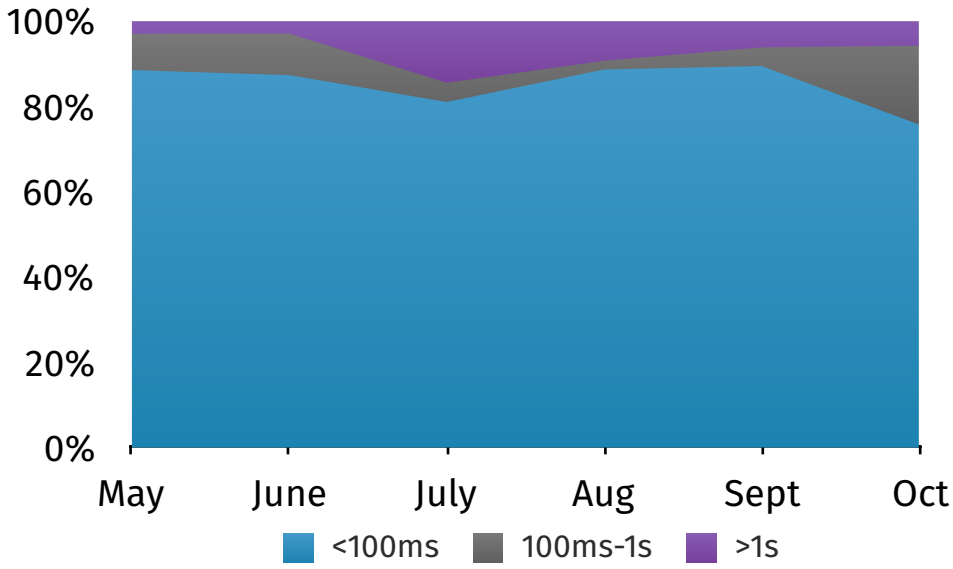



FIGURE 5.2. ICE CANDIDATE GATHERING DELAY DISTRIBUTION

## 6.2. ICE Connectivity Check Delay

**20%**

**of the sessions set up in <100ms**

The ICE connectivity check delay is the time taken from the start of the connectivity check to the time when the chosen candidate is selected by the endpoint. This metric encompasses the network delay between the endpoints, i.e. if a pair of endpoints are geographically far apart, the checks will take longer.

Another factor that affects the checks is how often are they sent, i.e. the interval between two consecutive checks (pacing interval). If the pacing interval is long, the checks will take longer.

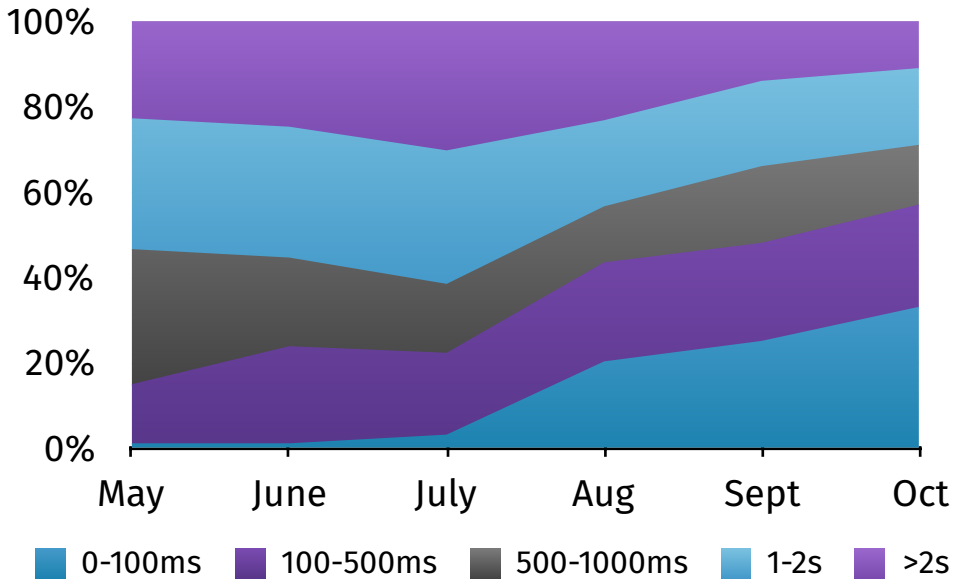



FIGURE 5.3. ICE CONNECTIVITY CHECKS DELAY

In July, Chrome rolled out M52, which reduced the pacing interval, and this had a big impact on the connectivity checks delay. The number of sessions that were setup in under 100ms grew from under 2% in June to 20% in October (see Figure 5.3 for details). It should be noted that the traffic on callstats.io in the same period increased by 8 times (July-October).

# 7. Network and Media Metrics

Multimedia applications observe the congestion cues, and react to the changes in the cues, by modifying the encoding and sending rates to match the available end-to-end path capacity. The goal is to minimise losses at the receiver while maintaining a stable throughput. Losses are caused by congestion or bit-errors, and are detrimental to the perceived video quality. Although, real-time communication is tolerant to a small amount of losses, a burst of packet losses should be avoided.

In this section, we discuss the network and media metrics that are used by engineers building the next generation of codecs or networking stacks. These metrics rely on percentiles, which can be confusing, if the concept is unknown. You can read more about percentiles at: www.dummies.com/education/math/statistics/what-percentile-tells-you-about-a-statistical-value/.

## 7.1. 95-percentile Fractional Loss

**77%**

**sessions have no packet loss**

Callstats.io measures fractional losses every 500ms, and at the end of a session, we calculate the 95 percentile loss. For example, a 5-minute (300s) call, the 95-percentile represents the 15 worst seconds of the session, regarding fractional loss. If the 95-percentile metric is low (less than 5% packet loss), then 95 percent of the session had no or fewer than 5% packet loss. Consequently, if the 95-percentile is high (>33%), then 5% of the session would have terrible quality of experience.

Figure 7.1 shows the 95-percentile fractional loss in the reporting period (Aug-Oct 2016). We observed that only 23% of the sessions had any losses. And, that less than 3% of the sessions had more than 20% losses for about 5% of the time. That is pretty low!

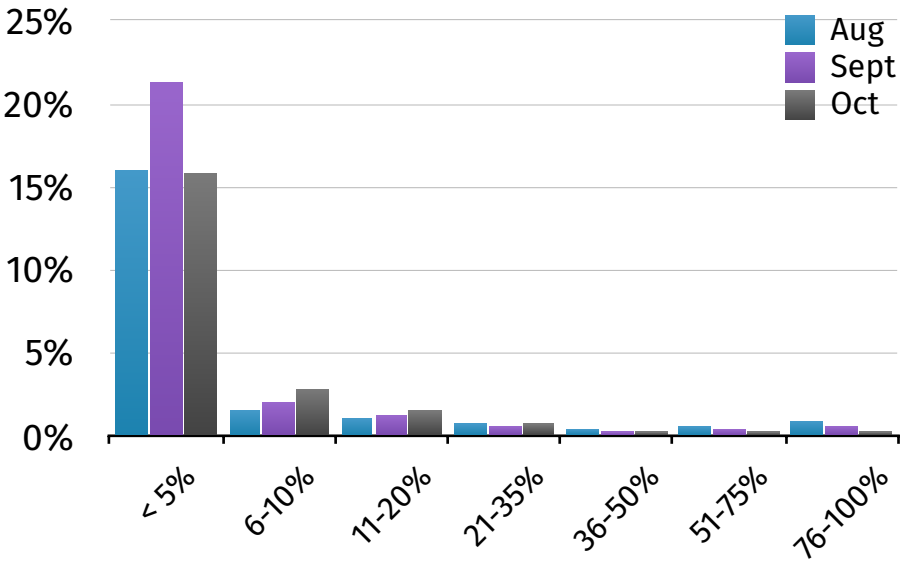



FIGURE 7.1. 95-PERCENTILE FRACTIONAL LOSS

## 7.2. 95-percentile RTT

Real-time media sessions expect that the end-to-end latency should be low. For good quality audio sessions, ITU-T suggested that the mouth-to-ear delay be under 150ms. That is the time to capture, packetise, network delay, and finally render the audio. The complete process should be completed within 150ms of capturing the packet. Roughly this corresponds to about 100-120ms network latency, assuming 20ms packetisation interval and 10ms for the remaining operations.

In Real-time Transport Protocol (RTP), we can measure the Round Trip Time (RTT). For simplicity, systems assume the network latency as half of the RTT. Thus, the expected RTT should be under 240ms (2 x network delay = 2 x 120ms). We measure the 95-percentile RTT, i.e., 95 percent of the session had an RTT lower than the indicated value. Again, for a five minute session, this period would be less than 15 seconds.

**80%**

**of sessions have less than 240ms RTT**

Figure 7.2 shows the cumulative distribution of the 95-percentile RTT across all sessions. About 50% of the sessions (median) have an RTT lower than 90ms. Finally, 80 percent of the sessions have less than 240ms RTT.

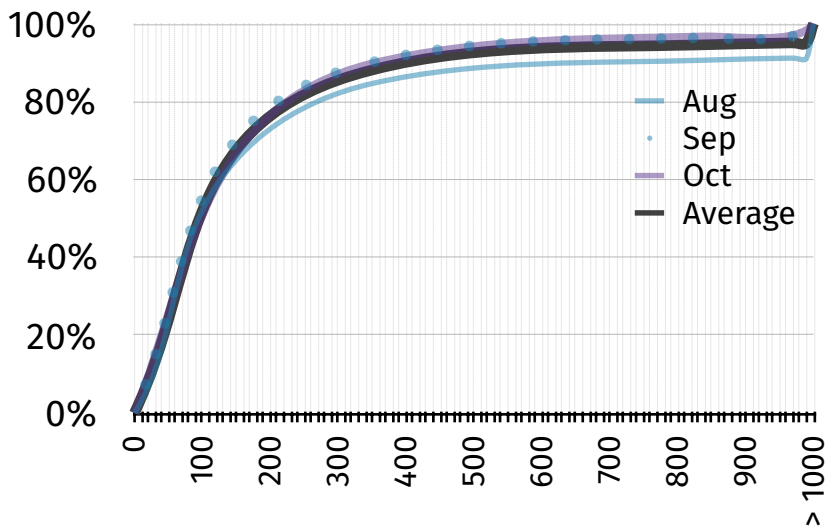



FIGURE 7.2. 95-PERCENTILE RTT IN MS

## 7.3. Audio codecs

WebRTC supports two audio codecs: G.711 and Opus. In the reporting period about 91% of the sessions use Opus, and 9% use G.711. The G.711 codec is fixed rate codec, with an average bitrate of 64 kbps. We observe a correlated 5-11% of the sessions have an average bitrate of 64 Kbps. Opus has a large dynamic range and adapts the bitrate to the available end-to-end capacity. We observe about 70% of the sessions have an average audio bitrate between 30-60 Kbps, which results in quite good audio quality.

**80%**

**of the sessions have over 30 kbps audio bitrate**

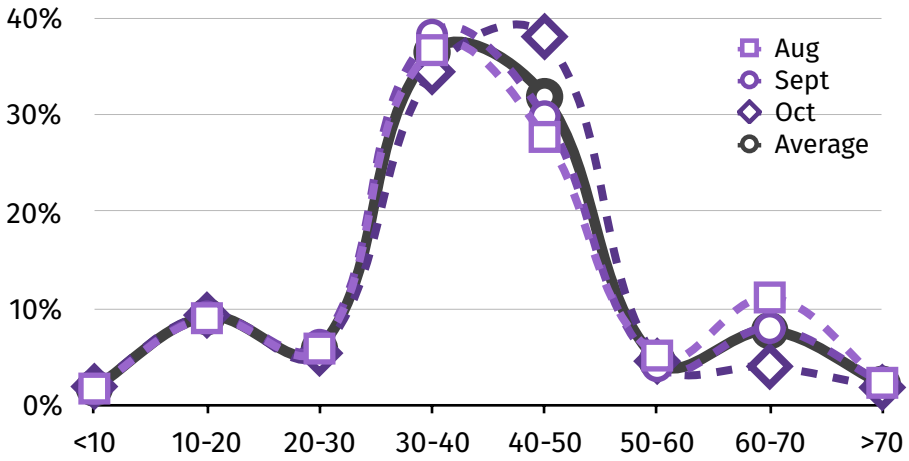



FIGURE 7.3. AVERAGE AUDIO THROUGHPUT IN KBPS

## 7.4. Video codecs

WebRTC mandated two video codecs: H.264 and VP8. Currently, H.264 is supported on Firefox, Chrome, and Edge. Although, Edge supports a different variant of H.264 than the other browsers and Chrome's H.264 support is behind a development flag. Due to these reasons, H.264 usage is abysmal, at about 5-10% in the reporting period. Although, VP9 is not mandated, some services are beginning to prioritise it over VP8. In October, we observed that VP9 usage was grown to about 8%.

**50%**

**of the sessions have over 1 Mbps video bitrate**

Since most of the sessions are over VP8, we measured the average throughput across all sessions, and present it in Figure 7.4. We observe that about about 15% of the session are below 250 Kbps, which may result in mediocre quality depending on the use-case (for example, small screens prefer lower resolutions). About 50% of the sessions use more than 1Mbps, which is generally considered high quality.

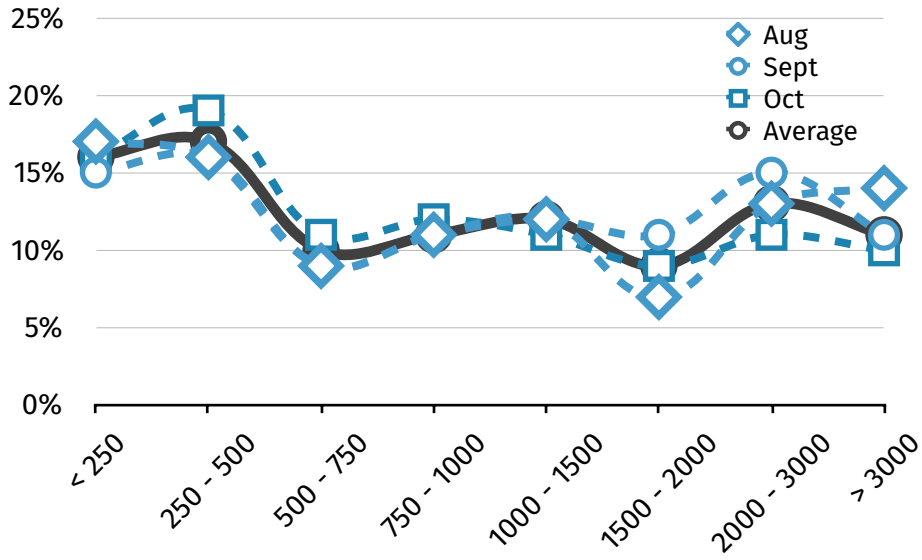



FIGURE 7.4. AVERAGE VIDEO THROUGHPUT IN KBPS

# Editors

**Varun Singh** is the CEO, Founder of **callstats.io**, contributor to several standards at the IETF and W3C. He has 14 years of industry and research experience and a Ph.D and M.Sc in Multimedia and Networking from Aalto University.

**Marcin Nagy** is the Lead Engineer and Co-Founder of **callstats.io,** and a Ph.D. Candidate at the School of Electrical Engineering at Aalto University. He has 8 years of industry and research experience in multimedia, network security and privacy and an M.Sc. from Aalto University.

**Lasse Lumiaho** is the marketing lead at **callstats.io**, with 7 years of industry experience in product management, growth, and marketing. He has a M.Sc. in User-Centred Product Development from Aalto University.

# Copyright