callstats.io

WebRTC METRICS REPORT 2017/02

Usage metrics from real world WebRTC deployments.

A collection of WebRTC statistics from January 2017 to June 2017 brought to you by callstats.io, the realtime communications monitoring platform.

Hi from Varun Singh, CEO

Thank you for downloading the callstats.io's industry report on WebRTC metrics. The second quarter of 2017 saw some exciting developments in WebRTC, including the long awaited Safari announcement to join the ecosystem. With WebRTC maturity, we're also seeing the application ecosystem grow bigger and more interesting with exciting use cases in e.g. IoT, VR/AR and automation.

This report is the third one in our series that will be published every quarter by callstats.io, with the next one planned for late 2017. Similar to the earlier reports, we are going through the last six months of sessions, from January 2017 to June 2017. In this period, our install-base has continued to grow 20% month on month while the world-wide monitored traffic grew by 5-times.

In 2017, we are making advancements on multiple fronts: improvements for the dashboard, more client libraries, and more nuanced metrics for diagnostics. Furthermore, we are investing more resources on machine learning, using AI to find correlations in our data across services, end-users, and networks.

This 15-page report includes 13 charts that provide easy-to-digest insights into the growing number of WebRTC deployments.

This report should help product managers and engineers design better real-time communication systems: chiefly, being mindful of potential bottlenecks as they plan for the growth of their application service.

This report answers to questions like:

- What operating system and browsers dominate WebRTC?
- How high is the set up time or time to first media?
- What is the impact of peer-to-peer or bridge topologies on the connectivity, round-trip time, fractional loss?

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-2017-02/.

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 anonymized WebRTC metrics collected by callstats.io.

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

TL;DR

- Chrome still dominates the WebRTC browser market. Likewise, the operating system distributions remain the same, with Windows leading.
- 40% of the connectivity checks finish in 100ms.
- 80% of the media sessions render media in less than 1s after starting the call.
- We have an even split between bridge and peer-to-peer traffic. 50% for each network topology.
- Average Round-trip time is lower for peer-to-peer sessions than the sessions using the bridge.
- Median average RTT for peer-to-peer sessions is 70ms and it is 140ms for bridge sessions.
- 95-percentile fractional loss is lowest for the bridge followed by peer-to-peer, and highest for TURN sessions.
- About 90% of the sessions on the bridge, 84% on the peer-to-peer, and 78% on the TURN have zero 95%-ile losses.

1. The WebRTC Media Pipeline

The metrics in this report 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.

When receiving these packets, the endpoint concatenates them 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.

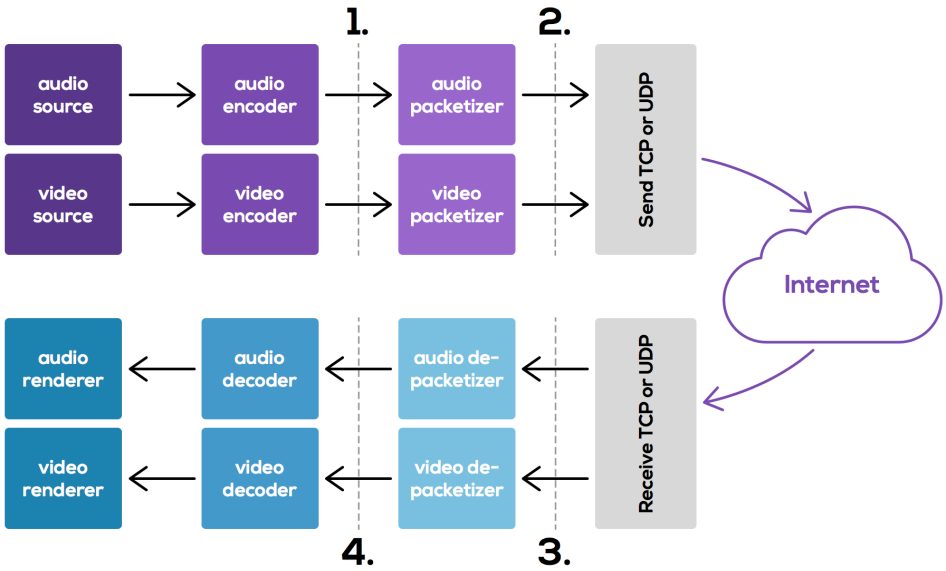


FIGURE 1.1. THE WEBRTC PIPELINE

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 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 summarized. Furthermore, the summarized 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. Safari's WebRTC support is available in Safari 11.0 and later. *Adapter.js* is still most important tool needed to iron out most of the inconsistencies between different browser implementations.

In the following section, we discuss the operating system and browser distribution metrics, and deep dive into Chrome and Firefox usage insights.

2.1. Operating system 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 operating systems and browser versions.

Figure 2.1 shows the distribution of the various operating systems and we observe that the operating system distribution remains fairly unchanged during Q2 of 2017.

While Windows is dominant as expected, other platforms should not be ignored during the development process. Linux, macOS, and Android each have a substantial market share, altogether reaching 50% of the operating system share.

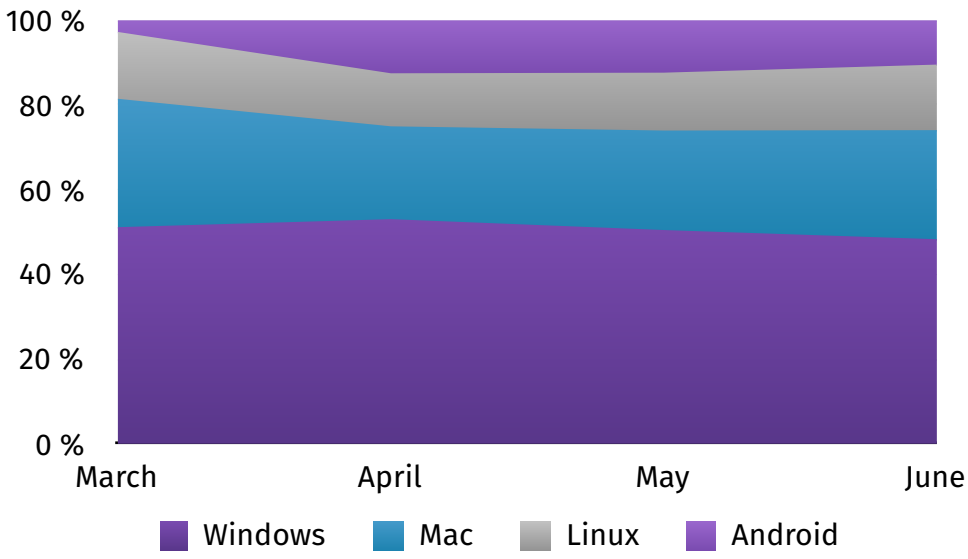
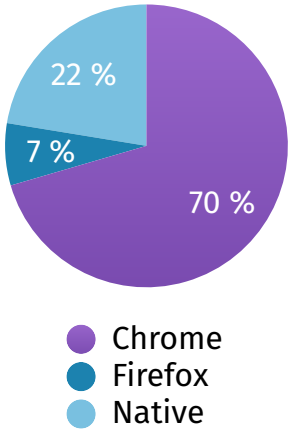


FIGURE 2.1. DISTRIBUTION OF OPERATING SYSTEMS



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 do not want to keep up with the browser updates and want to have more control over the expected user experience. These products and services roll out their WebRTC products as native desktop apps. Three most common tools for building native desktop apps are:

- React Native: https://facebook.github.io/react-native/versions.html
- NW.js: https://github.com/nwjs/nw.js/releases
- Electron: https://github.com/electron/electron/releases

In June, Safari added support for WebRTC, this is yet to make it to global availability and apps are still switching to it in production, especially since Safari does not support VP8 and requires universal support for the H.264 video codec.

2.3. Google Chrome (Chromium)

From March 2017 to June 2017, the Chromium project released three stable versions: M57 in mid-March, M58 in late-April, and M59 in early-June. Figure 2.2 shows the monthly variation of active sessions with each Chrome browser version.

In July 2017, the Chrome M58 version (released in late April 2017) grew to more than 80% usage share. This pattern is observed each time a

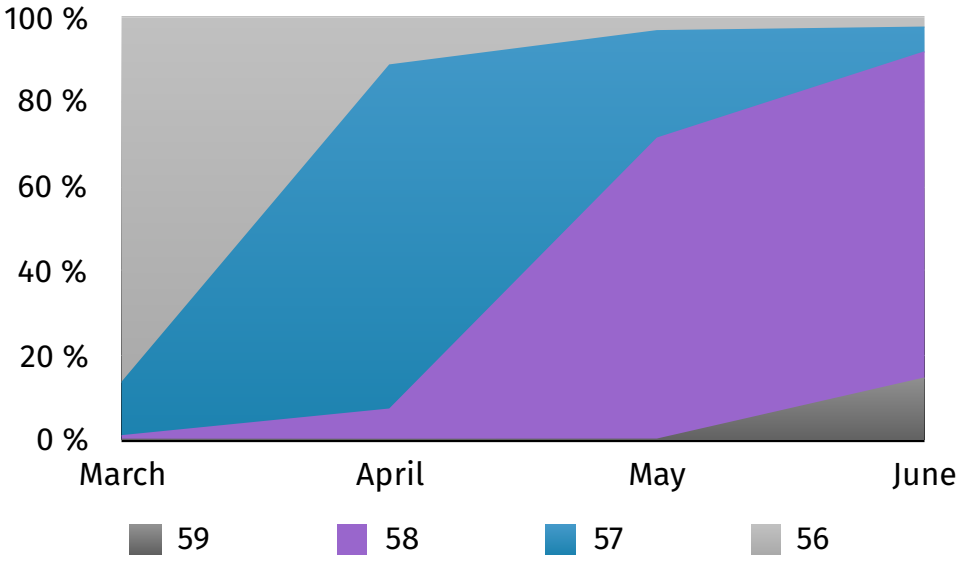


FIGURE 2.2. DISTRIBUTION OF GOOGLE CHROME VERSIONS

new stable version is released and consistent with the observations of the Google team.

2.4. Mozilla Firefox

In Q2/2017, Firefox (FF) released two stable browser versions, FF 52 in early March 2017, and FF 53 in mid-April. FF 52 replaced FF 45 as the new Extended Support Release (ESR), hence the “Other” category in our previous WebRTC report disappeared (which used to be FF45).

FF 54 was released in mid-June, when we compiled results for this report, in early July, FF 54 hardly shows any traffic for it. Figure 2.3 presents the monthly variation in the fraction distribution of the active sessions for a particular Firefox browser version. Firefox’s new stable releases are similar in their uptake to Chrome.

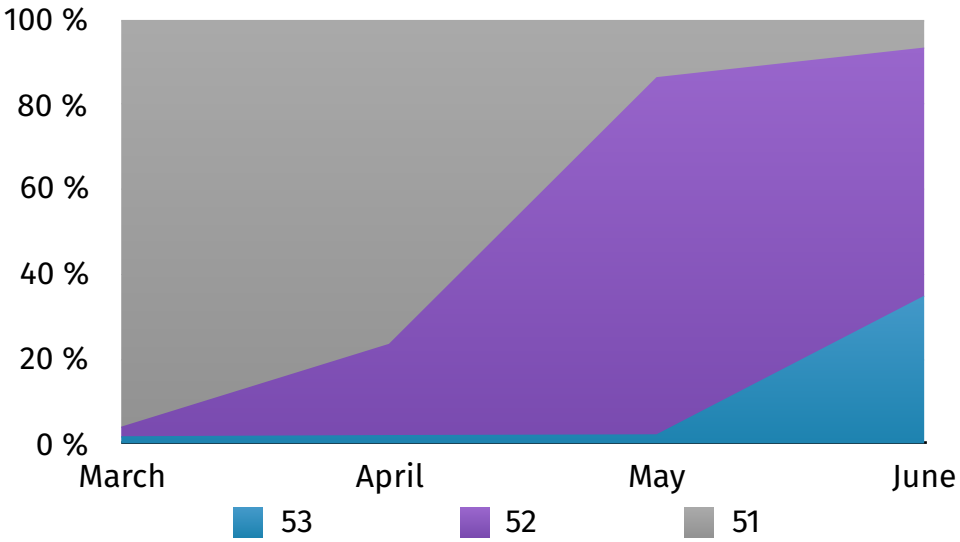


FIGURE 2.3. DISTRIBUTION OF FIREFOX VERSIONS

3. 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. Users and services expect the TTFM to be as low as possible.

Figure 3.1 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.

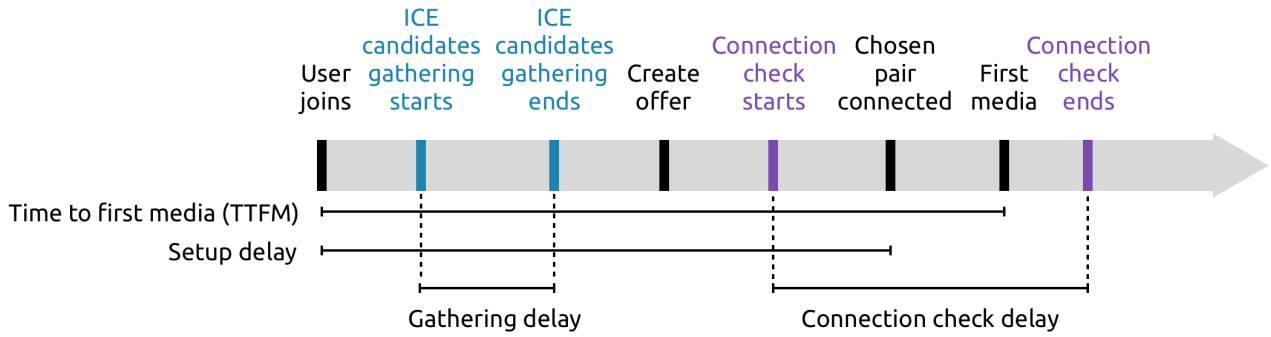


FIGURE 3.1. DECONSTRUCTING SET UP TIME

95%

of the cases gathering delay is below 100ms

3.1. Gathering delay

Gathering of ICE candidates is the first step in the connection establishment process. We observe that more than 95% of the cases the gathering process is completed within 100ms. The gathering delay is between 100ms and 1s for 2% of the cases, and over 1s for 3% of the cases.

3.2. Connectivity delay

Connectivity checks are performed for each candidate pair, where each local candidate is paired with every candidate received from a remote party.

Figure 3.3 shows the delay distribution for the connectivity process. Less than 40% of this process is finished within 100ms and 33% is finished between 100ms and 500ms.

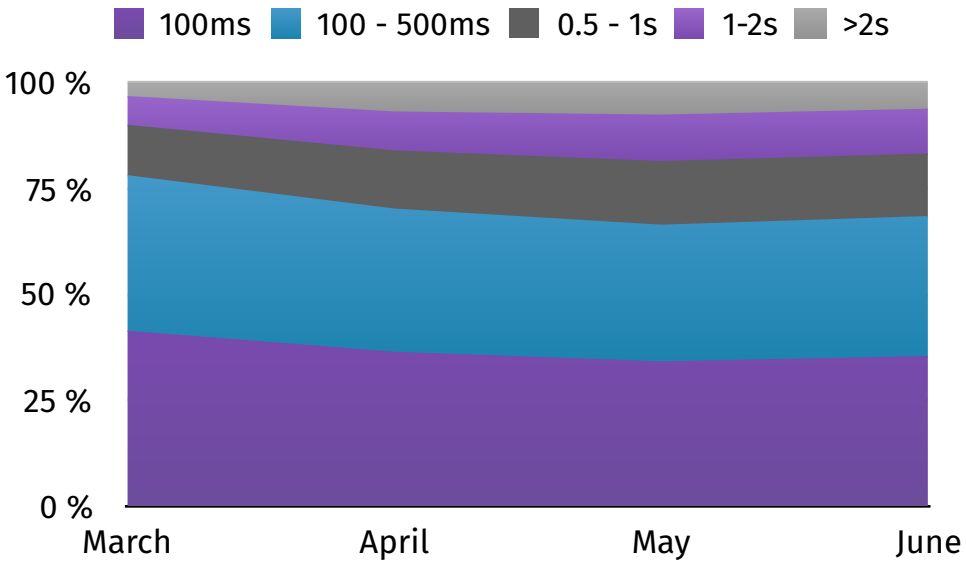


FIGURE 3.3. DISTRIBUTION OF CONNECTIVITY DELAY

3.3. Time to First Media

From an user experience perspective, TTFM is a very important connection setup-related metric. It signifies the amount of time it takes to render the first audible or visual frame from the remote party.

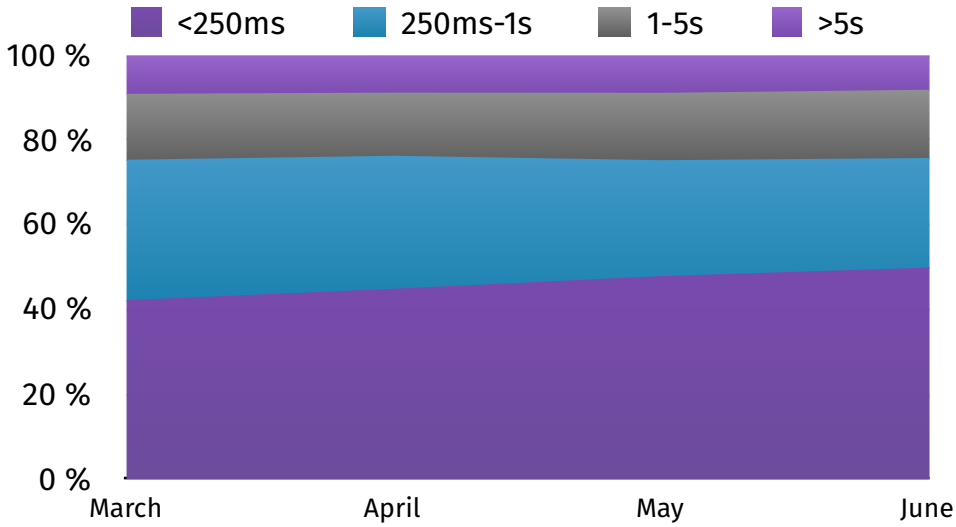


FIGURE 3.4. DISTRIBUTION OF TTFM

Figure 3.4 shows the distribution of TTFM, we observe that 50% of the connections have a very low TTFM: below 250ms. In 80% of the cases, the TTFM is below 1s, which is also a pretty good result. However, in 10% of the cases it takes more than 5s to present media to the user.

4. Connection Type

With WebRTC, a connection topology between participants can be constructed in multiple ways:

- Directly between peers, also known as Peer-to-Peer (P2P), this yields a mesh topology when there are several participants.
- Using a central media point (Bridge), this yields a start topology when there are several participants.

A conference using P2P is established between the participants directly. This has the advantage that it does not need any additional, possibly expensive infrastructure, like servers on the Internet. On the other hand, peer-to-peer is inefficient, when the conference consists of several participants, a full-mesh topology is formed, and the the same data is transmitted between several participants, vastly increasing the sending load on the network traffic and endpoint CPU (because the data might be encoded at different rates for each endpoint depending on the end-to-end path capacity).

If there are connectivity issues due to NATs or firewalls in the P2P topology, they are commonly solved by using TURN servers. When a TURN server is used, the participant first connects to a TURN server and then routes all their data through the TURN server to the other participant. To summarize, the P2P topology requires additional infrastructure as well to reliably connect participants in all scenarios.

The Figure 4.1 shows the distribution of connection per different connection types. More than half of the connections go through middleboxes. About 20% of the P2P traffic uses a TURN server, the rest is pure P2P.

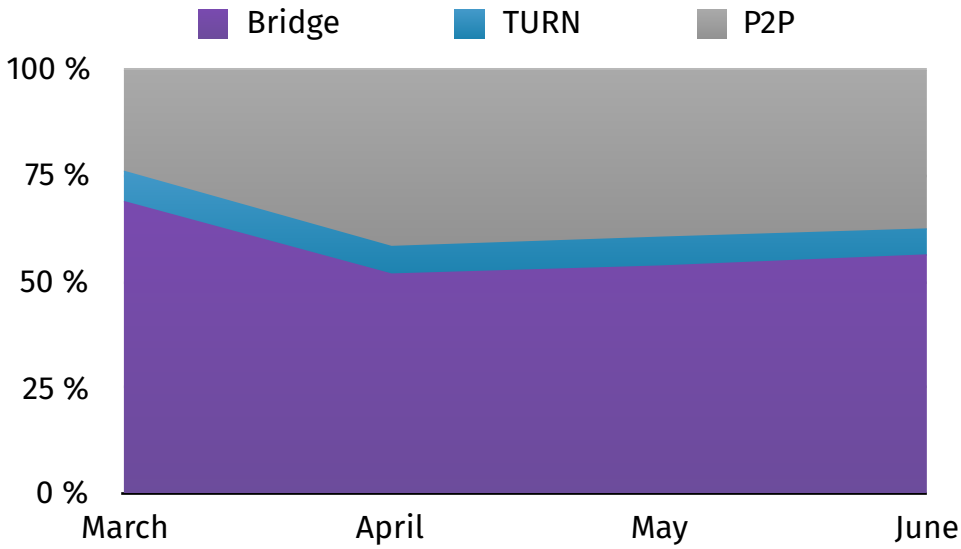


FIGURE 4.1. DISTRIBUTION OF SESSIONS BY CONNECTION TYPE

In the following sub-sections, we present our findings related to the usage of these different connection types.

4.1. Internet Protocol v6 and v4

Top-level IPv4 addresses were exhausted in 2011; between 2011 and 2015 the Regional Internet Registries (RIRs) also announced IPv4 address depletion, and since 2010 efforts are underway to increase IPv6 usage. For this reason, callstats.io keeps track of IPv4 and IPv6 (ICE candidates) usage and we report it in our quarterly reports. Compared to our previous reports, we are breaking down our analysis based on direct P2P or via an infrastructure (TURN and bridge) entity.

The IPv6 availability varies across connection types, on average in the reporting period, we observe that IPv6 is available for P2P in 7% of the connections, 15% for the Bridge, and less than 1% for the TURN servers. Overall, the IPv6 usage is low, around 5% of the sessions finally route traffic over IPv6.

The Figure 4.2 shows the distribution of IPv4 ICE candidates, the P2P distribution contains endpoints that gathered: host, srflx, prflx, and relay addresses. Consequently, in the P2P scenario, 25% have a single ICE candidate and 43% have up to 3 candidates. Conversely, the bridge connections often begin with at least 2 candidates: host and srflx/prflx in 60% of the scenarios.

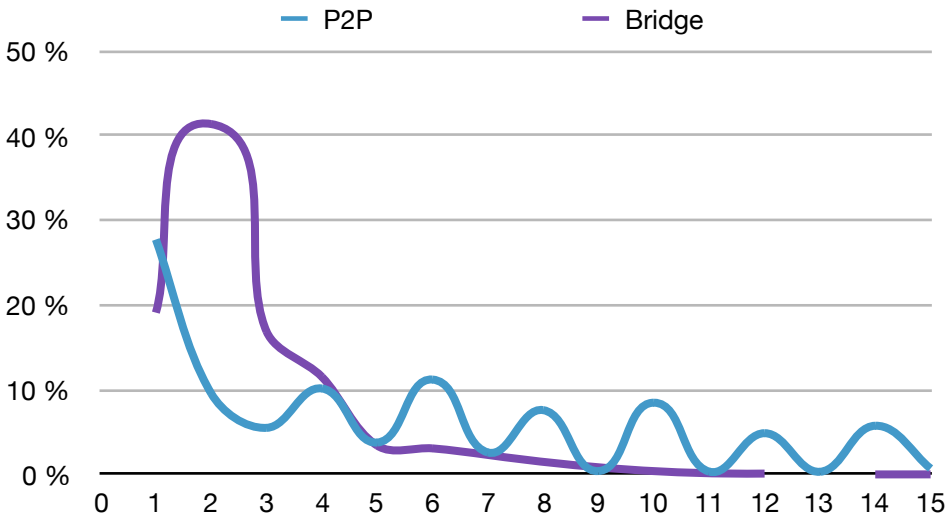


FIGURE 4.2. DISTRIBUTION OF AVAILABLE IPV4 CANDIDATES

4.2. TTFM based on connection type

In Section 3.3, we show that 70% of the connections have a TTFM lower than 1 second. We were curious if the values differed for different connection types. Figure 4.3 shows that P2P connections tend to have very low TTFM, more than 80% of the sessions have TTFM lower than 250ms! Comparatively, on the bridge, 46% of the sessions have TTFM lower

than 250ms and about 75% of the bridge sessions have a TTFM lower than 1 second.

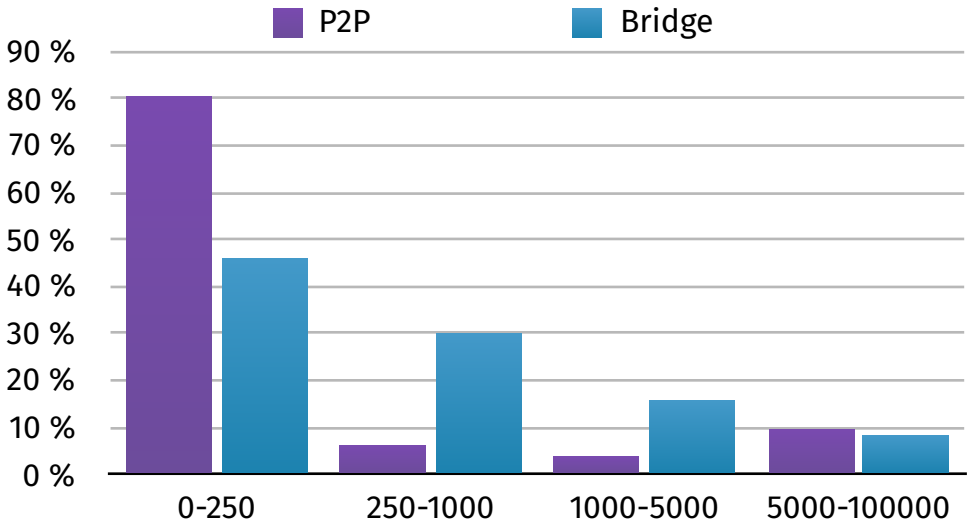


FIGURE 4.3. DISTRIBUTION OF TTFM BY CONNECTION TYPES

4.3. RTT based on connection type

Our observations in Sections 3.3 and 4.2 show that the TTFM for the bridge topology is higher than in the P2P topology. We were curious about the placement of the infrastructure (e.g., the bridges, TURN servers), i.e., they are placed further away from the endpoints, yielding a higher TTFM. Figure 4.4 shows the RTT distribution based on different connection types.

We observe that the RTT is quite different between connection types, longer for sessions for the bridge, which explains the variation in TTFM. P2P has generally the lower RTT with the median below 70ms and 83% are below 240ms! In contrast, the bridge connections have

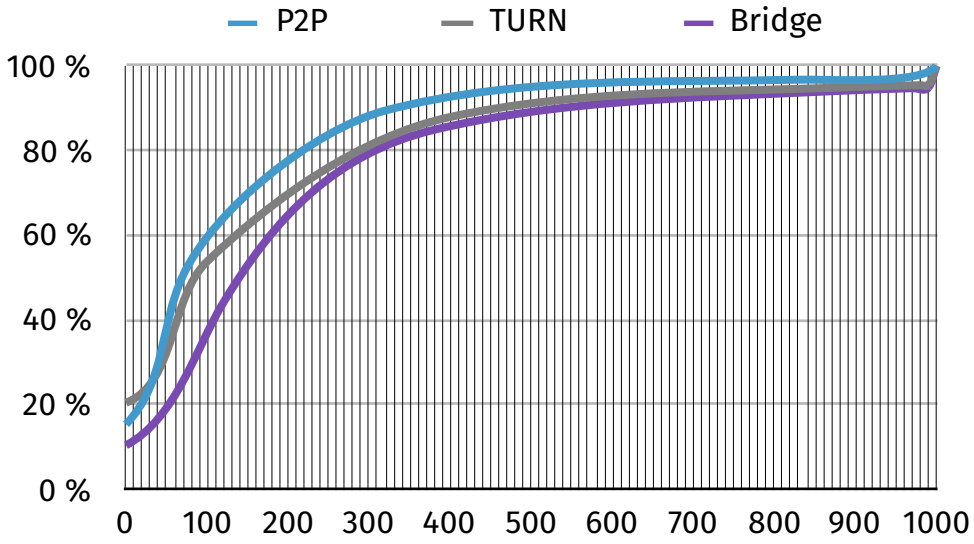


FIGURE 4.4. DISTRIBUTION OF RTT BY CONNECTION TYPES

slightly higher RTT, the median is 140ms and 73% of connections are below 240ms.

On the other hand, the RTT distribution for TURN sessions is pretty interesting, because 50% of the sessions are below 80ms, which is similar to the P2P connection type; however, we observe a big chunk with higher RTT with only 74% of connections being below 240ms, which is similar to the bridge sessions. One possible explanation is that the TURN servers are present in specific locations and when the users are in the same geography, it yields low RTT for those participants; however, it yields high RTT for participants when TURN is located further away.

Compared to the last report, we observe an increase in RTT. This was analyzed with respect to app usage, distance between participants, and browser versions. While app usage and distance had no significant effect. We found that Chrome 56 had a lower RTT compared to later Chrome versions (57, 58, 59). The timeframe of the deployment of these versions coincides with the time the RTT changes as well, which strengthens the claim. Still, from the Chrome change logs there is no indication on why this would be the case.

4.4. Fractional loss based on connection type

We measure fractional loss at 500ms intervals and summarise the median, average, and the 95-percentile value at the end of each session. If the metric is low (less than 5% packet loss), then the session experienced no major media quality degradation and is likely to be seen as good quality from a user perspective. Consequently, if the metric is high (>33%), then the user sees and hears errors, for example, the video stutters, blacks out, or has artefacts.

Figure 4.5 shows the 95%-ile fractional loss, and in general we observe that the averages are consistent with the month-by-month analysis in

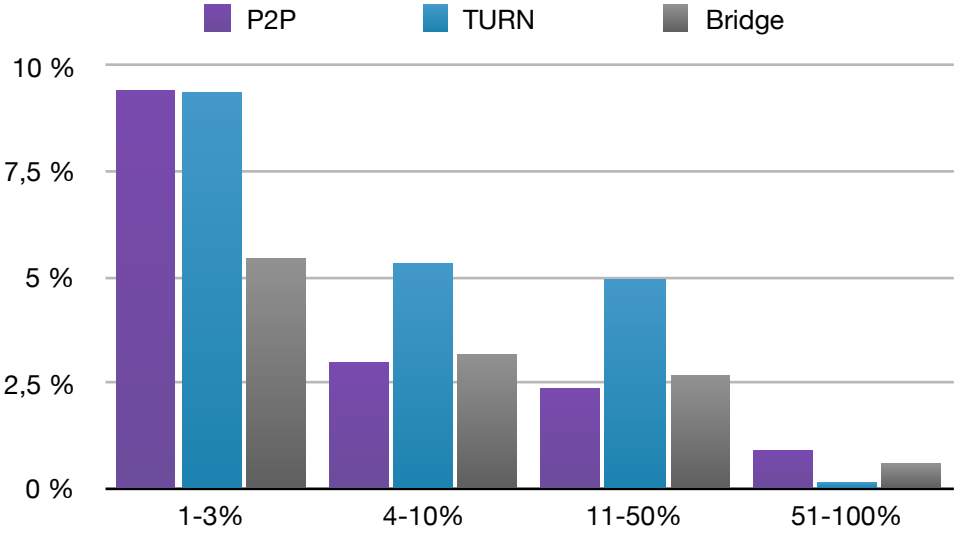


FIGURE 4.5. FRACTIONAL LOSSES BY CONNECTION TYPE

the previous report. In this report, since we are measuring across connection types, we observe that about 90% of the sessions on the bridge, 84% on the P2P, and 78% on the TURN have no 95%-ile losses. Surprisingly, we observe higher losses on the TURN server, which on further inspection showed that the TURN servers were under-performing, we observed a correlated loss across sessions on the same TURN server (based on the IP address of the TURN server).

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.

Lennart Schulte is a Software engineer at **callstats.io**, and a Ph.D candidate at the School of Electrical Engineering at Aalto University. He received his diploma degree in computer science from RWTH Aachen, Germany.

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-centered Product Development from Aalto University.

Copyright


callstats.io