WebRTC

REALTIME COMMUNICATION FOR THE OPEN WEB PLATFORM

What was once a way to bring audio and video to the web has expanded into more use cases than we could ever imagine.

NIKLAS BLUM,
SERGE LACHAPELLE,
AND HARALD
ALVESTRAND, GOOGLE

In this time of pandemic, the world has turned to Internet-based, RTC (realtime communication) as never before. The number of RTC products has, over the past decade, exploded in large part because of cheaper high-speed network access and more powerful devices, but also because of an open, royalty-free platform called WebRTC.

In fact, over the past year, there has been a 100-fold increase of video minutes received via the WebRTC stack in the anonymous population that has opted into Google Chrome's statistics. WebRTC can be found in most Internet meeting services, social networks, live-streaming experiences, and even cloud-based gaming products.

WebRTC provides RTC capabilities to browsers and native apps. An open-source implementation and tutorials for this platform can be found at <https://webrtc.org>. It

includes audio and video codecs, and signal-processing functions such as bandwidth estimation, noise suppression, and echo cancellation.

This widely deployed communications platform powers audio/video calling, conferencing, and collaboration systems across all major browsers, both on desktop and mobile devices. This has enabled billions of users to interact. WebRTC has vastly expanded and facilitated the ability to create and deploy realtime, interactive services for startups and large-scale companies, and it can be found in commercial products and open-source projects alike.

HOW WEBRTC STARTED

The idea for WebRTC originated in late 2009, more than a year after the launch of Google's Chrome browser. The Chrome team looked for functionality gaps between the desktop and the web. While most of the discrepancies were already being addressed by ongoing projects, no solution existed for realtime communications. At the time, only Adobe's Flash and Netscape's NPAPI (Netscape Plugin API) provided RTC. Flash's offering was somewhat low quality and required a server license. Plug-ins are quite tricky for users to install, and few developers have the resources to handle deploying and updating plug-ins that work with three different browsers across several operating systems.

At about this time Google identified a company, Global IP Solutions (aka GIPS), that had the low-level components required for RTC. The GIPS components were licensed by several large customers and were present in products from Google, Skype, AOL, Yahoo!, Cisco, and others. By

combining these audio and video components with a JavaScript interface, Google believed it could solve the big “hole” in its web offerings and spur innovation in the RTC market. If a few lines of JavaScript code were all you needed to add RTC to a web app—and with no licensing, integration of components, or deep knowledge of RTC required—who knew what could happen?

GIPS was based in Sweden and the U.S. and had engineers in both Stockholm and San Francisco. Luckily for Google, its audio and video Hangouts product was already being worked on in Stockholm, and having the GIPS engineers join in further reinforced the Stockholm office’s strength as an RTC specialist within Google.

When the acquisition was completed in January 2011, the newly formed Chrome WebRTC team focused on integrating the code into Chrome and open sourcing all the key components at webrtc.org. From the beginning the plan was to build something open for the web that would make RTC available for everyone.

ARCHITECTURE AND FUNCTIONALITY

A WebRTC peer may be a user endpoint (web browser, native app, etc.) or a server that acts as an intermediary between two or more endpoints. While many WebRTC services rely on a client-server architecture, many others are deployed in a peer-to-peer (P2P, aka connection-less) architecture.

WebRTC is both an API and a protocol. The WebRTC protocol is a set of rules for two WebRTC agents to negotiate bidirectional secure realtime communication. The WebRTC API²³ then allows developers to use the

WebRTC protocol.

The WebRTC API is specified only for JavaScript. The protocol to establish a connection between two WebRTC peers is a collection of other technologies, which can be split into signaling, connection management, security, and media transfer. These four steps usually happen sequentially. The prior step must be successful for the subsequent one to begin. Each step is actually made up of many other protocols.

As part of the WebRTC standards, many existing technologies that have been around since the early 2000s are combined and adapted for use in browsers and mobile applications.¹⁷

Figure 1 provides a high-level overview of the main components and technologies in WebRTC.

Android and iOS APIs are implementation-specific and not part of the standard, but they follow the same principles as the JavaScript APIs (webrtc.org open source implementation¹⁸). Audio and video capturing/rendering and network integration are specific to different operating systems.

PeerConnection API

The `RTCPeerConnection` API²¹ is the central part of the WebRTC specification dealing with connecting two applications on different endpoints to communicate using a peer-to-peer protocol. The communication between peers can be video, audio, or arbitrary binary data (see later in this article for clients supporting the `DataChannel` API).

In order to discover how two peers can connect, both clients need to provide a STUN (session traversal utilities

FIGURE 1: **WEBRTC LIBRARY COMPONENTS**

for NAT)⁹ or a TURN (traversal using relays around NAT) server³ configuration.¹¹ Their role is to provide an ICE (interactive connectivity establishment) candidate to each client, which is then transferred to the remote peer. This transferring of ICE candidates and exchange of other configuration information, such as media capabilities, is commonly called *signaling*.

Audio/video processing

WebRTC allows you to send and receive streams that include audio and/or video content. Streams can be added and removed at any time during a call; they can be either independent or bundled together. A common collaboration use case for RTC is to capture a computer's desktop content as a video feed and then include audio/video from the computer's webcam and microphone. The WebRTC protocol in general is codec agnostic. The underlying transport has been designed to support any codec format; however, the WebRTC user agent capabilities with regard to media codecs have been subject to standardization and are well defined.

The media functionality for processing audio and video provides the core of any WebRTC implementation. For audio communications and recording, Opus, G.711 μ -law/A-law algorithms, and DTMF (dual-tone multi-frequency) have been defined as mandatory codecs.¹⁶ The IETF standardization committees have agreed that WebRTC endpoints need to support the VP8 video codec and H.264 Constrained Baseline for processing video.¹³

Buffers in WebRTC implementations manage variability in packet arrival times, also called *jitter*, over the connection between peers. The logic of the buffering, managing of retransmission requests, and concealing data packets that have been lost or timed out is at the core of the signal processing work in WebRTC. These algorithms are constantly being developed and have seen major improvements over the past 10 years. The work greatly contributes to obtaining the best possible media quality when communicating over the Internet, especially

when peers are connected to networks with different throughput levels and quality.

Security and media transport

WebRTC connections must be encrypted. This is both a core part of the design and part of the standardization. Two existing protocols, DTLS¹² (Datagram Transport Layer Security) and SRTP² (Secure Realtime Transport Protocol), have been adopted for this.

DTLS allows you to negotiate a session and then exchange data securely between two peers. SRTP is designed for exchanging media; it does not have a handshake mechanism and is bootstrapped with the external keys exchanged via DTLS:

1. DTLS does the handshake over the connection provided by ICE. During the DTLS handshake, both sides offer a certificate.
2. The SRTP session is created from the keys generated by DTLS.
3. With these steps completed successfully, SRTP-encrypted media can be exchanged between WebRTC peers.

Media flows between WebRTC peers are by default based on UDP (User Datagram Protocol), meaning that the protocol has to handle unreliable delivery. To achieve the highest possible quality, the stack needs to make tradeoffs between latency and quality. Generally speaking, the more latency you are willing to tolerate, the higher-quality video you can expect. For realtime voice communication, ITU-T (International Telecommunication Union-Telecommunications Standardization Sector) has

defined the E-model,⁷ which says that users start being dissatisfied when the mouth-to-ear delay becomes greater than 250 ms.

Congestion control is the mechanism by which WebRTC figures out what quality is achievable, given the latency constraints. Practically speaking, congestion control is being used by a bandwidth estimator adapting the media-encoding parameters for bit rate and video resolutions or audio frame sizes. This lowers the quality but assures that media keeps flowing when users have low or varying bandwidth available.

In the early days of WebRTC, it took, even under good conditions, on average 40 seconds or more to establish a connection and reach video quality of 720 pixels (HD) resolution. By setting aggressive goals, the time was pushed down to 100 ms, thanks to a collaboration with researchers from the Polytechnic University of Bari. This collaboration led to a new congestion-controller design;⁴ figure 2 shows the result of launching the congestion-control algorithm.

Data channels

In addition to sending realtime audio and video data, WebRTC allows sending and receiving arbitrary data via so-called data channels. Use cases for data channels range from file transfer, gaming, and IoT (Internet of things) services to P2P CDNs (content delivery networks). The peer-to-peer data API²⁰ allows the creation of data channels. It extends the `RTCPeerConnection` API. SCTP (Stream Control Transmission Protocol)¹⁵ is used as the underlying protocol to transport data channels. It includes channel multiplexing,

FIGURE 2: **RAMP-UP TIME TO 1 MBPS VIDEO BIT RATE**

reliable delivery with TCP-like retransmission mechanism, congestion avoidance, and flow control.

STANDARDIZATION

At IETF 78 (summer 2010) in Maastricht, Google’s nascent WebRTC team had an informal lunch with engineers from Microsoft, Apple, Mozilla, Skype, Ericsson, and others to gauge the interest in building such an RTC platform for the

web. A quickly organized one-day workshop¹⁴ was held with the goal of understanding how such a standard should be written and defined. This led to intense activity in the W3C (World Wide Web Consortium) and the IETF, resulting in the formation of two working groups in May 2011: the IETF's RTCWeb⁵ and the W3C's WebRTC,²⁷ both with participation from across the industry.

WEBRTC IN 2020

The adoption of WebRTC has come a long way. Most modern services that use voice or video are either based on the WebRTC protocols or have the ability to use them in addition to the native protocols the service originally deployed with. Cisco's Webex service, for example, has a WebRTC client that lets people participate in conferences directly from their browsers without downloading additional software. Newer services, such as whereby.com and Jitsi, have been natively based on WebRTC from the outset. Even when no web browser is involved, major services use WebRTC for video transmission. For example, WebRTC enables the Amazon Ring product to view security camera and doorbell footage. Increasingly, new IoT products that stream voice and/or video are basing their network stacks on the WebRTC protocols.³

2020 was a year unlike any before. The need for RTC has been highlighted by Covid-19, as people across the globe have found new ways to work, educate, and connect with loved ones via video chat. WebRTC has suddenly become one of the most important sets of technologies allowing web browsers to make voice, video, and realtime data calls. It has allowed for an ecosystem of interoperable communications

apps to flourish: Since the beginning of March 2020, Chrome has seen a 100-fold increase in received video streams via WebRTC, excluding incognito and users opted by default out of sharing stats (see figure 3).

These successes would not have been possible without all the supporters that make an open source community. An important element of this success is all the code contributors, testers, bug filers, and corporate partners who helped make this ecosystem a reality.

FIGURE 3: VIDEO MINUTES IN CHROME OVER WEBRTC IN 2020

OUTLOOK

Google is a founding member of AOMedia (Alliance for Open Media) and has been active in defining the AV1 video bitstream for the RTC use case. As AV1 has become a standard, the video codec is being integrated into WebRTC. Chrome version 89 is shipping an AV1 software encoder providing AV1-to-web applications for RTC. AV1 provides another 30- to 50-percent bit-rate savings at the same quality compared with VP9, and is expected to offer another level of bandwidth efficiency and quality for video-calling services. Because of the complexity of the codec, hardware support will be of great importance to make it ubiquitously available. AV1 will be critical in facilitating RTC services to scale further and in allowing for higher-quality video experiences in the future.

WebRTC goes beyond voice and video communication. Emerging gaming, low-latency video streaming, AR/VR (augmented reality/virtual reality), and mixed-reality services are equally benefiting from and demanding low-latency media. For example, WebRTC enables the Stadia gaming service to bring cloud-based, low-latency, high-quality experiences to web browsers and televisions.

These use cases push the latency barrier, resulting in the need for further transport protocol optimizations. The corresponding standardization effort to cover this need is WebTransport,^{6,26} focusing on optimizing for super-low-latency client-server media streaming via the QUIC protocol.

As new use cases for WebRTC emerge, the WebRTC standardization is evolving into what is called WebRTC NV (Next Version).²⁵ NV will not be a completely new API but

will allow access to the lower-level media pipeline inside PeerConnections. Media will become accessible using the Streams¹⁹ and WebCodecs APIs.²² A first step in this direction is the already implemented Insertable Streams API²⁴ that provides the foundation for full E2EE (end-to-end encryption) multiparty conferencing in browsers.⁸

WebRTC's reach into mobile devices started through the native (i.e., nonweb) integration into mobile social media, messaging, and video calling apps. With emerging 5G networks, video calling will become even more of a commodity.

WebRTC's open architecture also allows for interesting innovations using machine learning and artificial intelligence to augment call quality and hide the effects of noise¹⁰ or network disruptions.¹

What started as a way to bring audio and video to the web has expanded into more use cases than could be imagined—from simple video calling to AR/VR experiences, cloud-based gaming, and massively scalable live streaming services; and from simple point-to-point video chat to multiuser conversations where quality is augmented through advanced machine-learning models. Most importantly, WebRTC is growing from enabling useful experiences to being essential in allowing billions to continue their work and education, and keep vital human contact during a pandemic. The opportunities and impact that lie ahead for WebRTC are intriguing indeed.

References

1. Barrera, P., Stimberg, F. 2020. Improving audio quality in Duo with WaveNetEQ. Google AI Blog (April 1); <https://>

- ai.googleblog.com/2020/04/improving-audio-quality-in-duo-with.html.
2. Baugher, M., McGrew, D., Naslund, M., Carrara, E., Norrman, K. 2004. The Secure Real-time Transport Protocol [SRTP], IETF RFC 3711; <https://tools.ietf.org/html/rfc3711>.
 3. Gross, G. 2020. WebRTC technologies prove to be essential during pandemic. IETF interview with Adam Roach [December 8]; <https://www.ietf.org/blog/webrtc-pandemic/>.
 4. Holmer, S., Lundin, H., Carlucci, G., De Cicco, L., Mascolo, S.; H. Alvestrand, ed. 2015. A Google congestion control algorithm for real-time communication, <https://tools.ietf.org/html/draft-alvestrand-rmcat-congestion-03>.
 5. IETF. Real-time communication in Web-browsers (RTCWeb) working group; <https://datatracker.ietf.org/wg/rtcweb/documents/>.
 6. IETF. 2021. WebTransport (webtrans); <https://datatracker.ietf.org/wg/webtrans/about/>.
 7. International Telecommunication Union-T. 2015. G.107: The E-model: a computational model for use in transmission planning; <https://www.itu.int/rec/T-REC-G.107-201506-I/en>.
 8. Iov, E. 2020. This is what end-to-end encryption should look like! (April 12). Jitsi blog; <https://jitsi.org/blog/e2ee/>.
 9. Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., Matthews, P. 2020. Session Traversal Utilities for NAT [STUN]. IETF RFC 8489; <https://tools.ietf.org/html/rfc8489>.
 10. Protalinski, E. 2020. Google Meet noise cancellation is rolling out now—here’s how it works. VentureBeat (June

- 8]; <https://venturebeat.com/2020/06/08/google-meet-noise-cancellation-ai-cloud-denoiser-g-suite/>.
11. Reddy, T., Johnston, A., Matthews, P., Rosenberg, J. 2020. Traversal using relays around NAT (TURN): Relay extensions to session traversal utilities for NAT (STUN), IETF RFC 8656; <https://tools.ietf.org/html/rfc8656>.
 12. Rescorla, E., N. Modadugu, N. 2012. Datagram Transport Layer Security, version 1.2. IETF RFC 6347; <https://tools.ietf.org/html/rfc6347>.
 13. Roach, A. B. 2016. WebRTC video processing and codec requirements. IETF RFC 7742; <https://tools.ietf.org/html/rfc7742>.
 14. RTC-Web Workshop. 2010; <http://rtc-web.alvestrand.com/>.
 15. Stewart, R., Ed. 2007. Stream Control Transmission Protocol. IETF RFC 4960; <https://tools.ietf.org/html/rfc4960>.
 16. Valin, J. M., Bran, C. 2016. WebRTC audio codec and processing requirements. IETF RFC 7874; <https://tools.ietf.org/html/rfc7874>.
 17. WebRTC for the Curious. 2020. What is WebRTC? [September 19]; <https://webrtcforthe curious.com/docs/01-what-why-and-how/>.
 18. WebRTC.org implementation. Google Git; <https://webrtc.googlesource.com/src/>.
 19. W3C. 2016. Streams API [November 29]; <https://www.w3.org/TR/streams-api/>.
 20. W3C. 2020. Peer-to-peer Data API [December 15]; <https://www.w3.org/TR/webrtc/#peer-to-peer-data-api>.
 21. W3C. 2020. RTCPeerConnection interface

- (December 15); <https://www.w3.org/TR/webrtc/#rtcpeerconnection-interface>.
22. W3C. 2020. WebCodecs (December 8); <https://wicg.github.io/web-codecs/>.
23. W3C. 2020. WebRTC 1.0: Real-time communication between browsers. W3C Proposed Recommendation (December 15); <https://www.w3.org/TR/webrtc/>.
24. W3C. 2020. WebRTC insertable media using Streams (September 1); <https://w3c.github.io/webrtc-insertable-streams/>.
25. W3C. 2020. WebRTC Next Version use cases (November 30); <https://www.w3.org/TR/webrtc-nv-use-cases/>.
26. W3C. 2020. WebTransport (December 9); <https://w3c.github.io/webtransport/>.
27. W3C. 2021. Web Real-Time Communications working group; <https://www.w3.org/groups/wg/webrtc>.

NIKLAS BLUM is a group product manager at Google. He leads the strategy and execution for the audio/video calling experience in Google's video communication products, including Google Meet, Google Duo, and Chrome/WebRTC. He has spent 15-plus years in the communications space. He holds a Ph.D. in service and software engineering from the University of Potsdam, Germany, and an MBA from ESMT, Germany.

SERGE LACHAPPELLE is director of product management at Google. He has spent more than 20 years in the video communications industry, starting as the cofounder of Marratech AB, which was acquired by Google in 2007. At

Google, Lachapelle started many video-calling initiatives, including Gmail Video Chat, Google Hangouts, WebRTC, Google Duo, and Google Meet. He holds a B.S. in computer science from Ecole Polytechnique de Montréal, Canada.

HARALD ALVESTRAND *is the standards coordinator for the WebRTC project at Google. He has used the Internet since 1984 and became an evangelist for open communication across company borders almost immediately. He has been a member of the board at ICANN, chair of the IETF, and area director of the IETF Applications Area. He holds a master's degree in electronics from the Norwegian University of Science and Technology (NTNU).*

Copyright © 2021 held by owner/author. Publication rights licensed to ACM.