Real-Time Communications for the Web

Cullen Jennings, Cisco
Ted Hardie, Google
Magnus Westerlund, Ericsson Research

ABSTRACT

This article provides an overview of the work that W3C and IETF are doing toward defining a framework, protocols, and application programming interfaces that will provide real-time interactive voice, video, and data in web browsers and other applications. The article explains how media and data will flow in a peer-to-peer style directly between two web browsers. This explains the protocols used to transport and secure the encrypted media, traverse NATs and firewalls, negotiate media capabilities, and provide identity for the media.

INTRODUCTION

The landscape of communications within browser contexts and between browsers and other systems has been dominated for some time by a set of proprietary protocols and plugins. That has hindered the developed of real-time interaction on the web by creating both non-overlapping walled gardens and a series of upgrade problems for sites which wish to support multiple systems. WebRTC aims to simplify the development and deployment of real time communication by providing a set of open standards that enable direct, interactive, rich communications between browsers and other clients. The Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) believe an open standard will best foster innovation on top of the basic components.

The core components WebRTC will enable are peer-to-peer audio and video streams, and a multiplexing peer-to-peer data channel. In the WebRTC model, a web server offers a JavaScript application using real-time facilities through a standard application programming interface (API) provided by the browser in which it resides or by another program using web technologies, such as a WebKit-based mobile game. The web server acts as a rendezvous server for signaling traffic until peer communications have begun.

With WebRTC in common browsers, an interactive component should be able to be added to an application on a website using the common API. WebRTC promises more than video chat buttons on websites. Early efforts at hackathon events have combined WebRTC with WebGL to great effect: live feeds being made into jigsaw puzzles, musicians playing together across the web, and a host of framing effects. Even more important may be the peer-to-peer data channel on the horizon, as it will allow web applications to share data with their peers without intermediation.

This article lays out the overall WebRTC system, showing how the downloaded application in a browser relates through it to the underlying protocols and network systems. The security challenges and issues around congestion control in transport contexts are called out as special sections, as is the extensibility model, but the basic aim is to provide context for those who plan on exploring WebRTC in depth.

ARCHITECTURE

The basic architecture for WebRTC involves at least three parties: an application provider and two peers. Each peer downloads a JavaScript application into a local web context (e.g., browser or mobile application). That application uses the WebRTC API [1] to communicate to the local context. For media exchange, each peer identifies local media sources as a set of tracks, negotiates the creation of a connection to the other peer, and passes the media along the opened connection.

The core API calls for this are GetUserMedia [2] and RTCPeerConnection [1]. Each GetUserMedia call returns a MediaStream, which consists of a set with zero or more synchronized media stream tracks, each track representing a local media resource such as a camera or microphone. The RTCPeerConnection represents a communication channel with a particular peer. The local entity adds one or more MediaStream to the RTCPeerConnection to have it transmitted to the peer. Adding a MediaStream to an RTCPeerConnection triggers a negotiation with the peer to agree on parameters, such as media format, resolution, and bandwidth limitations. The remote peer then adds a MediaStream to the RTCPeerConnection, there is another negotiation, and the local peer then gets a callback notifying it of the new or changed MediaStream and its tracks so it can start using it.

The WebRTC architecture does not mandate any particular protocol or function for performing rendezvous between peers. Establishing an

0163-6804/13/$25.00 © 2013 IEEE

RTCPeerConnection involves several steps, which are usually combined, with some being the sole responsibility of the web application:

- Requesting the establishment of an RTCPeerConnection to a peer
- The web application routing the request to the peer
- The peer approving or refusing the RTCPeerConnection
- Exchanging necessary parameters (this includes address information for the peers)
- Loose synchronization of attempts to establish a connection

To initiate an RTCPeerConnection, one peer gathers local information, which is supplied through the API as an Session Description Protocol (SDP) offer. The offer/answer negotiation with SDP is further described later in this article. The local information can include media and data channel information. The web application's servers then communicate with the intended peer. If the peer accepts the request, it creates its RTCPeerConnection object. The intended peer's RTCPeerConnection takes the offer and any locally added MediaStreams into account to produce an answer. When the initiating peer gets the answer, it can attempt to establish the actual network flow(s) for the RTCPeerConnection.

After an RTCPeerConnection is established, its contents can be renegotiated as required by adding or changing MediaStreams or data channels. Data channels [3] are created between peers using the RTCPeerConnection as well, using an underlying transport built as Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) over UDP.

An RTCPeerConnection always involves one pair of peers, but a peer can have multiple RTCPeerConnections carrying the same MediaStream to establish mesh-type conferences. Conferences can also be supported using central media processing nodes that provide media switching or the mixing of transmitted MediaStreams between sets of RTCPeerConnections.

Variations on the basic architecture include multiple application providers whose back-end services interoperate, and application providers which act as gateways to other services that accept and produce similar media. Gateways to IP telephony systems will be common, to permit WebRTC applications to exchange media with legacy VoIP systems.

Security Challenges

WebRTC provides a number of security challenges:

- JavaScript can be downloaded from any site without user consent.
- Users must be able to consent to the usage of media resources such as cameras and microphones.
- No significant amount of network traffic must be transmitted without consent from the destination.
- Confidentiality and authentication must be provided for any transmitted media to prevent man-in-the-middle attacks and eavesdropping.
- Users' private information, such as identity and geographical location, must not be disclosed when not desired, especially to third parties using the same web application.

In the WebRTC security architecture [4] the JavaScript application starts out as untrusted and potentially hostile. The basic premise of the security architecture is that the web browser or corresponding WebRTC implementation functions as a trusted computing base. Built on that base, the encrypted and authenticated transport of media, data, and server communications enable the development of trusted communications between the parties. Hooks for third-party user identity providers can prevent man-in-the-middle attacks even when using less than fully trusted applications. Using congestion control can prevent applications from using extremely unfair bandwidth shares at congested points.

Real-Time Media Transport

Real-time media, such as audio and video, is transported using Real-Time Transport Protocol (RTP). RTP is a well established standard for real-time media transport, which provides a number of functions, such as intra- and inter-media synchronization, packet loss detection, and transport feedback. RTP uses the concept of payload formats, which define how to encapsulate the encoded media in RTP packets, thus providing extensibility while enabling the best possible adaptations between the requirements of packet-based transport and the media encoding format.

RTP can be used over any lower-layer transport protocol that provides a datagram abstraction, making it usable directly over UDP and with a framing layer TCP. RTP can thus be used over whatever lower-layer transport flow has been established between the peers. The transport flow establishment using Interactive Connectivity Establishment (ICE) is described.

The RTP Control Protocol (RTCP) enables periodic transport feedback, event-driven transport and media feedback, and stream meta information, like stream inter-synchronization information. A number of extensions have been defined for RTP and RTCP.

End-to-end confidentiality, message authentication, and replay protection can be provided by Secure RTP (SRTP) [5]. The Audio Video Profile with Feedback (AVPF) early feedback profile provides significant improvements in the transmission of RTCP messages that are event-driven rather than periodic. WebRTC combines security and early feedback using the Secure RTP Profile for RTCP-Based Feedback (RTP/SAVPF) [6]. The key management for SRTP is provided by DTLS-SRTP [7], which is an in-band keying and security parameter negotiation mechanism. DTLS-SRTP enables perfect forward secrecy as well as possibilities for adding additional verification against a man-in-the-middle inserting itself in the key management step.

WebRTC also uses a number of standalone extensions to provide rich functionality. For conferencing this includes Full Intra Requests, which enables both joining conferences and switching between media sources by requesting that the media sender start sending media encod-

RTP can be used over any lower-layer transport protocol that provides a datagram abstraction, making it usable directly over UDP and with a framing layer TCP. RTP can thus be used over whatever lower-layer transport flow has been established between the peers.

The chief difficulty in setting up peer connections is that peers are often behind NATs or firewalls. WebRTC uses the ICE protocol to solve this problem. It allows various techniques to be tried simultaneously and uses whichever technique works best.

ed without any dependencies on the past media stream. Another conference extension is client-to-mixer audio levels, which provides a central mixer with per-packet voice energy levels that reduce delay and load on a node mixing or switching media streams. See [8] for a complete list of RTP/RTCP extensions.

Data Transport

In addition to providing real-time media transport over RTP, the WebRTC RTCPeerConnection provides transport of arbitrary data directly between the WebRTC endpoint peers. This data transport provides both reliable and semi-reliable message service, including multiple channels with prioritization support, so that low-priority large data objects can be prevented from blocking the transmission of higher-priority messages.

This data transport is realized using SCTP [9] with some extensions. SCTP has native support for message transport and multiple flows with prioritization. To ensure confidentiality and authentication of SCTP packets, they are sent protected by a DTLS association. This DTLS association is run over a lower-layer transport flow provided by ICE [10], commonly UDP.

Since SCTP can be sent natively over IP, this process may look unnecessarily complex. However, there are good reasons for this layering. UDP simply has the best chance of establishing a peer-to-peer transport flow when the endpoints are behind firewalls or network address translators (NATs). Enabling the usage of whatever datagram transport ICE successfully establishes maximizes the probability of a direct peer-to-peer data channel. Encapsulating SCTP inside DTLS, rather than running DTLS on top of SCTP, which is possible, means the SCTP transport protocol is also security protected.

Transport Issues

NAT and Firewall Traversal

The chief difficulty in setting up peer connections is that peers are often behind NATs or firewalls. WebRTC uses the ICE protocol [10] to solve this problem. It allows various techniques to be tried simultaneously and uses whichever technique works best.

The first technique is often called *hole punching*. The device inside the NAT sends a Session Traversal Utilities for NAT (STUN) packet to a server outside the NAT. The server reports back and informs the device inside the NAT of the IP address and port from which the packet appears to have come. This allows the device to discover the IP and port on the outside of the NAT used for that flow. The distant peer can send traffic to the external address of the hole. The STUN messages used to verify working connectivity include a secret that is passed in the signaling so that each of the two endpoints knows it is exchanging STUN messages with the other. As long as the NAT has "endpoint-independent mapping" behavior, as defined in [11], this approach works well.

The second technique uses a media relay. A peer forms a connection to a media relay box outside the NAT/firewall. The device requests a port on the relay box to which the far end can send data. The protocol used by WebRTC for media relays is the Traversal Using Relays around NAT (TURN) protocol. The downside of a media relay is that it adds latency and jitter to the media. In addition, someone needs to bear the cost of running the media relay. The media relay approach works with all types of NATs and many firewall deployments, as long as the firewalls do not block outbound TCP flows.

Sometimes the device is not actually behind a NAT for all address families and so may have an IP address can be used directly. IPv6 addresses, when present, are typically global and can be selected during the negotiation. IPv6 addresses may still be behind firewalls, though, and they can have limited reachability during this phase of IPv6 deployment, so they cannot be selected blindly.

In general, the STUN technique results in a high-quality, cheaper connection when it works, but the TURN technique has a higher success rate. ICE [10] is a technique that combines all these techniques and chooses the best one. With ICE, first both sides gather a bunch of possible address/port pairs that might work. This includes the local host's IPv4 and IPv6 addresses, addresses found by talking to STUN servers, and addresses of zero or more media relays. Both sides then exchange these addresses, along with an indication of how desirable they are. The combinations of the addresses for both sides form an ordered list of candidate flows that might work. The candidate flows are tested in order by both sides by sending STUN messages to the far end and awaiting a reply. ICE chooses the best working candidate.

Multiplexing Data and Media over One Transport Flow

WebRTC enables the multiplexing of all the RTP media streams and the data to share a single lower-layer transport. This minimizes the risk of NAT and firewall traversal failure, that is, the risk that transport flow is established for media but not data or vice versa. This also results in fate sharing between the data and real-time media. The shared flow does, however, prevent a flow-based quality of service (QoS) mechanism being applied to subflows (e.g., prioritizing audio and video and not the data).

This aggregate will most commonly use a UDP flow as lower-layer transport. That UDP flow can be used for one RTP session carrying multiple RTP media streams, ICE's STUN messages, and the data transport messages. Classical multiplexing with one UDP flow per RTP session or data channel is also supported in WebRTC to support existing deployments and flow-based QoS mechanisms.

As can be seen in Fig. 1, having everything over a single UDP transport flow results in the three protocols existing on the same level on the same lower-layer identifier (UDP flow). The actual demultiplexing of the three protocols is done by looking at the value of the first byte of the UDP payload. A value of 0 or 1 indicates a STUN packet, a value in the range 20–63 indi-

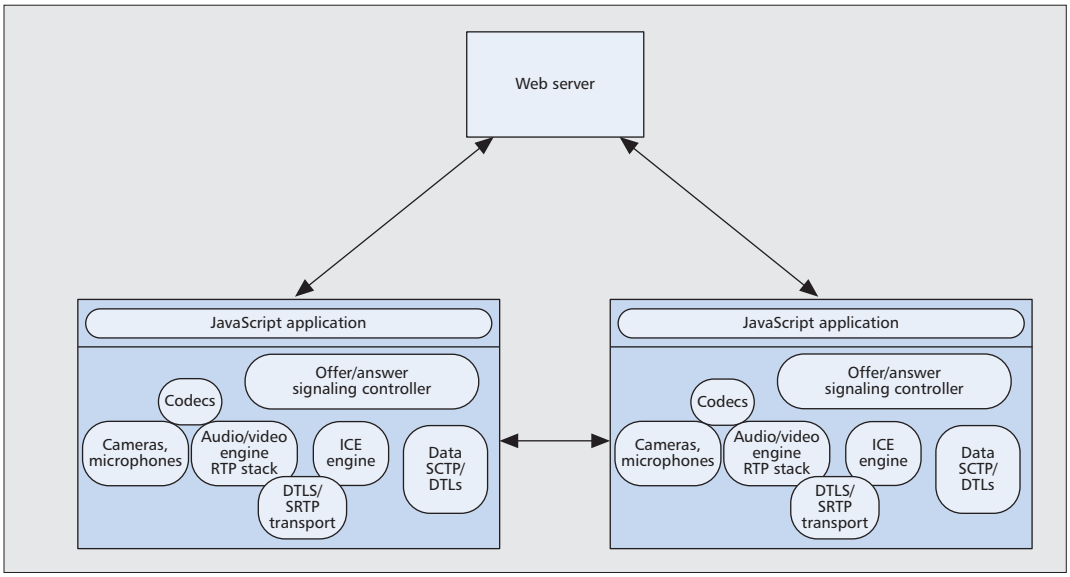

Figure 1. *Browser architecture.*

cates a DTLS packet, and a value of 128–191 indicates an SRTP/SRTCP packet. This multiplexing does produce some limitations: a single RTP session must carry all media streams, both audio and video. RTP has good support for multiple media streams within an RTP session; however, including multiple media types in one RTP session has some implications [12]. DTLS has two roles in this usage, as shown in Fig. 2: It encapsulates SCTP packets in its security protected data format, and it provides key management and parameter negotiation for the SRTP protection of the RTP session.

Congestion Control

WebRTC will operate over Internet paths that provide best effort services to the IP packets. In such an environment, some type of congestion control is required. Not only does congestion control ensure the highest possible quality for the WebRTC application, but it also prevents malicious servers from establishing high-bandwidth flows across bottlenecks that are shared with the targets of attacks and pushing the target's flows out of the way. The data channel uses SCTP and its congestion control algorithm. What to use for real-time media remains an open question. This is a difficult problem: A solution must try to keep delay and packet losses low, and provide some type of fairness between flows. The interaction between the real-time and data channel congestion controls also needs to be dealt with to prevent SCTP starving the real-time media in the same peer connection.

Security Solutions

Communication Consent

The web uses the Same Origin Policy [13] to try to prevent one server from using a web client's credentials and resources to attack another server. The core of the policy is that no script is allowed to send requests or data to a server other than the one from which it originated. Such a policy is clearly incompatible with WebRTC, which has the explicit goal of permitting communication with a peer that could be any host on the Internet. Another mechanism is required to ensure that a malicious script does not use WebRTC to initiate harmful or overabundant network traffic toward another host. A consent mechanism allows each browser to verify that the intended peer consents to communication.

WebRTC's mechanism for establishing consent for the media and data flows in the RTCPeerConnection is to reuse the ICE mechanism [10] already available for NAT and firewall traversal. A WebRTC implementation is required to verify communication with the peer for each transport flow by using STUN binding request/response pairs before sending any other type of packet. These STUN requests and responses use a browser-generated secret that is shared with the peer through the web application's rendezvous mechanism. The browser also cryptographically generates random 96-bit transaction identifiers, which are not shared with the JavaScript for each request, so that off-path attackers cannot forge responses.

To prevent a peer from continuing to send traffic for an indefinite time, the receiving peer's continued consent must be verified. This process uses the same mechanism as for the initial verification but in parallel with ongoing communication. If an authenticated response is not received within 30 s of the previous one, all media and data transmission in this connection will be halted.

Identity

In many communications use cases, it is important to be able to know the identity of the entity at the far end. Whenever media is encrypted, it is important to be sure that it comes from the intended sender and not from some man-in-the-middle. The media encryption in WebRTC is based on using SRTP with DTLS-SRTP key management. However, this simply results in a fingerprint each needs to check to understand that media has not been tampered with by a man-in-the-middle. This fingerprint needs to be

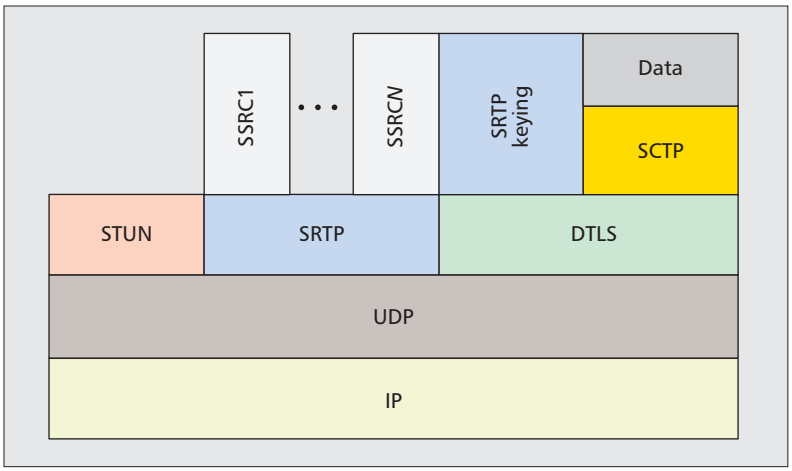

Figure 2. *Protocol layering of an RTCPeerConnection with both media and data channels over a UDP transport flow.*

cryptographically bound to a meaningful identity for the user to be able to know whether the encrypted media is going to and from the intended other party or a man-in-the-middle attacker.

A wide and changing variety of identity solutions are available. The WebRTC approach is to form an identity proxy in the browser, such that changing the identity provider protocol does not become a problem. The basic approach is that the user on a given browser uses their preferred identity provider and logs on to the provider in whatever way that provider uses. This log on is done in a security context in the browser that is separate from the JavaScript code running the RTCPeerConnection. This identity provision may be as simple as passwords and cookies, or it may involve multifactor authentication, perhaps leveraging the user's mobile device. Once the user is logged on in a separate security context in the browser, the identity provider can sign an assertion that binds the fingerprint from DTLS-SRTP to the identity of the user of that browser. The reason for the separate security context is to ensure that the JavaScript code from running the RTCPeerConnection cannot tamper with the DTLS-SRTP fingerprint before the assertion is created. This assertion, along with information about how to contact the identity provider to validate the assertion, is passed to the far side. The far side can check that the fingerprint in the assertion matches the fingerprint in the DTLS/SRTP connection and validate the assertion.

Media Negotiation

Both peers need ways of describing the capabilities of their media subsystems to the other side, and describing what actual media is being sent and received. The peers can then agree on a common set of codecs to use, the parameters for the codecs, how many audio and video streams are being used, transport information, and any RTP extensions they choose to use. WebRTC uses the IETF RFC 3264 offer/answer approach [14], which uses SDP to exchange this information. This is the same approach used in Session Initiation Protocol (SIP) and Jingle. One side creates an *offer* that describes roughly what it is willing to do. The other side looks at the offer and creates an *answer* that specifies which of the offered options have been selected. The offer/answer pair together describes the state of media subsystems, including what media is being sent and received, as well as the transport information such as the source and destinations of the media. SDP is used to negotiate the RTP media transport as well as the SCTP data channels.

The JavaScript application in one of the peers calls the API to create an SDP offer. This offer describes the media capabilities the browser is willing to send and receive. The application has a chance to modify this to remove or change some of the information in the SDP. Then the application tells the browser to be prepared to send and receive that media. At this point, the browser has reserved the resources for all the capabilities it is advertising in the offer and is prepared to receive media for any of them.

The application passes the offer to the other peer using a mechanism of the application's choosing. Typically, the offer is passed up to the web server, and then the web server uses an approach like web-sockets to pass the offer down to the other peer. The application on the other peer can pass this offer to the browser to have the browser create an answer. Once the application decides to use this offer, the browser can immediately start sending media to the other peer. At the same time, the answer is passed back to the other peer via a path through the web server.

When the peer that created the offer receives the answer, the application can pass this answer to the browser, and the browser can start sending media to the other side. If the answer is a final answer and no more answers will be honored for that particular offer, the browser can also free up any resources that were reserved in the offer but are not in use in the answer.

Media Encoding

The WebRTC framework allows peers to negotiate the codecs that will be used for real-time media. The Working Groups involved have agreed to select mandatory-to-implement audio and video codecs. This selection is meant to ensure that there are no interoperability failures among compliant systems; it is not meant to limit the total number of available encodings. As noted below, negotiation is a primary point of extensibility. For audio, the mandatory-to-implement codecs are G.711 and OPUS. A choice for video has not yet been made. VP8 and H.264 are being considered.

Future Extensibility

The initial work on WebRTC has been designed to enable easy deployment of real-time applications focused on rich media. Since the definition of "rich media" changes over time, the overall system is designed to have multiple points of extensibility. The first of these is the offer/answer negotiation mechanism, which allows peers to offer additional codecs or media types at any time. The second is the functionality of the basic API; GetUserMedia is not tied to a specific set

of media sources, so it can be used whenever new sources of media are available. WebRTC's mechanism for handling independent media tracks also allows for many different kinds of combinations at the presentation layer.

Outside the realm of "rich media," WebRTC's inclusion of a peer-to-peer data channel also provides a key opportunity for extension. Once such a data channel is available, the two peers can use it in a variety of application contexts, including real-time games, resource sharing, and computational collaboration. One way of looking at WebRTC is as a variant of peer-to-peer networking, with the hosting web site both providing the peers' clients and acting to connect specific peers. While its function goes beyond that of a traditional peer-to-peer enrollment server, an application provider can easily match that set of capabilities. That means it can enable not just peer connections, but peer networks. While the early applications contemplated by WebRTC use essentially full-mesh topologies for peer communications, it would not be difficult to extend WebRTC to use topologies like those of distributed hash table (DHT)-based peer-to-peer networks. With those new topologies would come new opportunities for extensions.

Conclusions

WebRTC relies on a variety of mechanisms with long histories: offer/answer negotiation, NAT/firewall traversal, RTP-based media exchange, peer-to-peer data channels, and the web itself. Combining them promises to create an open ecosystem that will make peer-to-peer applications both radically easier to deploy and far richer in their media content.

At the time of publication, several browsers have implemented the current version of WebRTC and demonstrated interoperability. There is an active group of application developers experimenting with the technology and API. This work is, in turn, driving the standards process. While the standards are not yet finalized, there are already innovative applications being built. As it matures, the authors expect WebRTC to have a profound impact on the user experience of the web.

Research Questions in WebRTC

There are some questions that have arisen during the engineering work of defining WebRTC that would benefit from research input. WebRTC is also a sufficiently complex system that different types of studies could be performed to verify the intentions of the design or study the impact of the system on the communication market or users' communication patterns and habits.

Real-time media congestion control is one issue for which WebRTC at this stage is unable to provide a full specification for a well functioning system. The IETF has recently responded to this shortcoming by starting a new working group, RTP Media Congestion Avoidance Techniques (RMCAT). This working group will need input such as algorithm proposals, information about what media degradations have the least subjective quality reduction, and analyses of proposed algorithms' fairness, stability and so on.

WebRTC currently relies on user consent to grant applications for access to local media resources to those intending to use the WebRTC APIs. This creates issues for user interface designers. It is well known that users do not seriously consider what they are doing when they are invited to click quickly through dialog boxes. Security can be seriously compromised. There is room for innovation in this space, along with studies to determine how successful the emerging implementations are.

The WebRTC security architecture could also benefit from studies to determine the overall security attained and identify security flaws or other issues arising in early deployments.

It is well known users do not seriously consider what they are doing when they are invited to click quickly through dialog boxes. Security can be seriously compromised. There is room for innovation in this space, along with studies to determine how successful the emerging implementations are.

References

[1] Bergkvist *et al.*, "WebRTC 1.0: Real-Time Communication between Browsers," Oct. 2012 http://dev.w3.org/2011/webrtc/editor/webrtc.html.
[2] Burnett and Narayanan, "Media Capture and Streams," Sept. 2012, http://dev.w3.org/2011/webrtc/editor/getusermedia.html.
[3] R. Jesup, S. Loreto, and M. Tuexen, "RTCWeb Datagram Connection," IETF Internet-Draft draft-ietf-rtcweb-data-channel-01, Sept. 2012.
[4] E. Rescorla, "RTCWEB Security Architecture," IETF Internet-Draft draft-ietf-rtcweb-security-arch-03, July 2012.
[5] M. Baugher *et al.*, "The Secure Real-time Transport Protocol (SRTP)," IETF RFC 3711, Mar. 2004.
[6] J. Ott and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)," IETF RFC 5124, Feb. 2008.
[7] D. McGrew and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)," IETF RFC 5764, May 2010.
[8] C. Perkins, M. Westerlund, and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP," IETF Internet-Draft draft-ietf-rtcweb-rtp-usage-04, July 2012.
[9] R. Stewart, "Stream Control Transmission Protocol," IETF RFC 4960, Sept. 2007.
[10] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," IETF RFC 5245, Apr. 2010.
[11] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," BCP 127, IETF RFC 4787, Jan. 2007.
[12] M. Westerlund, C. Perkins, and J. Lennox, "Multiple Media Types in an RTP Session," IETF Internet-Draft draft-ietf-avtcore-multi-media-rtp-session-00, Oct. 2012.
[13] http://www.w3.org/Security/wiki/Same_Origin_Policy
[14] J. Rosenberg and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)," IETF RFC 3264, June 2002/.

Additional Reading

WebRTC related APIs and Protocol Specification Overview:
- "Media Capture and Streams," Sept. 2012, http://dev.w3.org/2011/webrtc/editor/getusermedia.html.
- "WebRTC 1.0: Real-time Communication Between Browsers," http://dev.w3.org/2011/webrtc/editor/webrtc.html.
- "Overview: Real Time Protocols for Brower-based Applications," draft-ietf-rtcweb-overview.

NAT and Firewall Traversal functions used:
- "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," IETF RFC 5245.
- "Session Traversal Utilities for NAT (STUN)," IETF RFC 5389.
- "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," IETF RFC 5766.
- "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," BCP 127, IETF RFC 4787.

Media Codecs:
- ITU-T Rec. G.711, "Pulse Code Modulation (PCM) of Voice Frequencies.
- "Definition of the Opus Audio Codec," IETF RFC 6716.
- "VP8 Data Format and Decoding Guide," IETF RFC 6386.
- ITU-T Rec. H.264, "Advanced video coding for generic audiovisual services."

RTP and Media Transport:
- "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP," IETF Internet-Draft draft-ietf-rtcweb-rtp-usage-05.
- "Multiple Media Types in an RTP Session," IETF Internet-Draft draft-ietf-avtcore-multi-media-rtp-session-01.
- "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)," IETF RFC 5104.

Data Channel
- "RTCWeb Datagram Connection," IETF Internet-Draft draft-ietf-rtcweb-data-channel-02.
- "Stream Control Transmission Protocol," IETF RFC 4960.
- "Datagram Transport Layer Security Version 1.2.," IETF RFC 6347.

Media Negotiation and Session Establishment
- "JavaScript Session Establishment Protocol," IETF Internet-Draft draft-ietf-rtcweb-jsep-02.
- "SIP: Session Initiation Protocol," IETF RFC 3261.
- "Jingle," XSF XEP 0166.

Biographies

Cullen Jennings (fluffy@cisco.com) is a Cisco Fellow in the Video & Collaboration Group at Cisco. He is responsible for setting the technical direction for the next generation of Cisco's communication and collaboration products, including VoIP, SIP, security, and peer-to-peer technologies. He also participates in Cisco's development efforts for the Internet of Things. He holds a Ph.D. from the University of British Columbia in computer vision. His current focus is on moving voice and video communications into the cloud so that they become an integrated part of web applications, and can be used in everything from business process software to collaboration applications and games. This involves smoothly integrating a number of technologies ranging from HTML5 to highly scalable cloud services and federated authentication. Previously, he played a key role in developing Cisco's technical strategy in voice/video security, media quality, NAT traversal, and peer-to-peer services. He currently serves as Co-Chair of the IETF RTCWeb working group and is co-editor for WebRTC specification at W3C. He has authored many industry standards and patents, and is an author of the book *Practical VoIP* (O'Reilly). A longtime sailor and snowboarder, lately he has been learning to kitesurf.

Ted Hardie (ted.ietf@gmail.com) currently leads a small team within Google's networking operations organization. His research interests cover a broad range of Internet technologies, but are currently focused on software defined networking, novel data distribution methods, mesh and peer networks, locality of service, and identifiers for non-hierarchical systems. He first worked in the Internet field in 1988 when he joined the operations staff of the SRI NIC. He later became the technical lead for the NASA NIC, part of the NASA Science Internet project. After leaving NASA, he joined Equinix as its initial director of engineering before taking on the role of director of research and development. He was an early-stage executive at Nominum. While he was Qualcomm's director of Internet and wireless, he served the Internet community as a member of the Internet Architecture Board and as an Applications Area Director for the IETF. He later acted as managing director for Panasonic's U.S Wireless Research Laboratory and the vice president of R&D for IP Infusion. He is the author of multiple RFCs, and has been an invited speaker and document reviewer for both granting agencies and technical conferences. He received his Bachelor's degree from Yale and his doctorate from Stanford. He has been a Fulbright Fellow and a Yale-China Fellow, both in Hong Kong.

Magnus Westerlund (magnus.westerlund@ericsson.com) received his M.Sc degre in computer science from Luleå University of Technology, Sweden, in 1999. He has been an IETF participant since 2000. He has served as Chair of the Audio/Video Transport (AVT) and the Audio/Video Transport Extensions (AVTEXT) Working Groups. He also served as Transport Area Director for four years from 2006. He is currently a researcher at Multi-Media Technologies at Ericsson Research in Stockholm, Sweden. He is currently actively contributing to real-time media transport functionalities for WebRTC and other video communication systems. He is currently chairing the IETF Working Groups Real-Time Communication in Web browsers (RTCWeb) and Audio/Video Transport Core Maintenance (AVTCORE). He has worked on a number of projects related to real-time media transport and application signaling. He has co-authored a number of RFCs, including RTP payload formats for AMR, AMR-WB, AMR-WB+, G.719, and H.264, and the RTP extension for codec control messages. He is still an active technical contributor in the AVTCORE, AVTEXT, RTCWeb, 6MAN, and MMUSIC WGs. He has also been participating in the Third Generation Partnership Project, contributing to the packet-switched streaming service, and multimedia broadcast and multicast service (specifications.