Network Working Group
Internet-Draft
Intended status: Experimental
Expires: August 19, 2011

H. Alvestrand
Google

February 15, 2011

A Datagram Transport for the RTC-Web profile
draft-alvestrand-dispatch-rtcweb-datagram-01

Abstract

This document describes a combination and profiling of existing IETF protocols to provide a datagram service that is suitable as a generic transport substrate for the RTC-Web family of real-time audio/video applications.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 19, 2011.

Copyright Notice

Table of Contents

1. Introduction

When transporting audio / video and other realtime data between participants on the current Internet, there are a number of obstacles to be faced.

Among them are NAT boxes, firewalls, connection interruptions, the availability of multiple paths between participants, and capacity issues.

This memo describes a combination of existing protocols that can be used to achieve a seamless datagram transport service across this very heterogenous environment.

An overview of the effort of which this is a part can be found in the overview document, [overview].

2. Terminology

This draft uses a couple of commonly used terms in quite specific ways. The reader is advised to study these definitions carefully.

(TODO: Agree on terminology to use)

Session An association with two endpoints, between which datagrams flow.

Datagram A sequence of octets, of a given length. In this specification, a datagram does not carry addressing information.

Channel One means of transporting a datagram over a session. A session may have multiple channels at any time. <Question: Should this word be replaced by "transport", for more consistency with ICE?>

Endpoint One end of a session. This document does not distinguish between an initiator and a responder endpoint.

Control channel A means of communication between the endpoints of a session that does not require a transport to be active. Typically, authentication, authorization and negotiation is carried out over the control channel. The specification of the control channel is out of scope for this specification.

3. Service model

The basic model presented is a datagram model. On top of this one can layer various services, such as pseudoTCP (REF), RTP[RFC3550] or any other higher layer protocol that is capable of running across a datagram service. (If a TCP connection can be established between the parties, this is usually the preferred option for reliable, sequenced transfer. The use of this datagram service for reliable transfer should be considered an option available for the case where only UDP connectivity is available.)

The addressing model departs from the traditional Internet model in that end point addresses are not used for endpoint identification, only for channel establisment; instead, an initial packet exchange, using ICE [RFC5245], is used to bind a channel to a prenegotiated session.

The datagram service is not completely transparent; in particular, it is not possible to carry a datagram where the two highest bits of the first octet are zero and octet 5 to 8 contain the value 0x2112A442, since these datagrams are reserved for use of the STUN protocol (RFC 5389 section 6).

4. Channel types

4.1. UDP channel

An UDP channel is negotiated using ICE. Each datagram is simply carried as the content of an UDP packet.

4.2. TCP channel

A TCP channel consists of a TCP connection, over which are sent datagrams packaged according to RFC 4571 [RFC4571]. The binding of a TCP channel is done by executing an ICE negotiation over the first few packets passed across the TCP channel, as specified in ICE-TCP [I-D.ietf-mmusic-ice-tcp]

4.3. TLS channel

A TLS channel consists of a standard TLS negotiation, followed by passing datagrams over the TLS record layer[RFC5246] section 6.2. There is no extra length field. A TLS channel is bound to its session by <insert process description>.

4.4. DTLS channel

A DTLS channel is created by executing a DTLS [RFC5238] connection negotiation, followed by datagram exchange, where the datagrams are protected by DTLS mechanisms. The DTLS channel is bound to its session by <insert process>.

4.5. WebSockets channel

A WebSockets channel uses the WebSockets protocol[I-D.ietf-hybi-thewebsocketprotocol] to pass datagrams as binary packets.

4.6. Channels with relay

If there is no possibility of setting up a direct connection, a relay must be used. When both parties are reachable using UDP candidates, the specification from TURN [RFC5766]is used. <NOTE - more text needed here - in particular because for TLS and WebSockets channels, TURN does not apply.>

5. Channel setup, teardown and usage

The service model envisioned here is that all datagrams arriving on a session are considered equally valid. The session gives no guarantees against duplication, loss or reordering; such concerns are left to the higher protocol layers.

The expected normal usage is that two endpoints will exchange addressing information that can be used for a series of potential channels, that the endpoints will probe for working channels using ICE (RFC 5245), and use the "best" candidate, while using the STUN probing facilities to keep some number of "second best" candidates alive if the "best" candidate stops working.

A data-sending endpoint may unilaterally decide to start or stop using an established channel at any time. No negotiation is necessary.

A receiving endpoint will learn that a channel has been removed by not seeing any more STUN keepalive messages on that channel within <timeout>.

A session is considered closed when all channels that have been successfully established have timed out.

6. An URI scheme for datagram channels

This URI scheme is mainly included in order to make it easy for APIs that normally use URIs as what they use to refer to objects. It reflects exactly the information found in the SDP attributes defined in RFC 5245 section 15.

<NOTE IN DRAFT: This may be replaced with a JSON representation, an XML representation or the SDP representation in later drafts, if the WG so decides.>

The DGSESSION URI scheme specifies the information required for a session; it consists of two parts:

- An absolute reference, which includes the user name and password used to establish channels over this connection.
- A series of addressing hints, which include the data necessary to establish a channel.

<TODO: Fill out an URI registration template for the scheme>

Example:

dgsession:username:password?ipv4:12.34.56:udp:12345&
ipv6:2002::dead:beef:tcp:80&ipv4:12.34.56.78:tls:443

The sequence of addressing hints is an indication of the preference of the URL constructor for the sequence in which to try these candidates; the most preferred address is the one to the left.

Note that a DGSESSION URI is a capability; anyone with the URI will be able to connect to the entity. They should therefore be handled in the same way as (short-term) passwords, and never passed in the clear.

6.1. new section

7. IANA Considerations

This document registers the URI scheme from section Paragraph 1.

Note to RFC Editor: this section may be removed on publication as an RFC.

8. Security Considerations

As with all layered protocols, it is a matter for the application to decide which level security should be provided at. For instance, an RTP session protected using SRTP <ref> can be considered to not need any further safeguards against interception, modification or replay, so can be passed "in the clear" across any channel type here. For data without such protection, adequate measures need to be taken; in particular, it is trivially easy for someone with the ability to snoop and insert packets to insert fake packets into an established UDP channel.

The main defense against denial-of-service attacks is the fact that the ICE mechanisms were designed for low cost refusal of unauthorized connections.

9. Acknowledgements

Thanks to Markus Isomaki for reviewing version -00.

10. References

10.1. Normative References

[I-D.ietf-hybi-thewebsocketprotocol]
Hickson, I., "The WebSocket protocol", draft-ietf-hybi-thewebsocketprotocol-00 (work in progress), May 2010.

[I-D.ietf-mmusic-ice-tcp]
Perreault, S. and J. Rosenberg, "TCP Candidates with Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-tcp-08 (work in progress), October 2009.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, July 2006.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

10.2. Informative References

[overview] Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", November 2010.

Appendix A. Change history

A.1. Changes from alvestrand-00 to alvestrand-01

Added the WebSockets channel option. Made some changes and clarifications, mainly based on Markus Isomaki's review. Pointed out that the DGSESSION URI scheme has to represent exactly the semantics of the SDP extensions for ICE.

Author's Address

Harald Tveit Alvestrand
Google
Kungsbron 2
Stockholm, 11122
Sweden

Email: harald@alvestrand.no

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2011

H. Alvestrand
Google

March 13, 2011

Overview: Real Time Protocols for Brower-based Applications
draft-alvestrand-dispatch-rtcweb-protocols-01

Abstract

This document gives an overview and context of a protocol suite intended for use with real-time applications that can be deployed in browsers - "real time communication on the Web".

It intends to serve as a starting and coordination point to make sure all the parts that are needed to achieve this goal are findable, and that the parts that belong in the Internet protocol suite are fully specified and on the right publication track.

This work is an attempt to synthesize the input of many people, but makes no claims to fully represent the views of any of them. All parts of the document should be regarded as open for discussion, with the intended discussion forum being the "RTCWEB" WG (in formation).

Currently, discussion is on the rtc-web@alvestrand.no mailing list.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft will expire on September 14, 2011.

Copyright Notice

Table of Contents

1. Introduction

The Internet was, from very early in its lifetime, considered a possible veichle for the deployment of real-time, interactive applications - with the most easily imaginable being audio conversations (aka "Internet telephony") and videoconferencing.

The first attempts to build this were dependent on special networks, special hardware and custom-built software, often at very high prices or at low quality, placing great demands on the infrastructure.

As the available bandwidth has increased, and as processors and other hardware has become ever faster, the barriers to participation have decreased, and it is possible to deliver a satisfactory experience on commonly available computing hardware.

Still, there are a number of barriers to the ability to communicate universally - one of these is that there are, as of yet, no single set of communication protocols that all agree should be made available for communication; another is the sheer lack of universal identification systems (such as is served by telephone numbers or email addresses in other communications systems).

Development of The Universal Solution has proved hard, however, for all the usual reasons. This memo aims to take a more building-block-oriented approach, and try to find consensus on a set of substrate components that we think will be useful in any real-time communications systems.

The last few years have also seen a new platform rise for deployment of services: The browser-embedded application, or "Web application". It turns out that as long as the browser platform has the necessary interfaces, it is possible to deliver almost any kind of service on it.

Traditionally, these interfaces have been delivered by plugins, which had to be downloaded and installed separately from the browser; in the development of HTML5, much promise is seen by the possiblitiy of making those interfaces available in a standardized way within the browser.

Other efforts, for instance the W3C Web Applications and Device API working groups, focus on making standardized APIs and interfaces available, within or alongside the HTML5 effort, for those functions; this memo concentrates on specifying the protocols and subprotocols that are needed to specify the interactions that happen across the network.

2. On interoperability and innovation

The "Mission statement of the IETF" [RFC3935] states that "The benefit of a standard to the Internet is in interoperability - that multiple products implementing a standard are able to work together in order to deliver valuable functions to the Internet's users."

Communication on the Internet frequently occurs in two phases:

- Two parties communicate, through some mechanism, what functionality they both are able to support
- They use that shared communicative functionality to communicate, or, failing to find anything in common, give up on communication.

There are often many choices that can be made for communicative functionality; the history of the Internet is rife with the proposal, standardization, implementation, and success or failure of many types of options, in all sorts of protocols.

The goal of having a mandatory to implement function set is to prevent negotiation failure, not to preempt or prevent negotiation.

The presence of a mandatory to implement function set serves as a strong changer of the marketplace of deployment - in that it gives a guarantee that, as long as you conform to a specification, and the other party is willing to accept communication at the base level of that specification, you can communicate successfully.

The alternative - that of having no mandatory to implement - does not mean that you cannot communicate, it merely means that in order to be part of the communications partnership, you have to implement the standard "and then some" - that "and then some" usually being called a profile of some sort; in the version most antithetical to the Internet ethos, that "and then some" consists of having to use a specific vendor's product only.

3. Functionality groups

The functionallity groups that are needed can be specified, more or less from the bottom up, as:

- Data transport: TCP, UDP and the means to securely set up connections between entities.
- Data framing: RTP and other data formats that serve as containers, and their functions for data confidentiality and integrity.

- Data formats: Codec specifications, format specifications and functionality specifications for the data passed between systems. Audio and video codecs, as well as formats for data and document sharing, belong in this category.

- Connection management: Setting up connections, agreeing on data formats, changing data formats during the duration of a call; SIP and Jingle/XMPP belong in this category.

- Presentation and control: What needs to happen in order to ensure that interactions behave in a non-surprising manner. This can include floor control, screen layout, voice activated image switching and other such functions - where part of the system require the cooperation between parties. Cisco/Tandberg's TIP was one attempt at specifying this functionality.

- Local system support functions: These are things that need not be specified uniformly, because each participant may choose to do these in a way of the participant's choosing, without affecting the bits on the wire in a way that others have to be cognizant of. Examples in this category include echo cancellation (some forms of it), local authentication and authorization mechanisms, OS access control and the ability to do local recording of conversations.

Within each functionality group, it is important to preserve both freedom to innovate and the ability for global communication. Freedom to innovate is helped by doing the specification in terms of interfaces, not implementation; any implementation able to communicate according to the interfaces is a valid implementation. Ability to communicate globally is helped both by having core specifications be unencumbered by IPR issues and by having the formats and protocols be fully enough specified to allow for independent implementation.

One can think of the three first groups as forming a "media transport infrastructure", and of the three last groups as forming a "media service". In many contexts, it makes sense to use a common specification for the media transport infrastructure, which can be embedded in browsers and accessed using standard interfaces, and "let a thousand flowers bloom" in the "media service" layer; to achieve interoperable services, however, at least the first five of the six groups need to be specified.

4. Data transport

Datagram transport is the subject of a separate draft, "A Datagram Transport for the RTC-Web

profile".[I-D.alvestrand-dispatch-rtcweb-datagram] The basic approach is to use ICE as a setup mechanism, and to specify mechanisms to use ICE over connections that utilize UDP and TCP if needed to support a basic datagram-passing function with adequate security. In order to deal with complex NAT/firewall situations, relaying using TURN MUST be supported.

For octet-stream transport, TCP is used. (QUESTION: Do we need a TCP relay specification? The use of TURN over TCP and TLS is specified in the TURN RFC - is it suitable?)

(The role of Web Sockets [I-D.ietf-hybi-thewebsocketprotocol] needs to be clarified.)

The data transport MUST behave reasonably in the presence of congested networks; this is usually interpreted as reducing the send rate when congestion is encountered. TCP, when correctly implemented, does this automatically; this is not the case with UDP, and the RTP framing specification does not contain a congestion control component.

Determining an useful congestion handling mechanism is a high priority for work with this specification suite.

5. Data framing and securing

RTP [RFC3550]and SRTP [RFC3711]. The RTP/SAVP profile, defined as part of SRTP, is supported, and "extended RTCP", RTP/SAVPF [RFC4585], with its secured version RTP/SAVPF [RFC5124]is used in order to support codec functionality that depends on this RTP profile, such as

The implementation of SRTP used MUST support encryption using AES-CM with MIC, on both RTP and RTCP channels. <TODO: Add pointer to appropriate profile here> (Note that like for all mandatory-to-implement, there is no requirement that these protocols be used, just that it is possible to negotiate them.)

[OPEN ISSUE; We need to specify a securable format of passing data that is not RTP. This should probably be a profile on using TLS and/or DTLS, although specifying a "data codec" and using SRTP has been proposed too.]

6. Data formats

The intent of this specification is to allow each communications event to use the data formats that are best suited for that

particular instance, where a format is supported by both sides of the connection. However, a minimum standard is greatly helpful in order to ensure that communication can be achieved. This document specifies a minimum baseline that will be supported by all implementations of this specification, and leaves further codecs to be included at the will of the implementor.

NOTE IN DRAFT: The particular codecs named are NOT A DECISION. They are included to illustrate possible choices, and to check with the group that the references given are necessary and sufficient for the purpose of specifying an interoperable codec suite.

In audio, the OPUS codec[I-D.ietf-codec-opus] MUST be supported. For ease of interoperability with gateways to older equipment, G.711 U-law, audio/PCMU, defined in RFC 1890 [RFC1890] section 4.4.12, is also mandatory to implement. There is no third mandatory to implement.

In video, the VP8 codec [I-D.westin-payload-vp8] MUST be supported.

The Theora codec is also freely available. H.264/AVC and H.264/SVC [I-D.ietf-avt-rtp-svc] are widely enough used that it gives a wider range of communications partners if they are supported.

7. Connection management

This specification is silent on the definition of connection management protocols. It envisions that implementors will make a choice on whether to implement connection management protocols as a downloadable component, as a browser plug-in, or as a frontend/ backend split, where a part of the protocol machinery is downloaded into the browser and uses some mechanism (for instance WebSockets) to communicate back to a backend implementing the rest of the connection management protocol.

XMPP, and its Jingle component, has proved a versatile tool in building interoperable communities, and so has SIP. This suite requires that the browser support establishing and describing connections using a data format capable of representing the information needed by these two protocols, such as one that can be one-to-one transformed into SDP. The exact specification of this API is done elsewhere <insert reference when available>; this API is powerful enough that all interesting parameters of the transport mechanisms specified above are settable, and clear enough that how to connect the API to the protocols is obvious.

8. Presentation and control

The most important part of control is the user's control over the browser's interaction with input/output devices and communications channels. It is important that the user have some way of figuring out where his audio, video or texting is being sent, for what purported reason, and what guarantees are made by the parties that form part of this control channel. This is largely a local function between the browser, the underlying operating system and the user interface; this is being worked on in <insert reference here when available>.

9. Local system support functions

These are characterized by the fact that the quality of these functions strongly influences the user experience, but the exact algorithm does not need coordination. In some cases (for instance echo cancellation, as described below), the overall system definition may need to specify that the overall system needs to have some characteristics for which these facilities are useful, without requiring them to be implemented a certain way.

Local functions include echo cancellation, volume control, camera management including focus, zoom, pan/tilt controls (if available), and more.

Certain parts of the system SHOULD conform to certain properties, for instance:

- Echo cancellation should be good enough that feedback (defined as a rising volume of sound with no local sound input) does not occur.
- Privacy concerns must be satisfied; for instance, if remote control of camera is offered, the APIs should be available to let the local participant to figure out who's controlling the camera, and possibly decide to revoke the permission for camera usage.
- Automatic gain control, if present, should normalize a speaking voice into <whatever dB metrics makes sense here - most important that we have one only>

10. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

11. Security Considerations

Security of the web-enabled real time communications comes in several pieces:

- Security of the components: The browsers, and other servers involved. The most target-rich environment here is probably the browser; the aim here should be that the introduction of these components introduces no additional vulnerability.
- Security of the communication channels: It should be easy for a participant to reassure himself of the security of his communication - by verifying the crypto parameters of the links he himself participates in, and to get reassurances from the other parties to the communication that they promise that appropriate measures are taken.
- Security of the partners' identity: verifying that the participants are who they say they are (when positivie identification is appropriate), or that their identity cannot be uncovered (when anonymity is a goal of the application).

This specification addresses some, but not all, of these concerns, and makes some assumptions about the security considerations of other parts of the environment; it is up to the implementor to see that these security assumptions are warranted. In particular:

- We assume that the ICE security mechanism is a necessary and sufficient criterion for accepting that a connection attempt is from a communications partner. This means that we trust the randomness of ICE "usernames" and the security of ICE "passwords".
- We assume that the SRTP key exchange mechanisms and security profiles specified provide an adequate level of protection for audio and video media.

(there needs to be more text here)

12. Acknowledgements

13. References

13.1. Normative References

[I-D.alvestrand-dispatch-rtcweb-datagram]
Alvestrand, H., "A Datagram Transport for the RTC-Web profile", draft-alvestrand-dispatch-rtcweb-datagram-01 (work in progress), February 2011.

[I-D.ietf-codec-opus]
Valin, J. and K. Vos, "Definition of the Opus Audio Codec", draft-ietf-codec-opus-04 (work in progress), March 2011.

[I-D.ietf-hybi-thewebsocketprotocol]
Fette, I., "The WebSocket protocol", draft-ietf-hybi-thewebsocketprotocol-06 (work in progress), February 2011.

[I-D.westin-payload-vp8]
Westin, P. and H. Lundin, "Proposal for the IETF on "RTP Payload Format for VP8 Video"", draft-westin-payload-vp8-02 (work in progress), March 2011.

[RFC1890] Schulzrinne, H., "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

13.2. Informative References

[I-D.ietf-avt-rtp-svc]
Wenger, S., Wang, Y., Schierl, T., and A. Eleftheriadis, "RTP Payload Format for Scalable Video Coding", draft-ietf-avt-rtp-svc-27 (work in progress), February 2011.

[RFC3935] Alvestrand, H., "A Mission Statement for the IETF", BCP 95, RFC 3935, October 2004.

Author's Address

Harald T. Alvestrand
Google
Kungsbron 2
Stockholm, 11122
Sweden

Email: harald@alvestrand.no

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2011

C. Jennings
Cisco

March 7, 2011

Architecture and API Requirements for RTC Web
draft-jennings-rtcweb-api-00

Abstract

Internet browsers and other software applications are enabling support for real time interactive voice and video. This draft outlines a set of IETF protocols that can be used for this purpose and describes the overall architecture. It also identifies the requirements for an application programming interface to control these protocols.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 8, 2011.

Copyright Notice

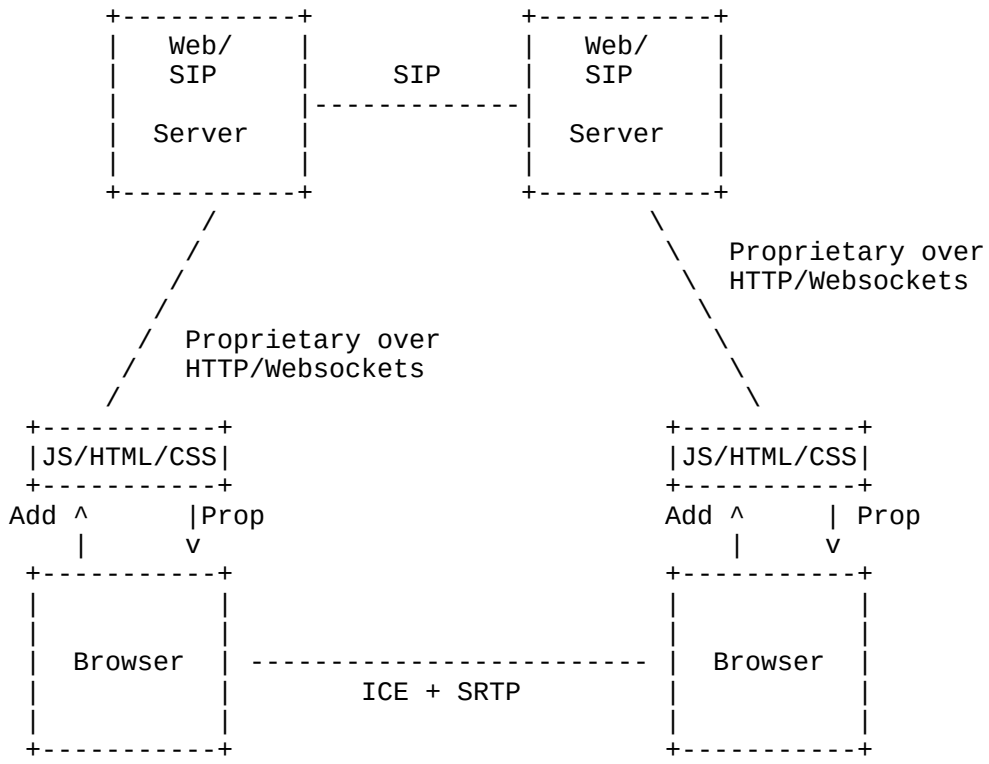
Table of Contents

1. Overview

This draft describes two models of how this would work, which are referred to as the advertisement proposal (AdProp) model and the offer answer (OffAns) model. Both of these models are useful in various situations, and they involve very similar code development efforts. This draft proposes an API and protocol set standardization that supports both models.

1.1. Advertisement Proposal Model

The AdProp model standardizes a way to send media between two browsers and standardizes an API in the browser, such that browser-based applications can find out the media capabilities of the browser and can tell the browser what media streams to send and receive. We use the term "browser app" to refer to a program that is running in the browser and using HTML, CSS, and JavaScript to control the browser. It is assumed that the browser app could communicate with the web server using existing approaches, and that the web server communicates with a SIP server as a way of federating to other websites or connecting to legacy VoIP systems. There are many different ways this model could be used, but the diagram below covers a fairly complex case that most other cases end up being a subset of. More use cases are discussed in section XXX.



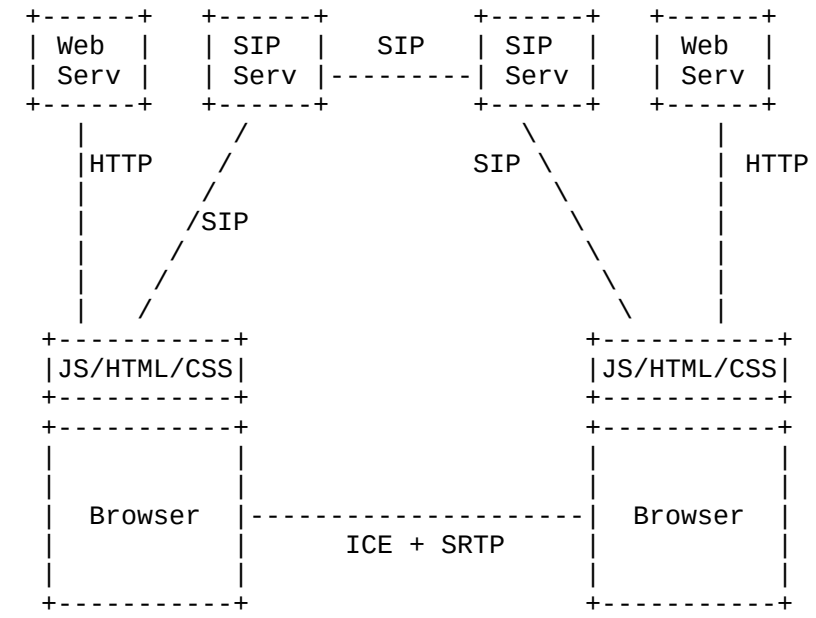
The API for this model has two distinct phases. First there is a Connection API that allows the browser app to use ICE to form a connection to the other browsers. This API assumes that the browser applications will be able to exchange ICE candidates lists by some out-of-band means -- most likely involving passing them up to the web servers over HTTP. The second stage is referred to as the AVT API. This API allows the browser apps to discover which codecs and capabilities the browser supports. It then allows the browser app to control which media streams the browser will send and receive. The browser describes its range of capabilities in an advertisement object. The browser app requests that a particular set of media streams be set up in a proposal to the browser. This is done as an atomic request which is either accepted or not. Partial acceptance has proven to be very difficult to deal with in the implementation of existing systems. The general overview and advantage of the AdProp model is discussed in draft-peterson-sipcore-advprop [I-D.peterson-sipcore-advprop].

The model above shows SIP as the protocol between the two web servers, but the API proposed would also work using Jingle or H.323 as the federation signaling protocol. It would also be possible to

implement the processing of SIP messages in the JavaScript in the browser application and then somehow tunnel the SIP messaging between the clients. XMPP over websockets has been proposed for this. The architecture and API in this draft would support all of these possibilities.

1.2. Offer Answer Model

The OffAns model standardizes a way to send media between the browsers, but it also selects an existing signaling protocol to negotiate and set up the media. The browser app would indicate to the browser that it wished to form a communication session with another entity, and then the browser would take care of the rest. A typical model for this is show below.

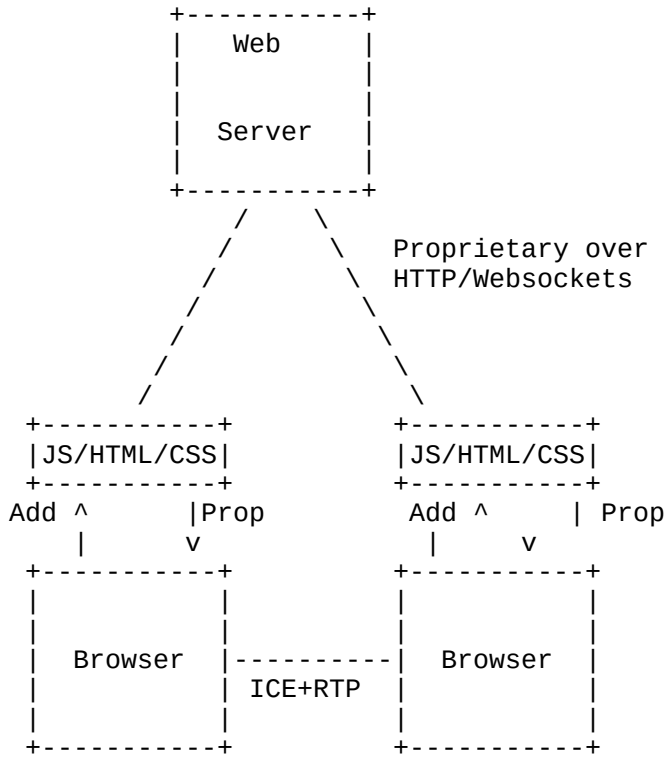


The major goal for this API is to be extremely simple to use in enabling a website for voice and video. On an iPhone today, one can simply put a tel URL on the web page and the iPhone can call it. That is a simple approach that web developers like and use. Since standards are involved, this proposal will have to be more complex. The API defines an HTML session element that can be used like a source element inside of an audio or video element. It also provides a JavaScript API to control the session and replace the user interface.

1.3. Use Cases

1.3.1. Facebook

Consider the case of a social networking site that allows IM between users and wants to also allow voice and video between them, but does not need to federate with others. The case could easily use the AdProp model. Assuming that it was only supported on browsers meeting a certain minimum functionality and it always uses the same capabilities, there is no need to even negotiate or share the advertisements between the two browsers. The browser app simply sets up the connection to the far end, and then uses a proposal for the media steam that is always the same.

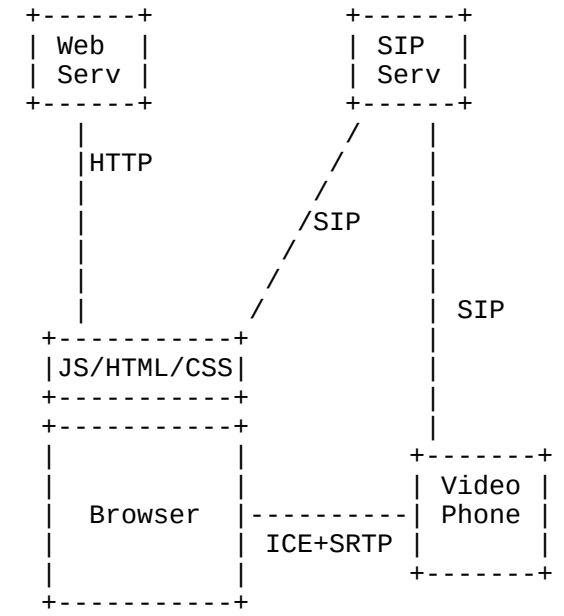


1.3.2. Webex

TBD

1.3.3. Amazon

Consider the case of a website that supports searching and displays advertisements related to the search. In this case clicking on the advertisement could directly connect the user with a sales agent at the company associated with the advertisement.



In this sort of case the people operating the web server do not need to deploy anything special to display the advertisement, and the company associated with the advertisement can use its existing call center, assuming it meets the legacy VoIP requirements outlined in section XXX.

The security issue of a browser sending a SIP packet to a device that does not meet the same origin policy is discussed in the section XXX, but the brief preview of the solution is that the SIP messages can use CORS REF much like a HTTP does.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Requirements

The section defines the set of protocols and selected subset profiles of these protocols that a browser would need to implement, and forms the requirements for the API to control these protocols. At a high level we split this into connection management, transports for real time media such as audio and video, transports for non media data, codecs support, and signaling protocols.

All of the data plane sessions are set up using ICE [REF] or ICE-Lite for security reasons, discussed in section XXX. Devices that could be deployed behind NATs, such as a web browser, are REQUIRED to support ICE while other devices that always deploy on public addresses can do ICE-Lite. The only mode of ICE REQUIRED is aggressive. Real time media is transported over RTP REF or SRTP [REF]. Support for multicast RTP is OPTIONAL. To support ICE, implementation needs to be able to do STUN REF and TURN REF. In addition, there is a strong interest to define a TURN-like protocol that looks like HTTP to intermediaries, so that media can be tunneled over HTTP. Support RTCPMUX REF is REQUIRED. RTP keep alive is done using RTCP as described in REF. The API needs to allow the DSCP REF for each RTP or media stream to be set. The API needs to allow the browser app to observer and control the SSRC values in the RTP.

Open Issue: There is a desire to be able to pass non media type data directly between browsers. For example, an application such as Second Life or gaming application may wish to pass small chunks of data such as player position with stringent real time requirements. There are several proposals for how to do this. The session would be set up using ICE, just as with RTP. One proposal is just to use a thin shim on top of UDP or DTLS to demux the packets from other packets such as RTP on the same connection. Another proposal is DTLS over DCCP over UDP with some appropriate congestion control scheme chosen for DCCP. Another proposal is to define a data codec to carry the data in RTP.

The mandatory to implement audio codecs are: PCMA, PCMU, telephone-event, and opus [REF]. The API needs to support the following OPTIONAL codecs: G729, G722, G7221, G723, AMR, AMR-WB, iLBC, L16 and opus. PCMU and PMCA codecs are REQUIRED to support 1 channel with a rate of 8000 and a ptime of 20. The mandatory to implement video codecs are: <to be chosen by working group - leading candidates for consideration are H.264-AVC and VP8>. The minimum profile and resolutions supported by the mandatory to implement video codecs are TBD. The API needs to support the following OPTIONAL codecs: H263-2000, H264, H264-SVC, raw and VP8.

The signaling protocol selected here is SIP though very little

overall architecture would change if the WG decided to use Jingle REF instead of SIP. The browser needs to implement the subset of SIP REF3261,3263,3264 and is required to support registration, invite, ack, cancel, bye, and update. Support for the following features is OPTIONAL: INVITES without an offer, re-invite, forking, S/MIME and sips. Support for the following is REQUIRED: sip over TLS, outbound proxy, 3xx redirects, early media, multipart mine REF 5621, update, identity 4916 & 4471, rport REF 3581, SIP keep alive as described in 5626.

Open Issue: define a TURN like protocol to tunnel RTP over HTTP

Open Issues: define a RTP mux protocol to multiplex RTP on top of a single UDP port. Would likely use SSRC as the demux code point.

Open Issue: Mandatory to implement video codec(s) and minimum profile.

Open Issue: Mandatory to implement audio codecs.

4. Connection API

It is expected this section will be removed from this draft and moved to a W3C draft but it is provided for reference at this point. The straw man API are many things including adequate error handling. The API would likely end up using exceptions for many things.

4.1. Session API

The session element can be used anywhere in HTML that the source element could be used. Fundamentally, this is an alternative way of setting up a source for an audio or video element.

Categories: None

Contexts in which this element can be used: same as source element

Content model: Empty

Content attributes:

src: URL to destination to create session with.

aor: Address of Record that identifies this user.

credential: Password or credential for the specified AOR.

proxy: URL for outbound proxy.

DOM interface:

```
interface Session : HTMLElement {

  attribute double volume; // control speaker volume
  attribute boolean mute; // control microphone
  attribute boolean sendVideo; // control camera

  attribute DOMObject videoPane;
  attribute DOMString aorUrl;
  attribute DOMString credential;
  attribute DOMString outboundProxyURL;
  readonly attribute DOMString remoteName;
  readonly attribute boolean secure;

  readonly attribute DOMString registrationState;
  // noRegistrar, registering, registered, registrationFailed
  attribute Function onRegisterStateChange;

  void open( in DOMString url ); // tel or SIP URL
  void close();
  void accept( boolean accept );

  readonly attribute DOMString sessionState;
  // noSession, openingSession, acceptingSession,
  // inSession, closingSession
  attribute Function onSessionStateChange;

  boolean sendKeyPress( in DOMString key ); // send DTMF or KPML
  attribute Function onReceiveKeyPress;
};
```

If the session will be able to display video, the DOM object for a video tag must be provided in the videoPane parameter of the constructor. If an aorUrl is provided, the session will attempt to register for incoming calls at the server using the provided credentials. If an outbound proxy is provided, all signaling for this session will use that proxy. The progress of the registration can be tracked with the onRegisterStateChange callback. The registrationState attribute will be a string with one of the following values: noRegistrar, registering, registered, or registrationFailed.

Open Issue: need to decide how to handle credentials and if they will be in the JavaScript. Similar issues for TURN server credentials.

To make a call, the open session method is called and the session state will change to "opening session".

Events:

Exceptions:

4.1.1. Session Example Incoming

The following HTML snippet would display a video pane with a user interface such that when the user clicked, it would create an audio video session by making a SIP call to "sales@example.com".

```
<video width=’320’ height=’240’ >
  <session src="sip:sales@example.com" >
</video>
```

4.1.2. Session Example Outgoing

The following HTML snippet would register to receive calls to the address "sip:fluffy@example.com". Furthermore it would use an outbound SIP proxy at sip.example.com.

```
<video width=’320’ height=’240’ >
  <session  aor="sip:fluffy@example.com"
            credential="password"
            proxy="sip:sip.example.com" >
</video>
```

4.2. Connection API

```
[NoInterfaceObject]
interface IceCandidate {
 DOMString foundation;
 unsigned short component-id; // always 1 ?
 DOMString transport; // udp
 unsigned long priority;
 DOMString type; // host, srflx, prflx, relay
 DOMString addressFamily; // v4 v6
 DOMString connectionAddress; // v4 or v6 ip address
 unsigned short port;
};
```

```
   [NoInterfaceObject]
   interface IceCandidateList {
    IceCandidate  candiate[];
    DOMString icePassword;
    DOMString iceUFragment;
   };

   [NoInterfaceObject]
   interface RelayServer {
     DOMString type; // stun turn
   };

   [NoInterfaceObject]
   interface StunServer : RelayServer {
     DOMString ip;
   };

   [NoInterfaceObject]
   interface TurnServer : RelayServer {
     DOMString ip;
     DOMString username;
     DOMString password;
   };

   [Constructor(in optional RelayServer relayServers[])]
   interface Connection {
     attribute int keepAlivetime; // default 30 seconds

     attribute RelayServer relayServers[]

     readonly attribute IceCandidateList candidateList;

     readonly attribute IceCandidate connectionNearEnd;
     readonly attribute IceCandidate connectionFarEnd;

     void open( IceCandidateList addressList );

     readonly attribute DOMString state;
     // creating,ready,connecting,open,closed

     attribute Function onready;
     attribute Function onopen;

     void send(in DOMString data);
     attribute Function onmessage; // implements MessageEvent interface
     attribute Function onerror;

     void close();
```

```
  attribute Function onclose;
};
```

The general usage for a browser that had a stun server at 192.0.2.1 would be to create a connection, wait for ICE to gather candidates and the state to change to ready, then send the ICE candidates list to the far side as shown in the following code.

Open Issue: Need to add more into this so that an application can understand what is going on and get information to provide status and debug problems as well as statistics. Also may need parameters to change the algorithm.

```
myConn = new Connection( [ {type:"stun",ip:"192.0.2.1"} ] );
myConn.onready = function() {
  myCandidates = myConn.candidateList;
  // send myCandidates to far side
}
```

Open issue: add text around setter calling function if in that state when set.

Later when the far side has sent its candidate list to this side, the browser app calls open to start opening the connection to the other side. Once the connection is open, the browser app can start sending and receiving data.

```
myConn.open( farSideCandidateList );
myConn.onOpen = function() {
  // can start sending data for far side
  myConn.send( "Hello" );
}
myConn.onmessage = function(e) {
  alert "Received data:" + e.data;
}
```

4.3. Audio Video API

Note this section is far from complete and is more just a sketch to get the flavor of the interface.

```
interface Advertisement {
  CodecAd codecs[];
  boolean rtcpMux; // default true
  boolean rtpMux; // default true
  boolean srtp; // default true
  DOMString protocols[];
  // RTP/AVP, RTP/AVPF, UDP/TLS/RTP/SAVP, UDP/TLS/RTP/SAPF
```

```
    srtpSuites[]; // AES_CM_128_HMAC_SHA1_32
  };

  interface CodecAd{
    string mediaType;
    int clockRate;
    float minBandwidth; // kbps
    float maxBandwidth; // kbps
    boolean canReceive;
    boolean canSend;
    boolean supportDscp;
  };

  interface TelEventDataCodecAd {
    int supportCodes[]; // defaults to 0-11 if not present
  };

  interface AudioCodecAd : CodecAd {
   int maxPacketTime; // ms
  };

  interface IlbcAudioCodecAd : AudioCodecAd {
   int modeList [];
  }

  interface G729AudioCodecAd : AudioCodecAd {
    boolean vadSupported;
  };

  interface G711uAudioCodecAd : AudioCodecAd {
    // G.711 PCMU must be 1 channel at rate of 8000
  };

  interface G711aAudioCodecAd : AudioCodecAd {
    // G.711 PCMA must be 1 channel at rate of 8000
  };

  interface L16AudioCodecAd : AudioCodecAd {
    int rates[];
    int channels[];
    DOMString emphasis[];
    DOMString channel-order[];
  };

  interface AMRAudioCodecAd : AudioCodecAd {
   DOMString modeSet;
   // bunch more needed here
  };
```

```
   interface VideoCodecAd : CodecAd {
    float maxFramerate; // fps
    int clockRate;
    int minXsize; int maxXsize;
    int minYsize; int maxYsixe;
    float minPar; float maxPar; float parList[];
    float minSar; float maxSar; float sarList[];
   };

   interface VP8CodecAd : VideoCodecAd {
      int versions[];
   };

   interface H264CodecAd : VideoCodecAd {
     unsigned short profile-levls[];
     unsigned short max-recv-level;
     int max-mbps;
     int max-smbps;
     int max-fs;
     int max-cpb;
     int max-dpb;
     int max-br;
     boolean redundant-pic-cap;
     DOMString sprop-parameter-sets;
     DOMString sprop-level-parameter-sets;
     boolean use-level-src-parameter-sets;
     boolean in-band-parameter-sets;
     boolean level-asymmetry-allowed;
     int packetization-modes[];
     int sprop-interleaving-depth;
     int sprop-deint-buf-req;
     long deint-buf-cap;
     int sprop-init-buf-time;
    // int sprop-init-buf-time;
     long max-rcmd-nalu-size;
     int sar-understood;
     int sar-supported;
   };


   interface Proposal {
     StreamProp streams[];
   };


   interface StreamProp {
     string mediaType;
     int clockRate;
```

```
    float minBandwidth; // kbps
    float maxBandwidth; // kbps
    boolean canReceive; // default true
    boolean canSend; // default true

    DOMString fingerprint; // RFC4572
    int pTime;
    DOMString protocol;
    // RTP/AVP, RTP/AVPF, UDP/TLS/RTP/SAVP, UDP/TLS/RTP/SAPF
    long ssrc;
    int dscp;
    DOMString srtpSuites;
    int srtpKdr;
    boolean srtpUnencryptedRtcp;
    boolean srtpUnauthenticated;
    DOMString srtpFecOrder; //FEC_SRTP, "SRTP_FEC"
    int srtpLifetime; // log base 2 of max packets with one kety
    DOMString srtpKeys[];
    int srtpMki[]; // MKI corresponding to srtpKeys at same index
  };

  interface VideoProp : StreamProp {
    int sizex;
    int sizey;
    float sar;
    float frameRate;
  };

  interface AudioProp : StreamProp {
    int pTime; // ms
  };

  interface Stats {
     StreamStats steam[];
  };

  interface StreamStats {
    // TODO RTCP stats
  };

  interface AVT {
    attribute Connection connection;
    readonly attribute Advertisement advertisement;
    readonly attribute Advertisement advertisementNoVideo;

    attribute DOMObject camera;
    attribute DOMObject mic;
    attribute HTMLVideoElement videoPane;
```

```
    readonly attribute Stats stats;

    readonly attribute Proposal proposal;
    boolean setProposal( Proposal newProp );
  };
```

Using this interface is fairly simple. First an AVT object is loaded and bound to an existing Connection object. It is also bound to cameras, microphones, and speakers, Then the current advertisement can be retried.

Open Issue: The SRTP keying should not be per stream.

```
var myAvt = org.w3c.device.load("device", "AVT", "1");

myAvt.connection = myConn; // the ICE formed connection
myAvt.camera = org.w3c.device.load("device", "camera", "1");
myAvt.mic = org.w3c.device.load("device", "mic", "1");
myAVT.videoPane = document.getElementById("myVideo");

mdAdv = myAvt.advertisement;
```

Open Issues: What's the best way to get an AVT object? How to get the other devices and wire them up to the AVT object?

Assume that the browser supports VP8 video at 720P and G.711. The myAvt object might look like:

```
{
   "codecs" :  [
      {
          "mediaType" : "PCMU",
          "clockRate": "8000",
          "maxPacketTime" : "60"
      },
      {
          "mediaType" : "PCMA",
          "clockRate": "8000",
          "maxPacketTime" : "60"
      },
      {
            "mediaType" : "VP8",
            "clockRate" : "90000",
            "maxXsize" : ""1440,
            "maxYsize" : "720",
            "parList" :  [ "1.0" ],
            "versions" : [ "1" ]
       } ],
   "protocols" : ["RTP/AVP", "RTP/AVPF" ]
};
```

Then, based on some knowledge about what the far end browser supports, the system would decide that it wants to use PCMU with VP8 at a QCIF resolution and 15fps. After forming a connection to the far end and waiting for the connection object to be in the ready state, it would construct the following proposal object and then send that proposal to the AVT systems as shown in the code below. Assuming the proposal is acceptable, the setProposal returns true and (returns false if it is not).

```
var proposal = {
  "streams" : [
      {
        "mediaType" : "VP8",
        "clockRate: : "90000",
        "protcol" : "RTP/AVP",
        "sizex" : "176",
        "sizey" : "144",
        "sar" : "1.0",
        "frameRate" : "15",
        "version" : "1"
      },
      {
        "mediaType" : "PCMU",
        "clockRate: : "8000",
        "pTime" : "20",
        "protcol" : "RTP/AVP"
      } ]
};


if ( myAvt.setProposal( proposal ) ) {
   // it worked
}
```

5. IANA Considerations

This document does not require any action of IANA.

6. Security Considerations

6.1. Attack Model

This architecture involves all the normal security consideration and attack models of HTTP, SIP and RTP but introduces yet another key issue. The assumption is that a user may browse to the attacker's website. The other assumption is that the browser is behind a firewall, and inside that firewall there are devices that would not have appropriate security models for the internet. For example, there could be SIP gateways that if sent an invite to call a 1-900 number would do so with no authentication or authorization. Whatever HTML/CS/Javascript is downloaded must not be able to send arbitrary packets to hosts behind the firewall or send SIP or RTP to devices that do not consent to communicate with the browser.

6.2. Media Security

The browser MUST enforce the constraint that no RTP or other media is sent to a given destination unless that destination completes an ICE connectivity check and proves it knows the secret generated by the browser. The browser must keep a list of locations it has attempted to contact with ICE in the previous 30 seconds and not contact any locations that have previously failed.

6.3. Signaling Security

The browser stops unwanted SIP signaling by using CORS REF. The same CORS headers used for HTTP will be added to the SIP signaling. Before the browser sends SIP signaling, it will preflight the SIP messaging using a SIP OPTIONS message. This is done the same ways CORS can preflight check an HTTP request.

7. Legacy VoIP Interoperability

There is no way to meet all the security requirements and maintain comparability with all legacy VoIP equipment. This draft tries to minimize the impedance mismatch. The requirements here would allow interoperability with legacy VoIP equipment as long as that equipment either directly supported, or was fronted by an SBC that supported, the following: SIP CORS extension, ICE or ICE-Lite, codecs from the mandatory to implement set, supported SIP invites containing an offer, and supported DTMF over RTP with telephone events.

A substantial fraction of VoIP equipment does all of this except for the CORS extensions. The item most commonly lacking is ICE-Lite but that is becoming increasingly prevalent, particularly on devices designed to sit on the edge of a domain and connect to remote UAs that may be behind NATs. For an edge device that was willing to receive SIP call from others, implementing the CORS is pretty trivial. When the UAS receives a SIP options request with an Origin header, it checks whether the header field value is on the white list, and if it is then the UAS copies the value to the Access-Control-Allow-Origin header field value in the response. For many situations the white list would be everything, while for others it would be just the list of websites that are expected to originate calls to this SIP device.

8. Acknowledgement

Thanks to Joe Hildebrand, Matt Miller, Matthew Kaufman, Eric Rescorla and Lyndsay Campbell for their review, comments and contributed

ideas.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

[I-D.peterson-sipcore-advprop]
Peterson, J. and C. Jennings, "The Advertisement/Proposal Model of Session Description", draft-peterson-sipcore-advprop-00 (work in progress), February 2010.

Author's Address

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 13, 2011

X. Marjou
JF. Jestin
France Telecom Orange
February 9, 2011

Requirements for interworking between RTC-Web and SIP-RTP protocols
draft-marjou-dispatch-rtcweb-sip-rtp-interwk-reqs-00

Abstract

In the context of [RTC-Web], some work is emerging to make real-time communications possible in a web browser. This document defines a minimal set of requirements so that such applications interoperate with SIP-RTP applications.

Table of Contents

1. Introduction

In the context of [RTC-Web], some work is emerging to make real-time communications possible in a web browser. Such work will allow to use the UDP protocol to transport real time data.

This document defines a minimal set of requirements so that RTC-Web applications interoperate with applications based on SIP ([RFC3261]) and RTP ([RFC3550]) protocols.

On the one hand, bringing real-time communication capability in the web browser RTC-Web promises to offer great value to end-users, developers and service providers. This value comes from the ubiquity of the web browser and the web architecture, from the simple programatic model the web offer and indeed the innovation perspective of such solution.

On the other hand, SIP and RTP protocols are broadly used to implement real-time or near real-time applications. This is particularly true for voice, video, instant messaging, presence and content sharing and when considering available implementations in devices (hard phone, mobile phone...), network infrastructures (e.g. SIP based architecture) and service provider interconnection gateways.

Allowing both solutions to interoperate promises RTC-Web solution to have greater value as it will allow this solution to reach legacy SIP multimedia devices and networks and vice versa.

Section 4 describes some use cases. Section 5 reports the different possible architectures. Section 6 finally states a set of requirements the ongoing RTC-Web solution definition should fulfil to be able to interoperate with SIP based multimedia applications.

Note: This document does not directly address RTC-Web service provider interconnection except if this interconnection is based on SIP-RTP.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [RFC2119].

3. Definitions

RTC-WEB: the term [RTC-Web] refers to the ongoing work on defining a solution that enables real time applications such as bidirectional audio and video within web applications

SIP-RTP: the term SIP-RTP is a generic term which refers to SIP and RTP ecosystem protocol stack. This includes, non exhaustively: SIP, SDP, RTP, RTCP, SRTP...

Note: To be adjusted to fit with the current on going [RTC-Web] charter.

4. Use cases

This section presents some use cases involving interworking between RTC-Web and SIP applications. These use cases include scenarios where real-time audio and/or video are exchanged.

4.1. SIP Multimedia application reachability extension

Alice wants to access its services. Her service provider A (e.g. atlanta.example.org) hosts these services on SIP-RTP servers. Alice can use a web browser implementing an RTC-Web extension to reach its service.

4.2. RTC-Web applications with integration of SIP device

Bob wants to access its services. His service provider B (e.g. biloxy.example.org) hosts these services on RTC-Web servers. Bob can use a device implementing an SIP-RTP extension to reach its service.

4.3. RTC-Web and SIP service provider interconnection

All the users of service provider A (e.g. atlanta.example.org) use an RTC-Web application. All the users of service provider B (e.g. biloxi.example.org) use a SIP-RTP application. Both service providers want to make communications possible between all these users.

This use case is typically an inter service operators use case.

5. Possible RTC-Web/SIP interworking architectures

This section outlines different architectures to realize RTC-Web/ SIP-RTP interworking. This section does not pretend to be exhaustive

in term of architecture description but intends to propose families of models any kind of solution should fit in.

These architectures satisfy the use cases listed above. However, It must be noted that depending on the considered use case, additional components may be necessary.

In this section, the name SIP used alone is a shortcut for SIP and SDP protocols. Similarly, RTP used alone is a shortcut for RTP, RTCP, and SRTP protocols.

5.1. SIP-RTP Stack in the Browser

This architecture consists in directly implementing a SIP-RTP protocol stack in the browser, enabling a direct connection between an RTC-Web application in a browser and a SIP-RTP phone.

Architecture with SIP-RTP in the browser:

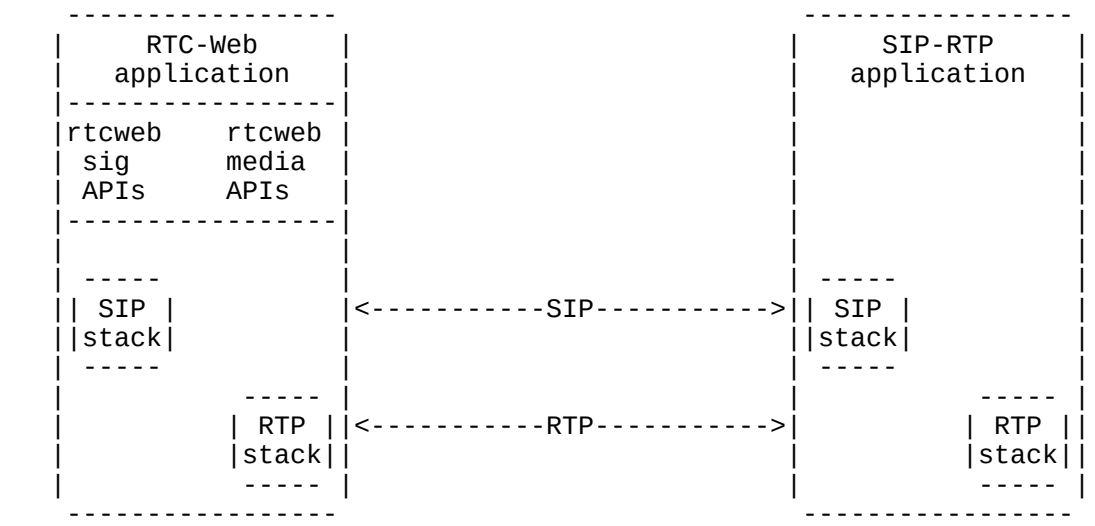


Figure 1

5.2. New Signalling Scheme and RTP Stack in the Browser

This architecture consists in implementing a new signalling scheme and an RTP stack in the browser.

A new signalling scheme means refers to two possible models:

- o Session management data set by API and transported by an application protocol (e.g. HTTP or WebSockets). Figure 2 illustrates such architecture with XXX as the session management data. The HTTP stack shown in the figure is the regular HTTP stack available by default in all web browsers. Having SIP (or part of it) embedded in HTTP in one possible implementation, as indicated in [draft-sinnreich-sip-web-apis-01].
- o A session management protocol different from SIP (e.g. XMPP, MEGACO). Figure 3 illustrates such architecture with YYY as the signalling protocol.

Both models relax constraints on the technology choice to implement the RTC-Web solution but add constraints on end-to-end compatibility with SIP-RTP applications by requiring the implementation of a gateway to map one protocol into another one.

Architecture with HTTP in the browser:

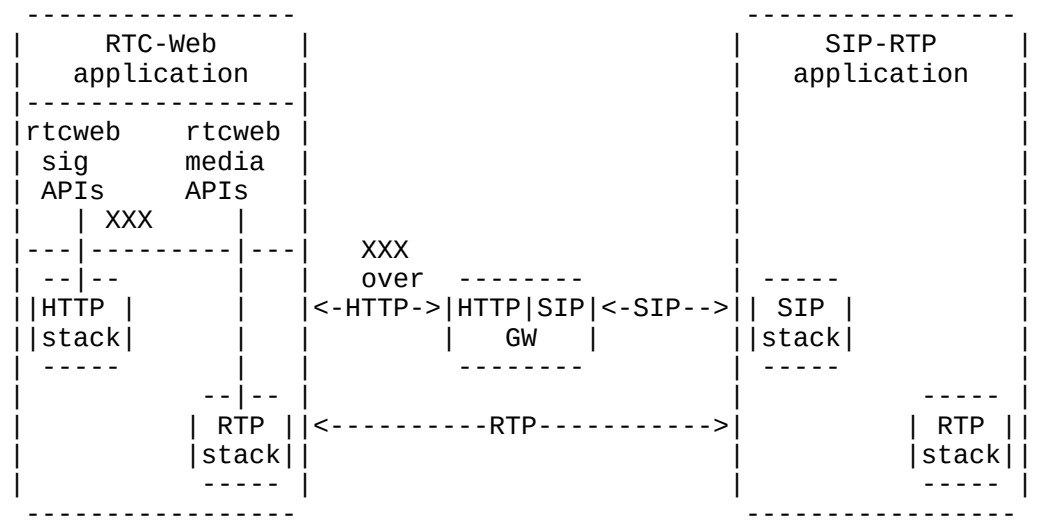


Figure 2

Architecture with another protocol than SIP or HTTP in the browser:

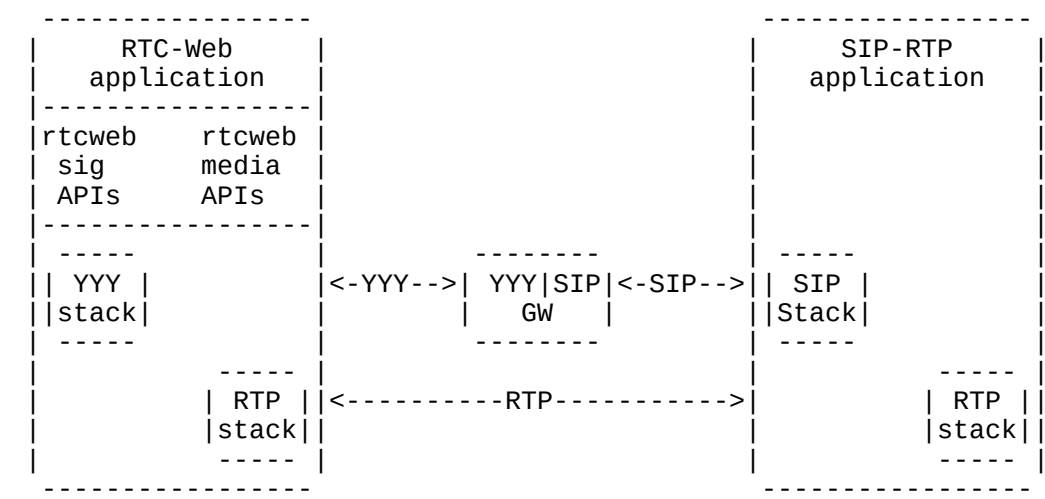


Figure 3

5.3. New Signalling Scheme and New Media Protocol in the Browser

This architecture consists in implementing different protocols in RTC-Web and SIP-RTP frameworks, both for at the signalling level and at the media level.

Such architecture requires interworking work (protocol mapping, gateway) both for the signalling and the media protocols.

This architecture relaxes constraints on the technology choice to implement the RTC-Web solution but adds constraints on end-to-end compatibility with SIP-RTP applications by requiring the implementation of gateway(s) to adapt protocols and media payloads.

Architecture with another protocol than RTP as a media protocol:

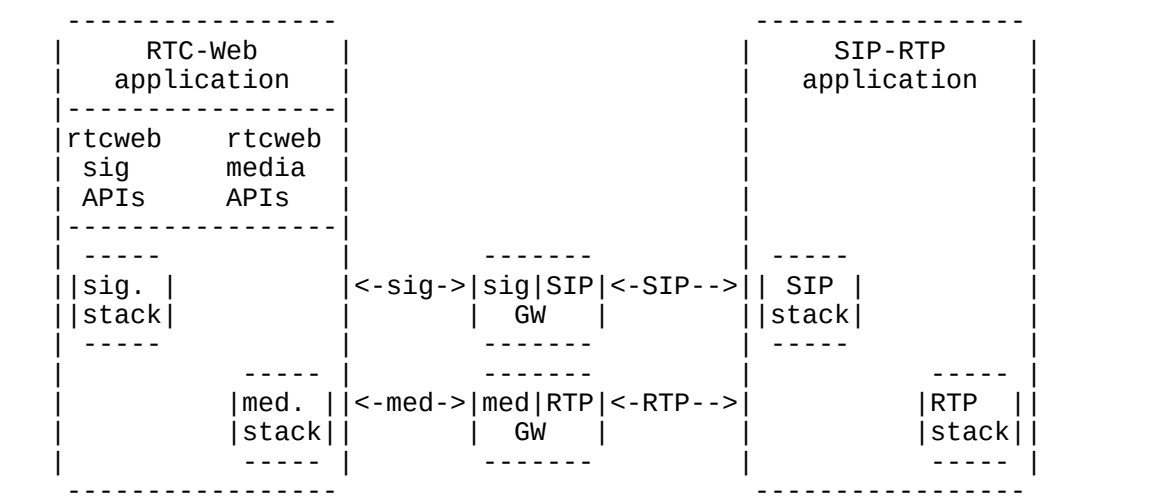


Figure 4

5.4. Analysis with regards to interworking

Using a full SIP-RTP stack in the browser (Section 5.1) would undoubtedly be the best solution with regards to interworking: it would avoid specifying new protocols and it would thus avoid the control plane interworking problem described in [RFC3439] (i.e. no need for protocol mapping). It nevertheless requires a granular API to configure and access the protocol stack.

Using the RTP protocol suite but another than the SIP protocol (Section 5.2) add the burden of interworking efforts at the signalling level. The level of complexity of this gateway depends on how much the signaling protocol will look like SIP. However, having HTTP (or WebSocket) as a protocol transporting the signaling data is attractive due to the central role played by this protocol in Web environments.

Using both new signalling and media protocols in the browser (Section 5.3) has been presented above for the sake of exhaustiveness but this solution is not attractive for SIP-RTP interworking: it increases the interworking efforts by requiring work at the media level (new media protocol, complexity and cost of interworking gateways...), whereas adding no identified advantages with regards to the existing RTP/UDP protocol suite.

6. Requirements

Whatever the architecture solution the RTC-Web will retain, a reasonable way-forward is to specify its protocols and APIs taking care of interworking with SIP-RTP devices. As such the following requirements are proposed to the RTC-Web working group:

6.1. Generic requirements:

GENERIC-REQ-1 The [RTC-Web] solution MUST be designed in a such way it allows interworking with SIP-RTP applications both at the signalling and media level.

6.2. Signalling level requirements:

SIG-REQ-1 The [RTC-Web] solution MUST be designed in a way it allows interoperability with SIP based multimedia applications. This is typically applicable for identifiers, credentials, state machine, and message types.

SIG-REQ-2 The [RTC-Web] solution MUST include a way to negotiate media format as in Offer/Answer model used in SIP ([RFC3264])

SIG-REQ-3 The [RTC-Web] solution MUST include a way to interoperate with ([RFC5939])

SIG-REQ-4 The [RTC-Web] solution MUST allow end to end codec negotiation between RTC-web device and SIP device

SIG-REQ-5 The [RTC-Web] solution MUST include a compatibility/ mapping with SDP([RFC4566])

SIG-REQ-6 The [RTC-Web] solution SHOULD NOT require SIP-RTP extensions.

6.3. Media level requirements:

MEDIA-REQ-1 The [RTC-Web] solution MUST be designed in a way it does not mandate a gateway at media level when interworking with SIP based multimedia application, consequently it must be based on RTP/RTCP protocol suite over UDP for real-time media.

MEDIA-REQ-2 The [RTC-Web] solution MUST be compatible with a media gateway architecture and not rely exclusively on a peer to peer (between RTC-Web devices)...

MEDIA-REQ-3 The [RTC-Web] solution MUST allow the configuration of some media-related parameters per session (e.g. buffer size, packetization...).

6.4. Codec level requirements:

CODEC-REQ-1 The [RTC-Web] solution MUST allow codecs available in existing SIP-RTP applications. A non exhaustive list is the following: G.711, G.722, AMR, AMR-WB, H.264.

7. Security Considerations

SEC-REQ-1 RTC-Web and SIP-RTP interworking solution MUST NOT compromise inherent security feature(s) developed and used for both RTC-Web and SIP-RTP solutions.

8. IANA Considerations

None.

9. Acknowledgements

Thank you to Bruno Chatras, Christophe Eyrignoux, and Sebastien Cubaud who provided early feedback.

10. References

10.1. Normative references

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,

A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3960] Camarillo, G. and H. Schulzrinne, "Early Media and Ringing Tone Generation in the Session Initiation Protocol (SIP)", RFC 3960, December 2004.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5939] Andreasen, F., "Session Description Protocol (SDP) Capability Negotiation", RFC 5939, September 2010.

10.2. Informative references

[I-D.sinnreich-sip-web-apis] Sinnreich, H. and A. Johnston, "SIP APIs for Communications on the Web", draft-sinnreich-sip-web-apis-01 (work in progress), June 2010.

[RFC3439] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, December 2002.

[RTC-Web] RTC-Web, "http://rtc-web.alvestrand.com/".

Authors' Addresses

Xavier Marjou
France Telecom Orange
2, avenue Pierre Marzin
Lannion 22307
France

Email: xavier.marjou@orange-ftgroup.com

Jean-Francois Jestin
France Telecom Orange
2, avenue Pierre Marzin
Lannion 22307
France

Email: jeanfrancois.jestin@orange-ftgroup.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 8, 2011

C. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University

March 7, 2011

RTP Requirements for RTC-Web
draft-perkins-rtcweb-rtp-usage-00

Abstract

This document discusses usage of RTP in the context of RTC-WEB work. It discusses important factors of RTP to consider by other parts of the solution, it also discusses which RTP profile to support and which RTP extensions that should be supported.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 8, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document discusses RTP in the context of RTC-WEB. This include RTP's requirement on underlying transport, for example when it comes to provide multiplexing. It discusses which RTP profile that should be supported and what RTP extensions that should be supported. The importance of congestion control and media adaptation is also discussed. This document is intended as a starting point for discussing RTP features in RTC-WEB.

The work in the AVT WG has all been about providing building blocks and not specify who should use which building blocks. Selection of building blocks and functionalities can really only be done in the context of some application(s). RTC-WEB will greatly benefit in interoperability if a reasonable set of RTP functionalities and extensions are selected. For RTC-WEB we have selected RTP extensions that are suitable for a number of applications that fits the context. Thus applications such as VoIP, audio and video conferencing, and on-demand multi-media streaming are considered. Applications that rely on multicast transport has not been considered likely to be applicable to RTC-WEB, thus extensions related to multicast have been excluded.

The document is structured into different topics. For each topic one or several recommendations from the authors are done. When it comes to extensions or need for implementation support we use three levels to indicate the importance for it to be included in an RTC-WEB specification. We see it as likely that everything that is included is in fact mandated to be implemented.

REQUIRED: Absolutely needed functionality to make the solution work well. Also functionality of low complexity that provide high value has also been categorized as required.

RECOMMENDED: Should be included as its brings significant benefit, but the solution can potentially work without it.

OPTIONAL: Something that is useful in some cases, but not always a benefit.

When this documents discusses RTP it always include RTCP unless explicitly stated otherwise. This as RTCP is a fundamental and integral part of the protocol.

2. Requirements from RTP

This section discusses some requirements RTP/RTCP [RFC3550] puts its

underlying transport, the signalling etc.

2.1. RTP Session Multiplexing

RTP has three fundamental points of multiplexing. The first one is the RTP session, which is used to separate media of different kind or purpose. Such as Audio and Video, or the document camera and the speaker camera in video conference. This multiplexing point does not have an identifier within the RTP protocol, instead it relies on the lower layer to separate the different RTP session. Thus the most common RTP session separation is different UDP port numbers, but also IP address or other identifiers maybe used to achieve this separation. The second multiplexing point is the SSRC that separates different sources of media within a single RTP session. The third is the RTP Payload type, which identifies how the media from a particular source is encoded.

These multiplexing points area fundamental part of the design of RTP and is discussed in Section 5.2 of [RFC3550]. From that list the ones that are directly related to the importance of the RTP session as concept are 4 and 5 (from RFC 3550):

"4. An RTP mixer would not be able to combine interleaved streams of incompatible media into one stream."

"5. Carrying multiple media in one RTP session precludes: the use of different network paths or network resource allocations if appropriate; reception of a subset of the media if desired, for example just audio if video would exceed the available bandwidth; and receiver implementations that use separate processes for the different media, whereas using separate RTP sessions permits either single- or multiple-process implementations."

Point 4, has to do with media of different kind or purpose. The processing that can happen in an RTP mixer, translator or in an end-point is dependent on the purpose and media type of the stream. Thus there is an importance of separating such streams from each other. This could of course be achieved by other methods, like tagging SSRC values with their purpose, however there are reasons why this was not chosen. First of all it is not the simple solution, as this require additional signalling, and possibly synchronization between session peers. In addition there is the issue point 5 raises.

Point 5 has to do with enabling quality of service or traffic engineering between the media flows in different RTP sessions. By using different transport layer ports, QoS mechanism that are capable of operating on the 5-tuple (Source address, port, destination address, port, and protocol) can be used without modification on RTP.

Due to these design principle implementors of various services or applications using RTP has commonly not violated this model. If one choses to violate it today one fails to achieve interoperability with a number of existing services, applications and implementations.

Lets assume one overloads multiple RTP sessions into one by tagging the SSRC to belong to different purposes. If one would gateway that design into a legacy system, then there would be a significant issue with SSRC collision. This as the legacy system would not know about the need to avoid using the same SSRC in the different RTP sessions.

There are also various RTP mechanism that has the potential for issues if one don't have a clear separation of RTP sessions:

Scalabilty: RTP was built with media scalability in consideration. The simplest way of achieving separation between different scalability layers are placing them in different RTP sessions, and using the same SSRC and CNAME in each session to bind them together. This is most commonly done in multicast, and not particular applicable to RTC-WEB, but gatewaying of such a session would then require more alterations and likely stateful translation.

RTP Retransmission in Session Multiplexing mode: RTP Retransmission [RFC4588] does have a mode for session multiplexing. This would not be the main mode used in RTC-WEB, but for interoperability and reduced cost in translation support for different RTP Sessions are required.

Forward Error Correction: The "An RTP Payload Format for Generic Forward Error Correction" [RFC2733] and its update [RFC5109] can only be used on media formats that produce RTP packets that are smaller than half the MTU if the FEC flow and media flow being protected are to be sent in the same RTP session, this is due to "RTP Payload for Redundant Audio Data" [RFC2198]. This is because the SSRC value of the original flow is recovered from the FEC packets SSRC field. So for anything that desires to use these format with RTP payloads that are close to MTU needs to put the FEC data in a separate RTP session compared to the original transmissions.

RTCP behavior also becomes a factor in why overloading RTP sessions is problematic. The extension mechanisms used in RTCP depends on the media streams. For example the Extended RTCP report block for VoIP is of suitable for conversational audio, but clearly not useful for Video. This has three impacts, either one get unusable reports if they are generated for streams where there are little purpose. This is maybe less likely for the VoIP report, but for example the more

detailed media agnostic reports it may occur. It otherwise makes the implementation of RTCP more complex as the SSRC purpose tagging needs not only to be one the media side, but also on the RTCP reporting. Also the RTCP reporting interval and transmission scheduling will be affected.

As a conclusion not ensuring that RTP sessions are used for its intended purpose as a multiplexing point does violate the RTP design philosophy. It prevents the usage of certain RTP extensions. It will require additional extensions to function and will significantly increase the complexity of the implementation. At the same time it will significantly reduce the interoperability with current implementations.

2.2. Signalling for RTP sessions

RTP is built with the assumption of an external to RTP/RTCP signalling channel to configure the RTP sessions and its functions. The basic configuration of an RTP session consists of the following parameters:

RTP Profile: The name of the RTP profile to be used in session. The RTP/AVP [RFC3551] and RTP/AVPF [RFC4585] profiles can interoperate on basic level, as can their secure variants RTP/SAVP [RFC3711] and RTP/SAVPF [RFC5124]. The secure variants of the profiles do not directly interoperate with the non-secure variants, due to the presence of additional header fields.

Transport Information: Source and destination address(s) and ports for RTP and RTCP must be signalled for each RTP session. If RTP and RTCP multiplexing [RFC5761] is to be used, such that a single port is used for RTP and RTCP flows, this must be signalled.

RTP Payload Types and Media formats: The mapping between media type names (and hence the RTP payload formats to be used) and the RTP payload type numbers must be signalled. Each media type may also have a number of media type parameters that must also be signalled to configure the codec and RTP payload format (the "a=fmtp:" line from SDP).

Support for exchanging RTCP Bandwidth values to the end-points will be necessary, as described in "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth" [RFC3556], or something semantically equivalent. This also ensures that the end-points have a common view of the RTCP bandwidth, this is important as too different view of the bandwidths may lead to failure to interoperate.

2.3. (Lack of) Signalling for Payload Format Changes

As discussed in Section 2.2, the mapping between media type name, and its associated RTP payload format, and the RTP payload type number to be used for that format must be signalled as part of the session setup. An endpoint may signal support for multiple media formats, or multiple configurations of a single format, each using a different RTP payload type number. If multiple formats are signalled by an endpoint, that endpoint must be prepared to receive data encoded in any of those formats at any time. RTP does not require advance signalling for changes between formats that were signalled as part of the session setup. This is necessary for rapid rate adaptation.

3. RTP Profile

The RTP profile REQUIRED to implement is "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124]. Which will mean implicit support for AVPF [RFC4585], AVP [RFC3551] and SAVP [RFC3711].

The AVPF part of SAVPF is required to get the improved RTCP timer model, that allows more flexible transmission of RTCP packets as response to events, rather than strictly according to bandwidth. This also saves RTCP bandwidth and will commonly only utilize the full amount when there is a lot of events to send feedback on.

The S part of SAVPF is for support of SRTP. This provides media encryption, integrity protection, replay protection and a limited form of source authentication. It does not contain a specific keying mechanism. So that and the set of security transforms will be required to be selected. It is possible that a security mechanism operating on a lower layer than RTP can be used instead and that should be evaluated. However, the reasons for the design of SRTP should be taken into consideration in that discussion.

4. RTP and RTCP Guidelines

RTP and RTCP are two flexible and extensible protocols that allow, on the one hand, choosing from a variety of building blocks and combining those to meet application needs, and on the other hand, create extensions where existing mechanisms are not sufficient: from new payload formats to RTP extension headers to additional RTCP control packets.

Different informational documents provide guidelines to the use and particularly the extension of RTP and RTCP, including the following:

Guidelines for Writers of RTP Payload Format Specifications [RFC2736] and Guidelines for Extending the RTP Control Protocol [RFC5968].

5. RTP Optimizations

This section discusses some optimizations that makes RTP/RTCP work better and more efficient and therefore are considered.

5.1. RTP and RTCP Multiplexing

Historically, RTP and RTCP have been run on separate UDP ports. With the increased use of Network Address Port Translation (NAPT) this has become problematic, since maintaining multiple NAT bindings can be costly. It also complicates firewall administration, since multiple ports must be opened to allow RTP traffic. To reduce these costs and session setup times, support for multiplexing RTP data packets and RTCP control packets on a single port [RFC5761] is REQUIRED.

Note that the use of RTP and RTCP multiplexed on a single port ensures that there is occasional traffic sent on that port, even if there is no active media traffic. This may be useful to keep-alive NAT bindings.

5.2. Reduced Size RTCP

RTCP packets are usually sent as compound RTCP packets; and RFC 3550 demands that those compound packets always start with an SR or RR packet. However, especially when using frequent feedback messages, these general statistics are not needed in every packet and unnecessarily increase the mean RTCP packet size and thus limit the frequency at which RTCP packets can be sent within the RTCP bandwidth share.

RFC5506 "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences" [RFC5506] specifies how to reduce the mean RTCP message and allow for more frequent feedback. Frequent feedback, in turn, is essential to make real-time application quickly aware of changing network conditions and allow them to adapt their transmission and encoding behavior.

Support for RFC5506 is REQUIRED.

5.3. Symmetric RTP/RTCP

RTP entities choose the RTP and RTCP transport addresses, i.e., IP addresses and port numbers, to receive packets on and bind their respective sockets to those. When sending RTP packets, however, they

may use a different IP address or port number for RTP, RTCP, or both; e.g., when using a different socket instance for sending and for receiving. Symmetric RTP/RTCP requires that the IP address and port number for sending and receiving RTP/RTCP packets are identical.

Using Symmetric RTP and RTCP [RFC4961] is REQURIED.

5.4. CNAME generation

The RTCP Canonical Name (CNAME) provides a persistent transport-level identifier for an RTP endpoint. While the Synchronization Source (SSRC) identifier for an RTP endpoint may change if a collision is detected, or when the RTP application is restarted, it's RTCP CNAME is meant to stay unchanged, so that RTP endpoints can be uniquely identified and associated with their RTP media streams. For proper functionality, RTCP CNAMEs should be unique within the participants of an RTP session.

The RTP specification [RFC3550] includes guidelines for choosing a unique RTP CNAME, but these are not sufficient in the presence of NAT devices. In addition, some may find long-term persistent identifiers problematic from a privacy viewpoint. Accordingly, support for generating the RTP CNAME as specified in "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)" [I-D.ietf-avt-rtp-cnames] is RECOMMENDED, since this addresses both concerns.

6. RTP Extensions

There are a number of RTP extensions that could be very useful in the RTC-WEB context. One set is related to conferencing, others are more generic in nature.

6.1. RTP Conferencing Extensions

RTP is inherently defined for group communications, originally assuming the availability of IP multicast. In today's practice, however, overlay-based conferencing dominates, typically using one or a few so-called conference bridges or servers to connect endpoints in a star or flat tree topology. Quite diverse conferencing topologies can be created using the basic elements of RTP mixers and translators as defined in RFC 3550.

An number of conferencing topologies are defined in [RFC5117] out of the which the following ones are the more common (and most likely in practice workable) ones:

1) RTP Translator (Relay) with Only Unicast Paths (RFC 5117, section 3.3)

2) RTP Mixer with Only Unicast Paths (RFC 5117, section 3.4)

3) Point to Multipoint Using a Video Switching MCU (RFC 5117, section 3.5)

4) Point to Multipoint Using Content Modifying MCUs (RFC 5117, section 3.6)

RTP protocol extensions to be used with conferencing are included because they are important in the context of centralized conferencing, where one RTP Mixer (Conference Focus) receives a participants media streams and distribute them to the other participants. These messages are defined in AVPF [RFC4585] or in "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)" [RFC5104].

6.1.1. RTCP Feedback Message: Full Intra Request

The Full Intra Request is defined in Section 3.51 and 4.3.1 of [RFC5104]. It is used to have the mixer request from the currently distributed session participants a new Intra picture. This is used when switching between sources to ensure that the receivers can decode the video or other predicted media encoding with long prediction chains. It is RECOMMENDED that this feedback message is supported.

6.1.2. RTCP Feedback Message: Picture Loss Indicator

The Picture Loss Indicator is defined in Section 6.3.1 of [RFC4585]. It is used by a receiver to tell the encoder that it lost the decoder context and would like to have it repaired somehow. This is semantically different from the the Full Intra Request above. It is RECOMMENDED that this feedback message is supported.

6.1.3. RTCP Feedback Message: Temporary Maximum Media Stream Bit Rate Request

This feedback message is defined in Section 3.5.4 and 4.2.1 in [RFC5104]. This message and its notification message is used by a media receiver, to inform the sending party that there is a current limitation on the amount of bandwidth available to this receiver. This can be for various reasons, and can for example be used by an RTP mixer to limit the media sender being forwarded by the mixer (without doing media transcoding) to fit the bottlenecks existing towards the other session participants. It is RECOMMENDED that this

feedback message is supported.

6.2. RTP Header Extensions

The RTP specification [RFC3550] provides a capability to extend the RTP header with in-band data, but the format and semantics of the extensions are poorly specified. Accordingly, if header extensions are to be used, it is REQUIRED that they be formatted and signalled according to the general mechanism of RTP header extensions defined in [RFC5285].

As noted in [RFC5285], the requirement from the RTP specification that header extensions are "designed so that the header extension may be ignored" [RFC3550] stands. To be specific, header extensions must only be used for data that can safely be ignored by the recipient without affecting interoperability, and must not be used when the presence of the extension has changed the form or nature of the rest of the packet in a way that is not compatible with the way the stream is signaled (e.g., as defined by the payload type). Valid examples might include metadata that is additional to the usual RTP information.

The RTP rapid synchronisation header extension is recommended, as discussed in Section 6.3.

Currently no other header extensions are recommended. But we do include a list of the available ones for consideration below:

Transmission Time offsets: [RFC5450] defines a format for including an RTP timestamp offset of the actual transmission time of the RTP packet in relation to capture/display timestamp present in the RTP header. This can be used to improve jitter determination and buffer management.

Associating Time-Codes with RTP Streams: [RFC5484] defines how to associate SMPTE times codes with the RTP streams.

Audio Levels indications: There is ongoing work to define RTP header extensions for providing audio levels both from a media sender to an mixer [I-D.ietf-avtext-client-to-mixer-audio-level], and from a mixer to a receiver[I-D.ietf-avtext-mixer-to-client-audio-level].

6.3. Rapid Synchronisation Extensions

Many RTP sessions require synchronisation between audio, video, and other content. This synchronisation is performed by receivers, using information contained in RTCP SR packets, as described in the RTP specification [RFC3550]. This basic mechanism can be slow, however,

so it is RECOMMENDED that the rapid RTP synchronisation extensions described in [RFC6051] be implemented. The rapid synchronisation extensions use the general RTP header extension mechanism [RFC5285], which requires signalling, but are otherwise backwards compatible.

7. Improving RTP Transport Robustness

There are some tools that can robustify RTP flows against Packet loss and reduce the impact on media quality. However they all add extra bits compared to a non-robustified stream. These extra bits needs to be considered and the aggregate bit-rate needs to be rate controlled. Thus robustification might require a lower base encoding quality but has the potential to give that quality with fewer errors in it.

7.1. RTP Retransmission

Support for RTP retransmission as defined by "RTP Retransmission Payload Format" [RFC4588] is RECOMMENDED.

The retransmission scheme in RTP allows flexible application of retransmissions. Only selected missing packets can be requested by the receiver. It also allows for the sender to prioritize between missing packets based on senders knowledge about their content. Compared to TCP, RTP retransmission also allows one to give up on a packet that despite retransmission(s) still has not been received within a time window.

7.2. Forward Error Correction

Support of some type of FEC scheme to combat packet loss is beneficial, but is application dependent and also claimed to be mostly encumbered. For further discussion.

8. RTP Rate Control and Media Adaptation

It is REQUIRED to have an RTP Rate Control mechanism using Media adaptation to ensure that the generated RTP flows are network friendly.

The biggest issue is that there are no standardized and ready to use mechanism that can simply be included in RTC-WEB. Thus there will be need for the IETF to produce such a specification. A potential starting point for defining a solution is "RTP with TCP Friendly Rate Control"[rtp-tfrc].

9. RTP Performance Monitoring

RTCP does contains a basic set of RTP flow monitoring points like packet loss and jitter. There exist a number of extensions that could be included in the set to be supported. However, in most cases which RTP monitoring that is needed depends on the application, which makes it difficult to select which to include when the set of applications is very large.

10. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

11. Security Considerations

RTP and its various extensions each have their own security considerations. These should be taken into account when considering the security properties of the complete suite. We currently don't think this suite creates any additional security issues or properties. The usage of SRTP will provide protection or mitigation against all the fundamental issues by offering confidentiality, integrity and partial source authentication. We don't discuss the key-management aspect of SRTP in this document, that needs to be done taking the RTC-WEB communication model into account.

In the context of RTC-WEB the actual security properties required from RTP are currently not fully understood. Until security goals and requirements are specified it will be difficult to determine what security features in addition to SRTP and a suitable key-management, if any, that are needed.

12. Acknowledgements

13. References

13.1. Normative References

[I-D.ietf-avt-rtp-cnames]
Begen, A., Perkins, C., and D. Wing, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", draft-ietf-avt-rtp-cnames-05 (work in

progress), January 2011.

[I-D.ietf-avtext-client-to-mixer-audio-level]
Lennox, J., Ivov, E., and E. Marocco, "A Real-Time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", draft-ietf-avtext-client-to-mixer-audio-level-00 (work in progress), February 2011.

[I-D.ietf-avtext-mixer-to-client-audio-level]
Ivov, E., Marocco, E., and J. Lennox, "A Real-Time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", draft-ietf-avtext-mixer-to-client-audio-level-00 (work in progress), February 2011.

[RFC2733] Rosenberg, J. and H. Schulzrinne, "An RTP Payload Format for Generic Forward Error Correction", RFC 2733, December 1999.

[RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, December 1999.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.

[RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, February 2008.

[RFC5109] Li, A., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, December 2007.

[RFC5117] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 5117, January 2008.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

[RFC5450] Singer, D. and H. Desineni, "Transmission Time Offsets in RTP Streams", RFC 5450, March 2009.

[RFC5484] Singer, D., "Associating Time-Codes with RTP Streams", RFC 5484, March 2009.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC5968] Ott, J. and C. Perkins, "Guidelines for Extending the RTP Control Protocol (RTCP)", RFC 5968, September 2010.

[RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.

13.2. Informative References

[RFC2198] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A., and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

[rtp-tfrc] Gharai, L., "RTP with TCP Friendly Rate Control

(draft-gharai-avtcore-rtp-tfrc-00)", March 2011.

Authors' Addresses

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
United Kingdom

Email: csp@csperkins.org

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

Phone: +46 10 714 82 87
Email: magnus.westerlund@ericsson.com

Joerg Ott
Aalto University
School of Electrical Engineering
Espoo 02150
Finland

Email: jorg.ott@aalto.fi

RTCWEB
Internet-Draft
Intended status: Informational
Expires: August 12, 2011

J. Rosenberg
M. Kaufman
M. Hiie
F. Audet
Skype

February 8, 2011

An Architectural Framework for Browser based Real-Time Communications (RTC)
draft-rosenberg-rtcweb-framework-00

Abstract

This document defines an architectural framework for browser-based real-time communications (RTC). We propose a media component model, where the browser provides an API abstraction which models media components and connections. The underlying protocols within the browser provide for a minimum set of functionality related to transport of media.

Table of Contents

1. Introduction

Real-time communications (RTC) remains one of the few - if only - classes of desktop applications that is not yet possible using the native capabilities of the web browser. These applications run natively on the desktop, or are powered by plugins. The functionality provided by these desktop clients is rich and complex - ranging from user interface, to real-time notifications, to call signaling and call processing, to instant messaging and presence, and of course - the real-time media stack itself, including codecs, transport, firewall and NAT traversal, security, and so on.

Given the breadth of functionality in today’s desktop RTC clients, careful consideration needs to be paid to how that functionality manifests in the browser. What functionality lives within the browser itself? What functionality lives on top of it - either in client-side Javascript or within servers? What protocols are spoken by the browser itself? What protocols can be implemented within the Javascript? What protocols need to be standardized, and which do not? Pictorially, the question is what protocols, APIs, and functionality reside within the box marked "Browser RTC Function" in Figure 1. Indeed, the central question is what functionality resides in that box, as the functionality will ultimately dictate the protocols that interface to it, and the APIs which control it.

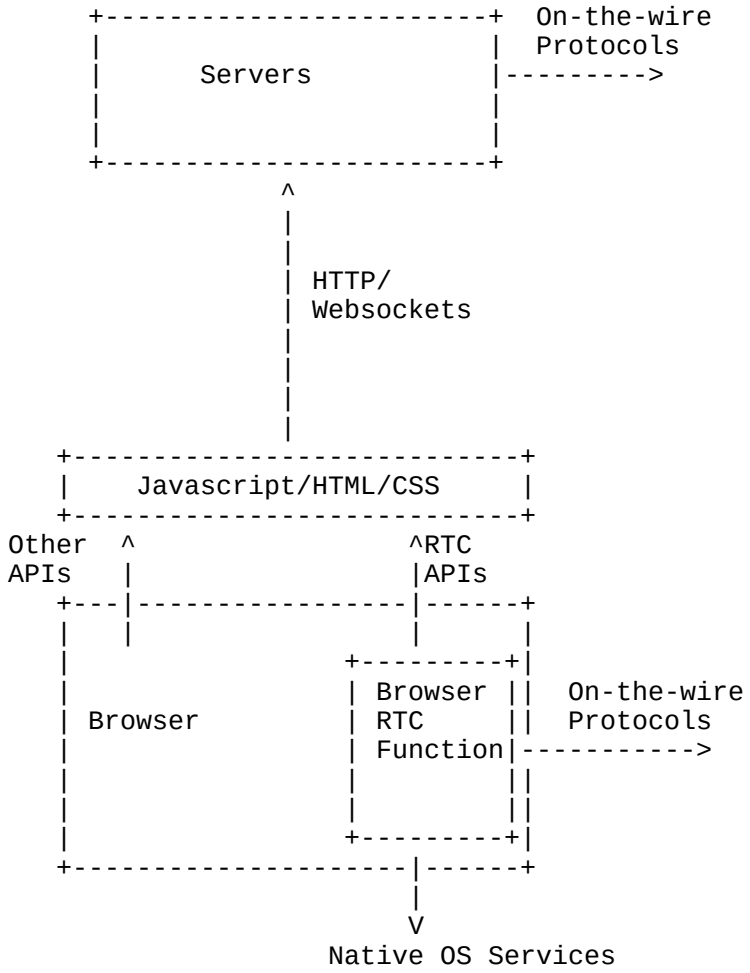


Figure 1: Browser Model

2. The Media Component Model

It is our position that the functionality that manifests within the box be a media component model. In this model, the browser

implements the necessary functionality to perform the real-time processing of media, starting from capture/render, through encapsulation in real-time transport protocols sent over the Internet. This functionality must be built into the browser, rather than within Javascript, due to its tight timing requirements and complexity. Furthermore, the functionality manifest as a set of loosely coupled components, each of which performs some aspect of the real-time processing. Each component has APIs which allow that component to be configured (with sensible defaults where appropriate), along with APIs that allow applications to gather information and statistics about the performance of that module.

The modules would include the codec itself, the acoustic echo canceller (AEC), the jitter buffer, audio and video pre-processing modules, and network transport components (including encryption and integrity protection of media) which speak specific transport protocols (such as the Real-Time Transport Protocol (RTP)). The media component model is purposefully minimalistic. It opts for maximizing the functionality that lives outside of the browser itself - within Javascript or servers. In particular, only functionality which is real-time - which cannot be done using Javascript or server functionality - resides within the browser itself. As explained in Section 5, this facilitates innovation, differentiation, and development velocity - all of the key characteristics that have made the web what it is.

As an example, a codec component implementing Opus [I-D.ietf-codec-opus] might be represented by a Javascript object with properties that mirror the configuration settings of the codec itself - the sample rate (one of narrowband, mediumband, wideband or super-wideband), the packet rate (number of frames per packet), the bitrate (which can vary between 6 and 40kbps), a slider that adjusts the packet loss resilience, a Boolean which indicates whether inband FEC should be used, and another Boolean which indicates whether to apply silence suppression. Of course, all of these parameters might have reasonable defaults so that non-expert programmers can just make it work. However, an advanced programmer could force a mode or change a setting as needed. After all, the Opus codec itself makes these parameters tunable exactly because there is no one right value; the correct setting depends on the application scenario and needs of the developer.

3. The Role of Signaling

It is our view that signaling is accomplished using a combination of existing client-server web protocols (HTTP, COMET, and websockets) and standards-based server-to-server protocols, such as SIP. A view

of the "browser RTC Trapezoid" is shown in Figure 2.

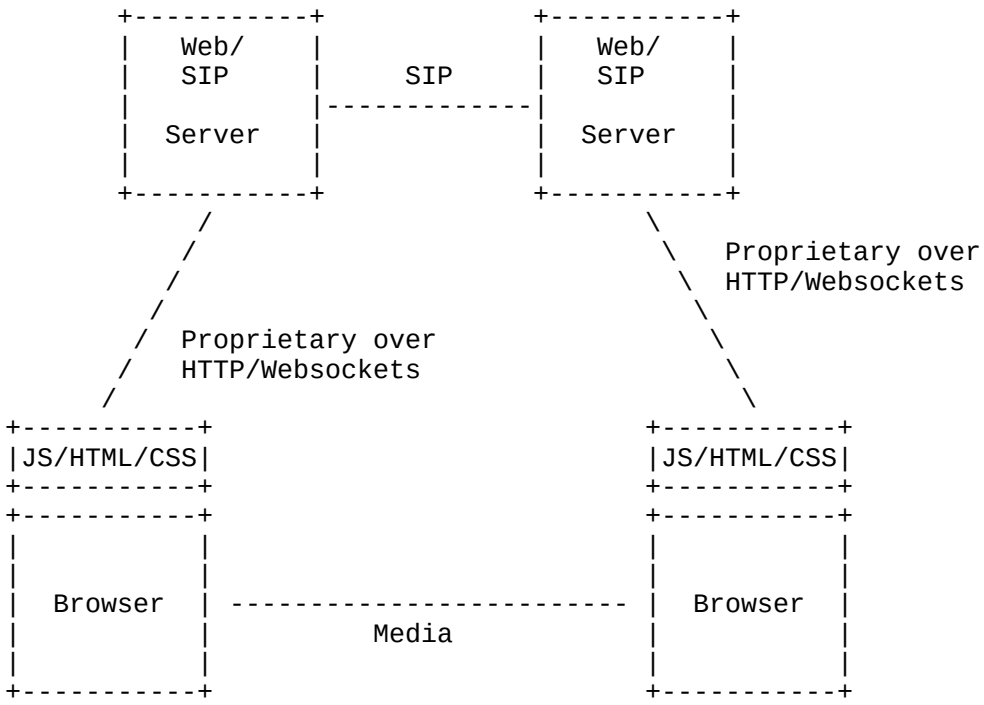


Figure 2: Browser RTC Trapezoid

In this example, a call is placed between two different providers. They use a SIP-based interface to federate between them. However, each of their respective browser-based clients signals to its server using proprietary application protocols built ontop of HTTP and Websockets. For example, provider A might offer simple calling services, and have a very simple web services interface for placing calls:

http://calling.providerA.com/call?target=joe@providerB.com&myIP=1.2.3.4:4476

Which takes only the called party and local IP/port as arguments. Provider A's server infrastructure - some combination of web and SIP servers built in any way it likes - uses the identity of the target, along with previously-known information on the capabilities of the caller's browser learned through a web-services registration, to

generate a SIP INVITE. This arrives at provider B's server infrastructure, which alerts its browser-based client of the incoming call. Provider B might be an enterprise service provider, and offer much richer features and signaling. Provider B uses a websocket interface to the browser, providing it the identity of the caller, the list of available codecs, and so on. B's service provider offers web-services based APIs for answering the call, declining it, sending to voicemail, redirecting to another number, parking it, and so on.

APIs within the browser allow each side to instruct the browsers to send media, including selection of media types and codecs. In this model, there is no SIP in the browser. It is our view that SIP has no place within the browser.

SIP is an application protocol - providing call setup, registration, codec negotiation, chat and presence, amongst other features. For each and every new feature that is desired to run between a SIP client and a SIP server, a new standard must be defined and then implemented. The feature set is indeed vast, considering the wealth of potential endpoints, ranging from simple consumer "voice only" clients, to richer videophones, to voice and video multiparty conferencing (including content sharing), to low-end enterprise phones, to high end executive admin phones, to contact centers endpoints, and beyond. Each of those requires more and more SIP extensions in order to function. This has resulted in a growing number of specifications, with diminishing returns of interoperability and feature velocity. As an example, the BLISS working group in IETF was formed to tackle some basic business phone features - including line sharing, park, call queuing, and automated call handling. Each of these individual features requires one or more specifications, and needs to be designed to meet the needs of all of the participants in the process.

There are two important consequences of this. First, the requirement of standardization acts as a huge deterrent to innovation. Indeed, in many ways, it is anathema to the very notion of how the web is supposed to work. In the web model, the provider can define arbitrary content to render to users, craft arbitrary UI, and define arbitrary messaging from the browser back to the server, all without standardization or change to the web browser. Google does not need to wait for the browsers to implement IMAP in order to provide mail service. Facebook does not need the browser to have XMPP or SIP to enable presence and instant messaging. Why is call processing any different? Why should Skype or any other real-time communications provider be constrained by standardized application protocols? Each provider should be able to design and innovate what it needs, and not be constrained by the functionalities of the application protocols burned into the browser.

While it is true that standardization will be required in order to extend these features between domains, that standardization process can be the successor - not the predecessor - to successsful deployment and usage of the feature within a domain. Furthermore, many features and services do not need to be extended between domains. Many of the BLISS features are good examples of this.

Inclusion of SIP in the browser for client to server signaling will also harm interoperability. Unfortunately, SIP interoperability betweend endpoints and servers has been relatively poor; working only for basic call setup, teardown, and basic features. Important concepts like configuration remain poorly standardized and almost never interoperate. The web has certainly had interoperability problems, but the nature of those problems is different. In the web, content providers often need to code differently for different browsers, but at least they can deliver their application functionality. On the other hand, with SIP phones, many cases features simply do not and cannot work, and this cannot be resolved through software development on behalf of the SIP provider. Interoperability is improved when there are fewer standards and not more. Instead of adding SIP and its extensions to the browser, application providers can use the tools that are already there - HTTP and websockets, and then define whatever signaling functions they desire ontop, without interoperability consequences.

Make no mistake - SIP remains important as a glue between service providers, and between server infrastructure within service provider networks. However, in a web context, there is simply no need for SIP support in the browser.

4. The Role of Media Transport

Unlike signaling, media transport does need to be in the browser, for two important reasons:

1. It operates in real-time and does not fit well with the programming model of Javascript

2. It needs to flow between endpoints directly - over UDP - in order to achieve low latency, and therefore requires standardization in order to interoperate with other providers or endpoints

The second point is important. Unlike most other web protocols, real-time media needs to be sent from the browser client to recipients other than the origin server or domain from which the web content came from. This is essential for ensuring low latency operations - one of the key metrics of quality in Voice over IP

systems. In some cases, the recipient will be another browser endpoint from the same provider. However, it could be a desktop client or mobile client from the same provider, or as shown in Figure 2, it could be a browser endpoint or desktop endpoint from another service provider. In all cases, a direct connection - indeed a direct UDP connection - is important whenever possible.

From a security perspective however, the browser cannot just have an API that tells it to send arbitrary UDP datagrams or even standardized-format voice (or worse - video) media packets to an arbitary IP address. The former introduces the opportunity for malicious JavaScript to craft packets that mimick other application protocols and send them to arbitrary endpoints (for example, an enterprise SNMP server). The latter would introduce a substantial opportunity for denial-of-service attacks. Malicious Javascript could tell the browser to "spam" an unwitting recipient with high bandwidth video. In the voice literature, this is referred to as the voice hammer attack [RFC5245]. In existing voice systems, this attack is possible but not likely due to the closed nature of most of the software and systems. In a web environment, where all it takes is one line of malicious Javascript, the attack becomes almost a certainty.

To avoid this attack, a simple handshake can be utilized. The browser should support a simple STUN-based [RFC5389] connection handshake. The exchange of the STUN transaction ID prior to transmission of media prevents the attack.

5. Benefits of the Media Component Model

There are several important benefits of the media component model proposed here.

5.1. Enabling Innovation

One of the reasons why the Web has been successful as a user interface platform is the short turn-around time to deploy new versions of web-based services. Often, these new versions are experiments that vary small details which are important to make the service successful. It is the fine granularity of user interface elements in HTML and related technologies that allow this experimentation with details. As there is no agreed-upon configuration of real-time audio/video communication technologies that always delivers the best result, we think that it is essential to give the application developers the same benefit of short turn-around time and ability to experiment with details. Therefore, the real-time communication primitives offered by user agents to web

applications/services should be fine-grained enough to allow for enhanced configurations and possibly new scenarios. Also, these interfaces to the primitives should allow gathering real-world data in enough detail on how the primitives are operating, to enable the feedback loop of deploy-measure-reconfigure-redeploy.

One of the areas where perhaps the most innovation can be expected is signaling - one only needs to look at the plethora of standards around SIP. Proposing user-agent vendors to implement all these standards is a sure way to make the common denominator across user agents marginal. Instead, the browser already has a programmability model (JavaScript) that can handle all these use cases, and more, provided the programming environment has access to the underlying media components as we propose here. Drawing again parallels from user interface development, there is an undecided problem of what should be executed by the user agent, and what by the web servers (e.g. validation). Similar gray boundary between the client and the server exists in the field of real-time communications. Therefore we propose to leave standardization of signaling out of scope for this activity, and let the web service providers define signaling as they see fit.

5.2. The Importance of Flexibility

There are obviously tradeoffs between built-in functionality and programmability. It is often tempting to provide the web page author with a simple and relatively inflexible way of expressing their intent so as to minimize the page author’s effort and accelerate adoption. As an example, the "<blink>" tag was adopted much more rapidly than it would have been if blinking text could have only been implemented by writing a JavaScript timer task to manipulate the DOM style objects.

On the other hand, such built-in functionality comes at two important costs. First, each browser implementation must implement the functionality, and the more which is moved from JavaScript to built-in functionality the more code must be present for that implemention. Second, and more important, the page author is now restricted to the subset of functionality which is provided by these browser implementations.

The "<video>" tag as it currently stands is an excellent example. While it does make it possible for a page author to embed video playback within a page without relying on external plug-ins (and without knowing much more than the URL of the video they wish to play), it also leaves the implementation of advanced functionality - such as adaptive multi-bitrate streaming - in the hands of the browser developer, not the page author. Unless all vendors agree on

a standard for transmission of such videos (including things like the file format for manifest information), this advanced functionality will be not available across the browser landscape. Most importantly, the logic - the actual decisions about when to switch rates and why - becomes buried deep inside the browser, hard or potentially impossible to adjust for various circumstances.

An alternative approach for adaptive multi-bitrate video streaming was recently adopted by the Flash Player. The video object simply has an API for receiving bits to be played back. The script engine (and thus the script author, usually through the use of a pre-existing library) becomes responsible for determining which bits to download and which bits to pass to the video object. This enables adaptive multi-bitrate HTTP streaming video, but it also enables any number of other uses, many of which were not even contemplated by the providers of that API. It also means that upgrades to this logic come in the form of new script libraries, and not in the form of an upgrade to the Flash Player itself.

We advocate a similar approach here whenever it is possible. With the exception of the passing of real-time data to and from the media components (which we believe must communicate directly in order to meet real-time latency constraints) we advocate placing all of the logic outside of the browser itself and instead into the hands of the page author through JavaScript APIs. These APIs may be more complex to use for some cases, but they minimize the implementation effort on the part of the browser vendor and can provide functionality that has not yet been contemplated.

An example of this might be the peer-to-peer NAT traversal problem. Rather than having an API for "browser, please use ICE [RFC5245] to open a connection to another peer" we would instead have APIs like "browser, please send an ICE-compatible STUN [RFC5389] probe to the following candidate address". This allows the actual logic, the sequencing, the choice of what to implement at the client and what to offload to the server, to be in the hands of the JavaScript developer. We expect that libraries to implement common functionality (such as ICE, which could be built ontop of this) will become readily and freely available, and so in short order the extra work required for a page author to work with these lower level APIs becomes insignificant.

6. Interoperability with Existing VoIP Gear

In order for Browser-based Real-Time Communication to be successful, it is essential to ensure a good level of interoperability with existing VoIP gear. This means that a strong baseline for

interoperability of end-to-end media needs to be defined.

The amount of VoIP gear currently deployed is very substantial for both VoIP Service Providers and Enterprise IP Telephony. In both cases, media is transported on RTP/RTCP [RFC3550] using codecs such as G.711 and G.729. Signaling for call control uses SIP [RFC3261], H.323, H.248/Megaco, and a wide range of proprietary protocols. Inter-domain, the signaling protocol is mostly SIP.

Interoperability at the signaling level can be handled by gateways, and is outside the scope of this paper. Media interoperability however needs to be addressed. It is not acceptable to rely on servers to convert media from one transport (and codec) to another because it introduces significant challenges. First, it requires a large number of servers to do the actual transcoding, which increases cost. Second, it affects the routing of media by adding an additional leg to the transport, which increases end-to-end delay, and therefore decreases voice quality. If there is codec translation, it decreases voice quality even further. And finally, it can potentially complicate end-to-end security.

Interoperability means working with reality, and not just standards. As such, it is important that browsers support basic RTP transport for voice and support the G.711 codec. Furthermore, they should interoperate with network-based session border controllers, which are the most commonly deployed technique for NAT traversal in existing networks. They should also support security, based on SRTP [RFC3711].

7. Informative References

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[I-D.ietf-codec-opus] Valin, J. and K. Vos, "Definition of the Opus Audio Codec", draft-ietf-codec-opus-02 (work in progress), February 2011.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time

Applications", STD 64, RFC 3550, July 2003.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

Authors' Addresses

Jonathan Rosenberg
Skype
Monmouth, NJ
US

Email: jdrosen@skype.net
URI: http://www.jdrosen.net

Matthew Kaufman
Skype
Palo Alto, CA
US

Email: matthew.kaufman@skype.net

Magnus Hiie
Skype
Palo Alto, CA
US

Email: magnus.hiie@skype.net

Francois Audet
Skype
Palo Alto, CA
US

Email: francois.audet@skype.net