RTCWEB
Internet-Draft
Intended status: Standards Track
Expires: August 22, 2013

M. Thomson
Microsoft

February 18, 2013

Data Channels for RTCWEB
draft-thomson-rtcweb-data-00

Abstract

RTCWEB have selected SCTP over DTLS over UDP with ICE for peer-to-peer data channels. There is some debate over the best way to negotiate channels. This proposal is a nose-to-tail description of an alternative to existing proposals.

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 22, 2013.

Copyright Notice

Table of Contents

1. Introduction

RTCWEB [I-D.ietf-rtcweb-overview] has defined the use of the Stream Control Transmission Protocol (SCTP) [RFC4960] over Datagram Transport Layer Security (DTLS) [RFC6347] over UDP with Interactive Connectivity Establishment (ICE) [RFC5245] for peer-to-peer data channels.

This document describes a proposal for how this protocol stack is used. The proposal attempts to reconcile the following basic requirements:

- the ability to have data channels used interchangeably with websockets, after establishment
- the ability to use as many SCTP features as possible

Like other proposals, this proposal uses an API that is largely interchangeable with the WebSockets API [REF:TBD]. Of course, that alone is insufficient because the way that data channels are created is completely different to websockets [RFC6455]. Only the general usage of the API follows the WebSockets API, channel establishment requires a very different process.

Furthermore, not every application will care for compatibility with the WebSockets API. For those applications, additional properties are exposed to enable valuable SCTP features.

In these aspects, all data channel proposals are the same. The details differ. For example, this one doesn't need an in-band protocol. It even avoids the need for negotiation, except where it is needed. If not for the fact that the WebSockets API designers - in their infinite wisdom - decided to distinguish text from binary, it wouldn't even need to use a PPID to identify textual messages.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

2. Overview of Operation

A data channel is a bidirectional communication medium between WebRTC peers.

Every data channel is bound to a specific SCTP stream number. The same SCTP stream identifier is used for both directions of the data channel. Though SCTP streams are unidirectional, and this concept doesn't hold any particular meaning for SCTP, this simplification ensures that channels can be created with minimal overhead.

Each data channel has a set of properties that governs how it sends messages. Unlike other SCTP APIs where properties like reliability settings are set on a per-message basis, this API places these properties on the data channel. This allows the API to behave exactly like the WebSockets API when sending messages. Details of the available data channel properties are included in Section 4.

There are three ways that a data channel can be created. All three result in an object representing the data channel being provided to the application. Each differs in the manner of delivery and how properties are selected for the data channel.

1. The application can request the creation of a new data channel directly. The browser selects appropriate properties for the channel, using any values provided by the application and providing defaults for others.

   This triggers a notification to the application that indicates that it needs to renegotiate the session.

2. Offer/answer negotiation can trigger the creation of a new data channel. In this case, the session description provided in an offer or answer describes the properties of the channel.

3. Messages can arrive on an SCTP stream that does not have a data channel allocated. If messages arrive on a stream, the browser provides default values for all stream properties.

Channel creation can fail if there are an insufficient number of available SCTP streams. This is based on either a local unwillingness to receive more streams, or based on knowledge of the unwillingness of the peer to receive more streams.

Creation can also fail if the application specifies a stream ID that is already in use. These should trigger the appropriate error mechanisms (exceptions or something).

Channels are closed by sending a RE-CONFIG chunk that includes Incoming and Outgoing SSN Reset Request parameters, as defined in [RFC6525]. Closing a channel doesn't permit the sending of a code and message as exposed in the WebSockets API, any values that are provided by the application are discarded.

3. (In)consistent Properties

The creation of the first data channel will require offer/answer negotiation. This is necessary to ensure that the SCTP association is created, including ICE, the DTLS handshake, plus any authentication that might be required.

Once an SCTP association is live, data channels can be used to exchange messages immediately after they are created. The drawback is that messages arrive at the peer without any information about what properties the sender attaches to the corresponding data channel.

How much property consistency matters to the application will depend on the application. If the application is performing SS7 signaling using M3UA [RFC4666], this is unlikely to matter, but some applications could rely on having consistent data channel properties.

3.1. Negotiation

The safest (and slowest) way to establish new channels with consistent properties is to negotiate them. This is performed using an offer/answer exchange. The application is able to choose where and when this negotiation occurs. If there is an existing data channel, then this provides a low latency path for performing this negotiation.

The negotiation includes a description for every SCTP stream that a peer is sending that includes all of the data channel properties (see Section 4), so that the receiver can create a data channel with the same properties. The browser creates a data channel with the described properties and provides that to the application. If the data channel description appears in an offer, the answer describes the data channel that is used on the same stream number.

An offer or answer that includes a description for a data channel that already exists, then the properties of that data channel are not modified. An answer MUST include the properties of the existent data channel, not the channel that the offer describes.

3.2. Dealing with Mismatched Properties

An application that chooses to send on data channels prior to negotiation will cause the receiving peer to create a data channel with a default configuration. Applications can handle this in a number of ways:

- The application on the receiving peer can create data channels in the same order as the sender to ensure consistent properties. This is possible because stream identifiers are assigned in the same way by both peers.

- The application on the receiving peer might apply application-specific default values for all non-negotiated channels.

- The application on the receiving peer might not care about having consistent data channel properties. Note that data channel properties only apply to the sending of messages.

Note: It is possible to provide an application with information about the values that are in use by a peer. This would in most cases be possible after negotiation, though some properties are revealed when new messages arrive. Of course, this is a lot of effort after the application has already effectively declared that it doesn't care. Given that the application could exchange this information using the data channel(s) it has convenient, adding new APIs seem of very low value.

[[Irrelevant API Note: Adding a data channel triggers a notification to the application that it should renegotiate the session. Normally, the 'negotiation needed' state is cleared when the negotiation commences (or completes?). If the application decides to send packets, then the damage is done and there is no point negotiating. That being the case, a data channel could removed from the set of unnegotiated things upon sending a packet. Negotiation from this point isn't going to change anything.]]

4. Available Data Channel Properties

The following properties are exposed to the application. All of these properties can be set during the creation of a data channel. Once the channel object is created, these properties are mutable, with the exception of "streamId".

streamId The SCTP stream ID to use for the channel. If not provided by the application, the lowest valued stream ID that is not already in use by a data channel is selected. If the channel is created as a result of negotiation or packet arrival, the stream ID has already been chosen.

binaryPPID The SCTP payload protocol identifier (PPID) that is used for binary messages. Textual messages are always sent using a PPID that indicates textual content, so this value only determines the PPID that is attached to binary messages. This field is a 32-

bit number.

Unless otherwise specified, channels use the PPID for WebRTC binary data channels. Channels created in response to the receipt of a message use the PPID from the received message, unless the message uses the PPID for WebRTC textual data channels, which causes the binary PPID to be selected instead. Details on the newly defined PPIDs are included in Section 6.

reliabilityTime The amount of time (in milliseconds) that the browser will attempt to retransmit messages for reliable delivery. Together with reliabilityRetransmissions, this enables unreliable or partially reliable transmission of data. The default value for this property is the largest number available (e.g., Number.POSITIVE_INFINITY).

reliabilityRetransmissions The numer of retransmissions that the browser will make for any packet reliable delivery. Together with reliabilityTime, this enables unreliable or partially reliable transmission of data. The default value for this property is the largest number available (e.g., Number.POSITIVE_INFINITY).

label The label to assign to the data channel. A default value is selected by the receiving browser.

protocol A protocol label that identifies the protocol used on the data channel. This property is undefined unless set by the application.

binaryType The BinaryType defined in The WebSockets API determines whether binary data is provided to the application as Blob or ArrayBuffer objects. The default value is "blob". Note that changing this might not have an immediate effect if messages have started to arrive prior to the change.

All these properties are specific to each message that is sent. Changes to mutable properties take effect for the next message that is sent on the channel.

This isn’t a W3C WebRTC document, so specifics of the API aren’t really relevant, but it is imagined that these properties could be passed in a dictionary to the method that creates a new data channel and exposed as attributes on the data channel object.

5. SDP Format

The properties described above appear in SDP. All properties are

declarative. Though the specifics of the syntax doesn't matter much, the following example could indicate something that would work:

```
m=application 12345 SCTP/DTLS 0 1 2 5
c=IN IP6 ::1
a=fmtp:0 binaryPPID=177;label=control
a=fmtp:1 label=chat
a=fmtp:2 label=characters;reliabilityTime=2000;protocol=lrudfb
a=fmtp:5 label=bullets;reliabilityTime=5000
```

[[Formal SDP grammar TBD]]

This assumes that the SCTP port number (inside the DTLS encapsulation) is fixed, so that this doesn't need to be indicated anywhere. I'm not sure whether asking IANA for a port allocation makes sense though.

6. Payload Protocol Identifiers

Two SCTP payload protocol identifiers are defined for WebRTC data channels.

The WebRTC textual data channel PPID (number TBD) is used for all messages that are identified as being textual. The payload of messages marked with this PPID MUST be UTF-8 encoded text.

The WebRTC binary data channel PPID (number TBD) is used as the default PPID for new data channels.

7. IANA Considerations

This document probably should register the PPIDs, but I don't really have time to do that right now.

8. Security Considerations

I thought about this, and I can't think of any specific security considerations. I could blather on about willingness to accept streams and large volumes of data, but that's pretty lame.

I'm sure something will turn up eventually.

9. Acknowledgments

This document is a rush job. If it survives for any longer than a couple of weeks, no doubt this section will come in handy.

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.

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

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, February 2012.

10.2. Informative References

[I-D.ietf-rtcweb-overview] Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-05 (work in progress), December 2012.

[RFC4666] Morneault, K. and J. Pastor-Balbas, "Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer (M3UA)", RFC 4666, September 2006.

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

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

Author's Address

Martin Thomson
Microsoft
3210 Porter Drive
Palo Alto, CA 94304
US

Phone: +1 650-353-1925
Email: martin.thomson@skype.net