RTCWeb Working Group	R. Jesup
Internet-Draft	Mozilla
Intended status: Informational	S. Loreto
Expires: May 03, 2012	Ericsson
	M. Tuexen
	Muenster University of Applied Sciences
	October 31, 2011

RTCWeb Datagram Connection
draft-jesup-rtcweb-data-01.txt

Abstract

This document investigates the possibilities for designing a generic transport service that allows Web Browser to exchange generic data in a peer to peer way. Several, already standardized by IETF, transport protocols and their properties are investigated in order to identify the most appropriate one.

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 May 03, 2012.

Table of Contents

1. Introduction

The issue of how best to handle non-media data types in the context of RTCWEB is still under discussion in the mailing list; there have been several proposals on how to address this problem, but there is not yet a clear consensus on the actual solution.

However it seems to be a general agreement that for NAT traversal purpose it has to be:
FOO/UDP/IP
or most likely:
FOO/DTLS/UDP/IP (for confidentiality, source authenticated, integrity protected transfers)
where FOO is a protocol that is supposed to provide congestion control and possible some type of framing or stream concept.
Moreover there has been a clear interest for both an unreliable and a reliable peer to peer datagram based channel.
This document provides Requirement and use cases for both unreliable and reliable peer to peer datagram base channel, provide an overview of the pro and cons of the different proposed solutions, and finally analyze in more detail the SCTP based solution.

2. Requirements

This section lists the requirements for P2P data connections between two browsers.

Req. 1 Multiple simultaneous datagram streams must be supported. Note that there may 0 or more media streams in parallel with the data streams, and the number and state (active/inactive) of the media streams may change at any time.

Req. 2 Both reliable and unreliable datagram streams must be supported.

Req. 3 Data streams must be congestion controlled; either individually, as a class, or in conjunction with the media streams, to ensure that datagram exchanges don't cause congestion problems for the media streams, and that the rtcweb PeerConnection as a whole is fair with competing streams such as TCP.

Req. 4 The application should be able to provide guidance as to the relative priority of each datagram stream relative to each other, and relative to the media streams. [TBD: how this is encoded and what the impact of this is.] This will interact with the congestion control.

Req. 5 Datagram streams must be encrypted; allowing for confidentiality, integrity and source authentication. See the security spec [xxx] for detailed info.

Req. 6 Consent and NAT traversal mechanism: These are handled by the PeerConnection's ICE [RFC5245] connectivity checks and optional TURN servers.

Req. 7 Data streams MUST provide message fragmentation support such that IP-layer fragmentation does not occur no matter how large a

message/buffer the Javascript application passes down to the Browser to be sent out.

Rec. 8 The data stream transport protocol MUST NOT encode local IP addresses inside its protocol fields; doing so reveals potentially private information, and leads to failure if the address is depended upon.

Req. 9 The data stream protocol SHOULD support unbounded-length "messages" (i.e., a virtual socket stream) at the application layer, for such things as image-file-transfer; or else it MUST support at least a maximum application-layer message size of 4GB.

Req. 10 The data stream packet format/encoding MUST be such that it is impossible for a malicious Javascript to generate an application message crafted such that it could be interpreted as a native protocol over UDP - such as UPnP, RTP, SNMP, STUN, etc.

Req. 10 The data stream transport protocol MUST start with the assumption of a PMTU of 1280 [*** need justification ***] bytes until measured otherwise.

Req. 11 The data stream transport protocol MUST NOT rely on ICMP being generated or being passed back, such as for PMTU discovery.

3. Use cases.

3.1. Use cases for unreliable datagram based channel

U-C 1 A real-time game where position and object state information is sent via one or more unreliable data channels. Note that at any time there may be no media channels, or all media channels may be inactive.

U-C 2 Non-critical state updates about a user in a video chat or conference, such as Mute state.

3.2. Use cases for reliable channels (datagram or stream).

Note that either reliable datagrams or streams are possible; reliable streams would be fairly simple to layer on top of SCTP reliable datagrams with in-order delivery.

U-C 3 A real-time game where critical state information needs to be transferred, such as control information. Typically this would be datagrams. Such a game may have no media channels, or they may be

inactive at any given time, or may only be added due to in-game actions.

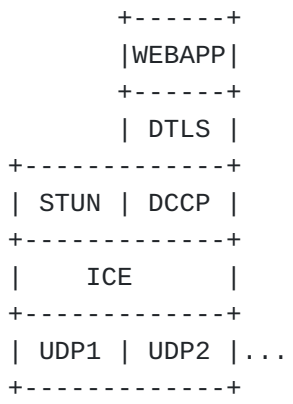
U-C 4 Non-realtime file transfers between people chatting. This could be datagrams or streaming; streaming might be an easier fit

U-C 5 Realtime text chat while talking with an individual or with multiple people in a conference. Typically this would be datagrams.

U-C 6 Renegotiation of the set of media streams in the PeerConnection. Typically this would be datagrams

4. Protocol alternatives

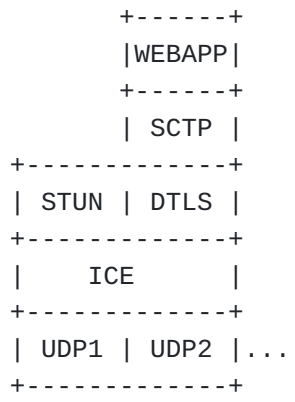
4.1. Datagrams over DTLS over DCCP over UDP.



DCCP [RFC4340] adds to a UDP-like foundation the minimum mechanisms necessary to support congestion control. It is a unicast, connection-oriented protocol with bidirectional data flow.
The main downside of DCCP is the uncertainty of the DCCP implementations. Moreover DCCP only meets the requirements for the unreliable data channel, so in order to satisfy the reliable data channel requirements there is a need to build a new mechanism on top of DCCP or use a different protocol for it.
The main advantage of DCCP is that the Congestion Control (CC) methods are modularly separated from its core, that allows each application to choose a different congestion control methods it prefers. Each congestion control method is denoted by unique ID (CCID): a number from 0-255. CCIDs 2, 3 and 4 are current defined; CCIDs 0, 1, and 5-255 are reserved. The end-points negotiate their CCIDs during connection initiation and achieve agreement through the exchange of feature negotiation options in DCCP headers.
CCID 2 [RFC4341] denotes Additive Increase, Multiplicative Decrease congestion control with feature modelled directly on TCP. It achieves the maximum bandwidth over the long term consistent with the use of

end-to-end congestion control but reduces the congestion window by half upon congestion detection. Applications that prefer a large amount of bandwidth as feasible over the longer terms should use CCID2.
CCID 3 [RFC4342], TCP friendly rate control mechanism(TFRC), denotes an equation-based and rate controlled congestion control mechanisms designed to be reasonably fair when competing for bandwidth with TCP like flows. It shows much lower variation of throughput over time compared with TCP that makes CCID 3 more suitable than CCID 2 for applications such as streaming media content that prefers to minimize abrupt changes in the sending rate.
CCID 4 [RFC5622] is a modified version of CCID 3 and designed for applications that use a small fixed segment size, or change their sending rate by varying the segment size. It denotes TFRC-SP (TCP friendly rate control for small packets)
Both CCID 3 and 4 uses the TCP throughput equation for CC; the former is based on the calculation of loss event rate but the later also include nominal packet size of 1460 bytes, a round trip estimate in TCP throughput calculation. In contrast to CCID 3, the CCID 4 sender imposes a minimum interval of 10 ms between data packets regardless of the allowed transmit rate.

4.2. Datagrams over SCTP over DTLS over UDP.

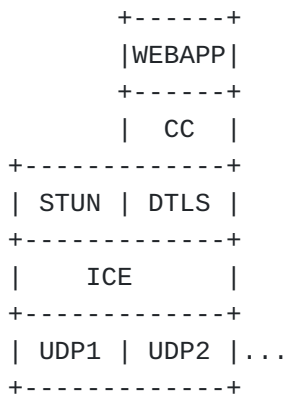


An SCTP [RFC4960] based solution will provide several interesting features among the others: Multistreaming, Ordered and Unordered delivery, Reliability and partial-Reliability [RFC3758], and Dynamic Address Reconfiguration [RFC5061].
Moreover SCTP provides the possibility to transport different "protocols" over multiple streams and associations using the ppid (Payload Protocol Identifier). An application can set a different PPID with each send call. This allows the receiving application to look at this information (as well as the stream id/seq) on receiving to know what type of protocol the data payload has.

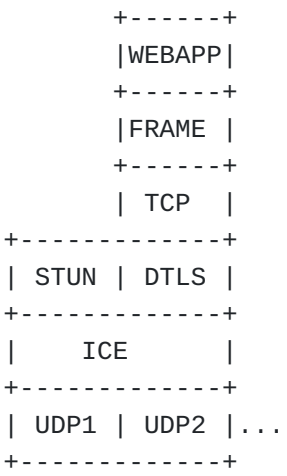
The SCTP feature so seems to satisfy all the requirements for both the unreliable and the reliable scenario.
There are SCTP implementations for all the different OS: Linux (mainline kernel 2.6.36), FreeBSD (release kernel 8.2), Mac OS X, Windows (SctpDrv4) and Solaris (OpenSolaris 2009.06), as well as a user-land SCTP implementation based on the FreeBSD implementation).
The SCTP solution is analyzed in more detail in Section 5.

4.3. A new protocol on top of UDP.

This option requires to at least build a congestion control (CC) mechanism on top of the plain UDP.
In designing it we have to follow carefully the guidelines provided in [RFC5405].



4.3.1. TCP over DTLS over UDP.



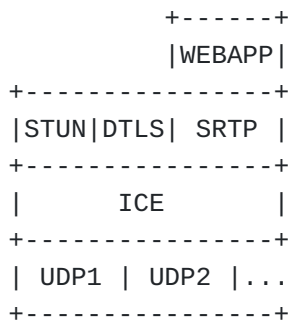
Layering TCP atop DTLS or UDP is an approach that has been suggested several times in the past, including TCP-over-UDP [I-D.baset-tsvwg-tcp-over-udp] and UDP-Encapsulated Transport Protocols [I-D.denis-udp-transport].

A similar mechanism has also been used for Google Talk's peer-to-peer file transfer protocol, implemented in the libjingle library as "PseudoTcp" [http://code.google.com/p/libjingle/source/browse/trunk/talk/p2p/base/pseudotcp.cc]. In this implementation, a lightweight userspace TCP stack has been developed with support for a fairly minimal set of TCP options, namely delayed acknowledgements, Nagle, fast retransmit, fast recovery (NewReno), timestamps, and window scaling. Some features have been removed, such as urgent data. And as in the aforementioned drafts, the TCP header has been tweaked slightly to remove fields redundant with the UDP header, namely source/destination port and checksum.

The advantage of this approach is clear; TCP is well-known and understood, and its congestion control is by definition TCP-fair. User-space implementations of TCP exist, e.g. as PseudoTcp, which has considerable deployment experience. It is also possible to support multiplexing of datagram flows with this approach, either by adding a stream identifier to the TCP header (in place of the port numbers, perhaps), or doing the same in a higher-level framing layer.

This approach has some disadvantages as well; since TCP is a stream, rather than datagram-oriented protocol, some framing needs to be added on top of TCP to provide the necessary datagram interface to the calling API. In addition, TCP only provides reliable delivery; there is no provision for a unreliable channel. This deficiency could be remedied by providing a separate protocol for unreliable channels. Such a protocol could be a lightweight datagram protocol that implements the sequence number and other fields necessary to run TFRC-SP.

4.4. A RTP compatible protocol.



When sending RTP with DTLS, rather than encapsulating RTP in DTLS, SRTP keys are extracted from the DTLS key material, allowing use of SRTP's

more efficient format. This same approach could be extended for sending of application data as a RTP payload.
The benefits of this approach are centered around the ability to reuse the existing mechanisms for audio/video data for application data, and the resultant simplification that occurs. If everything ends up RTP, and RTP provides information about loss and timing, we can have common encryption, congestion control, and multiplexing mechanisms for all types of data. For example, we could use RTP SSRC to demux different application data streams, and RTCP NACK to faciliate reliable delivery.
On the other hand, RTP has a number of semantics associated with it that aren't necessarily a good fit for arbitrary application data. While the RTP timestamp, sequence number and SSRC fields are meaningful, there are a number of other header fields that may not make sense, and the applicability of RTP's notions of timing, media playout, and control feedback is unclear.

5. Datagrams over SCTP over DTLS over UDP.

5.1. User Space vs Kernel implementation.

Even kernel implementation of SCTP are already available for all the different platforms (see Section 4.2), there are compelling reasons that incline towards for a SCTP stack that works well in user land. The main reason is deployability.
There are many applications today that are expected to run on a wide range of old and new operating systems. Web browsers are an excellent example. They support operating systems 10 years old or more, they run on mobile and desktop operating systems, and they are highly portable to new operating systems. This is achieved by having a fairly narrow portability layer to minimize what needs to be supported on old operating systems and ported to new ones. This creates a strong desire to implemented as much functionality as possible inside the application instead of relying on the operating system for it.
This leads to a situation where there is a desire for the SCTP stack to be implemented in the user space instead of the kernel space. For many applications that require support of operating systems without SCTP (insert whatever stack order is - UDP, DTLS - whatever), there is no way to deploy this unless it can be implemented in the application. The traditional reasons for kernel implementations, such as mux of many application using transport port numbers, does not particularly apply here where that level of multiplexing between application was provided by the underling UDP that is tunneled over. The requirement is:
It MUST be possible to implement the SCTP and DTLS portion of the stack in the user application space.

5.2. The envisioned usage of SCTP in the RTCWeb context

The appealing features of SCTP in the RTCWeb context are:

-

the congestion control which is TCP friendly.

- the congestion control is modifiable for integration with media stream congestion control.

- support for multiple channels with different characteristics.

- support for out-of-order delivery.

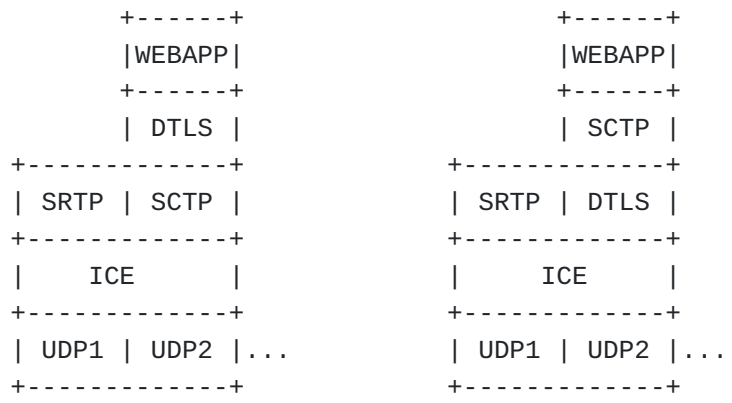
- support for large datagrams and PMTU-discovery and fragmentation.

- the reliable or partial reliability support.

Multi streaming is probably also of interest.
Multihoming will not be used in this scenario. The SCTP layer would simply act as if it were running on a single-homed host, since that is the abstraction that the lower layers (e.g. UDP) would expose.

5.3. SCTP/DTLS/UDP vs DTLS/SCTP/UDP

The two alternatives being discussed in this subsection are shown in the following Figure 6.



The UDP encapsulation of SCTP used in the protocol stack shown on the left hand side of Figure 6 is specified in [I-D.ietf-tsvwg-sctp-udp-encaps] and the usage of DTLS over SCTP is specified in [RFC6083]. Using the UDP encapsulation of SCTP allows SCTP implementations to run in user-land without any special privileges, but still allows the support of SCTP kernel implementations. This also requires no SCTP specific support in middleboxes like firewalls and NATs. Multihoming and failover support is implemented via the ICE layer, and SCTP associations are single-homed at the SRTP layer. The SCTP payload is protected by DTLS, which provides confidentiality, integrity and source authentication. SCTP-AUTH as specified in [RFC4895] is used to provide integrity of SCTP control information related to the user messages like the SCTP stream identifier. Please note that the SCTP control information (like the SCTP stream identifier) is sent unencrypted.

Considering the protocol stack on the right hand side of Figure 6, the usage of DTLS over UDP is specified in [I-D.ietf-tls-rfc4347-bis]. Using SCTP on top of DTLS is currently unspecified. Since DTLS is typically implemented in user-land, an SCTP user-land implementation has also to be used. Kernel SCTP implementations can't be used. When using DTLS as the lower layer, only single homed SCTP associations can be used, since DTLS does not expose any any address management to its upper layer. DTLS implementations used for this stack must support controlling fields of the IP layer like the DF-bit in case of IPv4 and the DSCP field. This is required for performing path MTU discovery. The DTLS implementation must also support sending user messages exceeding the path MTU. When supporting multiple SCTP associations over a single DTLS connection, incoming ICMP or ICMPv6 messages can't be processed by the SCTP layer, since there is no way to identify the corresponding association. Therefore the number of SCTP associations should be limited to one or ICMP and ICMPv6 messages should be ignored. In general, the lower layer interface of an SCTP implementation has to be adopted to address the differences between IPv4 or IPv6 (being connection-less) or DTLS (being connection-oriented). When this stack is used, DTLS protects the complete SCTP packet, so it provides confidentiality, integrity and source authentication of the complete SCTP packet.

It should be noted that both stack alternatives support the usage of multiple SCTP streams. A user message can be sent ordered or unordered and, if the SCTP implementations support [RFC3758], with partial reliability. When using partial reliability, it might make sense to use a policy limiting the number of retransmissions. Limiting the number of retransmissions to zero provides a UDP like service where each user messages is sent exactly once.

SCTP provides congestion control on a per-association base. This means that all SCTP streams within a single SCTP association share the same congestion window. Traffic not being sent over SCTP is not covered by the SCTP congestion control.

6. Message Format.

TBD if we need also to design a framing or not.

At time of writing nobody has identified a real need for masking of UDP, and DTLS is a much-preferred solution for encryption/ authentication.

More SCTP already provides sequence number information (e.g. stream sequence numbers). However there could be a need to support different kind of message (link in WebSocket where there is support to distinguish between Unicode text and binary frames), since for SCTP they are all user message. In the case there is this need a possibility is to map types to streams.

7. Security Considerations

To be done.

8. IANA Considerations

This document does not require any actions by the IANA.

9. Acknowledgments

Many thanks for comments, ideas, and text from Cullen Jennings, Eric Rescorla, Randall Stewart and Justin Uberti.

10. References

[RFC3758]	Stewart, R., Ramalho, M., Xie, Q., Tuexen, M. and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
[RFC4340]	Kohler, E., Handley, M. and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
[RFC4341]	Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, March 2006.
[RFC4342]	Floyd, S., Kohler, E. and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.
[RFC5622]	Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", RFC 5622, August 2009.
[RFC4895]	Tuexen, M., Stewart, R., Lei, P. and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, August 2007.
[RFC4960]	Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
[RFC5061]	Stewart, R., Xie, Q., Tuexen, M., Maruyama, S. and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, September 2007.
[RFC5245]	Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/ Answer Protocols", RFC 5245, April 2010.
[RFC5389]	

	Rosenberg, J., Mahy, R., Matthews, P. and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.
[RFC5405]	Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
[RFC6083]	Tuexen, M., Seggelmann, R. and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, January 2011.
[I-D.ietf-tls-rfc4347-bis]	Rescorla, E and N Modadugu, "Datagram Transport Layer Security version 1.2", Internet-Draft draft-ietf-tls-rfc4347-bis-06, July 2011.
[I-D.ietf-tsvwg-sctp-udp-encaps]	Tuexen, M and R Stewart, "UDP Encapsulation of SCTP Packets", Internet-Draft draft-ietf-tsvwg-sctp-udp-encaps-01, October 2011.
[I-D.baset-tsvwg-tcp-over-udp]	Baset, S and H Schulzrinne, "TCP-over-UDP", Internet-Draft draft-baset-tsvwg-tcp-over-udp-01, June 2009.
[I-D.denis-udp-transport]	Denis-Courmont, R, "UDP-Encapsulated Transport Protocols", Internet-Draft draft-denis-udp-transport-00, July 2008.

Authors' Addresses

Randell Jesup Jesup Mozilla EMail: randell-ietf@jesup.org

Salvatore Loreto Loreto Ericsson Hirsalantie 11 Jorvas, 02420 Finland EMail: salvatore.loreto@ericsson.com

Michael Tuexen Tuexen Muenster University of Applied Sciences Stegerwaldstrasse 39 Steinfurt, 48565 Germany EMail: tuexen@fh-muenster.de