WebRTC Data Streams

Justin Uberti
IETF 82.5

Recap

Add support to the PeerConnection API to let apps exchange arbitrary data with **low latency** and either **reliable or unreliable** delivery semantics.

This data exchange may be performed **with or without** transfer of audio/video media. Applications may establish **multiple data channels**.

Some Use Cases

- **Real-time games** (player state updates)
- **Real-time text**
- **Co-browsing** (mouse pointer updates)
- **Remote desktop** (mouse/keyboard events)
- **File transfer**
- **P2P broadcast**

Many of these cases will be multiparty (i.e. messages will be sent/received to/from many remote parties)

Requirements for API

from [I-D.jesup-rtcweb-data-02]

1. Multiple simultaneous datagram streams, in conjunction with 0 or more media streams
2. Reliable and unreliable
3. Congestion control
4. Prioritization

Changes Since Last Draft

- Tied DataStreams to a PeerConnection
- BaseStream base class
- DataStreams are unidirectional
- Added MTU for unreliable streams
- Better WebSocket alignment
  - Other send() variants
  - bufferedAmount replaces onreadytosend()
  - onsendresult removed

DataStream Design

- Familiar semantics, similar to MediaStream
  - Unidirectional
  - Label
  - Multiplexing
  - Signaling
  - Notifications
- Specific new features to satisfy use cases
  - Sending data (of course)
  - Reliable/unreliable support
  - Prioritization
- Reuses WebSocket idioms, where possible
  - Some fundamental differences (unreliable, unidirectional)

DataStream and MediaStream

- BaseStream class added to unify streams
  - removeStream
  - localStreams, remoteStreams
  - onaddstream, onremovestream
- addStream needs to be distinct (MediaStreamHints)

Changes to PeerConnection

```
interface PeerConnection {
   [...]
   // Creates an adds a data stream, either reliable or unreliable.
   // Label and reliability cannot be changed for a stream after it is created.
   // Will trigger new signaling.
   void addDataStream ( in DOMString label, in boolean reliable);
   // Removes a media or data stream from this PeerConnection.
   // Will trigger new signaling.
   void removeStream (in BaseStream stream);
   readonly attribute BaseStream[]  localStreams;
   readonly attribute BaseStream[]  remoteStreams;
   [...]
};
```

DataStream API

```
interface BaseStream {
  // Type of stream, either "media" or "data".
  readonly attribute DOMString type;
};

interface DataStream {
   [NoConstructor]
   // Label, like MediaStream's |label|. Maps to lower-level stream id.
   readonly attribute DOMString label;
   // Whether this stream has been configured as reliable.
   readonly attribute boolean reliable;
   // The relative priority of this stream.
   // If bandwidth is limited, higher priority streams get preference.
   attribute long priority;
   // States, as in MediaStream.
   const unsigned short LIVE = 1;
   const unsigned short ENDED = 2;
   readonly attribute unsigned short readyState;
   ...
```

DataStream API

```
  ...
  // Indicates the amount of buffered data, as in WebSockets.
  // Only applicable in reliable mode.
  readonly attribute unsigned long bufferedAmount;
  // Indicates the maximum size, in bytes, of outbound messages.
  // Only applicable in unreliable mode.
  readonly attribute unsigned long maxMessageSize;
  // Sends the supplied datagram, as in WebSockets.
  // Returns a nonnegative message id if the message can be sent.
  // Throws a TBD exception if the message is too large.
  long send(in DOMString message);
  long send(in ArrayBuffer data);
  // Called when a message is received, as in WebSockets.
  // Arguments: DOMString/ArrayBuffer message (depending on type of message)
  //            object metadata (with seqnum, timestamp)
  attribute Function onmessage;
};
```

Basic Flow

- ls = pc.**addDataStream** (type, label)
- <signaling>
- [remote] **onaddstream** (rs)
- ls.**sendMessage** (send datagram)
- [remote] **onMessage** (datagram received)

Full Example

```
// create and attach an unreliable data stream
var aLocalDataStream = local.addDataStream("myChannel", false);

// outgoing SDP is dispatched, including a media block like:
m=application 49200 <TBD> 127
a=rtpmap:127 application/html-peer-connection-data
a=datachannel:1 label:myChannel

// this SDP is plugged into the remote onSignalingMessage, firing onAddStream
// of a new unreliable DataStream with label "myChannel"
[remote] onAddStream(aRemoteDataStream);

// signaling completes, and the data stream goes active on both sides;
// we start sending data on the data stream
aLocalDataStream.send("foo");  // sends with seqnum S, timestamp T0

// the message is delivered
[remote] onmessage("foo", { seqnum: S, timestamp: T0 } );

// the data stream is discarded
local.removeStream(aLocalDataStream)

// new signaling is generated
m=application 49200 <TBD> 127
a=rtpmap:127 application/html-peer-connection-data

// resulting in onRemoveStream for the remote
[remote] onremovestream(aRemoteDataStream);
```

Multiparty Example

```
// create and attach a data stream
var aLocalDataStream = local.addDataStream("myChannel", false);

// outgoing SDP is dispatched, including a media block like:
m=application 49200 <TBD> 127
a=rtpmap:127 application/html-peer-connection-data
a=datachannel:1 label:myChannel

// remote side later replies with its own SDP
m=application 49200 <TBD> 127
a=rtpmap:127 application/html-peer-connection-data
a=datachannel:2 label:BsChannel
a=datachannel:3 label:CsChannel

// this results in onaddstream being fired for each data stream
onaddstream(remoteDataStreamB);
onaddstream(remoteDataStreamC);

// we start sending data on the data stream
aLocalDataStream.send("foo");

// the message is delivered to B and C
[remote-B] onmessage("foo", { seqnum: S1, timestamp: T0 } );
[remote-C] onmessage("foo", { seqnum: S2, timestamp: T0 } );
```

Open Issues

- Simple vs full API
  - Stefan's proposal
- Signaling
  - Consensus for multi-channel and reliability to be handled in browser
  - This implies some signaling of channels is needed, to ensure local and remote have agreement
  - Does SDP approach make sense?
- CC feedback to app
  - Reliable can use bufferedAmount
  - Can return error for unreliable
  - Is this sufficient?

Questions?