A proposal for statistics in WebRTC

This version: June 21, 2012

Design goals:

- Well defined metrics: WHAT was measured, WHEN it was measured, BY WHOM it was measured.
- Don't constrain implementation too much
- Make stats from remote endpoints available (if already passed in RTCP)
- Make the stats set extensible - new stats should require a define, not a redesign
- Support the stats currently in use by Hangouts

Proposal

interface RTCPeerConnection {
…. as before
+ RTCStatsRequest getStats([MediaStreamTrack|Object]? selector);
};

The "selector" may be a MediaStreamTrack that is a member of a MediaStream on the incoming or outgoing streams. The callback reports on all relevant statistics for that stream.
If the selector is blank or missing, stats for the whole PeerConnection are reported.
TODO: Evaluate the need for other selectors than MediaStreamTrack.

The returned structure contains a group of StatsElements, each reporting stats for one object that the implementation thinks is relevant for the selector. One achieves the total for the selector by summing over all the elements; for instance, if a MediaStreamTrack is carried by multiple SSRCs over the network, the getStats function may return one StatsElement per SSRC (which can be distinguished by the value of the "ssrc" stats attribute).

A PC MUST return consistent stats for each element in the array, adding new elements to the end as needed; this is needed so that an application can simply correlate a value read at one moment to a value read at a later moment.

(Note: This structure is modelled after the IndexedDB "IDBRequest" interface).

```
interface RTCStatsRequest {
    Object selector;
    RTCStatsReport[] statsElement;
    RTCStatsStructure data;
    enum {"working", "ready"} readyState;
    attribute Function? onsuccess;
}

interface RTCStatsReport {
   readonly statsElement local;
   readonly statsElement remote;
```

```
}

interface RTCStatsElement {
   readonly attribute long timestamp;
   readonly Dictionary stats { statName DOMString, statValue
                           int|bool|float|string }
}

This should give a Javascript structure like this:

[{local: {timestamp:xxx,
          stats:{packetsReceived:nnn, ...}},
  remote: {timestamp:xxx,
          stats:{packetsSent:nnn, ...}}},
 ...]
```

Stats may take time to collect, so the call should not be synchronous. (They should be on-system, but might require stuff like calling out to media processor components to collect). Stats need to be synchronized with each other in order to yield reasonable values in computation; for instance, if “bytesSent” and “packetsSent” are both reported, they both need to be reported over the same interval, so that “average packet size” can be computed as “bytes / packets” - if the intervals are different, this will yield errors.

Design principles

Stats should be as raw as reasonable. If reporting bytes sent, report bytes sent, not bytes sent per second. The caller can compute per second from other info.

Example query for “I get lousy sound, is it packet loss?”?
The sound track is audio track 0 of remote stream 0 of pc1.

```
pc1.getStats(RemoteStreams[0].audioTracks[0], function(stats) {
baseline = stats });
// … wait a byte
pc1.getStats(RemoteStreams[0].audioTracks[0], function(stats) {
   now = stats; processStats();
});

processStats = function() {
   // Real code would:
   // - Check that timestamp of “local stats” and “remote stats”
   //   are reasonably consistent.
   // - Sum up over all the elements rather than just accessing
   //   element zero.
   packetsSent = now[0]{‘remote’}.stats{‘packetsSent’} -
             baseline[0]{‘remote’}.stats{‘packetsSent’};
   // Equivalent:
   packetsSent = now[0].remote.stats.packetsSent -
                 baseline[0].remote.stats.packetsSent;
   packetsReceived = now[0][‘local’].stats{‘packetsReceived’} -
```

```
            baseline[0]['local'].stats{'packetsReceived'};
   fractionLost = (packetsSent - packetsReceived)  / packetsSent;
   // If fractionLost is > 0.3, we have probably found the culprit.
}
```

Alternative ways of computing the same thing are imaginable, given that baseline/now would also contain stats that are based on incoming RTCP reports (timestamped to that report).

Specific stats to track

Not all of these will necessarily be tracked in the first version.
As far as possible, the definitions from RFC 3550 and successors should be used for metrics.

AudioMediaStreamTrack

- SSRC
- Send codec in use (string)
- Bytes sent/received
- Packets sent/received
- Packets lost
- Last Sequence number (RTP timestamp)
- Jitter
- RTT (Round Trip Time)
- Energy level
- Jitter buffer size (rx)
- Echo Return Loss / Echo Return Loss Enhancment

VideoMediaStreamTrack

- SSRC
- SSRC groups (for Simulcast grouping etc)
- Send codec in use
- Bytes sent/received
- Packets sent/received
- Packets lost
- FIRs sent/received
- NACKs sent/received
- Packets retransmitted
- RTT
- Video width/height/fps
- Video current bitrate
- Video target bitrate

PeerConnection

- Bandwidth Estimation
    - Available send bw
    - Available recv bw
    - Total TX bitrate
    - Total RTX bitrate
- Connection Info
    - Total bytes sent/received

- Local candidate
- Remote candidate