Statistics API

Harald Alvestrand

Why statistics?

- It isn't needed to establish connections
- It isn't needed to transfer data

BUT

- Without it, how do you know your service is working?
- Without it, how can you tell whether your assumptions hold true?
- Without it, how do you know what to change?

Statistics are essential.

Primary target: Service provider

- Useful for diagnostics ("is data flowing"), but that's secondary
- Data must be:
  - Well defined
  - Comparable across events
  - Accessible
  - Possible to anonymize
  - Possible to aggregate
- In WebRTC apps, the service provider's only interface is the API

Don't reinvent the wheel

Stats have been collected in RTP-based applications since Day One

Lots of experience exists on its production and aggregation

Leverage that infrastructure

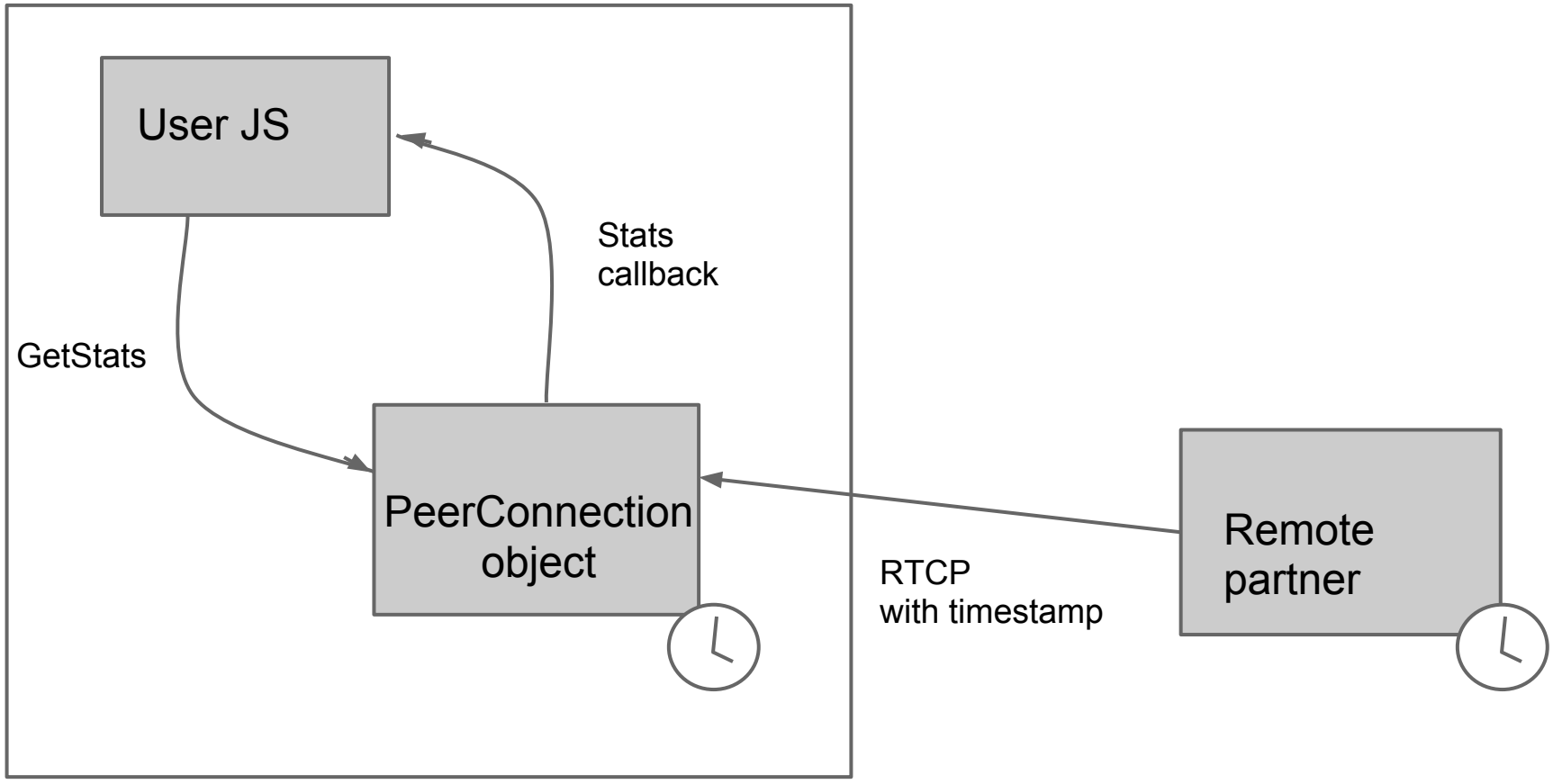
Local and Remote

- In a PeerConnection's MediaStreamTrack, data flows in one direction.
- Information flows in both directions - RTCP feedback.
- The recipient's view is extremely interesting to the sender application.
- When an app calls something other than itself, it has API access only to one end of the connection.
- -> RTCP-carried data must be exposed through the API.

Timing is critical

- We're measuring time-changing data.
  - We deliver packets, not packets per second.
  - Available bandwidth is a constantly changing guess
  - Inferring causality depends on observing sequences
- All measurements must be time-stamped at the measurement point.
- Remote data must be stored with originator's timestamp.

Model for Stats API - calls



Model for Stats API - Data

- Selector - points at a MediaStreamTrack
- Multiple data sets, indexed by observer
  - Local: Observed locally
  - Remote: Observed remotely, reported by RTCP
  - Extensions possible
- Data set mandatory info
  - Timestamp, observer (single value)
- Data items
  - Key/value pairs
  - Key is string, pointer to (IANA?) registry
  - Value is simple data object - string, int, bool....

```
Interface statsStructure {
  Object selector;
  readonly statsReports array [] {
    index enum {'local', 'remote'};
    value statsElements[];
  }
}


interface statsElement {
  readonly attribute long
timestamp;
  readonly enum reporter
    {'local', 'remote'};
    // Other values TBD.
  readonly Dictionary stats {
    statName DOMString,
    statValue int|bool|float|string
  }
}
```

First Version Data

- We can start with (some of?) RFC 3550 stats as "mandatory to implement" (packets, bytes)
- ID correlators like SSRC
- Observable items like remote IP:Port
- Keep It Simple

Extension Points (for later)

- Data items: Anyone with a credible definition can add something that can be observed
  - No app breaks because of too much data
  - Q: How do we know if a data item is unsupported, or just hasn't got a value yet?
- More observation points (if "local" and "remote" are not enough)
- Aggregated statistics (MediaStream, all PC)
- Schedule periodic callbacks?

Difficult Issues

- Not all information is available to browser
  - Delay in OS audio path is often not known
  - Available bandwidth is always an estimate
- Aggregation requires synchronized stats
  - We can't correlate exactly sender and recipient data
- Some Model Mismatches
  - Forward Error Correction streams: Where to count?
  - Removed streams: Where did stats go?
  - Multi-stream tracks: Counted once?

Issues Solved Elsewhere

- Periodic information collection
  - Javascript offers the "setInterval" function
- Storage and aggregation of returned data
  - Application dependent, application responsibility

Comments?