XMPP over WebSocket

Lance Stout

APPS AREA / XMPP WG

IETF 88, Vancouver

see-other-uri

```
<stream:error>
  <see-other-uri
          xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
    wss://example.com/other-endpoint
  </see-other-uri>
</stream:error>
```

see-other-uri

- Where to define it?
- Do we need to make it WebSocket specific?
  <see-other-uri xmlns="urn:ietf:params:xml:ns:xmpp-websocket" />
- What does it mean? Could I meaningfully bump a normal TCP session to use WebSocket or BOSH, or from WebSocket to BOSH?
- What use cases does this solve that existing HTTP redirects and load balancing methods would not?

Stream Start/End

- No streaming parser currently in browsers
- (Now would be the time to poke WHATWG/W3C about that with their talk of a new streams API)
- Have faced this situation before, eg in Flash.
- General problem: XMPP with framing. Other transports will use this too (eg, XMPP over WebRTC data channels).

Proposed Change

- Eliminate the current allowed case of multiple items per WebSocket message

- `<stream:start>`, `<stream:error />`, and `</stream:stream>` would always be in their own messages.

Resulting Code

```
function processData(data) {
    var streamData, start = false;

    data = data.trim();
    data = data.replace(/^(<\?.*\?>\s*)*/, '');
    if (data === '') return;

    if (data.match(self.streamEnd)) {
        self.emit('stream:end');
        return self.disconnect();
    }

    if (!self.streamStart) {
        var parts = data.match(/^<(\S+:)?(\S+) />);
        self.streamEnd = '</' + (parts[1] || '') + parts[2] + '>';
        start = true;
    }

    try {
        streamData = parse(self.streamStart + data + self.streamEnd));
    } catch (e) {
        return self.disconnect();
    }

    if (start) {
        self.streamStart = data;
        self.emit('stream:start', streamData);
    }

    // further library-specific dispatching, unchanged based on stream start/end mechanism
}
```

Alternative

- Use `<stream:open />` and `<stream:close />`

- Must specify all namespaces and lang on each stanza

Resulting Code

```
function processData(data) {
    var streamData;

    data = data.trim();
    data = data.replace(/^(<\?.*\?>\s*)*/, '');
    if (data === '') return;

    try {
        streamData = parse(data));
    } catch (e) {
        return self.disconnect();
    }

    if (streamData.match(‘urn:…?’, ‘close’) {
       self.emit(‘stream:end’, streamData);
       return self.disconnect();
    }

    if (streamData.match(‘urn:…?’, ‘open’) {
       return self.emit(‘stream:start’, streamData’);
    }

    // further library-specific dispatching, unchanged based on stream start/end mechanism
}
```

Compared

```
function processData(data) {
    var streamData, start = false;

    data = data.trim();
    data = data.replace(/^(<\?.*\?>\s*)*/, '');
    if (data === '') return;

    if (data.match(self.streamEnd)) {
        self.emit('stream:end');
        return self.disconnect();
    }

    if (!self.streamStart) {
        var parts = data.match(/^<(\S+:)?(\S+) /);
        self.streamEnd = '</' + (parts[1] || '') + parts[2] + '>';
        start = true;
    }

    try {
        streamData = parse(self.streamStart + data + self.streamEnd));
    } catch (e) {
        return self.disconnect();
    }

    if (start) {
        self.streamStart = data;
        self.emit('stream:start', streamData);
    }

    …
```

```
function processData(data) {
    var streamData;

    data = data.trim();
    data = data.replace(/^(<\?.*\?>\s*)*/, '');
    if (data === '') return;

    try {
        streamData = parse(data));
    } catch (e) {
        return self.disconnect();
    }

    if (streamData.match('urn:…?', 'close') {
       self.emit('stream:end', streamData);
       return self.disconnect();
    }

    if (streamData.match('urn:…?', 'open') {
       return self.emit('stream:start', streamData');
    }

    …
```

Compared

```
function processData(data) {
    var streamData, start = false;

    data = data.trim();
    data = data.replace(/^(<\?.*\?>\s*)*/, '');
    if (data === '') return;

    if (data.match(‘</stream:stream>’)) {
        self.emit('stream:end');
        return self.disconnect();
    }

    try {
        streamData = parse(self.streamStart + data + ‘</stream:stream>’));
    } catch (e) {
        return self.disconnect();
    }

    if (!self.streamStart) {
        self.streamStart = data;
        self.emit('stream:start', streamData);
    }

    …
```

```
function processData(data) {
    var streamData;

    data = data.trim();
    data = data.replace(/^(<\?.*\?>\s*)*/, '');
    if (data === '') return;

    try {
        streamData = parse(data));
    } catch (e) {
        return self.disconnect();
    }

    if (streamData.match(‘urn:…?’, ‘close’) {
       self.emit(‘stream:end’, streamData);
       return self.disconnect();
    }

    if (streamData.match(‘urn:…?’, ‘open’) {
       return self.emit(‘stream:start’, streamData’);
    }

    …
```

By assuming “stream” as prefix

Authentication

- Early consensus has been to use EXTERNAL if using authentication through HTTP headers/ cookies. Still applies? Does EXTERNAL carry too much assumption of using client certs?

- RFC 6120 mandates use of SCRAM-SHA-1 and the -PLUS variant. Browsers don’t expose channel binding information, but could add it if we present the use case for it.

TLS

- The browser is in control of TLS.
- Can’t check for XMPP specific information.
- Implications for the XMPP TLS manifesto?