Open *Source* | Open *Possibilities*

HTML5 Connectivity Methods and Mobile Power Consumption

Giridhar D. Mandyam

November 8, 2012

QUALCOMM INNOVATION CENTER, INC.

Introduction

- Web performance as a science has made great strides w.r.t. mobile browsers, for instance
  - JS engine performance
  - Graphics rendering assessment
  - Hardware-based co-optimizations (e.g. networking stack performance)
- Mobile device power consumption and the browsing experience is not sufficiently-studied
  - Difficult to assess in an automated manner
  - Web developers don't always have a specific focus on portable devices
    - Not a real differentiator ("My website is more power efficient than the next guy's!")

Introduction (cont.)

- Starting in ~2007, handset power consumption started being measured at the OS level beyond standard talk time/standby time
- Example is 1st-gen iPhone's advertised performance

	Preliminary (January 2007)	Final (June 2007)
Talk Time	5 hours	8 hours
Standby Time	-	250 hours
Internet Use	5 hours	6 hours
Video Playback	5 hours	7 hours
Audio Playback	16 hours	24 hours

"iPhone Delivers up to Eight Hours of Talk Time." Apple press release. http://www.apple.com/pr/library/2007/06/18iphone.html. June 18, 2007

Impacts to Mobile HTML5 Features

- Many new features have the potential to drain the battery quickly
  - Video playback (<video>)
  - WebGL
  - New connectivity methods
- This talk focuses on the last item – new connectivity methods in HTML5
  - WebSockets
  - WebRTC

WebSockets and Mobile Battery Life

WebSocket Introduction

- WebSockets introduced into HTML 5 as a means of establishing 2-way communication between a web-based client and a server
- WebSocket(URL) is main interface
  - Has a readyState read only attribute
    - Connecting, Open, Closing, Closed
  - Other read only attributes defined are extensions and protocol
    - These are currently to be spec'ed
  - Propagates three events
    - onOpen, onError, on Close
  - Two methods – send() and close()
    - send() can take three arguments – a string, a blob, or an ArrayBuffer
      - Can access read only attribute bufferedAmount (long) as part of send() handling
- Extensive browser support (Chrome, Firefox, etc.)

Example (Javascript)

```
var connection = new WebSocket('ws://QRTCserver.qualcomm.com');
//ws and wss are new URL schems for websocket and secure websocket respectively

// When the connection is open, send some data to the server
connection.onopen = function () {
  connection.send('Ping'); // Send the message 'Ping' to the server
};

// Log errors
connection.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};

// Log messages from the server
connection.onmessage = function (e) {
  console.log('Server: ' + e.data);
};
```

User Agent Requirements

- IETF has a corresponding spec (RFC 6455)
- UA is not originating a standard HTTPConnection upon WebSocket request
  - HTTP handshake over TCP connection
  - Same connection can be re-used by other web applications connecting to the same server
    - Server may be serving ws:// requests and http:// requests

Client Handshake

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Server Response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Issues With Respect to Keep-Alive

- Keep-alive mechanism not defined in API nor exposed to web developer
- HTTP keep-alive mechanisms do not really apply
  - IETF introduced PING and PONG control frames for keep-alive
    - Generally accepted to be server-initiated, although IETF standard does not prohibit client-initiation
- Ergo JS developers are sometimes advised to created their own keep-alive traffic

"In order to maintain presence, the chat application can send keep-alive messages on the WebSocket to prevent it from being closed due to an idle timeout. However, the application has no idea at all about what the idle timeouts are, so it will have to pick some arbitrary frequent period (e.g. 30s) to send keep-alives and hope that is less than any idle timeout on the path …"

- Wilkins, G. "Is WebSocket Chat Simple?", http://cometdaily.com/2010/03/02/is-websocket-chat-simple/, March 2, 2010

Issues with App-Layer Keep-Alive

- What if the application gets the interval wrong?
  - It could counteract modem mechanisms to preserve power (particularly over cellular connections)
- Example: UMTS Fast Dormancy
  - Handset modem-initiated change in radio resource state (Rel. '99 access)
  - Based upon modem criteria, device can send a message to RAN to downgrade RRC (radio resource control) state
  - Transition to radio idle state can result in power consumption reduction from 200 mA to less than 5 mA

What is the Potential Power Impact?

- Test setup running JS code from Firefox browser to WebSocket/AJAX server outside of firewall
  - Device – Lenovo Thinkpad T420
  - HSPA (AT&T) access
- Web client would start with WebSocket connection and fall back to XHR (AJAX) when battery level below pre-determined threshold
- Comparison
  - WebSocket keep alive message sent every 3 seconds
  - AJAX request sent every 20 seconds
  - Rate of power reduction reduced from .5% per minute to .2% per minute

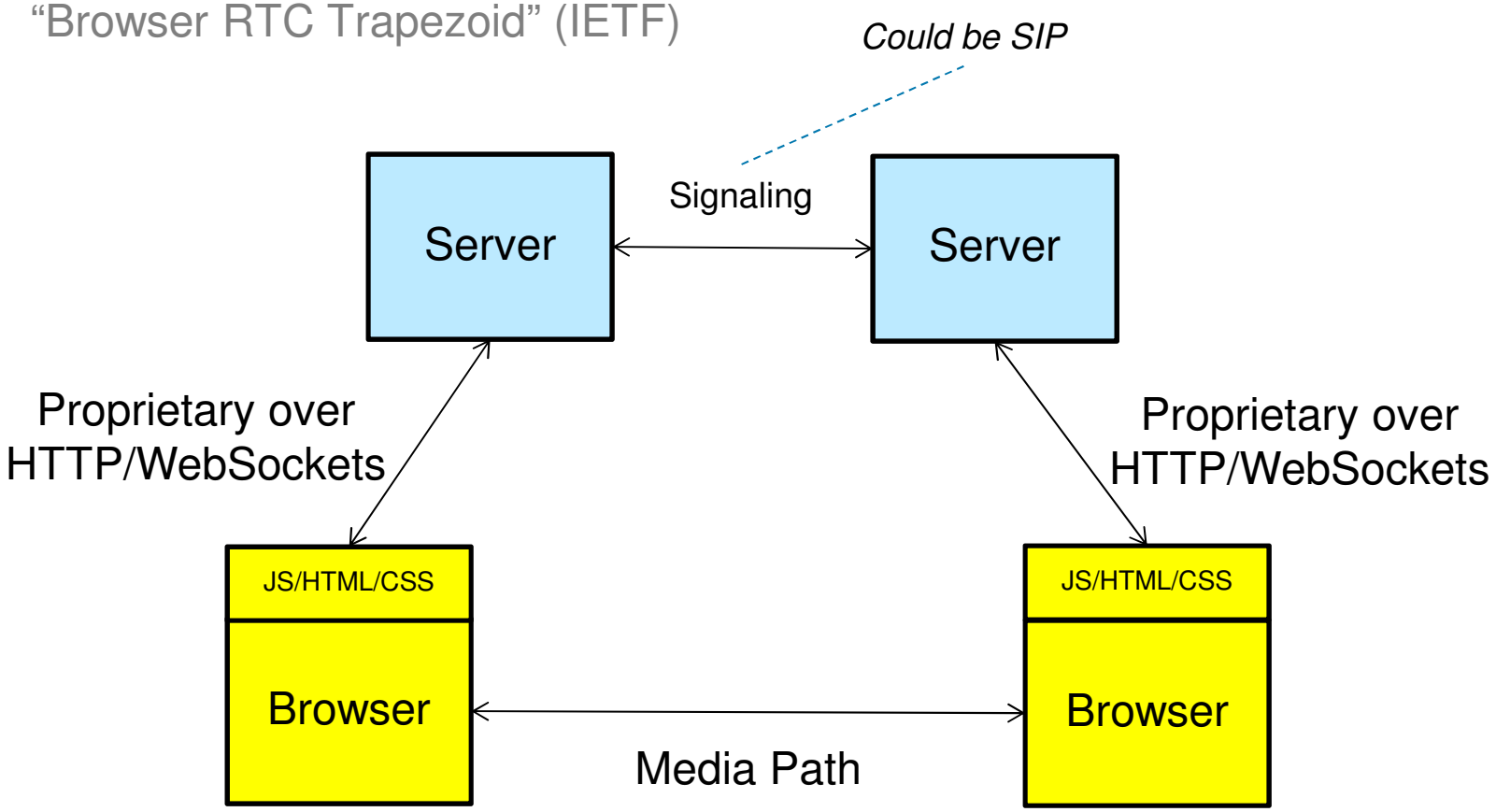
WebRTC and Cellular (LTE) Implications

Introduction

- WebRTC is shorthand for Web Real Time Communications
  - Standard for interoperability between browsers that can enable P2P streaming sessions
    - VoIP, video telephony, app-to-app streaming (opaque data streaming)
- WebRTC has two main standardization tracks
  - World Wide Web Consortium (W3C)
    - WebRTC working group currently defining API (DOM extensions) necessary for web developers to access communications capability from the user agent (browser)
  - IETF
    - RTCWeb working group in charge of defining underlying protocol specification
- WebRTC has some parallels to the WebSockets standardization effort
  - Client-server communications via the browser

Introduction (cont.)

"Browser RTC Trapezoid" (IETF)



WebRTC Goals

Javascript Specific

- API must provide the following functionality to web developers
  - Representation of multimedia streams with multiple possible origins
    - Pre-recorded, file-based, originating from local devices (camera, microphone)
  - Local recording (capture) capability
  - Connecting to remote peers using NAT-traversal
    - Dependence on IETF standards (ICE, STUN and TURN)
  - Sending locally captured or produced streams to remote peers
  - Receiving streams from remote peers
  - Sending/receiving arbitrary (opaque) data to remote peers
- Two main Javascript methods are defined to address these goals
  - getUserMedia() and PeerConnection()

Cellular-Specific Implications

- Ideally WebRTC sessions would leverage all QoS mechanisms available
  - DSCP settings on IP packets
  - Link-layer mechanisms
    - Bandwidth, per-packet delay guarantees
- The focus of WebRTC deployment will be on newer cellular technologies
  - LTE, i.e. “Long-Term Evolution”: also sometimes called “4G”
  - Based on orthogonal frequency division multiplexing (OFDM)
    - Requires statistical multiplexing of individual user’s traffic on the physical layer (air interface) regardless of the service involved
      - 3G communications allowed for circuit-switched connections for voice telephony

Cellular-Specific Implications (cont.)

- Telephony (video or voice) services can be considered a special case when trying to multiplex multiple users in a shared resource
  - LTE provides QoS Class Identifiers (QCI) for QoS-sensitive services
  - QCI 1 for LTE telephony involves a guaranteed bit rate, maximum packet delay and maximum packet loss
  - WebRTC sessions could leverage QoS or not
    - No QoS is sometimes referred to as an “over-the-top” (OTT) service
- QoS actually has implications to mobile battery life
  - Based on the scheduling of user traffic

LTE and VoIP Service Scheduling

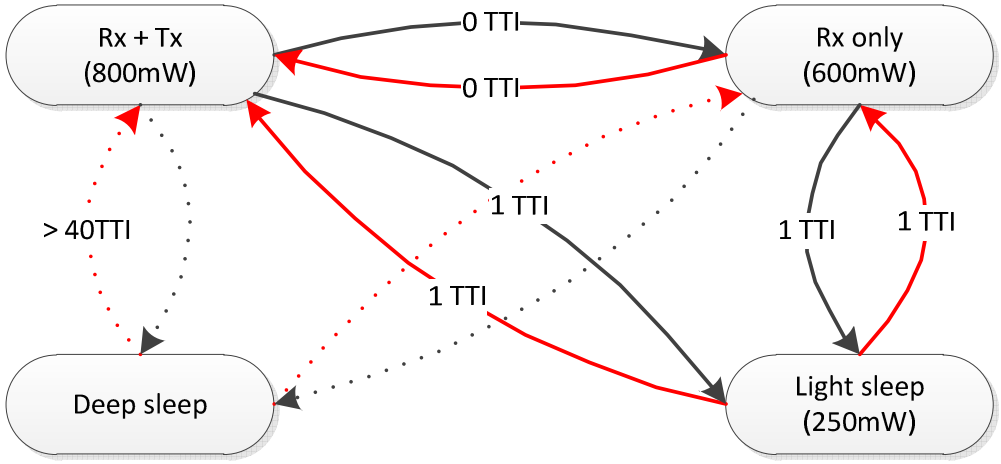
- Traffic model assumes a “Talk” and “Listen” state
- Power consumption tends to be highest during the “Talk” state
  - Simultaneously sending and receiving data means that the wireless device is run at highest power levels
- For upstream talk bursts, in LTE the handset sends scheduling requests (SR) to the base station
  - The base station schedules the user for both uplink (upstream) transmission and downlink (downstream) reception as a result
  - The handset monitors a downlink control channel to determine when it is allocated radio resources for transmission
- Downlink reception is also time multiplexed, meaning the handset can benefit from discontinuous reception (DRX)

LTE and VoIP Service Scheduling (cont.)

- Link allocations are performed on the basis of transmission time intervals (TTI's)
  - 1 TTI = 1 ms
- In general, there are two types of scheduling: dynamic and semi-persistent
- Dynamic
  - SR is sent whenever there is new data arrival
  - SR periodicity limits the UE transmission opportunities on the uplink
  - For downlink, the DRX duration limits the receive opportunities
- Semi-persistent (SPS)
  - The allocations are provided at 20 or 40 ms periodicity

Power State Modeling

- Power state 1: Handset actively monitoring downlink (DL) and transmitting on the uplink (UL)
- Power state 2: Handset actively monitoring DL, but there are no uplink transmissions. Transmit (Tx) chain is turned off in this state.
- Power state 3 (light sleep): Handset not monitoring DL or transmitting on UL. Both Receive (Rx) and Tx are turned off.
- Power state 4 (deep sleep): Handset not monitoring DL or transmitting on UL. RF and modem are both shut down. The only modem power consumption is due to leakage.



Power Difference Between Dynamic and Semi-Persistent Scheduling: impact to WebRTC

- Please refer to paper for detailed analysis
- Dynamic scheduling of VoIP results in 489.75 mW consumption
- Semi-persistent scheduling results in 394 mW
  - Savings of ~20%
- What does it mean for WebRTC?
  - WebRTC services that are deployed like any other OTT VoIP service will most likely not leverage semi-persistent scheduling
  - Applying cellular QoS for WebRTC should allow for SPS
    - Power consumption should approach operator-deployed VoIP services over LTE (VoLTE)

Ways Forward for Web Performance Efforts in the W3C

- Development of best practices for mobile web that consider power consumption
- Ensure minimum performance for implementations of new W3C battery API so that developers can leverage during runtime
  - It shouldn't be just for battery monitoring apps
- Indication to web developers whether cellular QoS is being leveraged in a persistent connectivity session
  - Explicit indication through an API is a possibility, but this may also be discerned through existing API's
- Explicit metrics regarding the state of the connection propagated through Javascript
  - Extending what was started with Navigation Timing, e.g. adding packet loss percentage

Thank You

QuIC
QUALCOMM INNOVATION CENTER, INC.