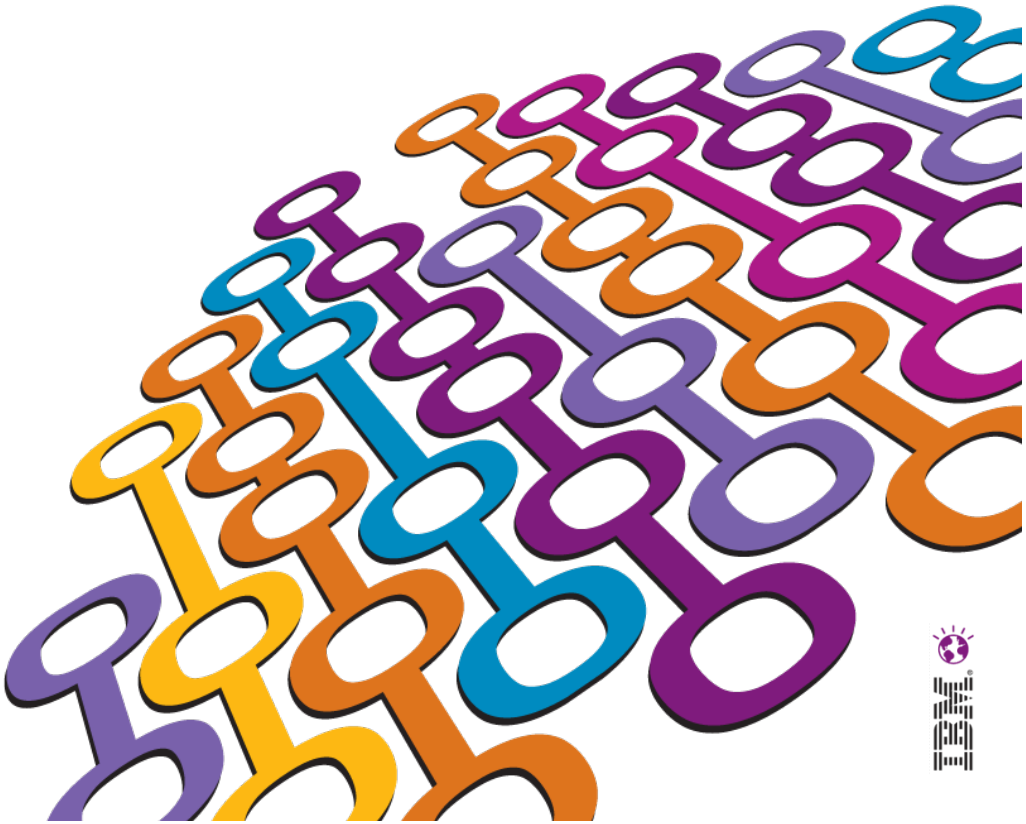
Low Latency Messaging for Mobile Apps – Or When HTTP and Push Notifications Are Simply Not Enough

Using MQ Telemetry Transport (MQTT) for Mobile Applications



Henrik Sjöstrand

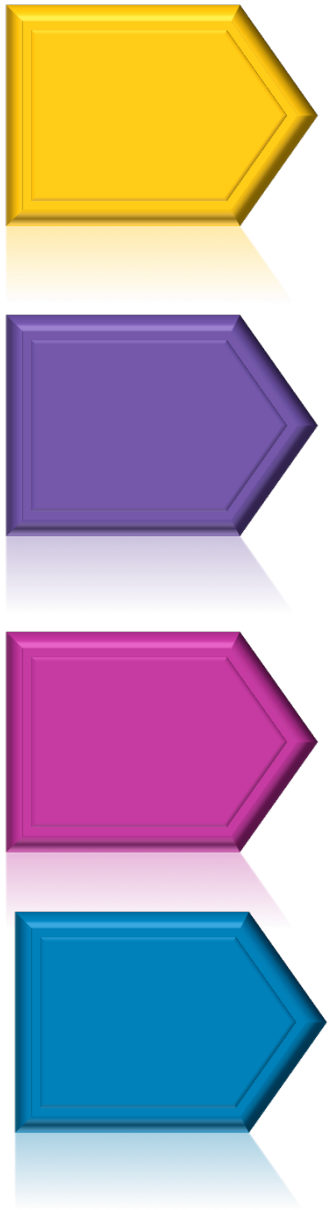
CTO Mobility Nordic, WebSphere Group

IBM Software Group, Sverige

henrik@se.ibm.com

Följ @HenrikSjostrand

Agenda



- Impact of Mobile on the traditional Web
- MQ Telemetry Transport (MQTT) overview
- Protocol deep dive
- How do you get started?

Four major waves of technology – Mobile is the next transformation

Back-Office Computing

Client-Server PC - 1981

World Wide Web and eBusiness

Confluence of Mobile, Social, Cloud, Big Data / Analytics

60's 80's 90's We are here

Mobile is Different from Desktop

Desktop		Mobile
Sit back and read	→	Read while moving
Document-oriented	→	Message-oriented
Large complex apps	→	Purpose-built mini-apps
Context-neutral	→	Context-aware
Task-driven	→	Notification-driven
Mains powered	→	Battery-powered
Predictable network response	→	Unpredictable network response

Mobile is Driving Changes to the Traditional Web

HTML4 → **HTML5**

- Applications need to be online
- Browser plugins for additional application function (e.g., Flash, Silverlight)

- Offline execution
- No plugins required.

HTTP → **WebSockets?**

- No built in reliability
  - Loses data on brittle networks
- Polling oriented
  - Unnecessary additional communications
  - Wastes battery life
- Chatty
  - Each message involves lots of back and forth
- One directional invocations
  - Client always initiates the communications.

- Bi-directional data exchange
  - Useful for push and streaming data (i.e. stock-ticker, media)
- **But missing key function**
  - A low level TCP socket; not a full blown messaging protocol
  - No assured delivery
  - No built-in publish / subscribe
  - Requires coding of message handling on top of it.

Notification Technologies for Mobile

- Ubiquitous (supported by every phone/telco)
- Expensive, telcos charge per message
- Limited text-based notification
- In the clear (traffic read by Telcos)

- Not interoperable; different per Mobile OS
  - APNS (Apple), GCM (Google), BBM (Blackberry)
- Requires a data plan
- Limited text-based notification
- Not suitable for confidential information (flows through Google, Apple, Blackberry)
- Transmission is one-way, not interactive
- No assured delivery

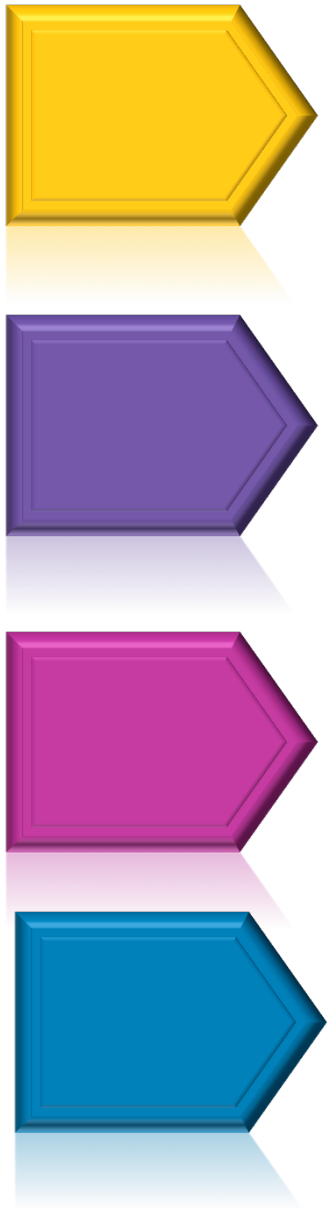
Facebook Messenger

*"One of the **problems** we experienced was **long latency** when sending a message. The method we were using to send was reliable but **slow**, and there were **limitations** on how much we could improve it. With **just a few weeks** until launch, we ended up building a new mechanism that maintains a persistent connection to our servers. To do this without **killing battery life**, we used a protocol called **MQTT** that we had experimented with in Beluga. MQTT is specifically designed for applications like sending telemetry data to and from space probes, so it is **designed to use bandwidth and batteries sparingly**. By maintaining an MQTT connection and routing messages through our chat pipeline, we were able to often achieve **phone-to-phone delivery in the hundreds of milliseconds, rather than multiple seconds**."*

-Lucy Zhang, Facebook Engineer 10/19/11 www.facebook.com/lucyz
(MQTT used by their 350M mobile users, 475 mobile operators)

http://www.youtube.com/watch?v=aJo5jG0eKtI&hd=1

Agenda



Impact of Mobile on the traditional Web

MQ Telemetry Transport (MQTT) overview

Protocol deep dive

How do you get started?

MQTT: Key Features

Open

- Open published spec designed for the world of “devices”
  - Invented by IBM and Eurotech
  - MQTT client code (C and Java) donated to the Eclipse "Paho" M2M project

Lean

- Minimized on-the-wire format
  - Smallest possible packet size is 2 bytes
  - No application message headers
- Reduced complexity/footprint
  - Clients: C=30Kb; Java=100Kb

Reliable

- Three qualities of service:
  - 0 – at most once delivery
  - 1 – assured delivery but may be duplicated
  - 2 – once and once only delivery
- In-built constructs to support loss of contact between client and server.
  - “Last will and testament” to publish a message if the client goes offline.
- Stateful “roll-forward” semantics and “durable” subscriptions.

Simple

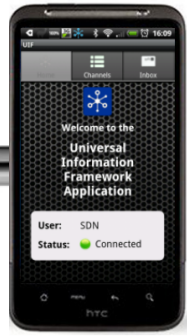
- Simple / minimal pub/sub messaging semantics
  - Asynchronous (“push”) delivery
  - Simple set of verbs -- connect, publish, subscribe and disconnect.

MQTT: Key Features

Scalable

- 240,000 concurrent clients tested with <5% CPU on a single IBM WebSphere MQ queue manager
- By comparison:
  - Apache Web Servers max out at 25,000 connections

Secure



- Direct connection between your enterprise and devices
- Network: TLS/SSL
- Authentication: JAAS
- Authorization: OAM

MQTT uses (much) less bandwidth than HTTP

IBM Hursley Lab

Scenario	HTTP	MQTT
1. Get a single piece of data from the server	302 bytes	69 bytes (~4 times)
2. Put a single piece of data to the server	320 bytes	47 bytes (~7 times)
3. Get 100 pieces of data from the server	12600 bytes	2445 bytes (~5 times)
4. Put 100 pieces of data to the server	14100 bytes	2126 bytes (~7 times)

European automobile manufacturer

Vehicle Telematics

Mobile Network Estimated Data Costs/Vehicle/Year*

HTTP	MQTT
220€/vehicle /year	23€/vehicle /year

*Comparison based on 100 messages/day, 200Bytes/Msg payload, 1-2€ /100MB TCP transfer costs.

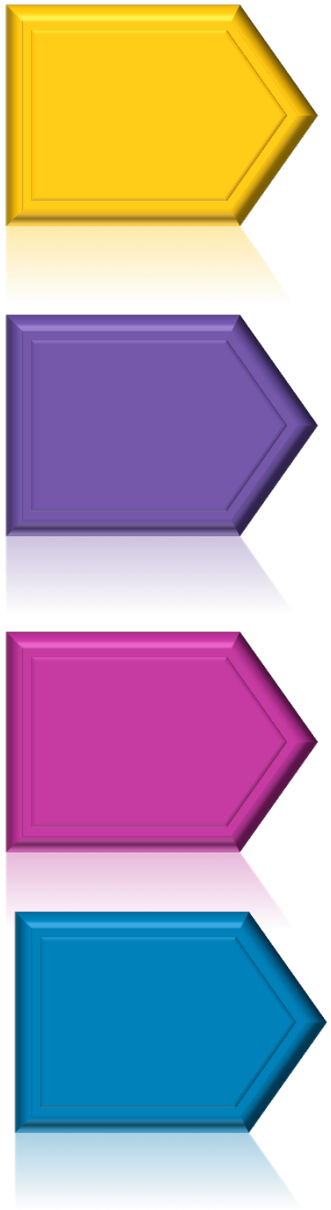
MQTT Low Power Usage

- Restriction on some devices, such as cell phones may simply be available power, how does MQTT use power?
  - HTC Desire Android mobile phone

	% Battery / Hour	
Keep Alive (Seconds)	3G	Wifi
60	0.77641278	0.0119021
120	0.38884457	0.0062861
240	0.15568461	0.00283991
480	0.07792208	0.00134018

- Protocol allows tuning to suit devices

Agenda



Impact of Mobile on the traditional Web

MQ Telemetry Transport (MQTT) overview

Protocol deep dive

How do you get started?

MQTT Protocol Deep Dive - Headers

- MQTT protocol flows:
  - Fixed header (2 bytes)
  - Variable header (optional, length varies)
  - Message payload (optional, length encoded, up to 256MB)

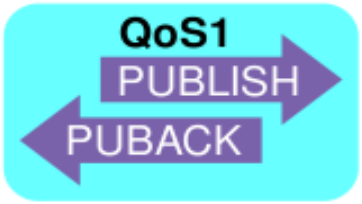
- Fixed header indicates the API call, the length of the payload and Quality of Service
- Variable header contents depends on API call defined in the fixed header
  - Message ID, Topic name, client identifier and so on.

Qualities of Service

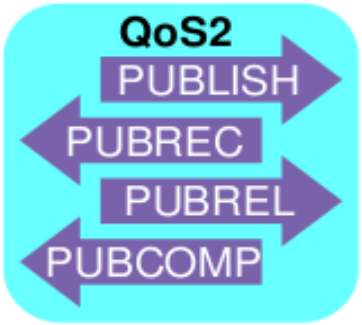
QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	<=1
1	0	1	At least once	Acknowledged delivery	>=1
2	1	0	Exactly once	Assured delivery	=1
3	1	1	Reserved		

- **QoS 0: At most once delivery (non-persistent)**
- – No retry semantics are defined in the protocol.
- – The message arrives either once or not at all.

- **QoS 1: At least once delivery (persistent, dups possible)**
- – Client sends message with Message ID in the message header
- – Server acknowledges with a PUBACK control message
- – Message resent with a DUP bit set if the PUBACK message is not seen

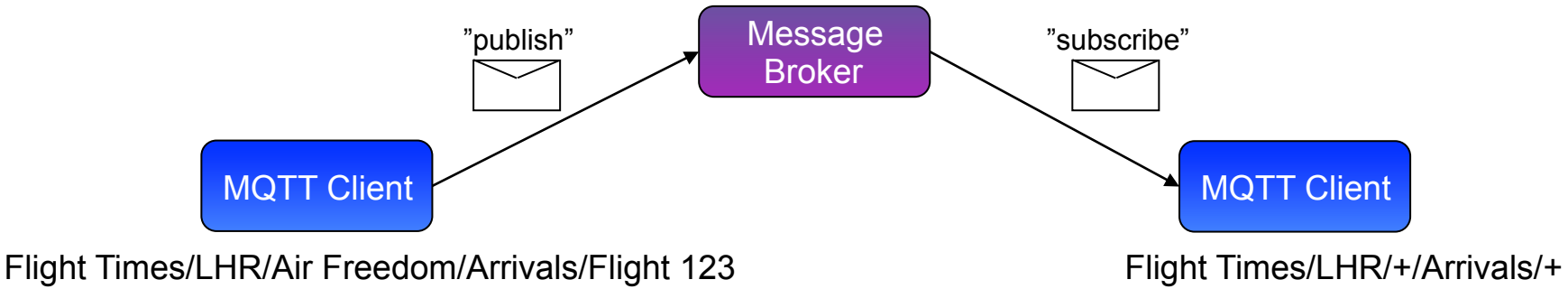


- **QoS 2: Exactly once delivery (persistent)**
- – Uses additional flows to ensure that message is not duplicated
- – Server acknowledges with a PUBREC control message
- – Client releases message with a PUBREL control message
- – Server acknowledges completion with a PUBCOMP control message



MQTT Topics

- All subscriptions are to a topic space
- All messages are published to an individual topic
- Topic names are hierarchical
  - Levels separated by “/”
  - Single-level only wildcards “+” can appear anywhere in the topic string
  - Multi-level (whole subtree) wildcards “#” must appear at the end of the string
    - Wildcards must be next to a separator
    - Can't use wildcards when publishing
- MQTT topics can be 64KB long



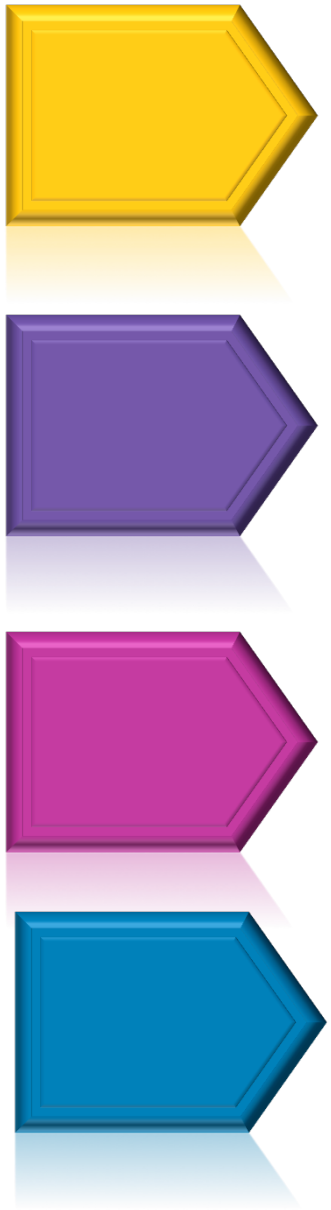
MQTT Keep Alive

- Protocol includes support for client and server to detect failed connections
  - At connection time, a keep alive can be specified
- If client does not send a PINGREQ request to the server, the server assumes the connection has failed.
- If client does not receive a PINGRESP after the keep alive time has passed after sending the request, the connection is failed.
- Maximum keep alive interval of 18 hours.
  - Can specify a value of 0 to disable keep alive

MQTT Last Will & Testament

- During connection, a Will message and topic can be specified
  - Abnormal disconnections will cause WMQ to publish the message
  - Clean disconnects will not cause the message to publish
- Can set the message as retained
  - Message is published to a subscriber when registering
- Useful to report the connection status of the client
  - Will message is a retained “down”
  - Upon connecting, client publishes a retained “up” message.

Agenda

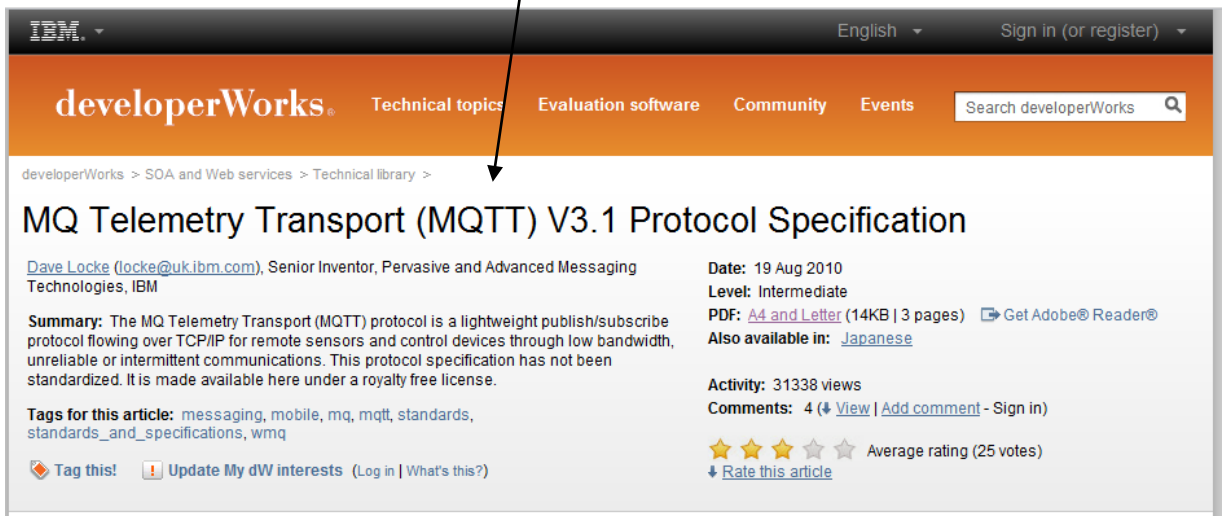


Impact of Mobile on the traditional Web

MQ Telemetry Transport (MQTT) overview

Protocol deep dive

How do YOU get started?



Servers/Brokers

- IBM WebSphere MQ (incl Telemetry)
- IBM WebSphere Message Broker
- IBM Lotus Expeditor micro broker
- Really Small Message Broker
- Mosquitto
- Eclipse Paho
- Eurotech Everyware Device Cloud (EDC)
- Pachube
- MQTT.js
- eMQTT
- m2m.io

Clients

- Device-specific
  - Arduino client for MQTT
  - mbed client for MQTT
  - Nanode MQTT
  - Netduino MQTT
- C
- C++
- Delphi
- Erlang
- Java
- JavaScript / node.js
- LotusScript
- Lua
- .NET
- Perl
- PHP
- Python
- REXX
- Ruby

eclipse.org/paho

paho

Paho is an m2m.eclipse.org project +1 28

WHAT IS PAHO?

The Paho project provides scalable open-source implementations of open and standard messaging protocols aimed at new, existing, and emerging applications for Machine-to-Machine (M2M) and Internet of Things (IoT).

Mailing List | Wiki | Bug Tracker | Source Code

For Constrained Networks

M2M and IoT systems need to deal with frequent network disruption and intermittent, slow, or poor quality networks. Minimal data costs are crucial on networks with millions and billons of connected devices.

Devices and Embedded Platforms

Devices and edge-of-network servers often have very limited processing resources available. Paho understands small footprint clients and corresponding server support.

Highly Scalable

Paho focuses on highly scalable implementations that will integrate with a wide range of middleware, programming and messaging models.

Documentation

Quick Start Guides

A list of quick start guides is maintained on the Paho Wiki.

Protocols

MQTT

The specification for the MQ Telemetry Transport protocol is available on

MQTT clients for Native and JavaScript Mobile Apps

- Native Android
  - Use the MQTT Java client and samples
  - This client provides synchronous and asynchronous APIs for MQTT
  - Available as open source from Eclipse Paho, or as supported code from IBM
  - IBM provides a sample Android service, that lets the MQTT client continue running even when the app is in the background

- Native iOS
  - Use the MQTT C Client and samples. This can be called from Objective C
  - This client provides synchronous and asynchronous APIs for MQTT
  - Available as open source from Eclipse Paho, or as supported code from IBM

- JavaScript
  - For applications running in Web Browsers (mobile or static)
  - For cross-platform hybrid apps (Android, iOS, Windows Phone or BB10)
  - Clients provide asynchronous APIs for MQTT

JavaScript options

- WebSocket implementation – the Messaging Client for JavaScript
  - Implementation is 100% JavaScript (single JavaScript file to include in your app)
  - Ideal for Browser-based apps (downloads with the app itself)
  - Supported on all modern Web Browsers
  - Runs in iOS 6.0 and Android 4.2.3, Blackberry 10, Windows Phone 8
  - Requires MQ 7.5.0.1 or MessageSight server implementation
- Cordova implementation – the Cordova plugin for Android
  - Uses Cordova technology to bridge between Javascript and the native Paho client
  - Runs on all versions of Android from 2.2 onwards
  - Allows app to continue running in the background
  - Runs against MessageSight and all versions of MQ from 7.1 upwards
- Both clients present the same API to app developer
  - Single line change in the HTML file to select which client to load

MQTT Broker Commercial alternative 1: IBM WebSphere MQ includes Telemetry component

- New WebSphere MQ Telemetry component – also known as MQXR ('eXtended Reach')
- Included in WMQ since v7.1
- Fully integrated / interoperable with WMQ
  - MQTT messages translated to standard WMQ messages
  - Administration included as part of WebSphere MQ Explorer
- Telemetry channels enable MQTT connections to the queue manager
  - Supports MQTTv3 protocol (most common in use)
- Highly scalable
  - 100,000+ clients
- Rich Security
  - SSL channels
  - JAAS authentication
- Ships with reference Java (for MIDP upwards) and C clients
  - Small footprint clients
  - other APIs and implementations of MQTT available via 3rd parties

http://www.ibm.com/software/integration/wmq/

WebSphere. software

MQTT Broker Commercial alternative 2: IBM MessageSight

- A secure, easy to deploy messaging server that is optimized to address the massive scale requirements of the machine to machine (M2M) and mobile use cases
- A ***million*** connections, and ***millions*** of messages per second
- Designed to sit at the edge of the enterprise, it can extend your existing messaging infrastructure or be used standalone.
- Connectivity to existing WebSphere MQ and to messaging clients using MQTT and JMS

The JavaScript Messaging API

© 2013 IBM Corporation

JavaScript Messaging API

Client Object – represents the connection with the server

- Properties
  - Hostname, Port, ClientId
- Events
  - onMessageArrived, onMessageDelivered, onConnectionLost
- Methods
  - Connect, Disconnect, Send, Subscribe, Unsubscribe

Message Object – represents data sent or received

- DestinationName
- Quality of Service (QoS)
- Retained flag
- Duplicate flag
- Payload (string or binary)

© 2013 IBM Corporation

Connecting to the Server


```
function connect(form) {
    try {
        client = new Messaging.Client(
            form.host.value,
            Number(form.port.value),
            form.clientId.value);
    } catch (exception) {
        alert("Exception:"+exception);
    }
    client.onConnect = onConnect;
    client.onMessageArrived = onMessageArrived;
    client.onConnectionLost = connectionLostCallback;

    client.connect();
}
```

Create client

Set callbacks

Connect to the server

© 2013 IBM Corporation

Sending a Message


```
function doSend(form) {
  message = new
      Messaging.Message(form.textMessage.value);

  message.destinationName =
      form.topicName.value;

  client.send(message);
}
```

Create a Message object

Set the Topic

Send the message

© 2013 IBM Corporation

Subscribing and Receiving


```
function subscribe(form)
  {     client.subscribe(form.subscribeTopicName.va
  lue);
}
```

Subscribe to a topic

```
function onMessageArrived(message) {
   var form = document.getElementById("output");
  form.receivedMessage.value =
       message.payloadString;
}
```

Display the message's payload

© 2013 IBM Corporation

Success and Failure

Success / Failure are signalled by callbacks registered when the API call is made. You can associate an InvocationContext with a callback

```
var subscribeOptions = {qos:2,
      invocationContext:{text:"Subscription 1"},
  onSuccess:onSuccess,
  onFailure:onFailure};
 client.subscribe("/World", subscribeOptions);
};

function onSuccess(resp) {
console.log("OK:"+resp.invocationContext.text);
};

function onFailure(resp)
  {    console.log("Failed:"+resp.invocationContex
  t+" "+resp.errorCode +" "+ +resp.errorReason);
};
```


© 2013 IBM Corporation

Asynchronous programming


```
Function connect(form) {
…
  client.connect({onSuccess:onConnect});
  // client won't yet be connected !
}

function onConnect() {
  console.log("it's now connected");
  client.subscribe("/World");
  message = new Messaging.Message("Hello");
  message.destinationName = "/World";
  client.send(message);
};
```

Warning: API calls like connect and subscribe are Asynchronous. You can't assume that they have completed until you have been called back. In single threaded environments (like phones) this won't happen till after you have relinquished control of your thread

© 2013 IBM Corporation

Links

- MQTT.org - documentation and software
  http://mqtt.org
- Eclipse paho project (IBM's MQTT reference implementation)
  http://www.eclipse.org/paho/
- IBM WebSphere MQ
  http://www.ibm.com/software/integration/wmq/
- IBM Worklight
  http://www.ibm.com/software/mobile-solutions/worklight/

Henrik Sjöstrand
CTO Mobility Nordic, WebSphere
IBM Software Group, Sverige
henrik@se.ibm.com

Följ @HenrikSjostrand