#SHAREorg

SHARE
Technology · Connections · Results

BIG Connectivity and Mobility with WebSphere MQ

Session 13923
Wednesday 14th August 2013

Chris J Andrews
IBM

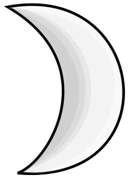
SHARE in Boston

Agenda

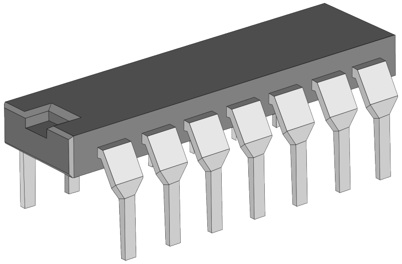
- Communication between Digital Devices
- MQTT
- WebSphere MQ Extended Reach (MQXR)
- MessageSight
- WebSphere MQ HTTP Bridge
- Live Demonstration of MQTT, MQXR and JavaScript

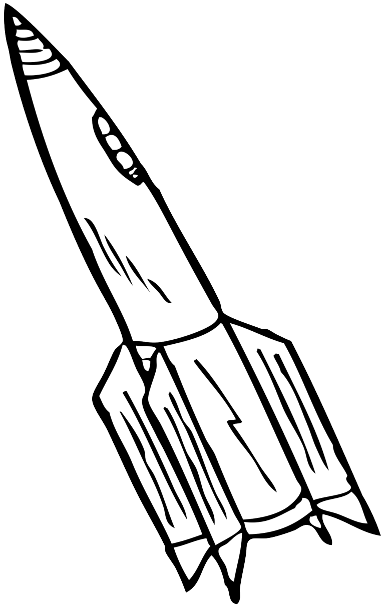
Embedded Digital Devices

Digital devices have been embedded into systems since the 1960s.



One of the earliest recorded uses was in the “Apollo Guidance Computer”.





The Inter-Continental Ballistic program of the 60s was responsible for a dramatic drop in the cost of Integrated Circuits.

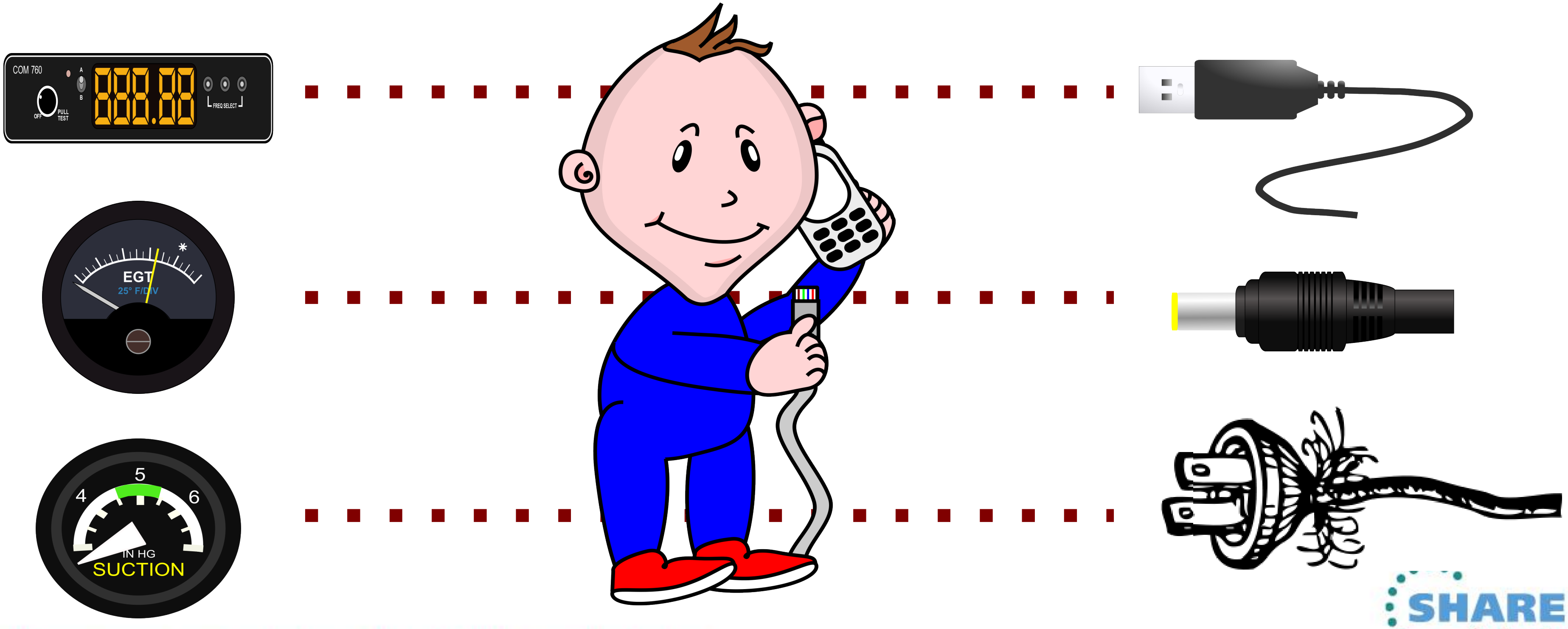
For example, the 'Minuteman' missile program alone is claimed to have reduced the price of nand gate ICs to $3 per unit, down from $1000 at the start of the program.

Universal Connectivity?

Until recently, common connectivity has not been high on the agenda!

For example, if you bought a sensor from manufacturer 'A', you plug it into a gauge also bought from manufacturer 'A'. The fittings and communication protocols were likely to be proprietry.

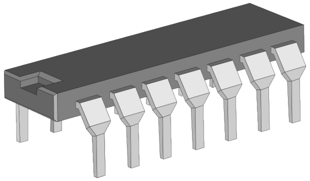
… And this has generally worked out fine.

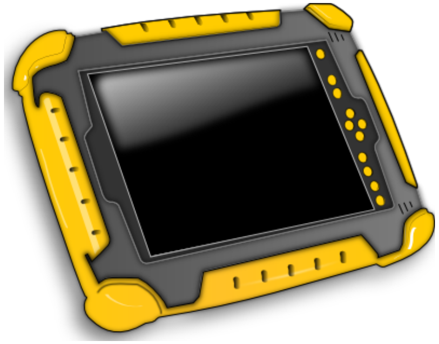


Embedded and Mobile Devices

However things are rapidly changing. The proliferation of devices has risen dramatically in recent times:

- Popularisation of custom embedded circuitry

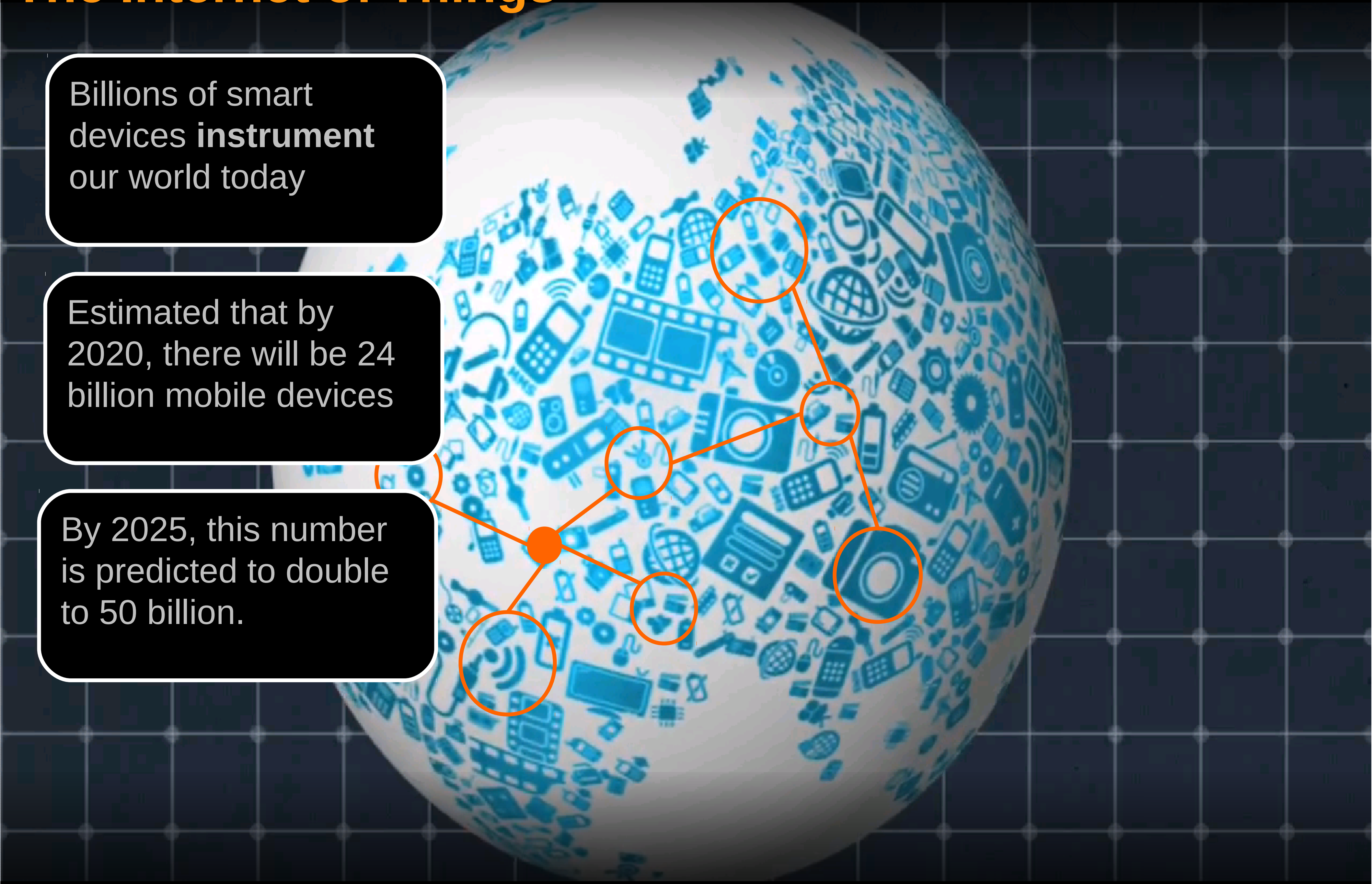




- Mobile Phones / Tablets

- Appreciation by industry as to the possibilities of making data available to the user

The Internet of Things



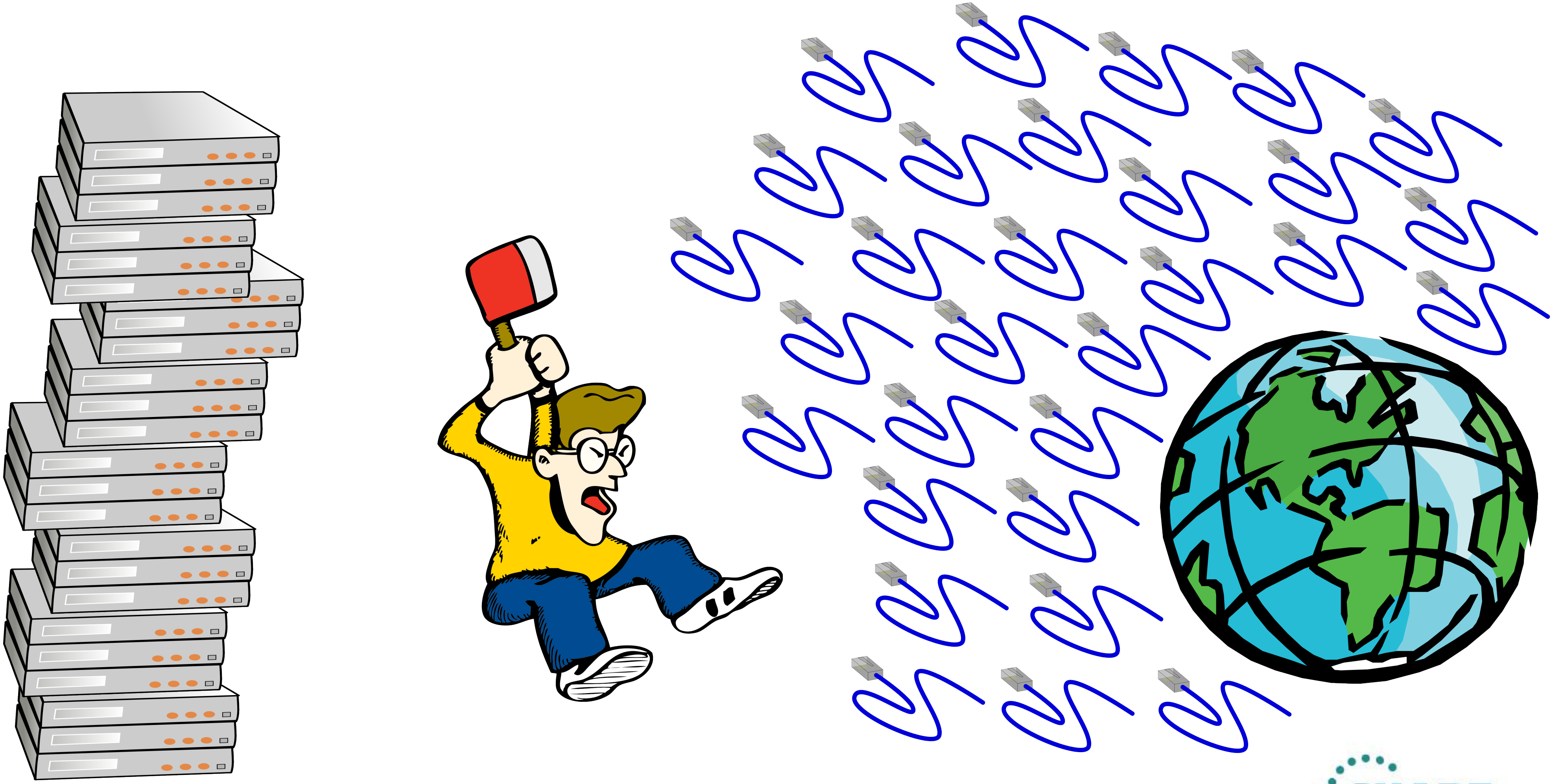
Billions of smart devices **instrument** our world today

Estimated that by 2020, there will be 24 billion mobile devices

By 2025, this number is predicted to double to 50 billion.

Connecting Billions of Devices

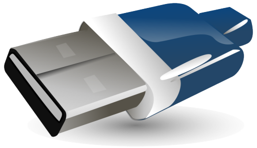
It is simply not practical to attempt to connect billions of devices using different proprietry solutions! This type of connectivity does not scale.

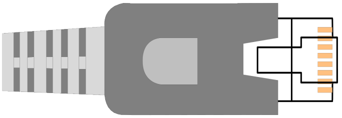


Connectivity, the Hardware Story

Thankfully, a small number of hardware standards have been granted mass popularity. For example:

Universal Serial Bus (USB)



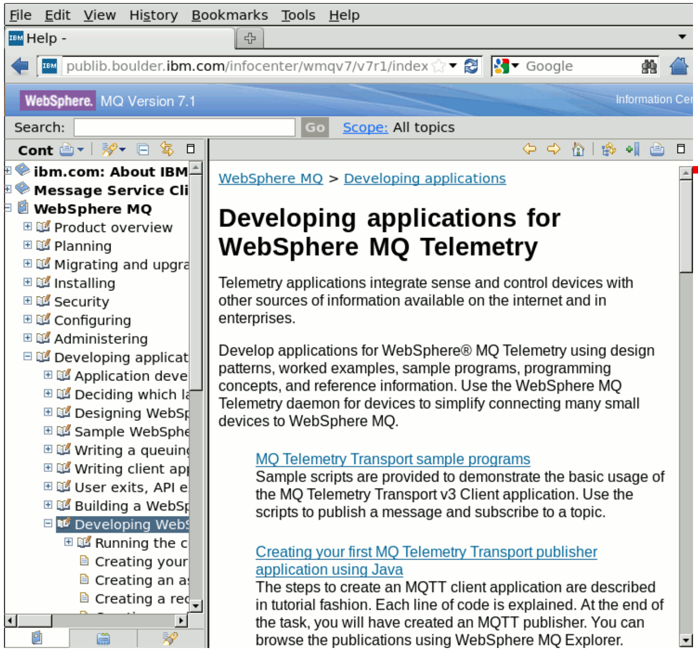


Ethernet (→ TCP/IPv4/v6)

Wireless Communications,
WiFi, Phone/Satellite comms (→ TCP/IPv4/v6)

The communication transport medium is more or less standarised. **But what about the software protocols?**

Internet Communication



WebSphere MQ > Developing applications

Developing applications for WebSphere MQ Telemetry

Telemetry applications integrate sense and control devices with other sources of information available on the internet and in enterprises.

Develop applications for WebSphere® MQ Telemetry using design patterns, worked examples, sample programs, programming concepts, and reference information. Use the WebSphere MQ Telemetry daemon for devices to simplify connecting many small devices to WebSphere MQ.

MQ Telemetry Transport sample programs
Sample scripts are provided to demonstrate the basic usage of the MQ Telemetry Transport v3 Client application. Use the scripts to publish a message and subscribe to a topic.

Creating your first MQ Telemetry Transport publisher application using Java
The steps to create an MQTT client application are described in tutorial fashion. Each line of code is explained. At the end of the task, you will have created an MQTT publisher. You can browse the publications using WebSphere MQ Explorer.

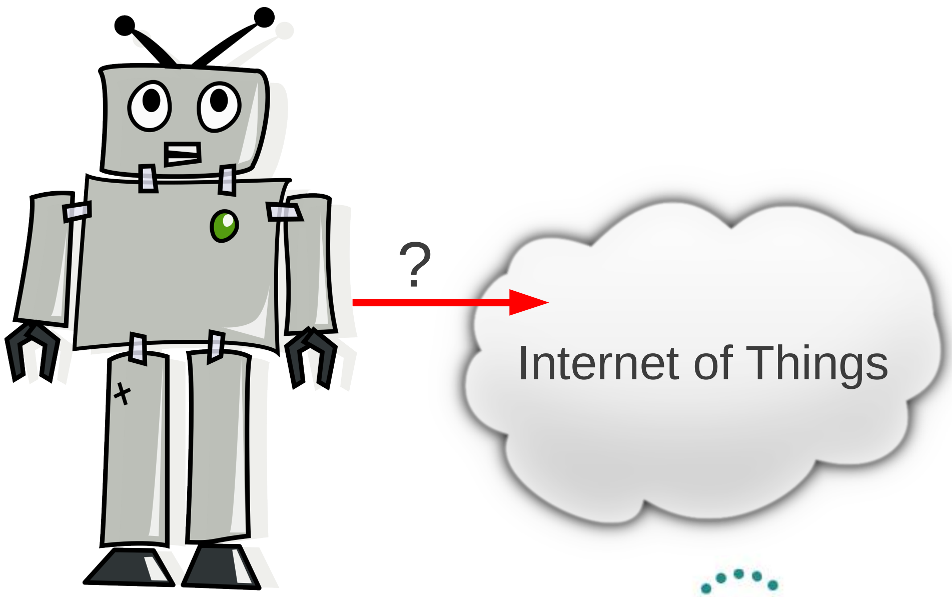
HTTP

WWW

HTTP serves as the de-facto protocol for communication between browsers and the internet

What protocol should machines use to communicate with each other?

A common Machine to Machine (M2M) protocol

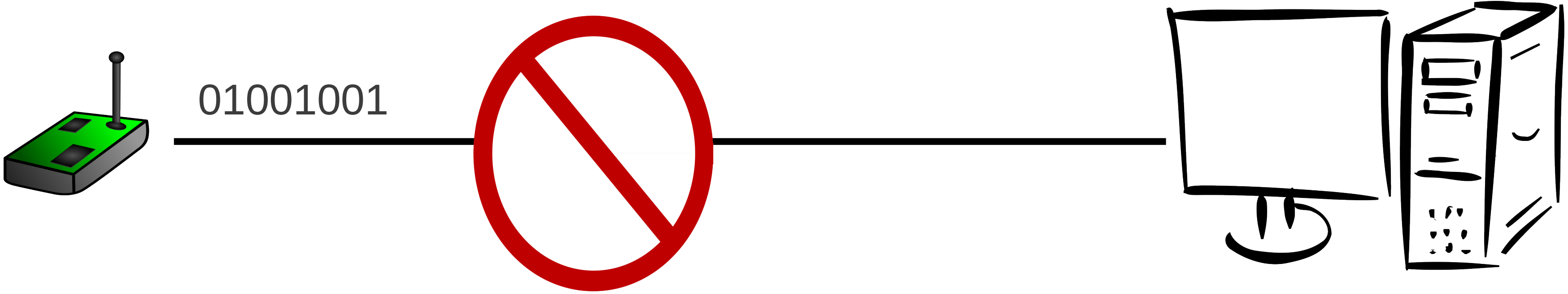


?

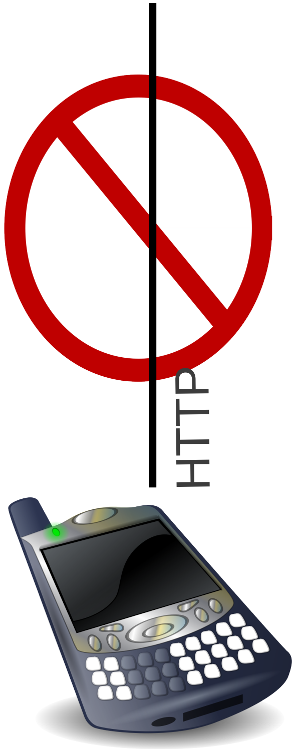
Internet of Things

Machine to Machine Communication

How about use an existing industry standard, such as the Java Message Service (JMS)?



No! There are several concerns with JMS on resource limited devices. However the main problem is that JMS is a Application Programming Interface – it makes no comment on the wire protocol itself.



What about HTTP then?

Client-Server request-response protocol, which is not optimised for:

- Intermittent Connectivity
- High power consumption to account for server polling
- Massive scalability, millions of devices

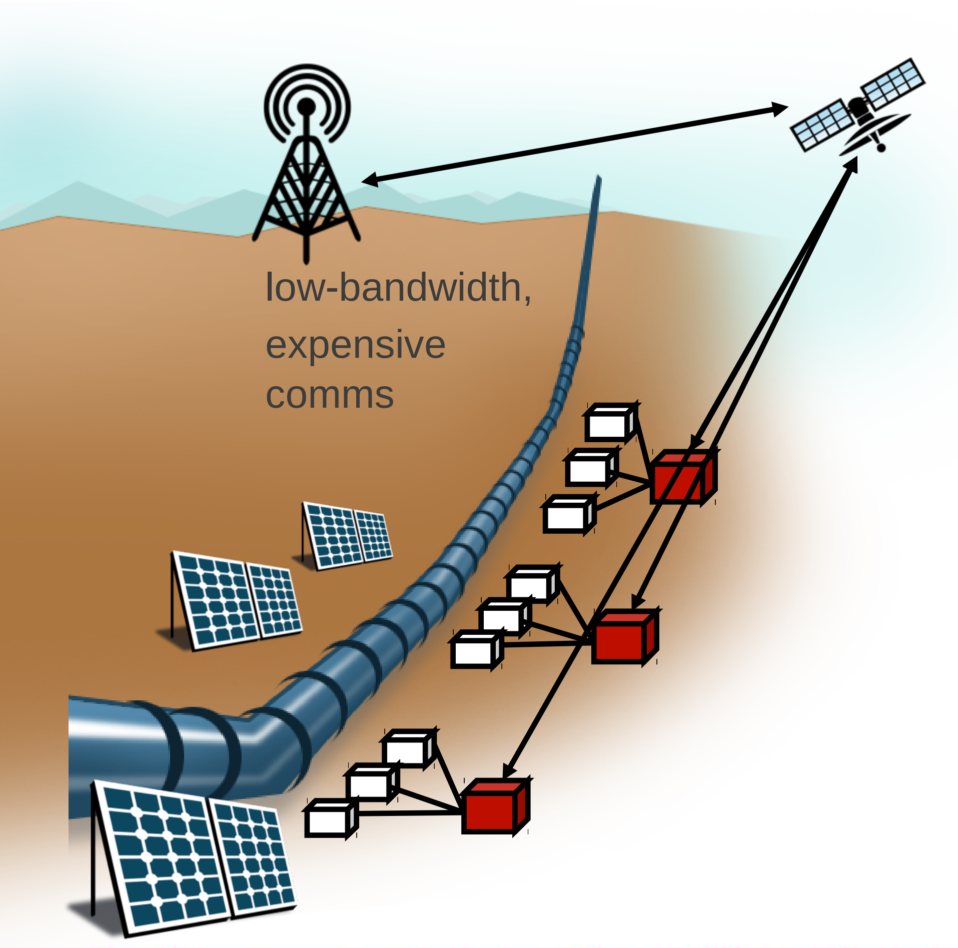
Agenda

- Communication between Digital Devices
- MQTT
- WebSphere MQ Extended Reach (MQXR)
- MessageSight
- WebSphere MQ HTTP Bridge
- Live Demonstration of MQTT, MQXR and JavaScript

MQ Telemetry Transport (MQTT)

To save inventing a new protocol every time a new embedded device came along, a common protocol was needed.

MQTT is that protocol. It traces its roots back to 1999, where Dr Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech) devised the protocol.



Design goals of MQTT:

- Works over unreliable communication networks
- Minimal data overhead (low bandwidth)
- Capable of supporting large numbers of devices
- Simple to interface the data with the traditional IT world
- Simple to developers to write applications to use

MQ Telemetry Transport (MQTT)

- Expect and cater for frequent network disruption – built for ***low bandwidth*****,** ***high latency*****,** ***unreliable*****,** ***high cost*** networks

- Expect that client applications may have very ***limited resources*** available.

- ***Publish/subscribe*** messaging paradigm as required by the majority of SCADA and sensor applications.

- Provide traditional messaging ***qualities of service*** where the environment allows.

- ***Published protocol*** for ease of adoption by device vendors and third-party client software.

MQTT Header

MQTT Header could be as little as 2 bytes! Structure is:

bit	7	6	5	4	3	2	1	0
Byte 1	Message Type				DUP flag	QoS Level		RETAIN
Byte 2	Remaining Length (at least one byte)							

Contrast with WebSphere MQ MQMD header structure:

```
struct tag MQMD {   MQCHAR4   StrucId;           // Structure identifier
                    MQLONG    Version;           // Structure version number
                    MQLONG    Report;            // Options for report messages
                    MQLONG    MsgType;           // Message type
                    MQLONG    Expiry;             // Message lifetime
                    MQLONG    Feedback;          // Feedback or reason code
                    MQLONG    Encoding;          // Numeric encoding of message data
                    MQLONG    CodedCharSetId;    // Character set identifier of message data
                    MQCHAR8   Format;            // Format name of message data
                    MQLONG    Priority;          // Message priority
                    MQLONG    Persistence;       // Message persistence
                    MQBYTE24  MsgId;             // Message identifier
                    MQBYTE24  CorrelId;          // Correlation identifier
                    MQLONG    BackoutCount;      // Backout counter
                    MQCHAR48  ReplyToQ;          // Name of reply queue
                    MQCHAR48  ReplyToQMgr;       // Name of reply queue manager
                    MQCHAR12  UserIdentifier;    // User identifier
                    MQBYTE32  AccountingToken;   // Accounting token
                    MQCHAR32  ApplIdentityData;  // Application data relating to identity
                    MQLONG    PutApplType;       // Type of application that put the message
                    MQCHAR28  PutApplName;       // Name of application that put the message
                    MQCHAR8   PutDate;           // Date when message was put
                    MQCHAR8   PutTime;           // Time when message was put
                    MQCHAR4   ApplOriginData;    // Application data relating to origin
                    MQBYTE24  GroupId;           // Group identifier
                    MQLONG    MsgSeqNumber;      // Sequence number of logical message within group
                    MQLONG    Offset;            // Offset of data in physical message from start of logical message
                    MQLONG    MsgFlags;          // Message flags
                    MQLONG    OriginalLength;    // Length of original message   }
```

MQTT Header Structure

Message Types:
CONNECT CONNACK
PUBLISH PUBACK
PUBREC PUBREL
PUBCOMP SUBSCRIBE
SUBACK UNSUBSCRIBE
UNSUBACK PINGREQ
PINGRESP DISCONNECT

DUP flag:
Used to indicate a redelivery message for one of the message types:
PUBLISH, PUBREL,
SUBSCRIBE, UNSUBSCRIBE

Quality of Service of a PUBLISH message

bit	7	6	5	4	3	2	1	0
Byte 1	Message Type				DUP flag	QoS Level		RETAIN
Byte 2	Remaining Length (at least one byte) (msg up to 127 bytes)							
Byte 3	*Remaining Length (msg up to 16KB)*							
Byte 4	*Remaining Length (msg up to 2MB)*							
Byte 5	*Remaining Length (msg up to 256MB)*							

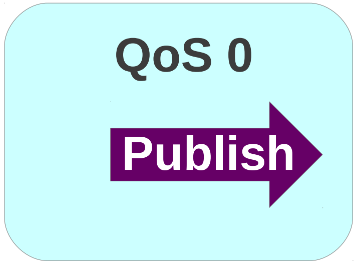
Variable **length** message (127 bytes maximum for the single byte length field), up to a maximum of 256MB for 4 length byte fields.

Indicates if a message should be **retained**, to be sent to new subscribers.

MQTT Qualities of Service

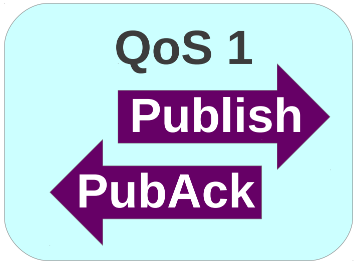
QoS 0: At most once delivery (non-persistent)

- No retry semantics are defined in the protocol.
- The message arrives either once or not at all.



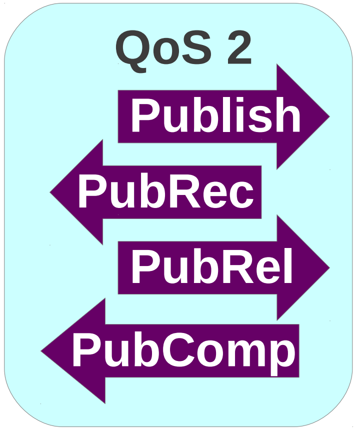
QoS 1: At least once delivery (persistent, duplicate messages possible)

- 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)

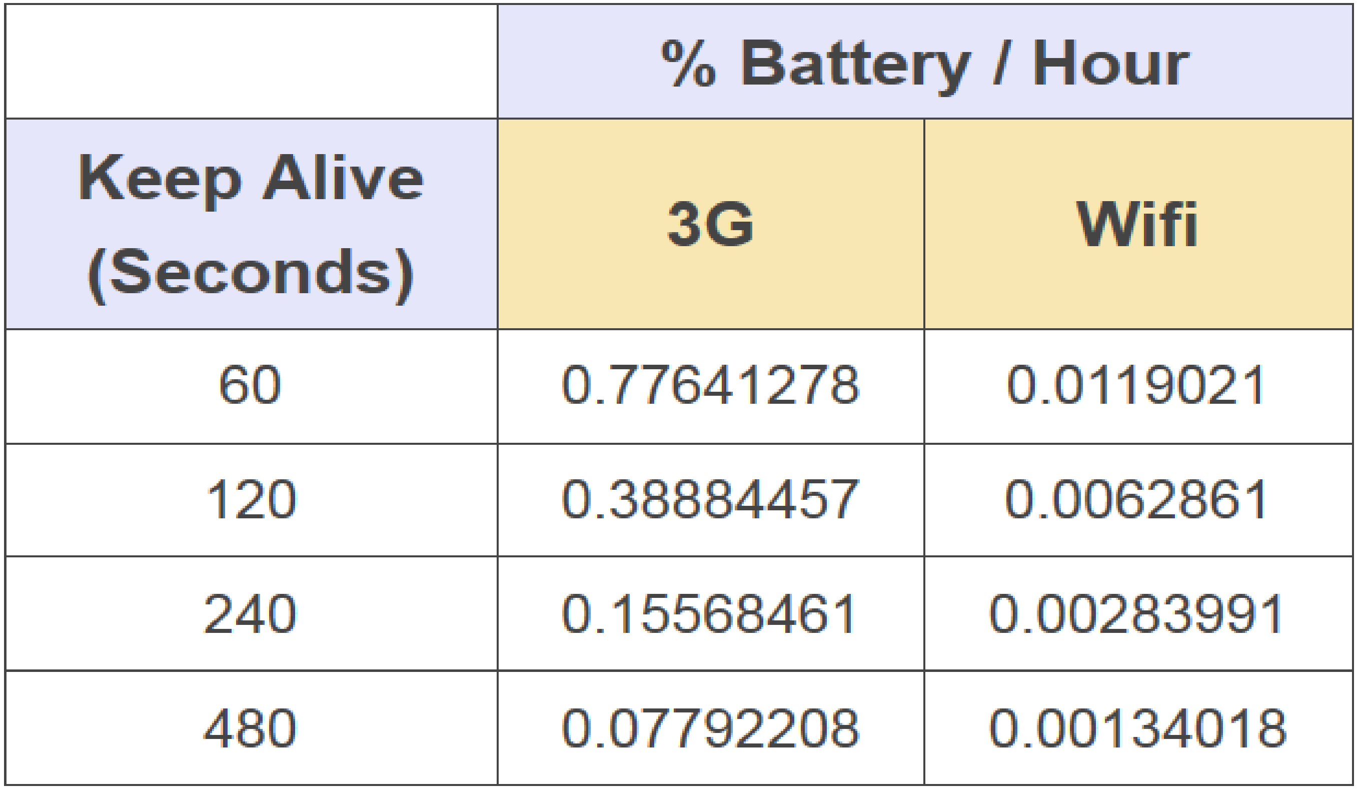
- Uses additional flows to ensure that message is not duplicated
- Server acknowledges with a PUBREC control message
- Client releases message with a PUBREL control message
- Server acknowledges completion with a PUBCOMP control message



MQTT Power Usage

How does MQTT use power?

- Example using a HTC Android mobile phone

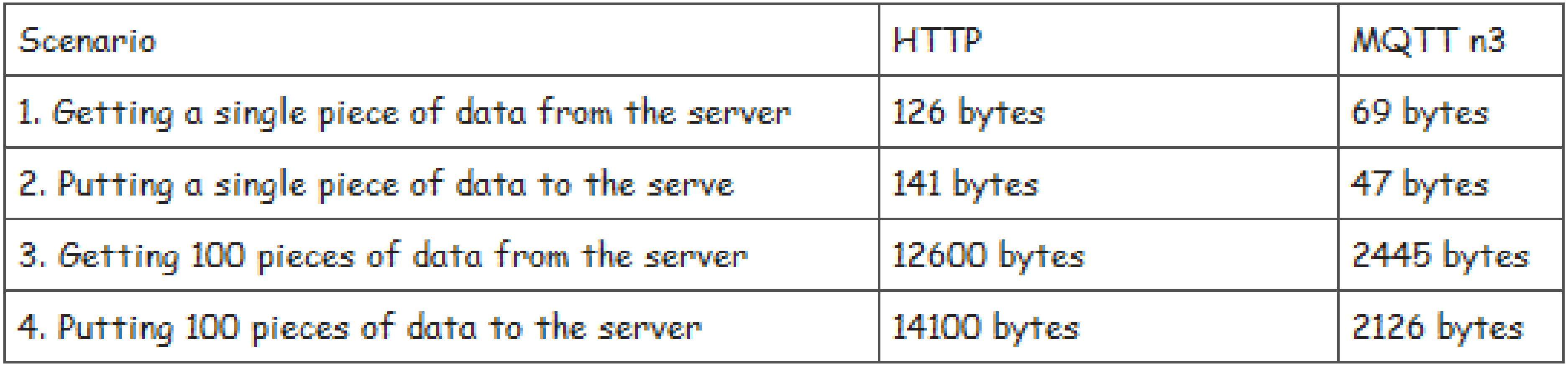


	% Battery / Hour	
Keep Alive (Seconds)	3G	Wifi
60	0.77641278	0.0119021
120	0.38884457	0.0062861
240	0.15568461	0.00283991
480	0.07792208	0.00134018

Protocol allows tuning to suit devices

MQTT Data Usage

How does MQTT compare to HTTP for data usage?

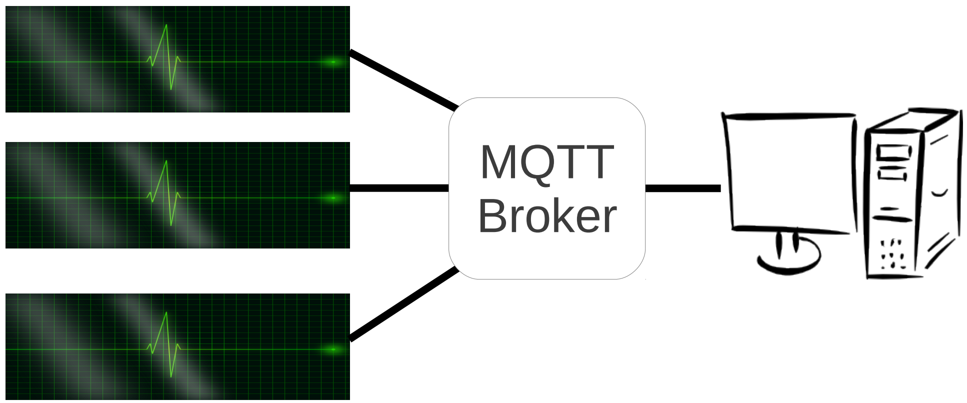


Scenario	HTTP	MQTT n3
1. Getting a single piece of data from the server	126 bytes	69 bytes
2. Putting a single piece of data to the serve	141 bytes	47 bytes
3. Getting 100 pieces of data from the server	12600 bytes	2445 bytes
4. Putting 100 pieces of data to the server	14100 bytes	2126 bytes

Very favourably – of the order of a 5x saving!

MQTT Sample Usage Applications

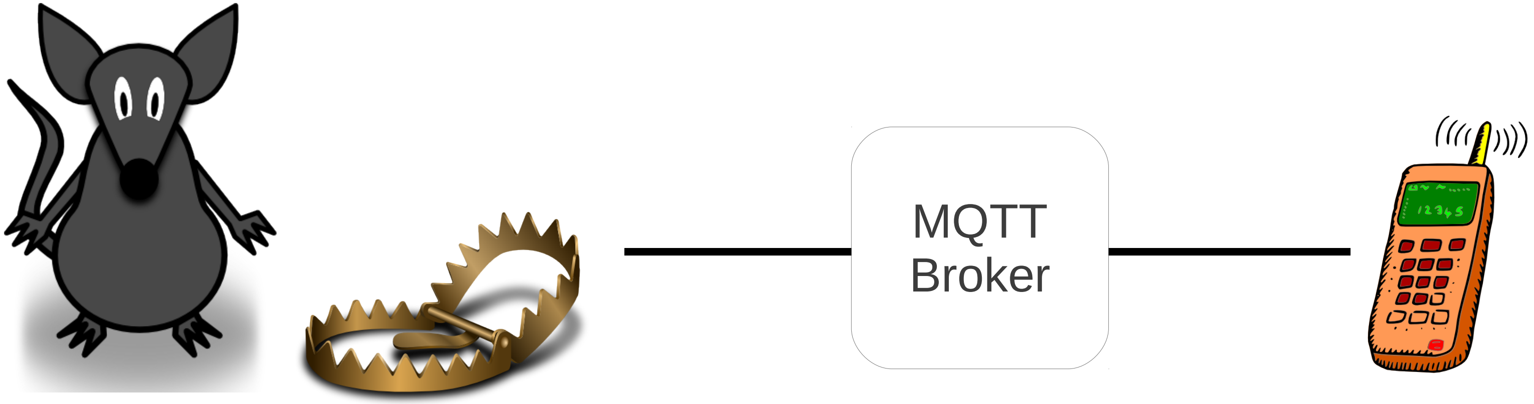
Medical devices in hospital equipment


MQTT Broker

Facebook Messenger

Low latency (milliseconds)
Low battery usage
Uses data sparingly
Implemented within weeks

The Andy Stanford-Clark Mouse Trap State Advisor


MQTT Broker

Agenda

- Communication between Digital Devices
- MQTT
- WebSphere MQ Extended Reach (MQXR)
- MessageSight
- WebSphere MQ HTTP Bridge
- Live Demonstration of MQTT, MQXR and JavaScript

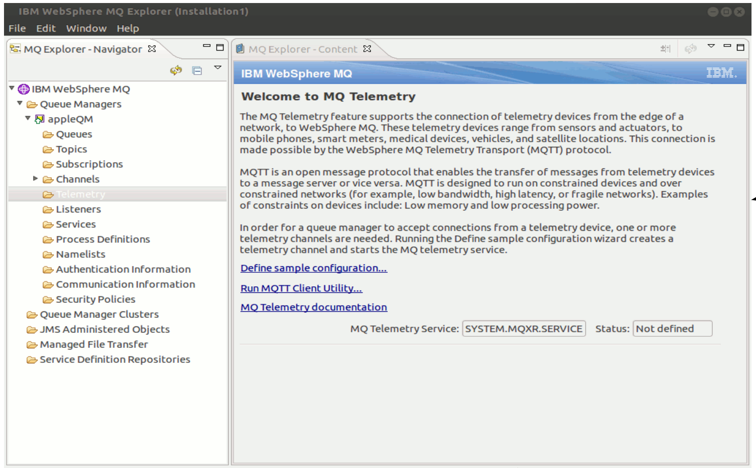
WebSphere MQ Telemetry

Supplied as a component of WebSphere MQ V7.1 and v7.5 on distributed platforms, under the component name “**WebSphere MQ Extended Reach**” (or MQXR).

MQXR brings MQTT protocol functionality to WebSphere MQ!

- Highly scaleable : tested with 200,000+ clients
- Security : SSL channels, JAAS authentication, WMQ OAM
- Ships with reference Java and C clients
  - Small footprint clients
  - other APIs and implementations of MQTT available via 3rd parties

WebSphere MQ Telemetry



Welcome to MQ Telemetry

The MQ Telemetry feature supports the connection of telemetry devices from the edge of a network, to WebSphere MQ. These telemetry devices range from sensors and actuators, to mobile phones, smart meters, medical devices, vehicles, and satellite locations. This connection is made possible by the WebSphere MQ Telemetry Transport (MQTT) protocol.

MQTT is an open message protocol that enables the transfer of messages from telemetry devices to a message server or vice versa. MQTT is designed to run on constrained devices and over constrained networks (for example, low bandwidth, high latency, or fragile networks). Examples of constraints on devices include: Low memory and low processing power.

In order for a queue manager to accept connections from a telemetry device, one or more telemetry channels are needed. Running the Define sample configuration wizard creates a telemetry channel and starts the MQ telemetry service.

Define sample configuration...

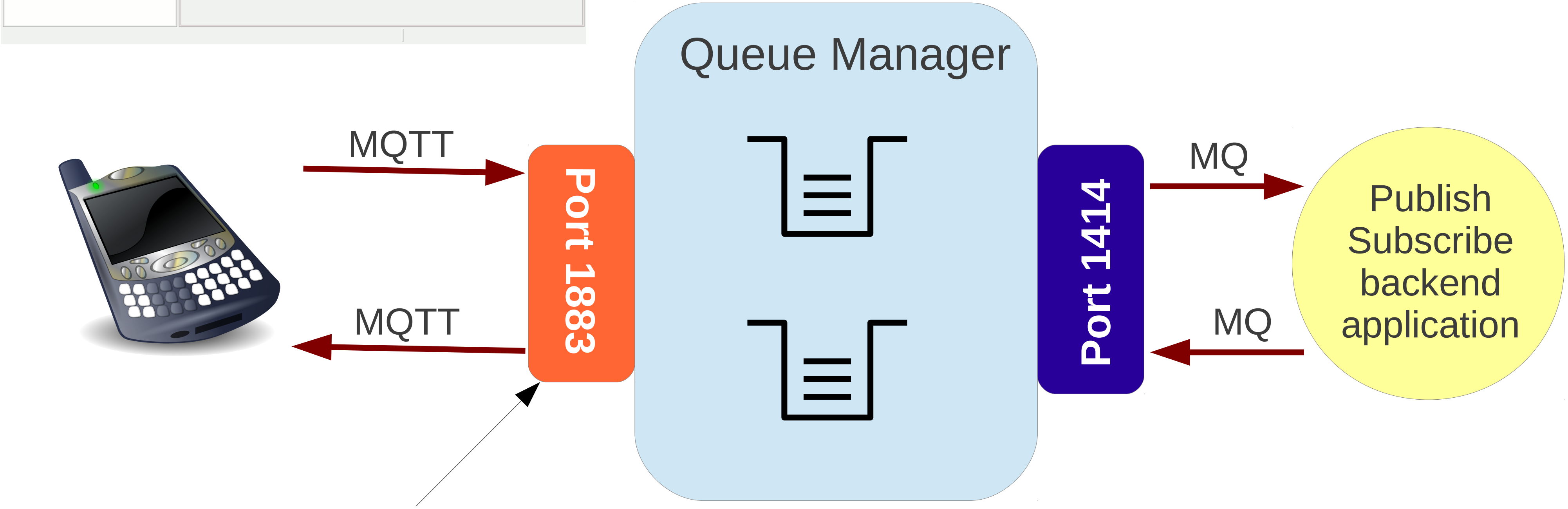
Run MQTT Client Utility...

MQ Telemetry documentation

MQ Telemetry Service: SYSTEM.MQXR.SERVICE Status: Not defined

Use WebSphere MQ Explorer to administer the WebSphere MQ Telemetry service – define Channels, start and stop the MQTT service.

Alternatively, it can be configured through 'runmqsc' commands.



WebSphere MQ MQTT Listener
IANA registered ports:
1883, 8883 for MQTT over SSH

MQTT through Javascript

As of WebSphere MQ 7.5.0.1, the WebSphere MQ MQXR component has support for MQTT v3.1 protocol over WebSockets.

This enables the use of MQTT through a WebSocket supporting web browser, meaning that MQTT can be used without preinstalling any software on a browser equipped device.

```
<script type="text/javascript" src="mqttws31.js"></script>
<script type="text/javascript">
  var clientId = "MyUniqueClientID";
  var client;

  function publishMessage() {
    client = new Messaging.Client(location.hostname, Number(1883), clientId);
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
    client.connect({onSuccess:onConnect});
  }

  function onConnect() {
    // Once a connection has been made, make a subscription and send a message.
    console.log("onConnect");
    client.subscribe("/TopicLocation");
    message = new Messaging.Message("My publish text!");
    message.destinationName = "/TopicLocation";
    client.send(message);
  }
… … …
</script>

<button type="button" onclick="publishMessage()" name="Connect">Publish a Message</button>
```

Reference the WMQ supplied MQTT javascript file

Connect to the MQTT server, and register callback functions

Subscribe to the Topic, and publish a message.

Write callback functions here

Invoke the Javascript function from HTML

WebSphere MQ Client Pack for Mobile - MA9B

Released in 1Q 2013, the “Mobile Messaging and M2M Client Pack” provides the following capabilities:

- Java implementation of the MQTT v3 protocol (“Paho” open source project client)

  Sample applications for Android

- C client implementation of the MQTT v3 client, compiled for Windows and Linux (x86) systems

- C client implementation of the MQTT v3 client, provided in source code form for iOS

WebSphere MQ Telemetry – Further Reading

MQTT homepage:
http://mqtt.org

MQTT Specification
http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html

WebSphere MQ and MQ Telemetry
http://www-01.ibm.com/software/integration/wmq/

Mobile Messaging & M2M Client Pack
http://www.ibm.com/developerworks/mydeveloperworks/blogs/c565c720-fe84-4f63-873f-607

MQTT: the Smarter Planet Protocol
http://andypiper.co.uk/2010/08/05/mqtt-the-smarter-planet-protocol/

Lotus Expeditor (micro broker)
http://www.ibm.com/software/lotus/products/expeditor/

Agenda

- Communication between Digital Devices
- MQTT
- WebSphere MQ Extended Reach (MQXR)
- MessageSight
- WebSphere MQ HTTP Bridge
- Live Demonstration of MQTT, MQXR and JavaScript

IBM MessageSight, Big Connectivity in a Box

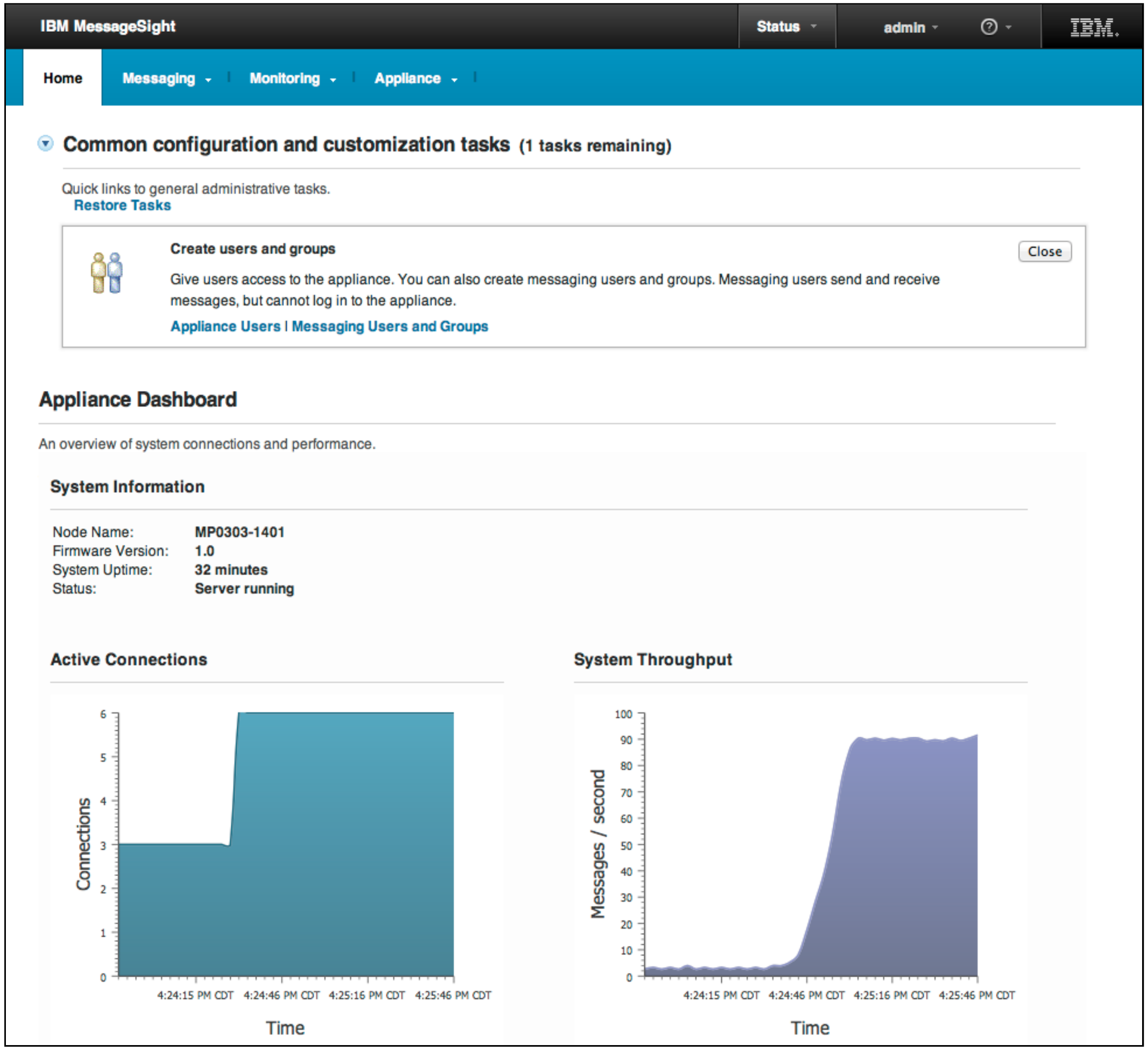
A secure messaging server appliance optimised to meet the demands of massive scale messaging of machine-2-machine and mobile use cases.

How massive? One applicance can achieve:

- 1 million concurrent connections
- 13 million non-persistent msg/sec
- 400K persistent msg/sec

IBM MessageSight – East of Use

- Up and running in 30 minutes
- Task oriented HTTP based UI guides administrator through the first steps
- Simple and scalable management through policies



Common configuration and customization tasks (1 tasks remaining)

Quick links to general administrative tasks.
Restore Tasks

Create users and groups

Give users access to the appliance. You can also create messaging users and groups. Messaging users send and receive messages, but cannot log in to the appliance.

Appliance Users | Messaging Users and Groups

Appliance Dashboard

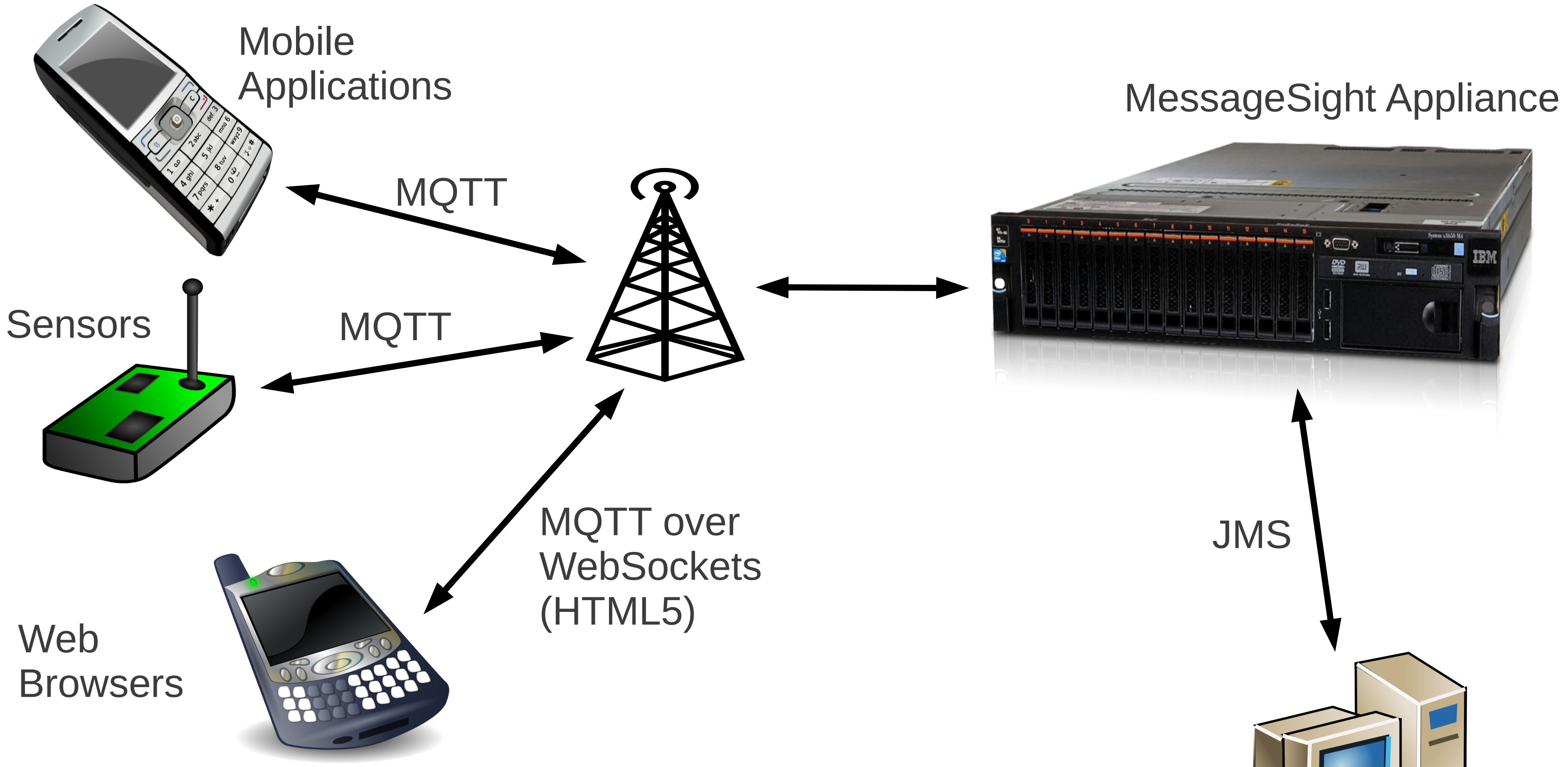
An overview of system connections and performance.

System Information

Node Name:	MP0303-1401
Firmware Version:	1.0
System Uptime:	32 minutes
Status:	Server running

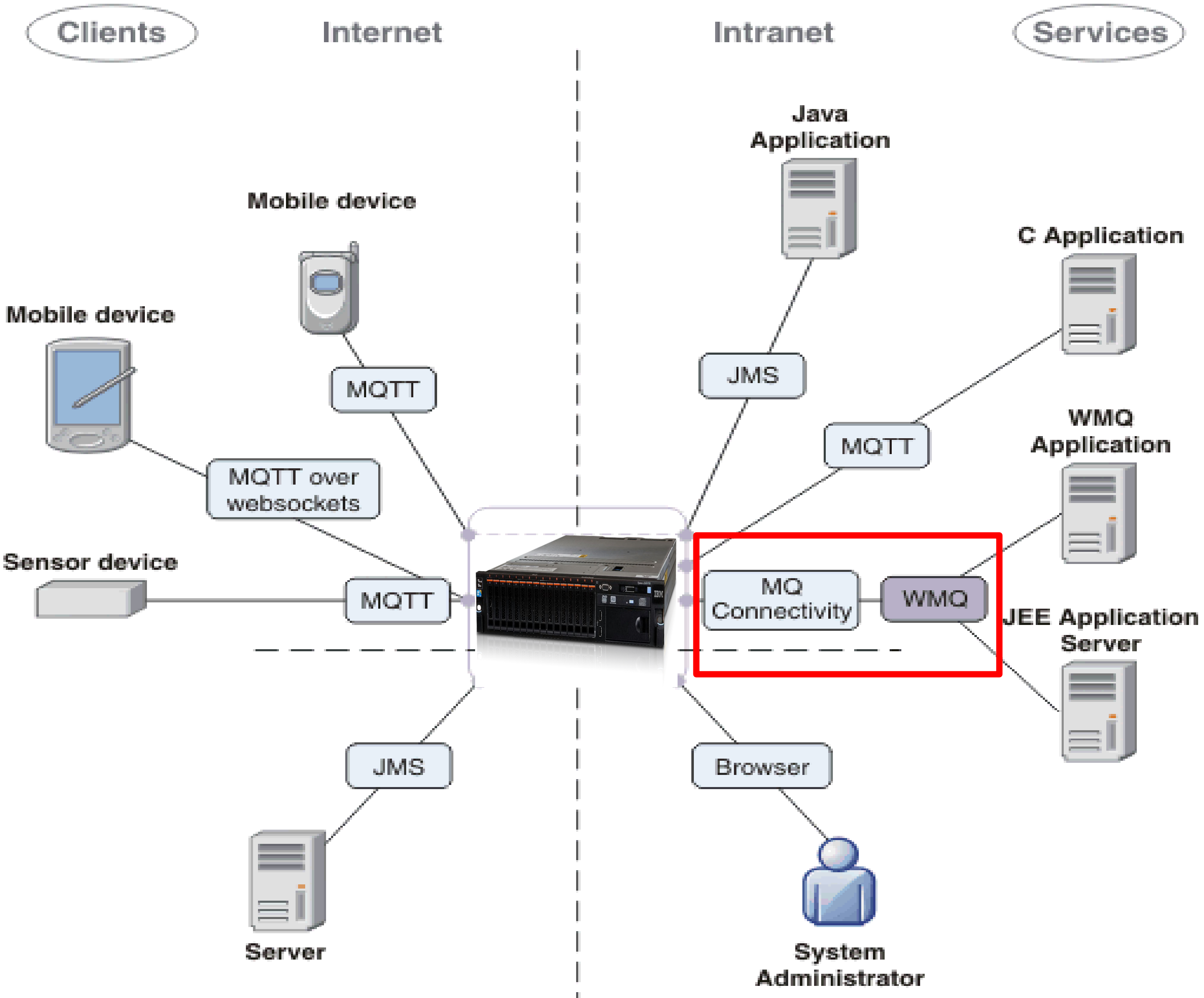
IBM MessageSight – Client Access

Utilises the MQTT messaging protocol

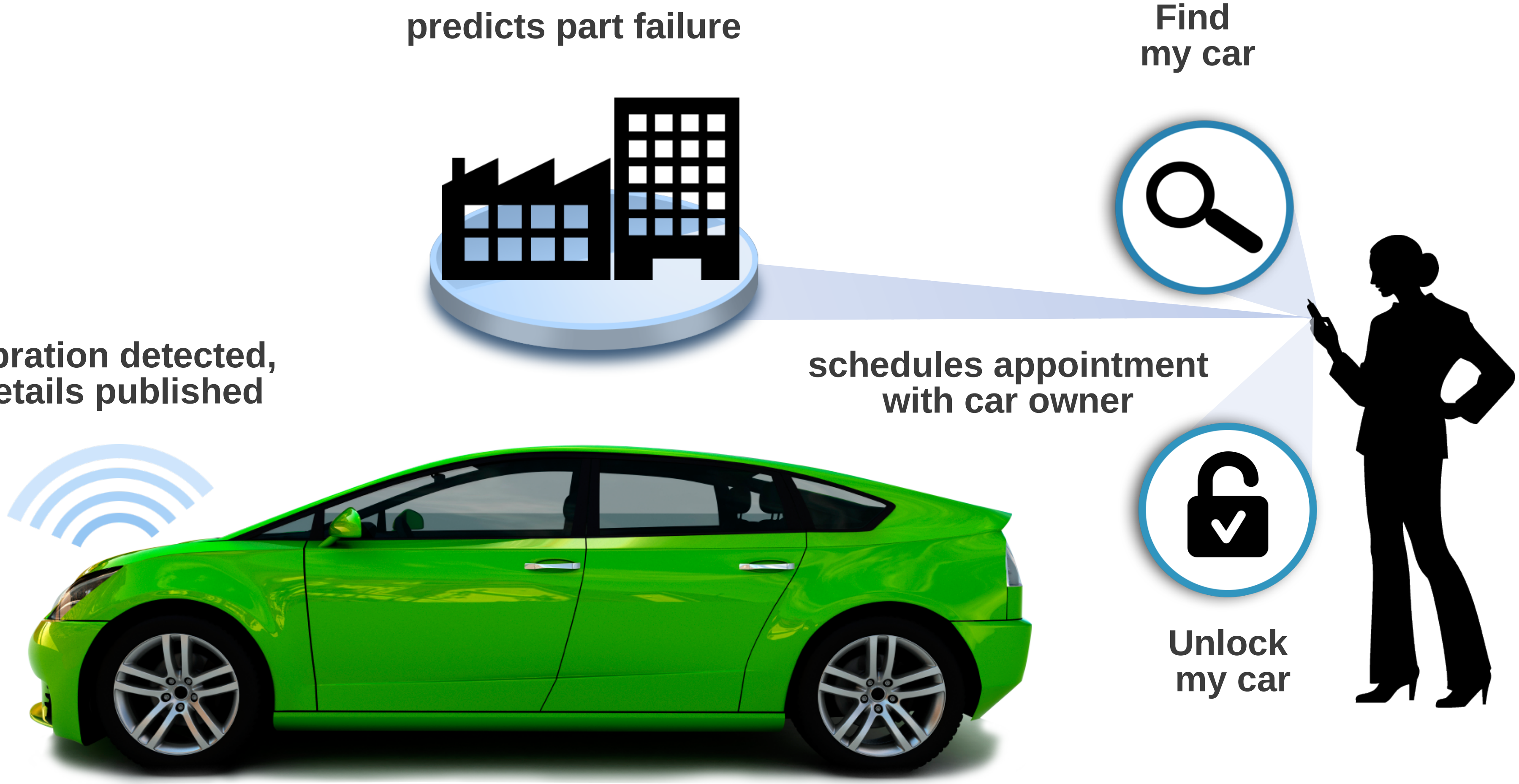


Also supports the Java Mesage Service (JMS) Application Programming Interface.

IBM MessageSight – Topology Configuration



IBM MessageSight - Example Use Case



predicts part failure

Find
my car

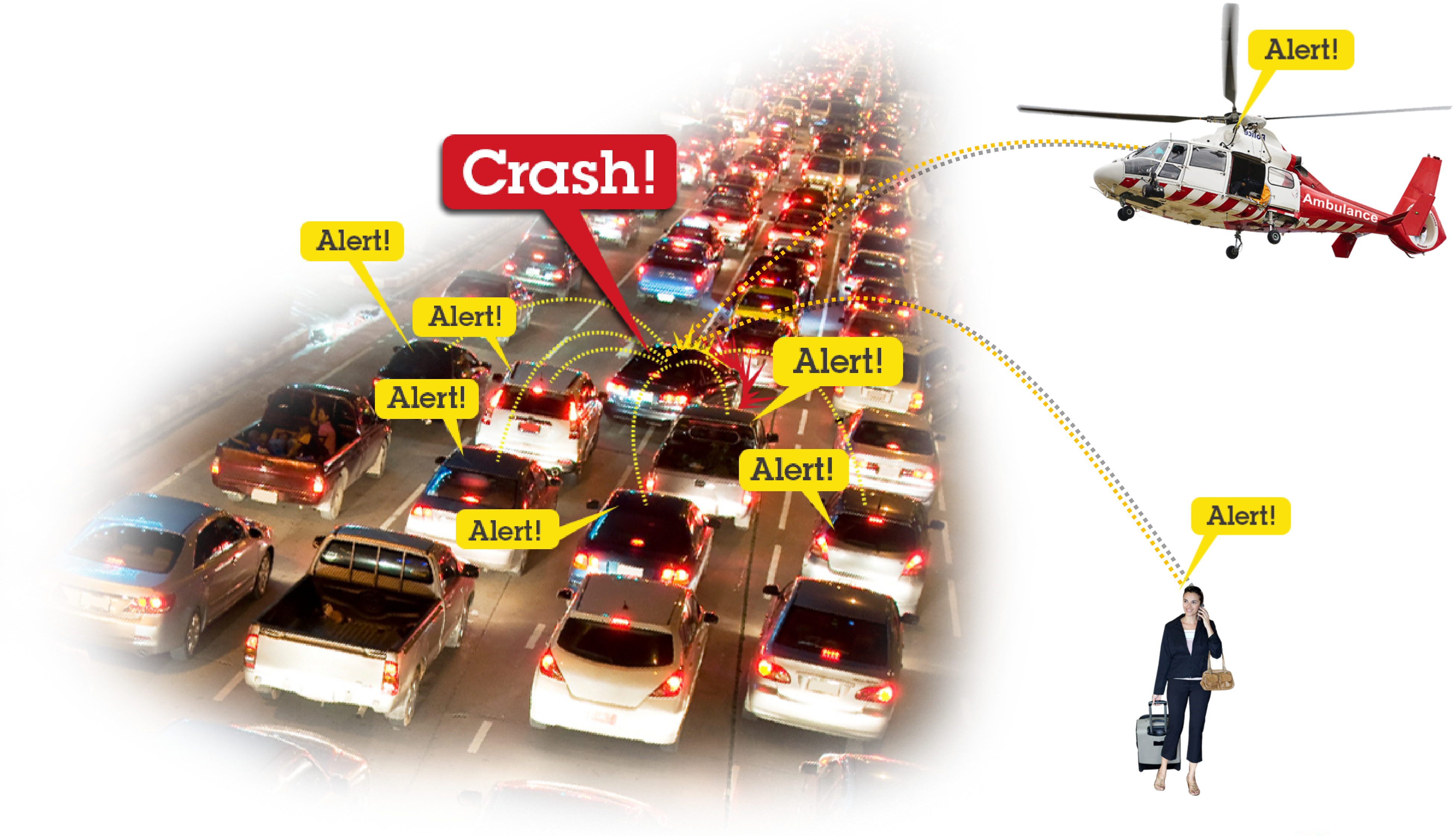
vibration detected,
details published

schedules appointment
with car owner

Unlock
my car

Car connected to MessageSight via mobile network

IBM MessageSight – Example Use Case

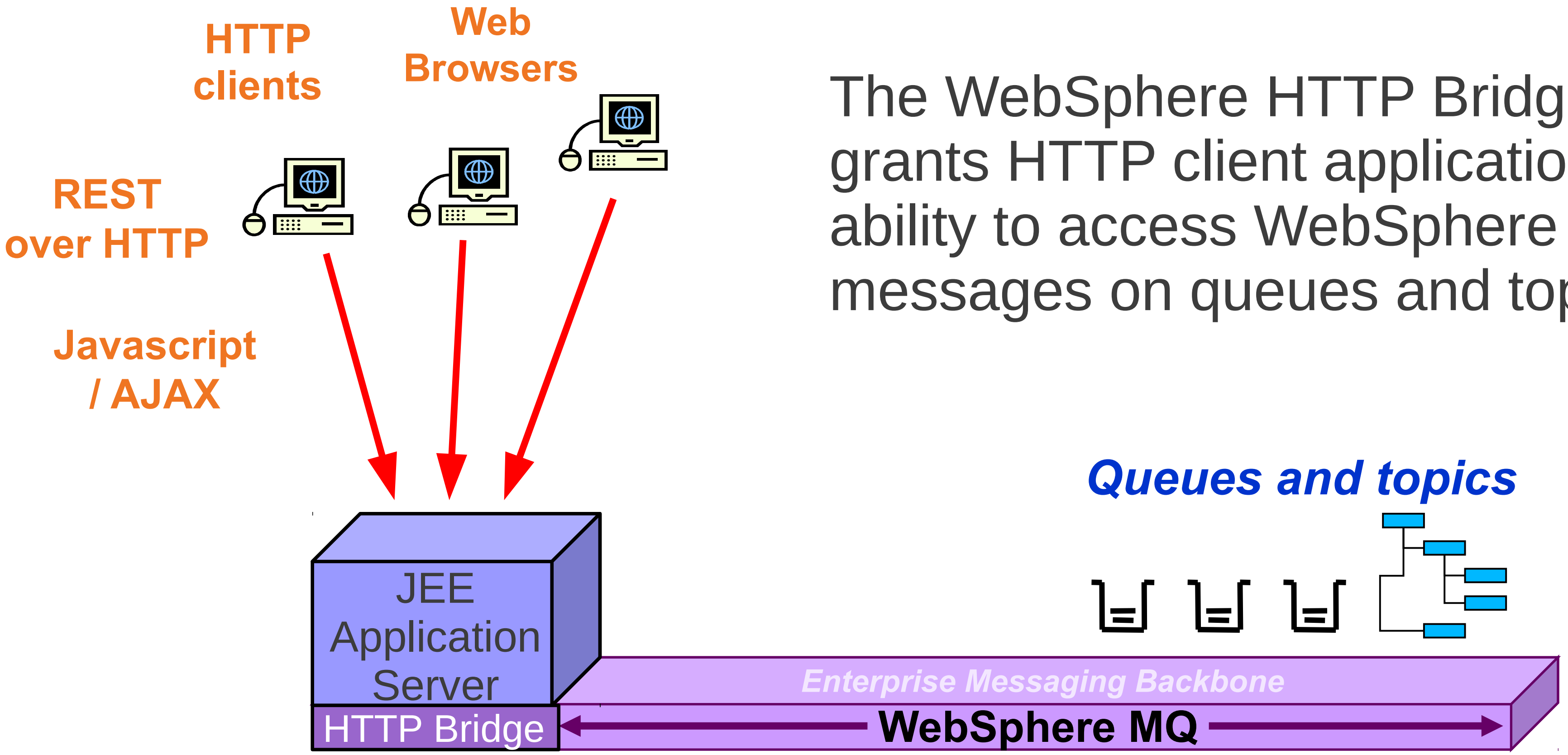


Agenda

- Communication between Digital Devices
- MQTT
- WebSphere MQ Extended Reach (MQXR)
- MessageSight
- WebSphere MQ HTTP Bridge
- Live Demonstration of MQTT, MQXR and JavaScript

WebSphere MQ HTTP Bridge

The WebSphere HTTP Bridge grants HTTP client applications the ability to access WebSphere MQ messages on queues and topics.

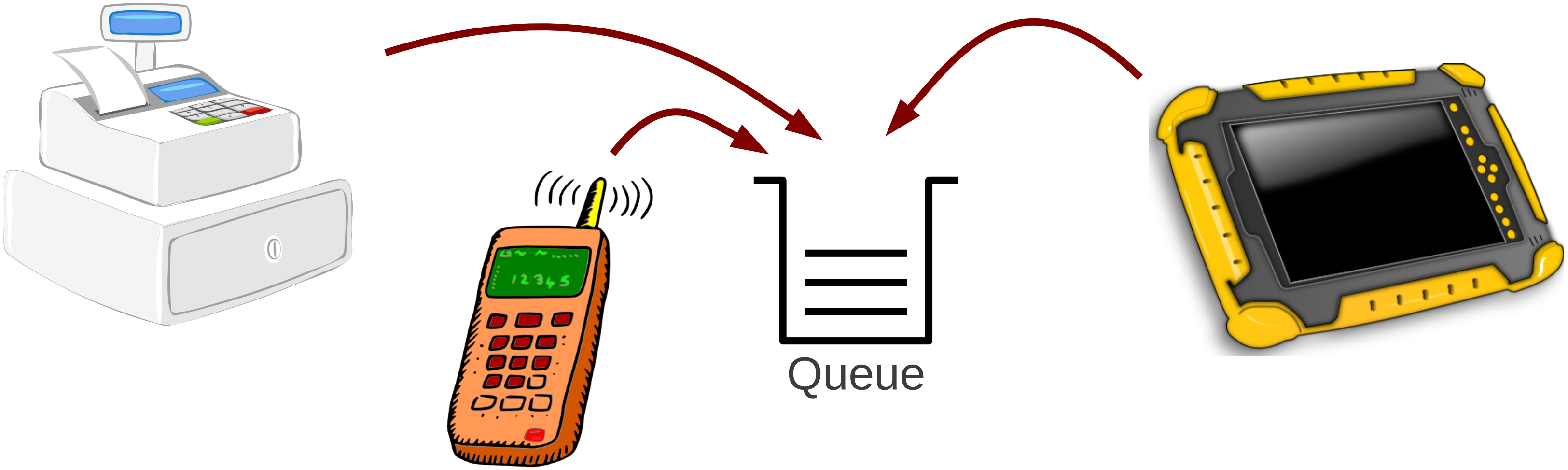


The HTTP Bridge comprises of a JEE Web application (servlet), which is to be installed into a JEE Application server in order to be used.

WebSphere MQ HTTP Bridge

The WebSphere MQ HTTP Bridge provides two key benefits:

1) **Zero Client Footprint**.
No WebSphere MQ MQI client libraries are required on the application host.
In addition, ***any*** platform which supports HTTP can access WebSphere MQ data.



2) **Simplifies access to WebSphere MQ messages** from browser based internet applications.
No WebSphere MQ programming knowledge is required to program the client applications

WebSphere MQ HTTP Bridge

How does data access work from HTTP?

HTTP Request	Result
POST	Puts a message to a queue or topic (MQPUT)
GET	Browses the first message on the queue (MQGET with browse)
DELETE	Receives a message from the queue (destructive MQGET), or creates a non-durable subscription from a topic
PUT	Not used

The HTTP request defines the location and name of the the queue or topic access point:

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: myhost.mydomain
```

WebSphere MQ HTTP Bridge

Example 1: **MQPUT**
Put a messsage to a queue, with message body containing a string message:

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: myhost.mydomain
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 60

Here is my message body that is posted on the queue.
```

This HTTP POST response is of the form:

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

WebSphere MQ HTTP Bridge

Example 2: **MQGET**
Destructively receive a message from a queue, waiting a maximum of 10 seconds:

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: myhost.mydomain
x-msg-wait: 10
x-msg-require-headers: correlID
```

This HTTP DELETE response is of the form:

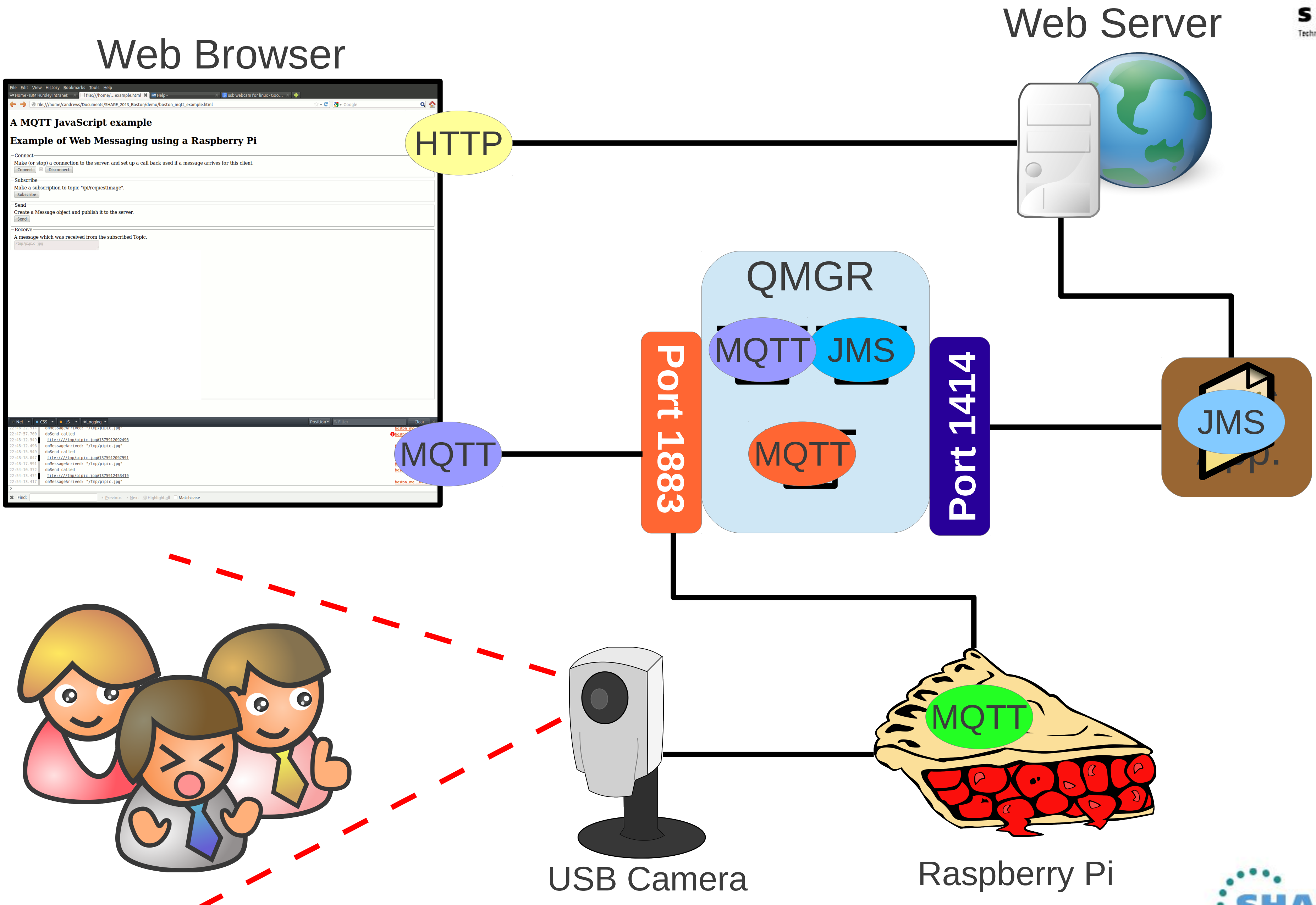
```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 60
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body from the queue.
```

Agenda

- Communication between Digital Devices
- MQTT
- WebSphere MQ Extended Reach (MQXR)
- MessageSight
- WebSphere MQ HTTP Bridge
- Live Demonstration of MQTT, MQXR and JavaScript

Mobile MQ Live Demonstration!



File Edit View History Bookmarks Tools Help

Home - IBM Hursley Intranet | file:///home/...example.html | Help - | usb webcam for linux - Goo...

file:///home/candrews/Documents/SHARE_2013_Boston/demo/boston_mqtt_example.html

A MQTT JavaScript example

Example of Web Messaging using a Raspberry Pi

Connect

Make (or stop) a connection to the server, and set up a call back used if a message arrives for this client.

Connect ☑ Disconnect

Subscribe

Make a subscription to topic "/pi/requestImage".

Subscribe

Send

Create a Message object and publish it to the server.

Send

Receive

A message which was received from the subscribed Topic.

/tmp/pipic.jpg

```
22:46:22.914  onMessageArrived: "/tmp/pipic.jpg"
22:47:57.760  doSend called
22:48:12.549   file:////tmp/pipic.jpg#1375912092496
22:48:12.496  onMessageArrived: "/tmp/pipic.jpg"
22:48:15.949  doSend called
22:48:18.047   file:////tmp/pipic.jpg#1375912097991
22:48:17.991  onMessageArrived: "/tmp/pipic.jpg"
22:54:10.372  doSend called
22:54:13.474   file:////tmp/pipic.jpg#1375912453419
22:54:13.417  onMessageArrived: "/tmp/pipic.jpg"
```

Net CSS JS Logging Position Filter Clear

Find: Previous Next Highlight all □ Match case

This was session 13923 - The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00				Extending IBM WebSphere MQ and WebSphere Message Broker to the Cloud	CICS and WMQ - The Resurrection of Useful
09:30	Introduction to MQ				Can I Consolidate My Queue Managers and Brokers?
11:00		MQ on z/OS - Vivisection	Hands-on Lab for MQ - take your pick!	MOBILE connectivity with Broker	Migration and Maintenance, the Necessary Evil. Into the Dark for MQ and Message Broker
12:15					
1:30	MQ Parallel Sysplex Exploitation, Getting the Best Availability From MQ on z/OS by Using Shared Queues	What’s New in the MQ Family	MQ Clustering - The basics, advances and what's new	Using IBM WebSphere Application Server and IBM WebSphere MQ Together	
3:00	First Steps With Message Broker: Application Integration for the Messy	What's New in Message Broker	**BIG Connectivity with mobile MQ**	WebSphere MQ CHINIT Internals	
4:30	What's available in MQ and Broker for high availability and disaster recovery?	The Dark Side of Monitoring MQ - SMF 115 and 116 Record Reading and Interpretation	MQ & DB2 – MQ Verbs in DB2 & Q-Replication performance	Big Data Sharing with the Cloud - WebSphere eXtreme Scale and IBM Integration Bus Integration	
6:00				WebSphere MQ Channel Authentication Records	

Copyright and Trademarks