Web APIs and Messaging for IoT

Frank Walsh

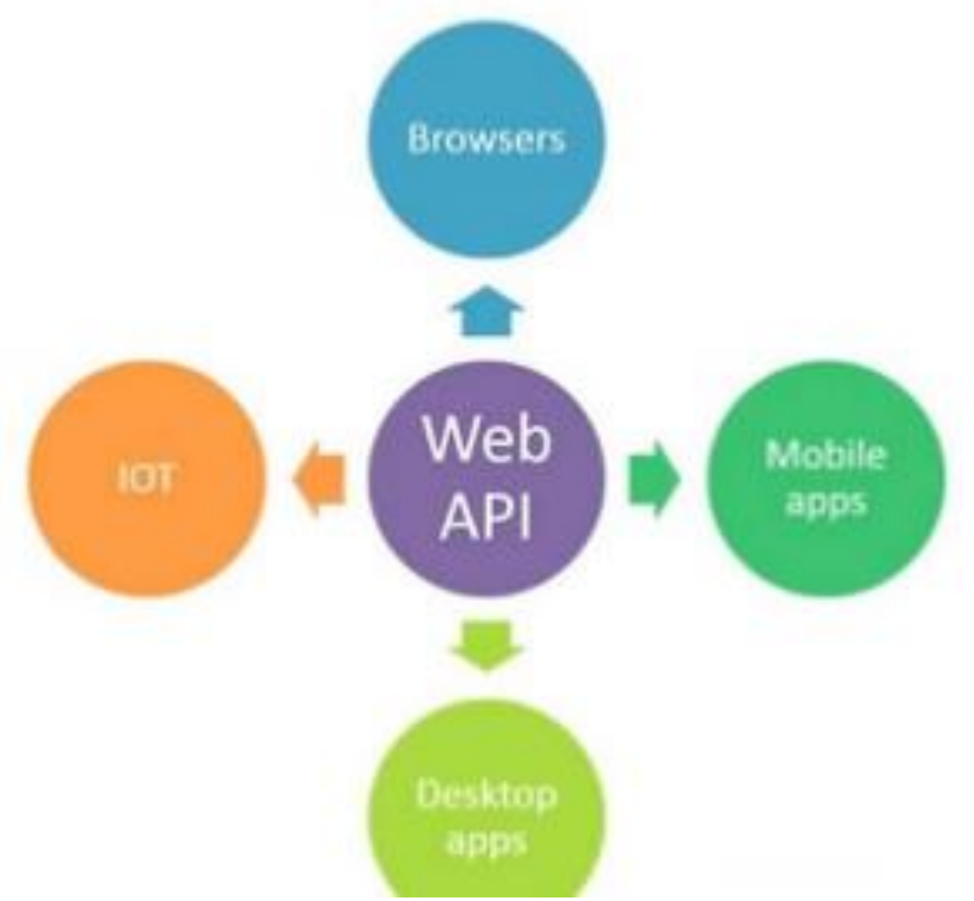
Arduino Web Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Arduino Web Page
10.0.0.20
Goog

Hello from Arduino!

A web page from the Arduino server

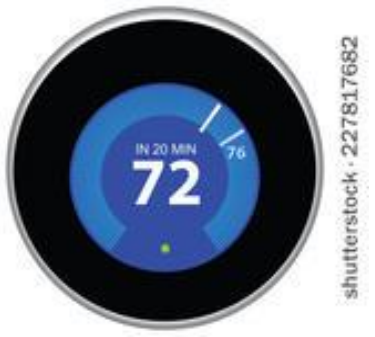
Web APIs

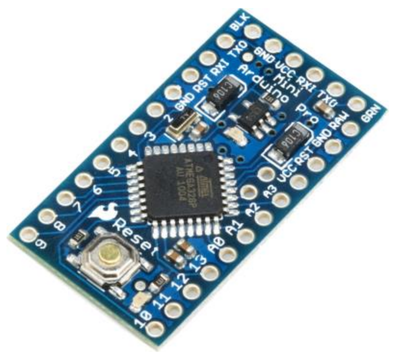
- Programmatic interface exposed via the web
- Uses open standards typically with request-response messaging.
  - Messages in JSON or XML
  - HTTP as transport
  - URIs
- Example would be Restful web service
- Typical use:
  - Expose application functionality via the web
  - Machine to machine communication
  - Distributed systems



What's a Web API

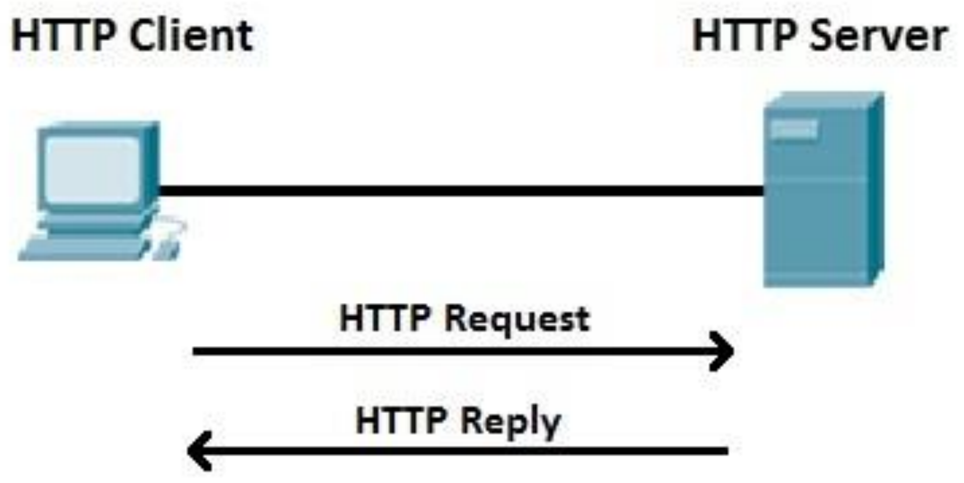
- Typically implements HTTP
  - processes HTTP requests
- Usually runs on machine connected to a network
  - Has an IP address
- Serves up resources
  - Ideally in a "restful" manner
- Many different types of devices can run a web API...





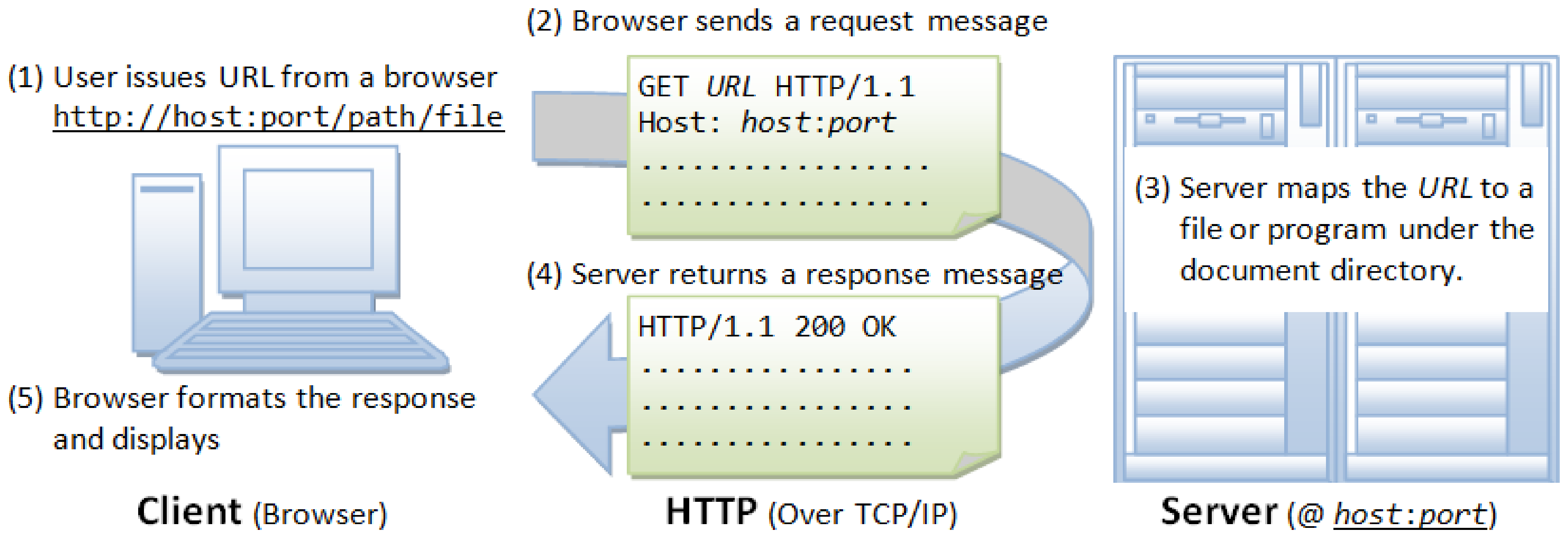
What’s HTTP

- HyperText Transfer Protocol
- Protocol used in World Wide We
    - http://www.wit.ie
- Your browser communicates usin HTTP (HTTP Client)
- Devices communicate using HTTI
- Simple, ubiquitous.



HTTP from browser

Browser:



(2) Browser sends a request message
(1) User issues URL from a browser http://host:port/path/file
GET *URL* HTTP/1.1
Host: *host*:*port*
....
....
(3) Server maps the *URL* to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
....
....
....
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ *host*:*port*)

URL

- A URL (Uniform Resource Locator) uniquely identifies a resource over the web. *protocol://hostname:port/path*
- There are 4 parts in a URL:
  - *Protocol*: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
  - *Hostname*: The domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
  - *Port*: The TCP port number that the server is listening for incoming requests from the clients.
  - *Path-and-file-name*: The name and location of the requested resource, under the server document base directory.
- Example, for http://www.nowhere123.com/docs/index.html
  - the communication protocol is HTTP
  - he host is www.nowhere123.com.
  - The port number was not specified, and takes default number, which is TCP port 80 for HTTP.
  - The path for the resource to be located is "/docs/index.html".

HTTP Protocol (Request)

- HTTP clients (e.g. a browser) translates a URL into a request message according to the specified protocol; and sends the request message to the server.
- For example, the browser translated the URL http://www.nowhere123.com/doc/index.html into the following request message:

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

HTTP Protocol (Response)

- When this request message reaches the server, the server can take either one of these actions:
  1. The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
  2. The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
  3. The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is below:

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
Content-Length: 44
Connection: close
Content-Type: text/html

 <html><body><h1>It works!</h1></body></html>
```

HTTP Query String

- Query string used to include data in a URL. For example

Query String

https://www.myhome.com/heating?status=on

- The server can use the query string to execute logic associated with that resource. In this example, it could be used to set the status of the resource (heating) to true.

HTTP Methods

- GET
  - Request objects without sending data
- POST
  - Modify objects with data that you are sending
- PUT
  - Create new objects with data that your are sending
- DELETE
  - Delete objects without sending data

External Data Representations
XML

- eXtensible Markup Language(XML)
- Same heritage as HTML(but XML is NOT HTML)
- XML data items are tagged with ‘markup’ strings
  - used to describe the logical structure of the data
- XML has many uses. For now we confine ourselves to external data representations
- Has many cool features including
  - Extensible
  - Textual
  - Kind of human readable and machine readable...

XML

```
<person id="123456789">
                <name>Smith</name>
                <place>London</place>
                <year>1984</year>
                <!-- a comment -->
</person >
```

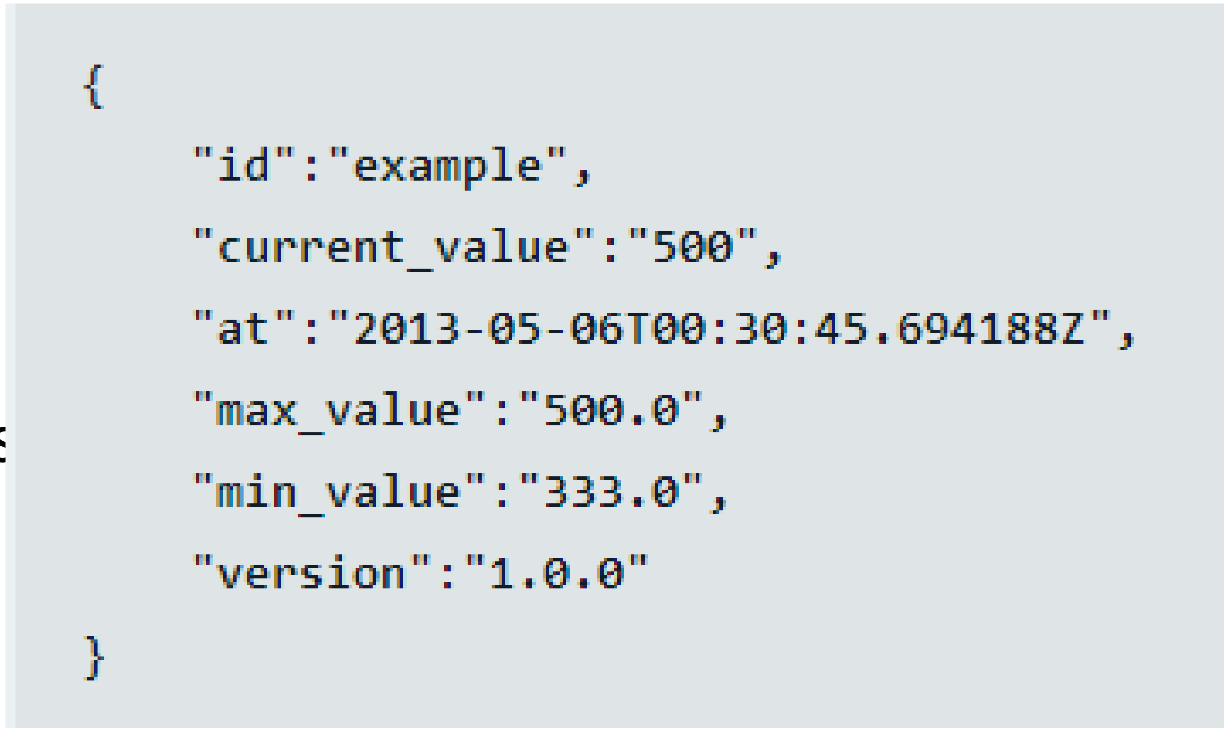
namespace

```
<person pers:id="123456789" xmlns:pers =
"http://www.cdk5.net/person">
        <pers:name> Smith </pers:name>
        <pers:place> London </pers:place >
        <pers:year> 1984 </pers:year>
</person>
```

- Above shows XML definitions of the Person structure.
  - As with xHTML, tags enclose character data.
  - Tags : <name>, <place>,<year>   data:"Smith", "London"…
- Namespaces provide a means for scoping names

JSON

- JavaScript Object Notation
- Lightweight text-based open standard designed for human readable data interchange.
- Can represent simple data structures and associative arrays.
- Good for serializing and transmitting structured data across a network



```
{
    "id":"example",
    "current_value":"500",
    "at":"2013-05-06T00:30:45.694188Z",
    "max_value":"500.0",
    "min_value":"333.0",
    "version":"1.0.0"
}
```

JSON

- JSON is a data interchange format technique
- A collection of name/value pairs.
- Application programming interfaces(APIs) exist for most programming languages

```
{
   person:{
      id:123456789,
      name:'Smith',
      place:'London',
      year:1984
   }
}
```

```
{ "temp" : 72.55 }
```

More on HTTP

- For an extensive overview, checkout:

  http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html

REST

- Short for Representational State Transfer
- Set of Principles for how web should be used
- Coined by Roy Fielding
    - One of the HTTP creator

•A set of principles that define how Web standards(HTTP and URIs) can be used.

Key REST Principles

1. Every “thing” has an identity
   - URL
2. Link things together
   - Hypermedia/Hyperlinks
3. Use standard set of methods
   - HTTP GET/POST/PUT/DELETE
   - Manipulate resources through their representations
4. Resources can have multiple representations
   - JSON/XML/png/...
5. Communicate stateless
   - Should **not** depend on server state.

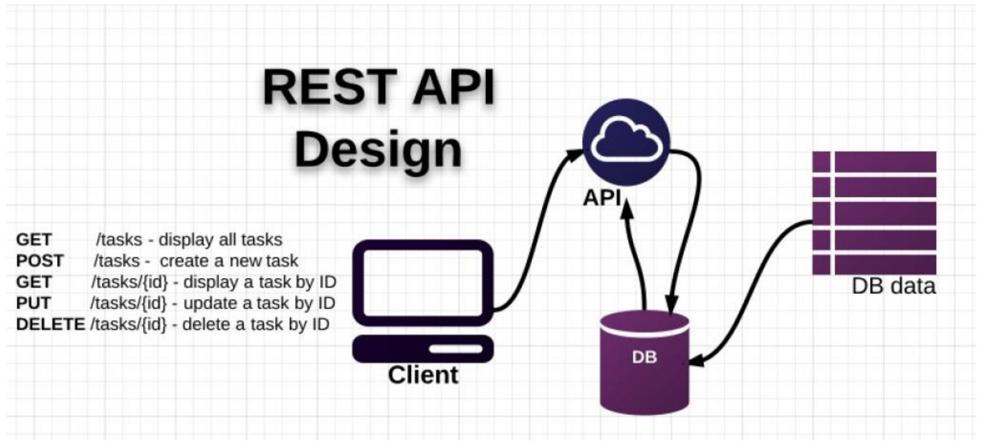
“API First” approach

- Collaboratively design, mockup, implement and document an API **before** the application or other channels that will use it even exist.
- Uses “clean-room” approach.
  - the API is designed with little consideration for the existing IT estate.
  - the API is designed as though there are no constraints.

source:
http://www.programmableweb.com/news/introduction-to-api-first-design/analysis/2016/10/31

Traditional API Design

- API design happens after the release of some a data-rich application
  - Existing application “wrapped” in API
- Created as an afterthought.
  - Tightly bound application needs data/function exposed as API.
  - Shoe-horned in as a separate entity.



```
GET       /tasks - display all tasks
POST      /tasks -  create a new task
GET       /tasks/{id} - display a task by ID
PUT       /tasks/{id} - update a task by ID
DELETE /tasks/{id} - delete a task by ID
```

Advantages of Web APIs

- Suits multi-device environment of today.
- An API layer can serve multiple channels/devices.
  - Mobile/tablet/IoT device
- Scalable, modular, cohesive and composeable
  - If designed properly(e.g. microservice/Rest architecture)
- Concentrate on function first rather than data

APIs in the Internet of Things

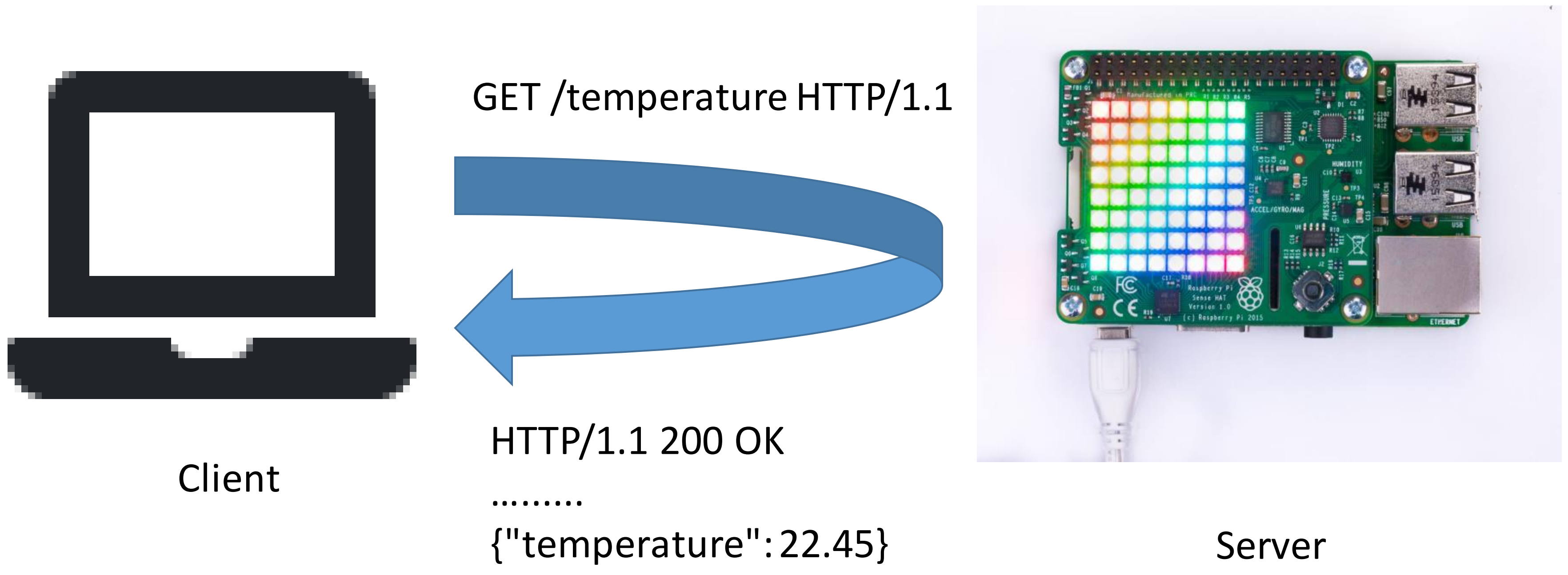
- Many new IoT devices emerging.
- Devices are limited on their own
  - Accompanying APIs invite innovation and generate value
- "Build a better mousetrap, and the world will beat a path to your door" - Ralph Waldo Emerson
  - e.g. Rentokil believe they have using APIs
  - Rentokil increased operational efficiency and compliance through the automatic notifications of a caught animal and its size.
  - Core to this are web APIs.

https://www.computerworlduk.com/it-business/rentokil-on-iot-rat-traps-cash-for-apps-incentives-apis-3612866/

HTTP Web API on an IoT devices

- Easy to set up a Web server on a Raspberry Pi(or smaller device):
  - Connects sensors/actuators to web
  - Access and Control your devices via the Web:
    - Web application program interface(Web API)

Demo



```
GET /temperature HTTP/1.1
```

```
HTTP/1.1 200 OK
.........
{"temperature": 22.45}
```

Indirect Messaging

Publish Subscribe

Using the “Middleman”

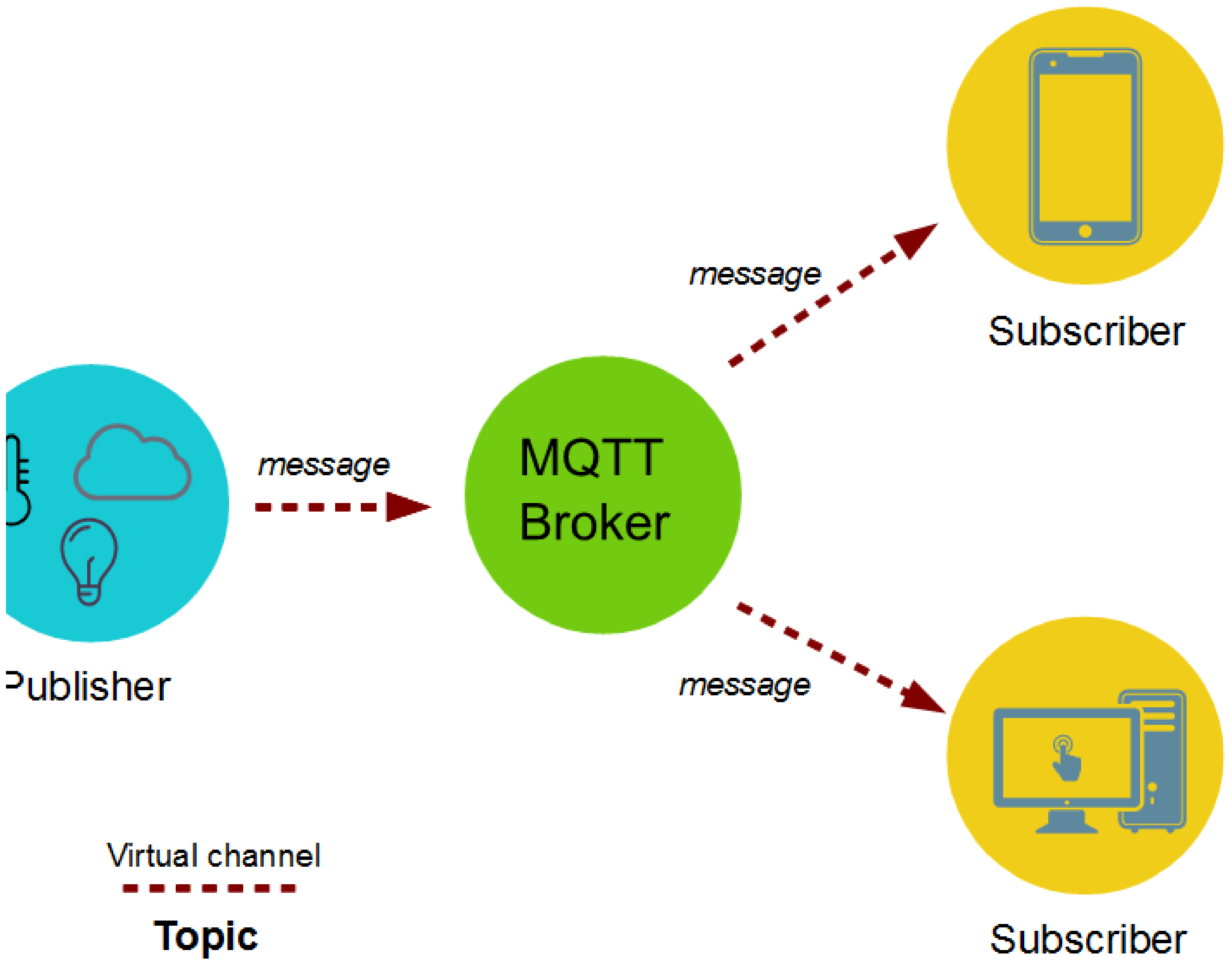
- Communication between processes using an intermediary
  - Sender → “The middle-man” → Receiver
  - No direct coupling
- Up to now, only considered Direct Coupling
  - Introduces a degree of rigidity
- Consider…
  - What happens if client or server fails during communication in Direct Coupling?
- Two important properties of intermediary in communication
  - *Space uncoupling*
  - *Time uncoupling*

MQTT

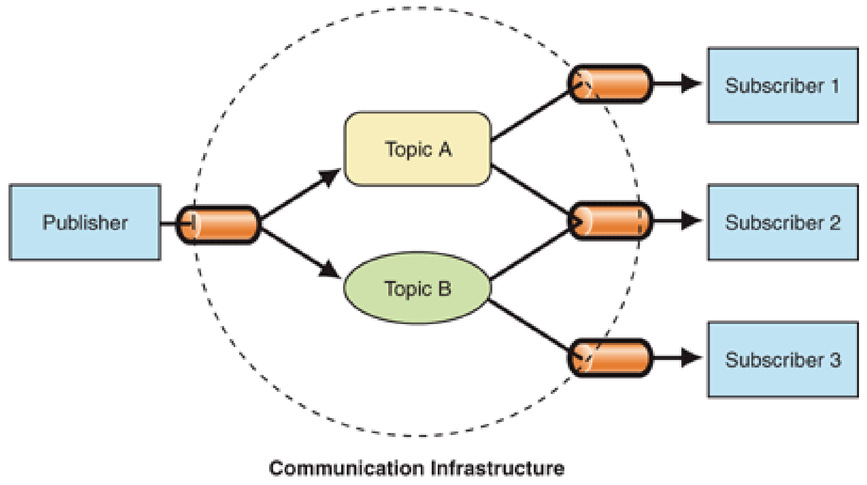
- MQ Telemetry Transport (MQTT)
- Telemetry
  - Remote measurements
- Created by IBM
  - from message queueing (MQ) architecture used by IBM for service oriented networks.
  - Telemetry data goes from devices to a server or broker.
  - Uses a **publish/subscribe** mechanism.
- Lightweight both in bandwidth and code footprint

MQTT – publish subscribe

- **Topics/Subscriptions:** Messages are published to topics.
  - Clients can subscribe to a topic or a set of related topics
- **Publish/Subscribe:** Clients can subscribe to topics or publish to topics.



Publish Subscribe Process



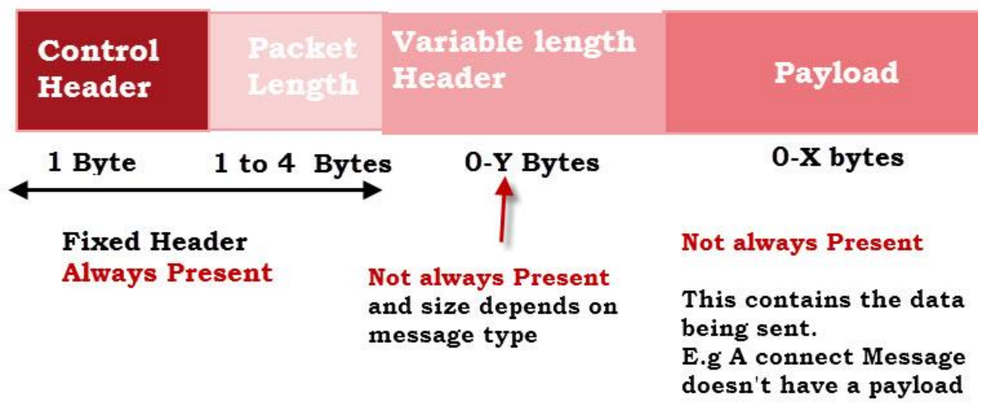
- A message is published once by a publisher.
- Many things can receive the message.
- The messaging service, or “broker”, provides decoupling between the producer and consumer(s)
- A producer sends (publishes) a message (publication) on a topic (subject)
- A consumer subscribes (makes a subscription) for messages on a topic (subject)
- A message server / broker matches publications to subscriptions
  - If no matches the message is discarded
  - If one or more matches the message is delivered to each matching subscriber/consumer

Publish Subscribe Characteristics

- A published messages may be retained
  - A publisher can mark a message as “retained”
  - The broker / server remembers the last known good message of a retained topic
  - The broker / server gives the last known good message to new subscribers
- A Subscription can be durable or non-durable
  - Durable: messages forwarded to subscriber immediately, If subscriber not connected, message is stored and forwarded when connected
  - Non-Durable: subscription only active when subscriber is connected to the server / broker

MQTT Characteristics

- MQTT protocol compresses to small number of bytes
  - Smallest packet size 2 bytes
  - Supports always-connected and sometimes connected
  - Provides Session awareness
  - “Last will and testament” enable applications to know when a client goes offline abnormally
  - Typically utilises TCP based networks e.g. websockets



MQTT Standard Packet Structure

MQTT Characteristics

- Three quality of service levels:
  - 0 = At most once (Best effort, No Ack),
  - 1 = At least once (Acked, retransmitted if ack not received),
  - 2 = Exactly once [Request to send (Publish), Clear-to-send(Pubrec), message (Pubrel), ack (Pubcomp)]
- Retained Messages
  - Server keeps messages even after sending it to all subscribers. New subscribers get the retained messages

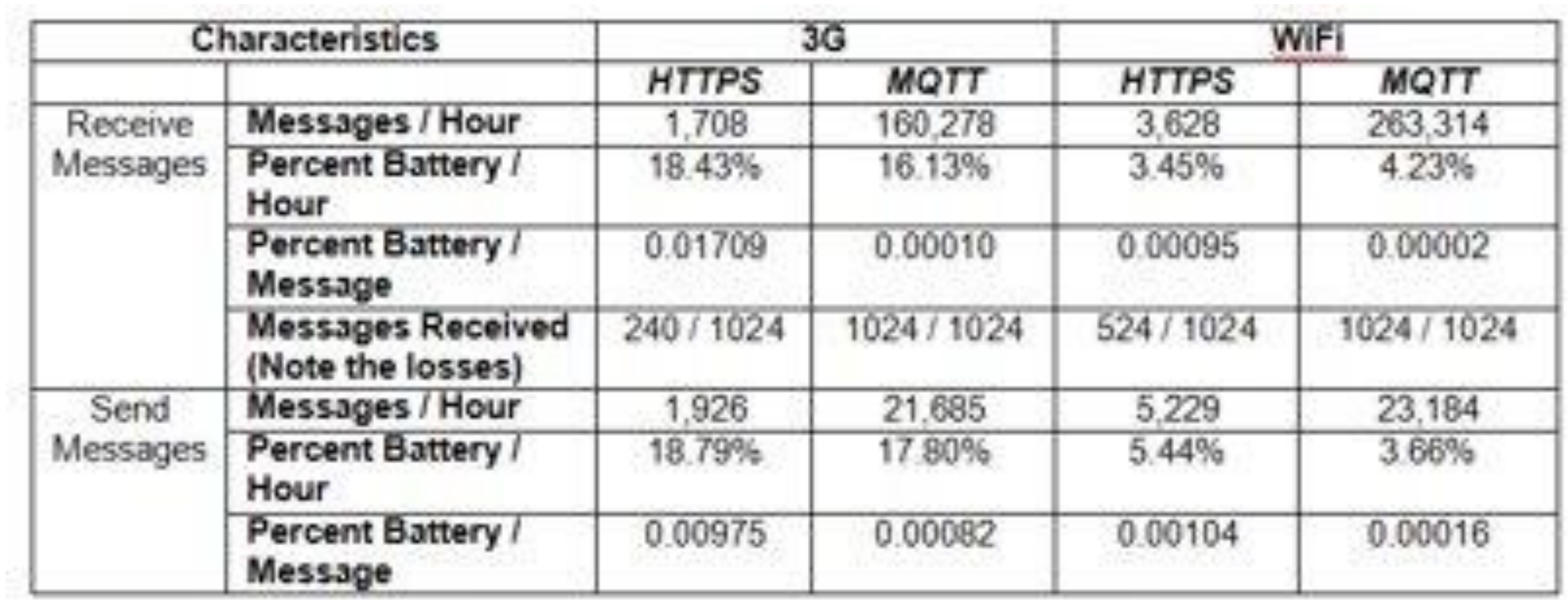
MQTT is Open Source

- Lots of implementations:
  - Mosquitto
  - Micro broker
  - Really small message broker (RSMB): C
  - Cloud services

MQTT vs HTTP

- Push delivery of messages / data / events
  - MQTT – low latency push delivery of messages from client to server and **server to client.** Helps bring an event oriented architecture to the web
  - HTTP – push from client to server but poll from server to client
- Reliable delivery over fragile network
  - MQTT will deliver message to QOS even **across connection breaks**
  - Decoupling and publish subscribe – **one to many delivery**

http://stephendnicholas.com/archives/1217

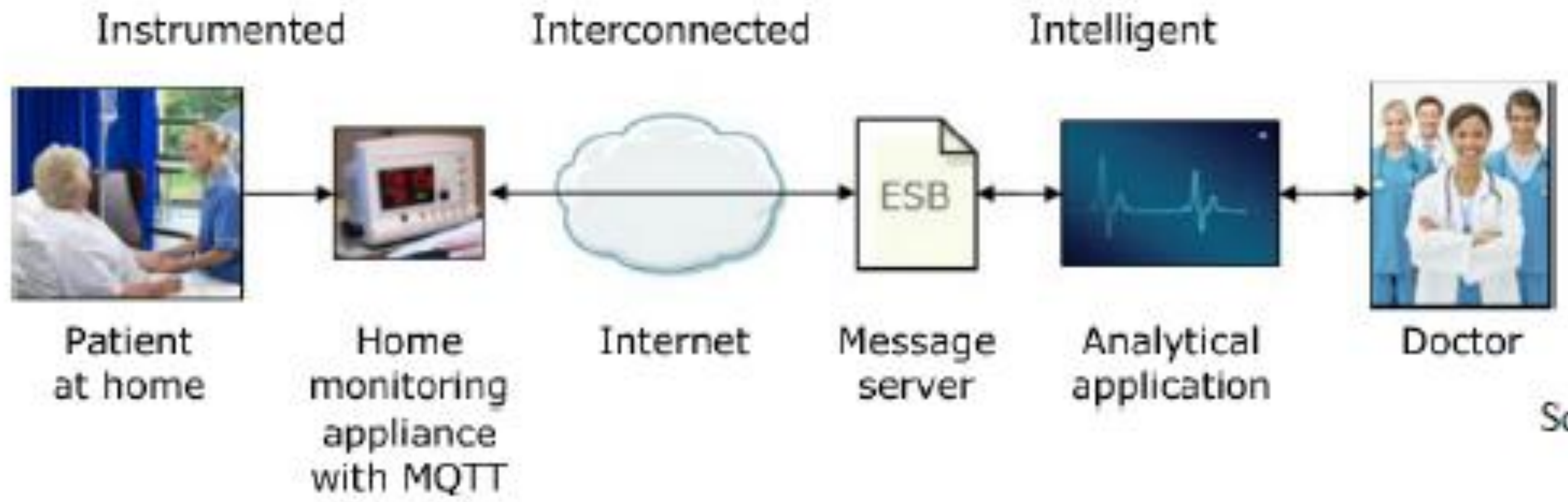


Characteristics		3G		WiFi	
		HTTPS	*MQTT*	*HTTPS*	*MQTT*
Receive Messages	**Messages / Hour**	1,708	160,278	3,628	263,314
	Percent Battery / Hour	18.43%	16.13%	3.45%	4.23%
	Percent Battery / Message	0.01709	0.00010	0.00095	0.00002
	Messages Received (Note the losses)	240 / 1024	1024 / 1024	524 / 1024	1024 / 1024
Send Messages	**Messages / Hour**	1,926	21,685	5,229	23,184
	Percent Battery / Hour	18.79%	17.80%	5.44%	3.66%
	Percent Battery / Message	0.00975	0.00082	0.00104	0.00016

*sending and receiving 1024 messages of 1 byte each.(source: https://www.ibm.com/developerworks

Application Example

- Home care monitoring solution
  - Home and patient instrumented with sensors.
    - E.g. door motion, blood pressure, pacemaker/defib.
  - Collected by monitoring service (broker) using MQTT
  - Subscribed by a health care service in the hospital
  - Alerts relations/health care profs. if anything is out-of-order



Source: Lampkin 2012