Aalto University
School of Science
Degree Programme of Computer Science and Engineering

HUANG, Fuguo

Web Technologies for the Internet of Things

Master's Thesis
Espoo, July 7, 2013

Supervisors: Professor Jukka K. Nurminen
Instructor: Dr. Mario Di Francesco

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: HUANG, Fuguo

Title:
Web Technologies for the Internet of Things

Date:	July 7, 2013	**Pages:**	54
Professorship:	Data Communication Software	**Code:**	T-110
Supervisors:	Professor Jukka K. Nurminen		
Instructor:	Dr. Mario Di Francesco		

Usually, the client-server model exchanges messages in a request/response pattern, where the server does not actively talk to clients. This is similar when a browser visits a web page, an HTTP request is sent to a web server. In order to synchronise the information between a browser and a web server, normally, there are polling or long-polling over Ajax requests. With Ajax, it increases the frequency of requesting the web server, with repeatedly connections and disconnections; or increases the workload of the web server to keep the requests open. The problem becomes severe when it happens in the scenario of Internet of Things (IoT) with a large amount of devices interconnected.

In effort to solve the Ajax abuse and real time communication problems in IoT, we provide a solution using WebSocket with its sub-protocol, the WebSocket Application Messaging Protocol (WAMP). In the thesis, we visualised the behaviours of IoT devices over an HTML5 application using publish/subscribe and Remote Procedure Call pattens providing by WAMP. Furthermore, we evaluate the current solution by comparing it with other possible solutions and protocols.

Keywords: Web, Technology, WebSocket, WebSockets, WAMP, Internet of Things, IoT, sensor, actuator, light, switch, JavaScript, HTML5

Language: English

Acknowledgements

Foremost, I would like to express my sincere gratitude to Prof Jukka K. Nurminen for supervising the thesis work. I especially wish to thank my instructor, Dr Mario Di Francessco, for his patience and motivation at all stages of the thesis.

I am grateful to Nico Kaiser for his contribution on the development of WAMP library. I am also thankful to my friends for their encouragement, and for helping me stay sane through the whole difficult period.

Most importantly, nothing would have been possible without the love and patience of my family.

Thank you!

Espoo, July 7, 2013

HUANG, Fuguo

Abbreviations and Acronyms

Ajax	Asynchronous JavaScript and XML
CoAP	Constrained Application Protocol
CORE	Constrained RESTful Environments
HTTP	Hypertext Transfer Protocol
ID	Identification
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
JSON	JavaScript Object Notation
MBWS	MessageBroker WebSocket Subprotocol
MQTT	Message Queuing Telemetry Transport
Pub/Sub	Publish/Subscribe
RPC	Remote Procedure Calls
REST	Representational State Transfer
RTT	Roundtrip Delayed Time
SOAP	Simple Object Access Protocol
STOMP	Simple (or Streaming) Text Orientated Messaging Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifiers
WAMP	WebSocket Application Messaging Protocol
XML	Extensible Markup Language
WSN	wireless sensor network

Contents

Chapter 1

Introduction

After the web turning from 1.0 to 2.0 where the network as platform, spanning all connected devices. Unlike Web 1.0, applications are as treated continually-updated services that get better the more people use it. This is done by consuming and remixing data from multiple sources, including individual users, while providing their own data and services in a form that allows remixing by others. [29]

Web browsers are now commonly installed in smart electronic equipments such as computers, smart phones, and tablets which can connect to the Internet. With Microsoft related operating systems (OS) such as Windows 7 and Windows 8, there might be Internet Explorer installed; With Apple related OS, such as Mac OS X and iOS, there will be Safari installed; With Google related OS, such as Android, there will be a WebKit[1] based browser(s) installed. Additionally, users are free of options to choose other web browsers.

Typical contents, presented by web browsers, are text and multi-medias such as image, audio and video. These contents are stored in other computers connected by the Internet. My work started at this point: a browser directly communicates with an internet-enabled device. To make the work interesting, the device should have good interoperability i.e. it should be able to communicate with sensors and actuator, or even with other devices.

1.1 Problem and Solution

Usability is another important element that mentioned in Web 2.0. Web pages are not just pages that have simple colours, logos, and animation, but they are more like desktop application. Thus, Web pages are turning to Web applications. [24]

[1] http://www.webkit.org

With the radical improvement of usability, Web applications contents have been presenting more dynamic. Data can be retrieved from multiple sources in real time and assembled on a single web application or page [24]. A web browser, one of the user agents that retrieves contents identified by Uniform Resource Identifier (URI) [1], is the gateway for users to the Internet.

With the purpose of presenting wonderful contents to users, a web browser communicates with a web server to fetch the contents. The synchronisation between a browser and a web server is, traditionally, based on HTTP request. For example, there are polling or long-polling over XMLHttpRequest (or Ajax). When we use Ajax, we retrieve data from a remote server by sending an HTTP request; For more other data, the web browser sends other HTTP requests. This is what we defined the problem in the thesis: With Ajax, it increases the frequency of requesting the web server, with repeatedly connections and disconnections; or rises the workload of the web server to keep the requests open. The problem becomes severe when it happens in the scenario of Internet of Things (IoT) with a large amount of devices interconnected.

However, this approach can radically improved by removing the redundant HTTP Requests. Then we transfer all data over a single TCP connection, which is what we called WebSockets. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections [33].

In effort to solve the Ajax abuse, bidirectional and real-time communication problems in IoT, we provide a solution using WebSockets with its subprotocol, the WebSocket Application Messaging Protocol (WAMP). In the thesis, we simulated and visualised the behaviours of IoT devices over an HTML5 application using publish/subscribe (Pub/Sub) and Remote Procedure Call (RPC) pattens, providing by WAMP. These two pattens cover the two most important interaction in IoT: notification and commanding.

Chapter 2

Background

The WebSocket Protocol enables full-duplex communication through a single socket between a client and a remote server. For this reason, this technology decreases the amount of opening ports on server sides, comparing with the traditional mean of retrieving resource. Moreover, this protocol is not just an enhancement of the current HTTP protocol; it, however, represents enormous advance, especially for real-time, event-driven communication [25].

2.1 Traditional Communication

Usually, the client/server model is designed as a request/response model, where the server does not actively communicate with clients. The client/server model is one of the network architectures to connect computers or applications. A client could be a computer, or an application, such as a web browser.

A server is, usually, a (powerful) computer that keeps some applications running as constantly as possible. Those applications could be a database, a web server (application), or a printer server (application) that links to a physical printer.

The HTTP protocol is a request/response protocol [14], where a client, or rather a user agent (UA) communicates with a server through a request and a response message(s). In other words, a client retrieves resources from a server in a request message, while the server hands over the contents in a response message. This is a general model that a client retrieves data from a server.

Generally, there are three ways to retrieve data [25]: polling, long polling and streaming, while all of these are uni-directional.

With polling, a client sends a request to a server, e.g. a web browser sends a request with Asynchronous JavaScript and XML (Ajax) [17] to a

web server and expects data. This is a solution of high quality, if the client knows exactly the time when the data is available. However, as you might notice, the server might not have the contents when the client requests, or the contents are temporarily not available. If this situation would happen in practise, the client would have repeated polling for data. This will result in a large redundant headers generated by HTTP protocol. Consequently, the approach leads a heavy workload to the server and will not be suitable in the scenario, such as, a sensor network having thousands of sensors.

Another method, Long polling, is an improved approach, with respect to the frequency of request. Compared with polling, long polling does similarly, expect that the server holds the client's request for a certain period until timeout. Within this period, the server will send a response(s) back to the client, if there is data available. This method reduces the frequency of HTTP requests, but it increases the workload of a server, such as, ports handling. Generally, an HTTP request is initiated over TCP/IP connections (but any other protocols that guarantee a reliable transport can be replaced) [14]. In TCP/IP connections, a port number is 16 bits [31], which means the maximum number of ports is $2^16 = 65536$. However, one characteristic in sensor network is scalability, which means the number of sensor nodes could yield to a larger amount than that of the ports. Even though all ports are dedicated to communication, it is insufficient to support a enormous sensor network.

As for the HTTP streaming, or known as HTTP server push, a server sends a response and keeps the connection open and alive, where the connection is triggered by a client's request. The drawback of this approach is that streaming is still encapsulated in HTTP, and thus buffer mechanism might increase the latency of messages delivering. Moreover, buffer is not an approach for an IoT device, considering the device's energy constraint.

2.2 The Internet of Things and WebSocket

In the concept of Internet of Things (IoT), the networked objects, called devices, are deployed worldwide and connected over the internet. Moreover, the IoT device is individually addressable, so that each device is interconnected and can be accessed through the standards of the web. With the purpose of performing a better communication with IoT devices, the WebSocket protocol enables two-way communication and is arranged to substitute the disadvantage that existing in the traditional client/server model.

The WebSocket protocol supports data frame including text and binary. Once the WebSocket connection has been successfully established, generally,

data frames transfer between a client and a server. However, in consideration of optimising WebSocket connections between a client and a server, the client can request that the server use a specific subprotocol by including the —Sec-WebSocket-Protocol— filed in its handshakes[33]. A subprotocol is a application-level protocols layered over the WebSocket protocol[33]. A subprotocol could be implemented by the servers and can be also versioned. The server chooses one or none subprotocol.

2.3 WAMP

WebSocket Application Messaging Protocol (WAMP)[1] is a subprotocol officially registered in WebSocket Protocol Registries [2] , which gives structure to messaging by providing two hight level messaging patterns: Publish/Subscribe (Pub/Sub) and Remote Procedure Calls (RPC).

WAMP fills in the blanks in the WebSocket protocol: a low-level specification which originally provides raw messaging communication (text or binary). Moreover, WAMP does not make any extra standard, but it bases on established Web standards: WebSocket, JSON and URI. The WebSocket standard is the default transport channel where WAMP is binding. With the WebSocket Standard, WAMP is assumed to be given a reliable, ordered, and full-duplex message channel.

2.3.1 JSON

JavaScript Object Notation (JSON) is a text format for the serialisation of structured data. It is derived from the object literals of JavaScript, as defined in the ECMAScript Programming Language Standard, Third Edition [ECMA] [9]. JSON has numbers, strings, booleans and null as its primitive types. Additionally, it has objects and arrays as structured types. JSON becomes popular to exchange data between in web services or applications. With JSON, WAMP message payload is serialisable and provides data types according to those of JSON.

2.3.2 URI

Uniform Resource Identifiers (URIs) provide a simple and extensible means for identifying a resource. URIs have a global scope and are interpreted consistently regardless of context [27]. WAMP binds to URIs by default,

[1] http://wamp.ws/

[2] http://www.iana.org/assignments/websocket/websocket.xml#subprotocol-name

then it assumes global assignments and resolution to be unique. URIs are used as IDs for both topics in Pub/Sub and procedures in RPC.

2.4 Other Registered WebSocket Subprotocols

Apart from WAMP, there are MessageBroker WebSocket Subprotocol (MBWS), Simple Object Access Protocol(SOAP) and Simple Text Oriented Messaging Protocol (STOMP) can be used as subprotocols, according to the WebSocket Protocol Registries.

MBWS is used by messaging clients to send messages to, and received messages from, an Internet message broker. The Internet message broker is also called a message broker which queues messages, sent by clients, for asynchronous deliveries to clients. [20].

Like WAMP, MBWS provides full-duplex and reliable transport; Unlike WAMP, MBWS does not obviously provide higher level messaging patterns. MBWS, however, defines a binary message frame starting with an MBWS binary header, and a text message frame starting with an MBWS text header. MBWS provides two additional features. Firstly, it supports connection recovery, which has not defined by the WebSocket protocol. Secondly, it supports metadata in MBWS message headers. MBWS has a light weight version, called MessageBrokerLight WebSocket Subprotocol (MBLWS) which only supports Message Metadata [20]. However, MBWS is still a draft.

SOAP is a lightweight protocol intending for exchanging structured information in a decentralised and distributed environment. SOAP uses XML technology to exchange data. [18] Microsoft, additionally, defines SOAP Over WebSocket Protocol Binding (MS-SWSB) [3], a subprotocol for the WebSocket Protocol. The subprotocol involves a Web Services Description Language (WSDL) and supports message exchange patterns (MEPs). With MEPs, SOAP supports one-way, two-way and request/response message exchange patterns, which involves Pub/Sub and RPC.

Simple (or Streaming) Text Orientated Messaging Protocol (STOMP) [4] is a frame based protocol. It refers to a STOMP frame includes a command, optional headers and a body. STOMP is text based with UTF-8 encoding by default, but it, optionally, support binary messages. As for message patterns, STOMP version 1.2 supports Pub/Sub. An example of Pub/Sub model using STOMP with WebSocket is discussed in [44].

[3] http://msdn.microsoft.com/en-us/library/hh536812.aspx

[4] http://stomp.github.io

	WAMP	**MBWS**	**SOAP**	**STOMP**
Data Representation	JSON	binary and text with MBWS headers	XML	binary and text with headers
Message Patten	Pub/Sub, RPC	not specified	MEP	Pub/Sub
Connection Recovery	no	yes	no	no
Metadata	currently no	yes	yes	no
Released	yes	draft	yes	yes

Table 2.1: WAMP and other WebSocket subprotocols

Table 2.1 is a comparison summary among WAMP, MBWS, SOAP and STOMP:

Chapter 3

Internet Of Things

3.1 Definition

The term, Internet of Things, was firstly mentioned by Kevin Ashton in once his presentation in 1999 [2]

Semantically, Internet of Things means "a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" [22].

3.2 Things: The Bridge between Real World and Virtual World

The "things" refers to the "objects" in the semantical explanation. The "things" are heterogeneous objects. The initial idea of the "things" started with Radio-Frequency IDentification (RFID) tags by Auto-ID Labs[1], a world-wide network of academic research laboratory in the field of networked RFID and emerging sensing technologies. Hereafter, other concepts of "things" emerged: NFC tags, Wireless Sensor and Actuator Networks (WSANs), Spime and Smart Items.

3.2.1 NFC and WSANs

NFC is a proximity technology that provides contactless communication by using ISO/IEC 14443 and ISO/IEC 18092. NFC tags utilise Reader/Writer mode, one of the modes that a NFC devices should supports [26], to store data and interact with users. A typical scenario is that a NFC tag acts

[1]http://www.autoidlabs.org

as a smart poster, where a user could place a NFC based mobile phone near a smart poster and thus would retrieve further information. e.g. if a smart poster stores movie poster data, it can forward a user to the movie trailer or its home page. Additionally, there are card emulation mode and Peer-to-Peer mode. Card Emulation Mode e.g. turns a mobile phone into a smart card such as a credit or debit card, and the NFC reader cannot distinguish the difference between the mobile phone and smart cards. Thus, with this features, it can implement a new way of payment system called Mobile Payments. NFC Peer-to-Peer (P2P) mode is designed based on the standards ISO 18092, which allows two NFC devices communicate mutually. [26]

WSANs are composed of large numbers of minimal capacity sensing, computing, and communicating devices and various types of actuators. WSANs constitute an important and exciting new technology with great potential for improving many current applications. It also creates new revolutionary systems in areas such as global-scale environmental monitoring, precision agriculture, home and assisted living medical care, smart buildings and cities, industrial automation, and numerous military applications. [40]

They are, together with RFID, are recognised as the atomic components that links the real world with the digital world.[41]

3.2.2 Spime and Smart Items

Spime and Smart Items are also considered to be “Things” Oriented Object. Spime is a neologism and firstly mentioned by Bruce Sterling, a science fiction writer, in 2005. Sterling stated that there will be a new kind of object - user-alterable, baroquely multi-featured and programmable - in the future, for which Sterling invented the term “Spime”. [41]. Sterling described that the Spime is one of the objects in Internet of Things since Spime enables a thing to be searchable. i.e. A thing can be searched by a search engine from a virtual world to a physical world. [41]

Spime is a bit theoretical, but we can find some of its prototype in the real world, and we call them Smart Items [3].

IoT is also derivative from the vision of “Internet”. Vasseur and Adam described Smart Objects in their book, Interconnecting Smart Objects with IP: The Next Internet [42]. A Smart Object is an item equipped with a form of sensor or actuator, a tiny microprocessor, a communication device and a power source. The sensor or actuator gives the smart object the ability to interact with the physical world. Smart Objects are interconnected using the Internet Protocol (IP).

According to IPSO Alliance, the IP stack is a light protocol that already

connects a huge amount of communicating devices and runs on tiny and battery operated embedded devices. This guarantees that IP has all the qualities to make IoT a reality. [3]

3.3 Sensors and Actuators

A sensor detects or measures the physical environment (such as temperature, speed, and location etc) and converts it to a form (such as a digital form) which can be read by human or used for other purpose. For example, a light sensor senses the brightness of the environment, then the sensor converts the brightness into a digital form (data). The data can be, further, transferred to other receivers.

Sensors can interconnect wirelessly, and form a network, which is called wireless sensor network (WSN). WSN monitors the environment or other physical data distributively. According to [10], WSN can be apply to a wide range of uses: monitoring space, things, and the encompassing space. Particularly, the first category, space, which is related to the thesis, covers light, temperature, humidity and other environment data.

Sensors perform tasks more than measuring environments. The second category applies on things or objects. Sensors, that being the case, are applied more in area of such as, the detection of structures breakages [5]. However, things and the environment are sometimes interconnected and interactive. This is the third category when sensors are used to e.g. monitor the wildlife habitats [10] and the interaction between traffic queuing and the frequency of traffic lights switching. In short, the WSN provides a good source of data for Internet of Things.

Sensors collect and transfer data for further use, but an actuator provides controlling functions to a system. e.g. a light switch can turn on and turn off a light; Here the light switch is an actuator. Sensors and Actuators work mutually can optimise processes. For example, sensors are installed in chemical production to bring much greater granularity to monitoring, where the sensor collects data and transfers them to computers which subsequently analyse the data and signal actuators to adjust the processes, such as ingredient mixtures, temperatures, or pressures [6]. The whole process can then be automated.

3.4 Internet-based Communication Protocols and Standards

3.4.1 TCP, UDP and IP

Transmission Control Protocol (TCP), one of the two major transport-layer protocols, provides a reliable bi-directional communication [8]. TCP addresses the communication problem between applications with the following major features:

- Transferring Data over a connection
- Connecting two endpoints
- Data is received the same as it is delivered
- Data is sent in sequence
- Reliable startup and shutdown

User Datagram Protocol (UDP) is the other major transport-layer protocol used in the Internet [8]. Compared with TCP, UDP is less complex and thus less reliable. For example, UDP is connectionless, meaning a message can be sent between two endpoints without prior arrangement; UDP sends and receives individual messages, meaning it does not divide a message into multiple parts.

Internet Protocol (IP) provides a communication mechanism for computers or devices [30]. IP distinguishes each computer or device in network by a unique 32-bits number known as Internet Protocol Address (IP address). TCP or UDP packets are usually encapsulated into IP packets.

3.4.2 HTTP

Hypertext Transfer Protocol (HTTP) is an application layer protocol exchanging or transferring contents, such as hypertext, in request/response pattern [14]. HTTP is assigned to port number 80 over either TCP or UDP connection [32]. HTTP communication, however, usually is established over TCP/IP connections with the default port TCP 80. HTTP can be also established other protocol, such as unreliable UDP.

When a browser requests resources from a server, usually the browser sends a HTTP request to the server. Then the server answers with an HTTP response. Before HTTP 1.1, a pair of request/response is, usually, sent in a single TCP connection. Generally, a TCP connection is expensive to create,

while most HTTP 1.0 or older connections use TCP at least efficiency, which leads to congestion and unwanted overhead [38]. To improve this, HTTP provides connection reuse mechanism [13] by using Keep-Alive in general headers. This mechanism has been further improved in HTTP 1.1, and thus all connections are set to persistency by default. This mechanism aims for reducing CPU and memory usage, pipelining, reducing network congestion, reducing latency the frequency of TCP opening handshakes and error reported improvement. With pipelining, the performance of HTTP 1.1 can be largely improved compared with HTTP 1.0. [28].

At this point, HTTP persistent connections seem to be perfect and the WebSocket protocol is unnecessary. The HTTP persistent connection, however, is restricted by its practical consideration. Firstly, Servers are, usually, maintained by setting time-out value. Consequently, the mechanism kills idle connections. e.g. Apache 2.0 server will set the timeout at 15 seconds by default [2], while Apache 2.2 reduces the amount down to 5 seconds[3]. Additionally, this implies that both clients and servers need to be capable to recover from any closed events and subsequently indicates delay in communication. Secondly, any client that use persistent connections is instructed to limit the number of simultaneous connections that they maintain to a given server at the maximum of two, for the purpose of improving HTTP response time and avoiding congestion.

For a browser, in order to synchronise the information with a web server, normally, there are polling or long-polling over Ajax. When we use Ajax, we retrieve data from a remote server by sending an HTTP Request; and if we would like to retrieve more data, we need to send other HTTP Requests. HTTP request/response will, finally, confront with the fact that it increases the frequency of requesting the web server, with repeatedly connections and disconnections; or increases the workload of the web server to keep the requests open. The problem becomes severe when it happens in the scenario of Internet of Things (IoT) with a large amount of devices interconnected. Figure 3.1 illustrates sequence diagram where a client (browser) fetches data from a server using HTTP.

However, this approach can radically improved by removing the redundant HTTP Requests. Then we transfer all data over a single TCP connection. This is what we called the WebSocket Protocol. We will expand more about the WebSocket Protocol in Subsection 3.4.3.

[2]http://httpd.apache.org/docs/2.0/mod/core.html#keepalivetimeout
[3]http://httpd.apache.org/docs/2.2/mod/core.html#keepalivetimeout

3.4.3 HTML5 and WebSockets

HTML5 is hypernym that consist of a large number of web technology [44]. HTML5 covers the following area: new semantics, new CSS styling, multimedia, Graphics & 3D, Device Access, Performance, offline & storage and connectivity. Amount all, the connectivity consists of communication technology such as Web Real-Time Communications (WebRTC), Server-Sent Events (SSE), Cross-Document messaging and WebSocket. WebRTC[4] provides real-time multimedia communication on web without requiring plugins; SSE provides an API for a browser to receive pushing notification from a server via an HTTP connection [21]; While Cross-Document messaging or Web Messaging largely expands the possibility and capability of communication of web browsers with least side-effect on security and privacy. Cross-Document messaging is designed to overcome the same-origin policy.

The goal of WebSocket technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g. using XMLHttpRequest or ¡iframe¿s and long polling) [5]. Older web technology, like HTTP, remains inefficiencies problems [44]. Firstly, it is not natively designed for nowadays Web Applications with rich contents and interactive demands; Secondly, overheads and unnecessary information cumulate swiftly as the interaction between the client and server continues. WebSocket improves communication problems by solving the above problems, while keeping simplicity. Both WebRTC and WebSocket are in effort to enhance the real-time communication. WebRTC provides APIs that allow web browsers communicate with each other directly, while WebSocket provides the real-time communication mechanism between a client and a server. Figure 3.2 illustrates sequence diagram where a client (browser) fetches data from a server using WebSockets.

3.4.4 Internet-based Communication Protocols and Standards with IoT

The currently architecture of IoT consists three layers [45]. From top to bottom, they are: application layer, network layer and perception layer. The perception layer consists of the unquietly addressable objects. The network layer connects the objects and connects objects to computers. The application layer makes the IoT data meaningful, useful or extensible.

[4]http://www.webrtc.org/reference/architecture

[5]http://www.websocket.org

TCP/IP and UDP ensure the objects communicable in IoT. Moreover, they ensure that the objects can be communicated outside the IoT network. Subsequently, the data, e.g. that sensors collected, is utilisable. For example, the data collected by sensors can be transferred to a data centre and can be further analysed.

Another issue about IP, Internet Protocol Version 4 (IPv4) is the dominant IP protocol in the Internet currently. IPv4, however, has exhausted [36] and thus could not meet the 'unquietly addressable' requirement in IoT. IP Version 6 (IPv6) is its successor. "IPv6 enables end-to-end communication, in which any IPv6 'smart things' can connect to any other IPv6 devices or system from any place and at any time. It enables the possible extension of the Internet to any device, sensor or actuator, which can become real nodes of the future Internet." [43]. It primarily provides the following features[11]:

- Expanding addressing capabilities to support a greater amount of addressable nodes.
- Simplifying header format to reduce the cost of packet handling.
- Improving support for extensions and options to allow greater flexibility in the future
- Flowing labelling capability to label packets and thus to handle, e.g., real-time services.
- Authentication and privacy capabilities

HTTP and WebSocket ensure that IoT can be used or extended by applications. For example, a web interface that can visualise temperature data. Subsequently, IoT network can be directly or indirectly connect to end users.

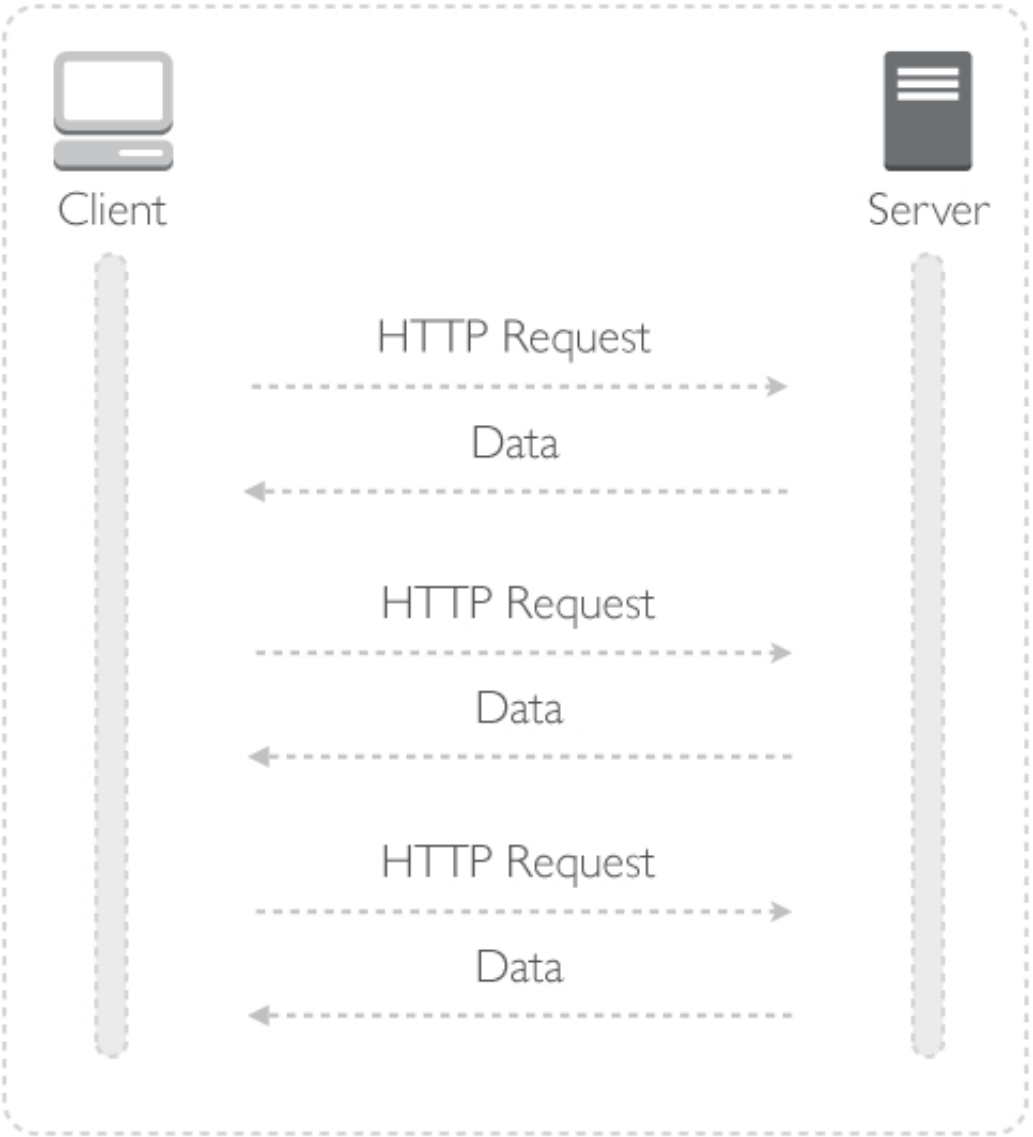

Figure 3.1: A Client Fetches Data from A Server with HTTP

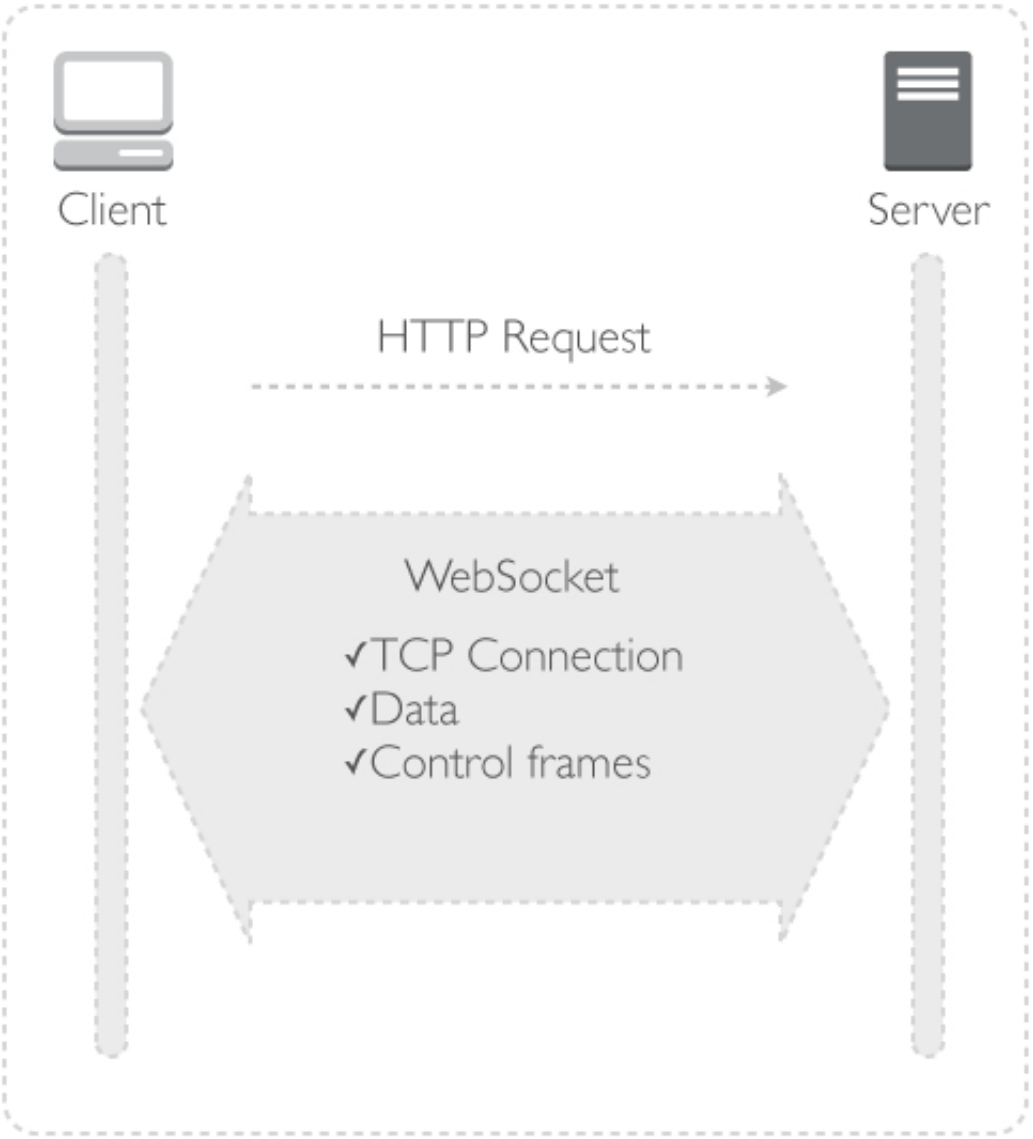

Figure 3.2: A Client Fetches Data from A Server with WebSockets

Chapter 4

Web Services for The Internet of Things

A Web Service "is a network accessible interface to application functionality, built using standard Internet technologies" [37]. In other words, an application can be accessed by another application, client or user over the Internet, via web technology such as HTTP, XML or SMTP. A Web Service, usually, provides services based on standards at the layer between service requesters and service providers. This means, it does not matter how are the services implemented, e.g. using C/C++, Java, Python or Javascript, or based on whatever platforms, e.g. Windows Server, Max OS Server or Unix Server.

When a request is sent to a web service layer, this layer will start a procedure which further requests the service provider. When the procedure is finished, the web service layer will response to the sender results (the response is sometimes optional and it relies on how is the web service implemented). The communication, between the service requesters and the web service layer, is usually based on standard protocols, which aims at cross-platform interoperability.

The next, we will talk about a sequence of protocols that can be used in Web Services.

4.1 REST

Representational State Transfer (REST) is introduced by ROY T. FIELDING in his doctoral dissertation in 2000. REST is a web architectural style. It is, however, not a standard or protocol.

An architectural style is "a coordinated set of architectural constraints that restricts the roles and features of architectural elements, and the allowed

relationships among those elements, within any architecture that conforms to the style. Thus, a style provides a name by which we can refer to a packaged set of architectural design decisions and the set of architectural properties that are induced by applying the style." [15]

REST was developed in parallel with HTTP protocols with the basic four constraints: resources identified by URI; resources manipulated by standard HTTP methods; resources with various representations; and stateless communication. Resources are data that can be addressed by URIs, which further implies resources are differed by URIs. A representation is a way to present resources. For example, a text, an Image or an HTML document. The representation, moreover, consists of metadata describing the data and, occasionally, metadata describing metadata.

All REST communications are stateless. Thus, each request involves all the necessary information that a receiver needs to understand the request. An example of stateless design is shown in Figure 4.1 [34]

Figure 4.1: Stateless Design

As it is illustrated, the request specifies which page the client fetches, i.e. page 2; while the response message contains what the current page is, what the next page is and also the requested page.

On the contrary, stateful design, as shown by Figure 4.2 [34], the server saves each client status and responses the corresponding status to the clients who request them. In 4.2, the client need not to specify which page it fetches, but just call the corresponding procedure. The server computes the result and returns the page to the client.

The replies are in XML format in Figure 4.1 and 4.2, but the REST reply is not restricted in XML document. It could be JSON or other data structure. Usually, REST is implemented relying on HTTP and other web technologies.

Figure 4.2: Stateful Design

4.2 SOAP

SOAP, a protocol designed for web services, specifies, firstly, a XML-based envelope to transfer information; Secondly, a set of rules for translating application and platform-specific data types into XML representations [37].

SOAP is largely based on XML standards and provides a standard way to structure XML messages. Thus, SOAP has two related applications: Remote Procedure Call (RPC) and Electronic Document Interchange (EDI). RPC is one way to do distributed computing, where one programme calls a function or method (called procedure) on another, optionally with parameters and receiving return value. EDI is used in automated transactions.

A SOAP message contains an optional SOAP header and a required SOAP body. The SOAP header consists of several information blocks and each of the block is a header. Each header explains how the message is to be parsed. The message body is the actual message in XML syntax.

SOAP provides a mechanism to handle errors which is called SOAP Faults. A SOAP fault consists of fault code, fault string, the fault actor and the fault details. The fault code has been standardised in namespace belong to `http://www.w3.org/2001/06/soap-envelope`; The fault string is an explanation of the error for human to read, while the fault detail is a further explanation about the error. The actor is a concept in SOAP Message Exchange Patterns (MEP), a pattern of message exchanged between SOAP nodes, which has been standardised at [4].

When a SOAP message is sent from a provider to a receiver, it passes one or more web services. Each of the web service is called intermediary. All these intermediaries that the SOAP message travels through is called a message path. We call each intermediary on a message path an actor. Thus, the fault actor is the unique identifier of message processing node where the

error occurs.

SOAP does not specify the means of SOAP message transports. Generally, SOAP can be transferred through HTTP, FTP, BSD sockets and SMTP etc.

4.3 CoAP

Constrained RESTful Environments (CORE) is a working group under Internet Engineering Task Force (IETF) foaming in 2010[1]. CORE has been contributing the major standardisation work for Constrained Application Protocol (CoAP).

CoAP is a protocol (in draft) used in constrained nodes and constrained networks and designed for machine-to-machine (M2M) applications [35]. A constrained node, usually, has a lower capacity controller with a small amount of ROM and RAM. Constrained networks refer to the networks, generally, with high packets error rates and a low throughput.

CoAP applications or endpoints interact via a request/response model. It also supports built-in services and resources discoveries. With CoAP, applications or endpoints can easily interact with an HTTP based web architecture. [35]

Unlike HTTP, CoAP stack is designed for constrained environment, while CoAP, additionally, differ from HTTP in terms of power consumption and overhead. Generally, CoAP power consumption bytes per-transaction are lower than those of HTTP, which implies longer battery lifetime. Figure 4.3 illustrates a comparison between HTTP and CoAP stack. CoAP consists of the concept of HTTP, but it has been re-designed with the respect to the low processing power and energy consumption constraints. [7]

CoAP relies on UDP. Moreover, CoAP cannot communicate with a browser directly without a gateway which translates CoAP to HTTP. The gateway can be an application. e.g. Copper[2], a firefox plug-in, currently, is available for users to browse and interact with Internet of Things devices.

4.4 MQTT

Message Queuing Telemetry Transport (MQTT)[3] is a machine-to-machine (M2M) / IoT connectivity protocol. There are two main specification for

[1] https://datatracker.ietf.org/wg/core/history/

[2] https://addons.mozilla.org/de/firefox/addon/copper-270430/

[3] http://mqtt.org

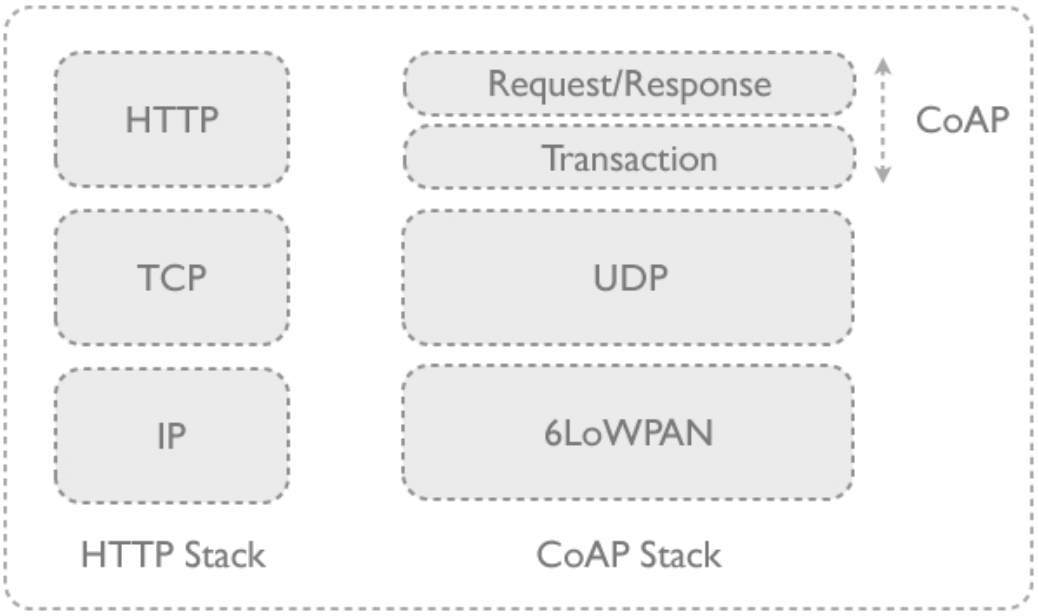

Figure 4.3: HTTP and CoAP Stack Comparison

MQTT: MQTT v3.1 and MQTT-S (MQTT for Sensor) v1.2. We will compare MQTT V3.1 with WAMP. Since MQTT-S does not work on TCP/IP network, it is out of scope of my project.

MQTT is designed as an extremely lightweight publish/subscribe messaging transport; while MQTT-S is a publish/subscribe messaging protocol for wireless sensor networks (WSNs) and is originally designed for non-TCP/IP networks.

MQTT V3.1 and WAMP are both working on TCP/IP network. Different from WAMP, MQTT V3.1 assumes the communications channel is more likely to be unreliable.

MQTT V3.1 has a two bytes message header and a UTF8 encoded payload. Message type, such as Pub/Sub, is defined in the header. Unlike WAMP, MQTT V3.1 does not yet support RPC. Additionally, MQTT V3.1 provides three levels of Quality of Services (QoS) control.

Level 0 is called "At most once delivery", when the message is delivered highly dependent to the underlying TCP/IP network. There is no any expectation that the recipient will sent back a response and the message arrives at the recipient at once or it just loses.

Level 1 is called "At least once delivery", when a response is expected to be sent back from the recipient, otherwise, the sender will re-send the message.

	WAMP	CoAP	MQTT
Transport Layer	TCP	UDP	TCP
Get Resource	over WebSockets	HTTP Requests	BSD Sockets
Browser	yes, HTML5 support	no, only via Firefox plugin currently	no, not directly
Server	WebSocket Server with WAMP support	CoAP Server	MQTT Server
RPC	yes	request/response model (through REST APIs)	no
Pub/Sub	yes	yes	yes
Proxy Support	through WebSockets	built-in	external proxy to translate protocol (implies delay)

Table 4.1: WAMP, CoAP and MQTT comparison

Level 2 is called "Exactly once delivery", when the message will be received and only received once by the recipient. More helper messages will be exchanged at this level to ensure the quality of delivery, and hence there is an increase in the network traffic [23].

WAMP, however, does not support such QoS control. Finally, WAMP has been officially registered in the WebSocket Subprotocol Name Registry, while MQTT is not a member.

4.5 Comparison

Table 4.1 is a summary of the comparison between WAMP, CoAP and MQTT.

Chapter 5

Patterns for Web-based IoT Applications

5.1 RPC

Remote Procedure Call (RPC) is one of the Inter-Process Communication methods for data exchange among threads or processes on different hosts. Unlike a local application case [39], where the code runs all locally, RPC solves the problem of excusing codes of the whole application which is distributed cross different networks or the Internet.

RPC can be further developed into a massage pattern [1] involving two roles, client and server. Usually, RPC pattern is initiated by a client, which sends an request to a remote server; The server processes based on the request, then responses to the client. An example of RPC pattern is shown in Figure 5.1. The client starts its procedure sequence when the programme is running. The procedure sequence starts with Procedure 1. The client, afterwards, needs the result from Procedure 2 which locates on the Server. Thus, it sends a request to the server. The server handles the logic when the request comes, and thus it starts Procedure 2. At the moment, the Client is waiting for the result until the server responses to the client. With the result, the client can now continue to execute the next procedure, Procedure 3.

RPC is commonly used in real world scenarios. For example, Facebook, a social network web platform, provides open APIs that allows third party applications to access its data. Here, Facebook plays the role of the server(s), while the third party applications play the role of the client. When a third party application (client) sends a request to Facebook asking, e.g. a list of

[1] http://wamp.ws/faq#rpc

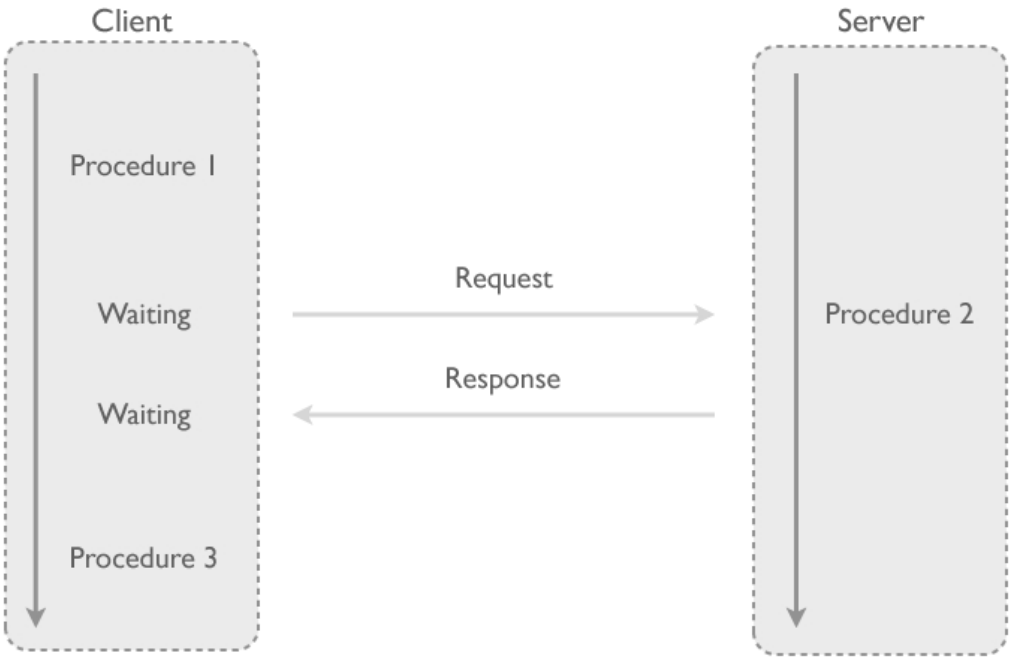

Figure 5.1: An Example of RPC Pattern

the user's friends, Facebook servers process the request. Facebook servers, then, response to the client, when the result, either success or error, is ready. The client can then use the list of friends to perform further procedures.

From the perspective of IoT, RPC pattern covers one of the two most important interaction in the IoT application architecture, i.e. commanding. In the case of IoT, nodes could locate on different locations from cities to countrysides. Commanding is helpful to configure a node remotely, e.g. setting the parameters (device identification, timeout and reporting rate). As for an actuator, RPC pattern can control its action. e.g. a light actuator, or called a light switch, has the actions of on and off. With commanding, the light can, thus, be switched on or off remotely.

5.2 Publish/Subscribe

Publish/Subscribe (Pub/Sub) is one of the messages exchanges patterns. Messages are exchanged based on publishing and subscribing. A sender of a message is called a publisher. Usually, each message contains a topic. A receiver filters and receives a message by the message topic. The receiver

who subscribes topics is called a subscriber.

Pub/Sub architecture consists, usually, at least publishers, subscribers and a broker(s). The broker maps the right messages to the right receivers and exchanges messages among publishers and subscribers. Generally, there is, on one hand, no restricted boundary between a publisher and a subscriber. This means, a publisher can also be a subscriber and receives messages from other publishers; while a subscriber can also be a publisher and publishes messages with topics. On the other hand, there is no order between publishing and subscribing. This means, a publisher can publish a message with a topic without subscribing any topic; or even no one subscribes the topic. The subscriber does not have to subscribe any topic either. This refers to, if the aforementioned situation happens, then messages are , e.g., simply discarded. All the message exchanging logics, however, are handled by the broker(s). The broker(s) should specifies and implements corresponding functions to different messaging situations.

An example of Pub/Sub pattern is shown in Figure 5.2. Client 1 plays roles in both subscriber and publisher in different stages. i.e. When it subscribes topic 1, client 1 is treated as a subscriber. Client 1, however, changes to a publisher when it publishes a message with the topic (topic 1). The broker analyses the message and maps the topic to the subscribers who are interested in the topics. In this case, both Client 1 and Client 2 are interested in the topic. Hence, the broker replies the message to both of them. Client 1, and thus, changes its role to a subscriber.

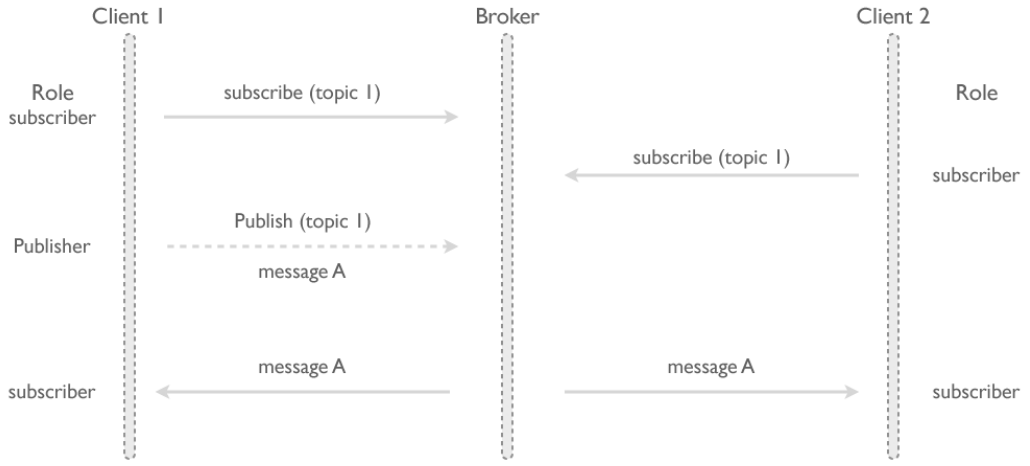

Figure 5.2: An Example of Publish/Subscribe Pattern

The concept of Pub/Sub pattern has been used in real world cases. Twit-

ter, for example, can be treated as a real world scenario, even though the implementation might be different in Twitter. When a new user, User 1, posts a message in Twitter, usually, it will not popup on any other users' home page, but it will popup on User 1's homepage or a similar page that represents the user. This can be understood that User 1 subscribes himself, but no one subscribes him.

User 1 can follow other users based on his interest. For example, User 1 follows User 2. User 1 will, then, receives User 2's post on User 1's homepage. Here, User 2 becomes a topic that User 1 subscribes; User 1 is the subscriber and User 2 is the publisher. Twitter, acts as a broker, handles the logic how the messages sent by User 2 delivers to User 1.

From the perspective of IoT, Pub/Sub pattern covers the other one of the two most important interaction in the IoT application architecture, i.e. notification. With notification, data can be sent interconnected, such as among nodes or from nodes to data centres. For example, one node, Node A, collects temperature data and humidity data. Node A then submits the collected data to the other node or data centre (called Node B) for the purpose of assembling. One strategy, which organises and manages the data sent by Node A, is to set the topic of the each message to the data type (i.e. temperature or humidity). Subsequently, Node B can easily categorise the data for further processing. There are, surely, other ways of exploiting Pub/Sub pattern.

5.3 Database-centric Approach

Database-centric is a software architecture where database plays critical role. Generally, database provides fault tolerant and reliable transactions. Database-centric Approach, in IoT services, subsequently, provides a dependable mechanism to record enormous IoT data.

An example of implementing a database-centric IoT service has been discussed in [16]. The approach merges the features of database-centric approach with the demands of IoT services, and provides customised features for IoT services. Thus, the approach, firstly, supports heterogeneous and multimedia contents; Secondly, it provides standard APIs for standard web services, and thus the IoT data can be utilised at optimally maximum; Finally, live notification is supported and security features are applied.

Chapter 6

Design and Implementation of Sample IoT Services

6.1 Smart Lighting and Control Application

6.1.1 Components

In the thesis, we built an HTML5 based application to demonstrate a smart lighting and controlling IoT service. The concept of the sample service consists the following elements: a light, a switch, a sensor and the sun. The light is a component that visualises the light status. The switch is a component that represents an actuator that can control the light. The sensor is a component that senses the environment brightness. The sun represents the environment. The change of the luminance from the sun affects on the sensor which then affects on the light. The luminance of the light will be, subsequently, adjusted. The service allows the light to keep the brightness of a space constantly, or to be adjusted by the switch: on, off or to dim. The relationship among the components is shown in Figure 6.1.

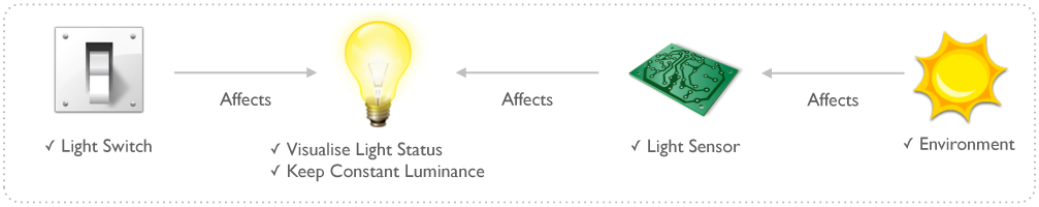

Figure 6.1: Components Relationship

6.1.2 Implementation of The Backend

With the concept above, we implemented an embedded web server based on a NodeJS server. We also attached a WebSocket server to the NodeJS server, and a WAMP server on top of the WebSocket server. The embedded web server, then, understands the WebSocket protocol and the WAMP subprotocol. The component layer structure is shown in Figure 6.2.

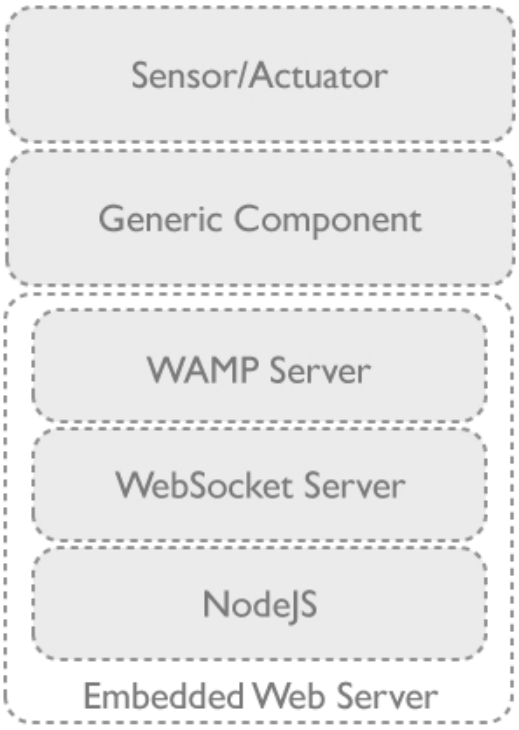

Figure 6.2: Component Layer Structure

We added a generic component layer on the embedded web server. The generic component is an abstract layer that describes information and generic configuration of a device, e.g., component type (temperature sensor, humidity sensor or light actuator etc), device ID, returnable, timeout, data sent rate and some other hardware parameters.

The generic component also implements a function that control the component events. For example, if the sensor should send the data in some fixed interval or the data should be sent only by signalled. In the thesis, the generic component covers sensor and actuator. Thus, we added another component layer onto the generic component. All the layers interconnected will, subsequently, play the role of sensor or actuator.

We start with the sensor. As a sensor is built on top of a generic component which handles parameters configuration and event abstraction, a sensor is capable of responding to configuration and events. As for a light sensor, the events are the followings:

- Get current sensor data. This function triggers a sensor to retrieve data from its environment. In the sample, the sensor senses light brightness data.
- Configuration. To configure a sensor with parameter. Sensor's action, however, is usually hardcoded and cannot be configured remotely. This is due to a security reason, that the currently implementation does not allow an action parameter (such as a function) transmitted remotely.
- Reset. To reset the configuration of a sensor to default.

As for actuator, it is built at the same layer as that for sensor, and thus, it also inheres features from generic sensors as the sensor does. In the sample, we configure an actuator into a light switch, and it can be described as the followings:

- Switch on. To switch on a light and regardless the light current state.
- Switch off. To switch off a light and regardless the light current state.
- Switch a light. To switch the light to its opposite state. This means, if a light is on, it, then, will be switched off and vice versa.
- Adjust luminance. To adjust a light luminance to a given value which should be in the range of a light acceptable value.
- Configuration. Similar to that of a sensor.
- Reset. Similar to that of a sensor.

In effect with simulation, we added an virtual component that simulates a light. Thus, a light status (on or off) can be check through this component.

Summary so far, the interface of a light sensor/switch (sensor/actuator) can be illustrated in Figure 6.3. in a generic way:

To view the architecture, at a higher level, which we have exposed so far, the light sensor and the light switch are connected to a component with an embedded web server, or separately. This architecture is shown in Figure 6.4.

The light sensor and the light switch are linked to a component respectively. The two components differ themselves by the topmost layer which we showed in Figure 6.2. With the embedded web server, the light sensor and

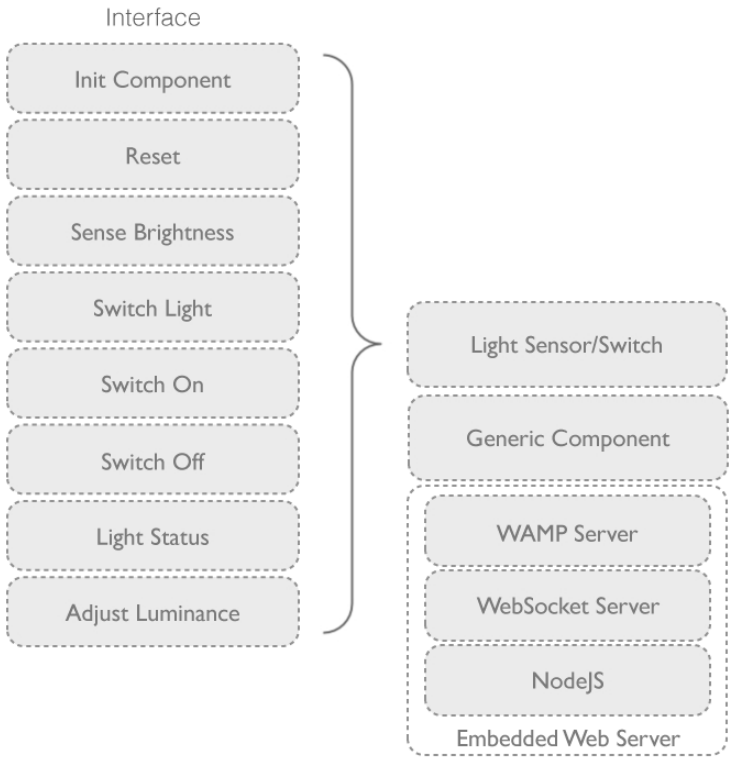

Figure 6.3: Component Interface

the light switch can communicate with each other and other components (such as the simulated component, light) over WebSockets. We can think this way, the component, which the light switch connected to, enhance the functionality that the light switch can do. For example, it controls the light via the interface switch on/off.

6.1.3 Implementation of The Front-end

However, the current system should be accessible from outside, such as by end users. We implemented this via a web front-end interface.

The structure is layered at front. Concerning the connection is over WebSocket, a WebSocket layer is arranged at the bottom. The WebSocket layer handles the connection with the embedded web server.

The communication coming from the embedded web server, is however, using WAMP protocol. Thus, we added another layer, WAMP layer, on top of the WebSocket layer. With these two layers, the front-end and the backend

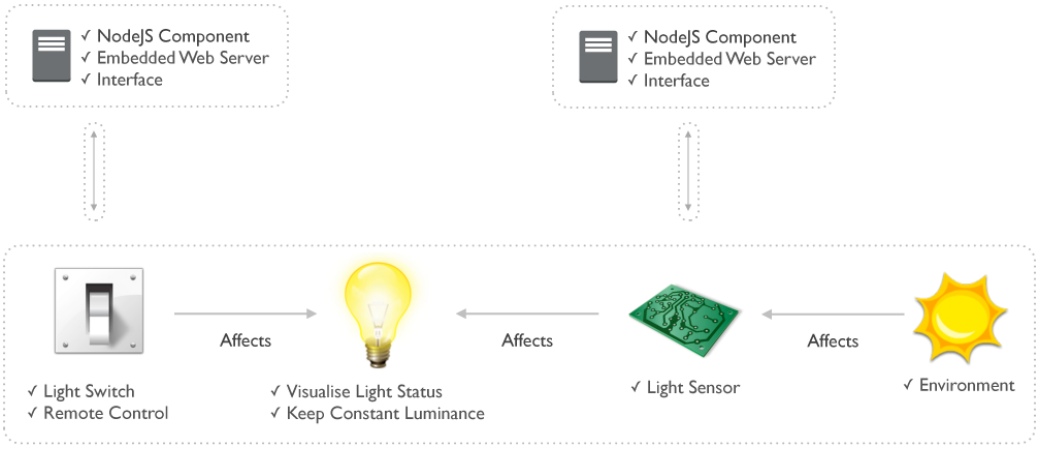

Figure 6.4: Components Connected to Embedded Web Server

can, now, communicate.

We also would like to control the backend, and thus we added another layer, APIs layer, on top of the WAMP layer. The APIs layer contains light sensor APIs and light switch APIs. As the name implies, programmers can control the backend through the APIs that this layer provides. For example, with the APIs, we can make a web interface that visualises the light switch which can response to users click event and subsequently control the light on and off; we can also visualise the light status on a web interface, so we can check the light status remotely.

This architecture is shown in Figure 6.5.

Figure 6.5: Front-end Layer Structure

6.1.4 Overall Structure

We have covered end users, front-end structure and backend structure. Now, we can link them together. This architecture is shown in Figure 6.6.

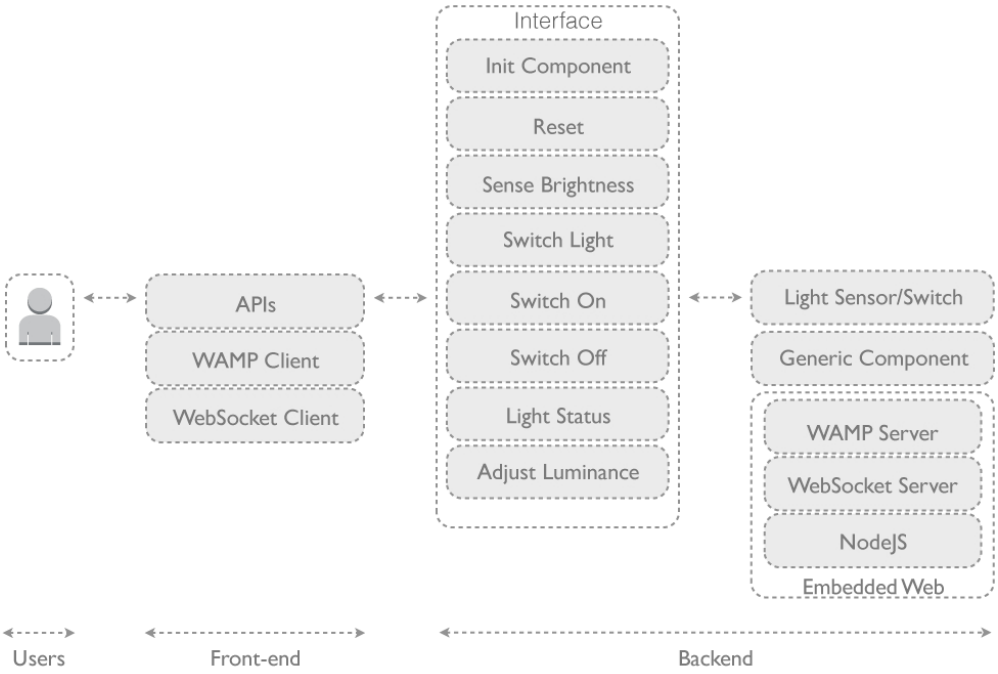

Figure 6.6: Overall Structure

The APIs layer connects users and the interface provided by the backend. WAMP Client talks with WAMP server through WebSocket Client which talks with WebSocket Server.

6.1.5 The Role of Messaging Patterns

Previously, we have discussed Pub/Sub and RFC. These two pattens, handled by WAMP layer, cover the two most important interactions in Internet of Things: notification and command. Here, we discuss how are they integrated in the sample. This architecture with message patterns is shown in Figure 6.7.

As the front-end is built with HTML5 technology, the sample is capable with any browser that supports HTML 5 feature (WebSocket). Thus, the architecture is designed in a way that end users can control the light switch and view the light status via a browser.

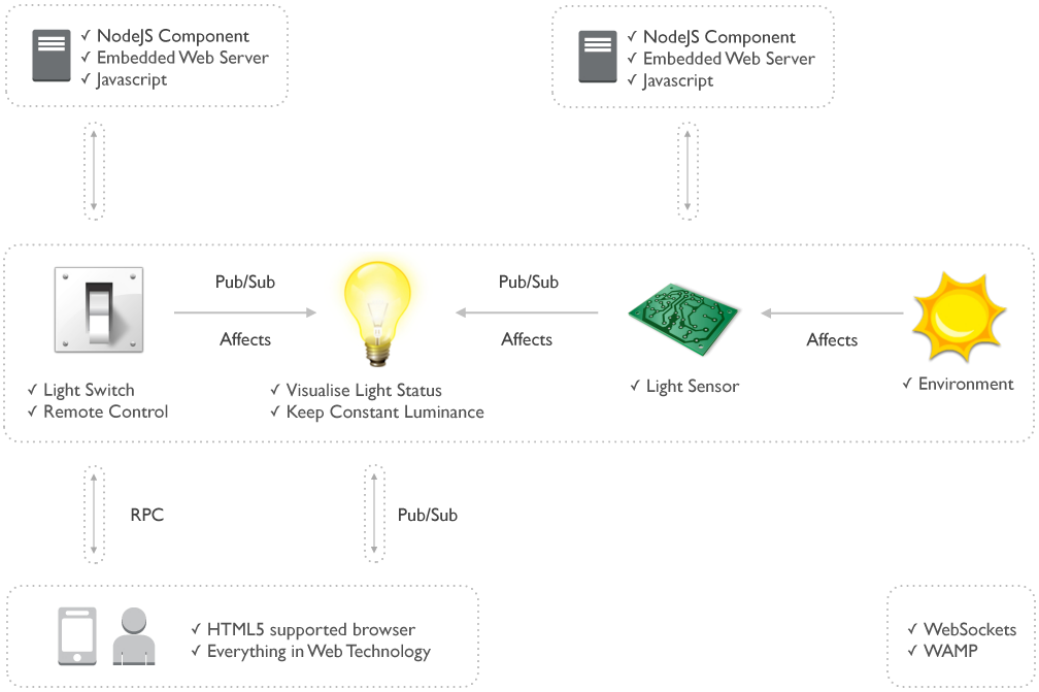

Figure 6.7: Overall Structure with Massage Patterns

When a end user control the light switch, such as turning on the light; turning off the light; or adjusting the luminance of the light, it sends the command using RPC patterns. RPC pattern is also applied in component initialisation and reset. When the user view the visualisation of the light status, it communicates with the components via Pub/Sub patterns.

The interaction between the light switch and the light is using Pub/Sub pattern. The light component subscribes a topic from the light switch. When the user e.g. turns on the light, the switch component publishes a message with the light on value (such as, true) to the light component. The light component, then, receives the message, processes the message and turn on the light. Luminance adjustment is similar.

The interaction between the light sensor and the light is, also, Pub/Sub pattern. The light component subscribes a topic from the light sensor, as opposed to that from the light switch. When the light sensor updates its collected data, the light receives the data and subsequently adjust the light luminance according to its setting.

6.2 Social Network IoT Sensing Application

In contrast to the WebSocket based sample: Smart Lighting and Control Application, we provided another sample IoT service, Social Network IoT Sensing Application, based on data centralised model using HTTP for communication.

6.2.1 Features

The sample application provides a way to visualise data collected by IoT devices remotely in nearly real-time. For example, a user deploys IoT devices at home. The devices, then, collect temperature, humidity and multimedia (image, video, and audio) data at home. The purpose of the design enables the user to inspect those data in a visualised way from a browser. Visualisation means, for example, a graph chart of the temperature of the user's home within the past 24 hours. Real time means, for example, the web interface updates corresponding to live notification, once the collected data have been processed.

Furthermore, the sample application is extensible by integrating third party APIs. For example, the sample could visualised IoT devices geolocations and geographic structures, with Google Map APIs; Notification could be extended into the area of social network, other than the owner exclusively.

The underlying infrastructure of application handles scaling such as distributing and clustering. This feature satisfies the same demand in IoT services.

6.2.2 Stack and Architecture

Social Network IoT Sensing Application is a CouchApp[1], a Javascript and HTML5 application served directly from CouchDB. CouchDB is a documented-oriented database. Data are stored in JSON documents and are accessed with a web browser via HTTP. CouchDB is, also, designed for data distribution and replication. In short, CouchDB provides scalability and flexibility. These features are meaningful to the challenges[16] - heterogeneity and scale - in IoT services. Its layer stack is shown in Figure 6.8.

CouchDB acts as an HTTP server, serving HTML directly to a browser, while manages heterogeneous data and horizontal scaling; CouchDB, moreover, provides live notifications and handles over the notifications to upper

[1]http://couchapp.org/page/index

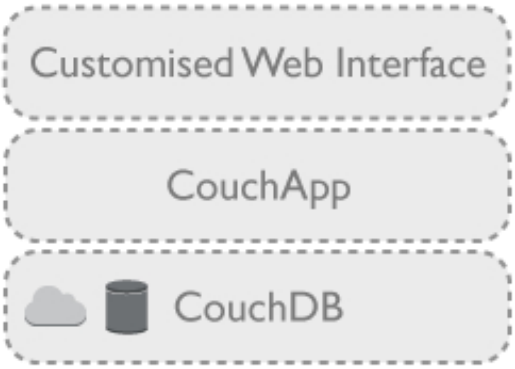

Figure 6.8: Social Network IoT Sensing Application Layer Stack

layers. External data, such as collected by IoT devices, connect to CouchDB via a gateway. The collection of the data and the purpose of the gateway have been discussed in [16].

CouchApp inherited these features from CouchDB. Furthermore, CouchApp is extensible. Thus, we have a third layer to extend the functionality and customise the usability. For instance, we used BackboneJS, Model-View (MV) Javascript library, to structure our application. Third party APIs, additionally, can be integrated into the third layer and thus enable, e.g. social network or geography functions. End users interact with the system via the third layer.

The architecture of the sample application is shown in Figure 6.9.

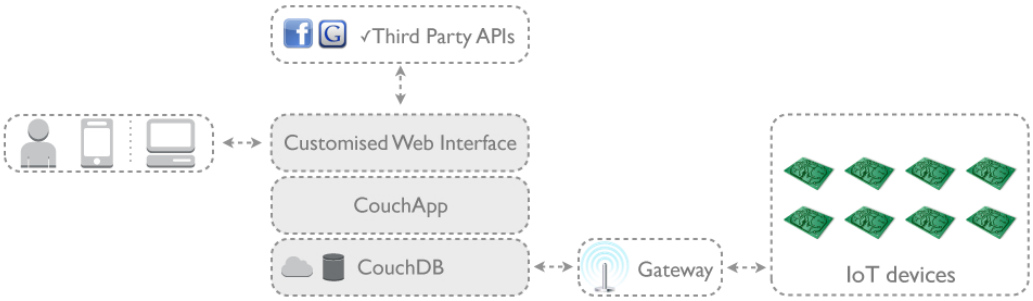

Figure 6.9: Social Network IoT Sensing Application Architecture

6.2.3 Retrieving Data

The architecture, which end users directly interacting with, is based on a HTTP web server, CouchDB. Thus, all connections are HTTP based. In order to keep notification alive, the browser has to maintain HTTP connections, negotiate with the web server, update HTTP connections and keep HTTP connection alive.

Chapter 7

Evaluation

7.1 Origin and Justification

We planed to build an IoT service with web technology that differed from the pervious solution, database-centric approach, which largely relies on Ajax for data exchange.

The prevision of the thesis was constructed in two parts: web technology and IoT; Its solution should integrate the two parts into a coherent service. Following the consideration of HTML5 connectivity features, we focused on the WebSocket protocol which fills the part of web technology. As for IoT, which covers a wide area, we selected one scenario, smart lighting control, with sensor and actuator from many IoT visions such as NFC applications, smart housing and phones connectivity.

Compared with other HTML5 communication technology, the WebSocket protocol is more mature, having ready made auxiliary libraries with a feasible amount of selection. The WebSocket protocol, additionally, supports subprotocols where WAMP fills the blank and provides the messaging patterns which furthermore covers the two most important interaction in IoT service.

The Sensor and actuator are two of the most common "things" in IoT architecture. The sensor collects data, while the actuator emits commands. We evaluated these two components matches the web technology, WebSockets with WAMP. We, further, collected several IoT scenario assumptions - high lever descriptions of what we thought might work. For example, we drafted using sensors in a phone, collected data and reported to data centre; Applying a searching function in real world; or smart building. We evaluated the phone case is a bit far away from web technology and the implementation might involve more mobile technology rather than web technology. As for

the searching function, it might involve more hardware configurations and physical experiments.

Among of all, thus, we trimmed down the assumptions and focused on the smart building scenario. We proposed several hypotheses - more granular descriptions of the assumption - which target on the following specific areas: sensor deployment, data collection, data transmission, endpoint visualisation and others, such as business model level cases. Moreover, we continuously decreased the amount of areas to work on. Data transmission is related to the thesis scope where it matches Pub/Sub and RPC patterns; While endpoints visualisation provides a human friendly interface to review data, and it, besides, performs simulation, such as, simulating the environment.

Finally, we started to experiment on the smart lighting and control application, an IoT services about sensors and actuators with WebSockets.

7.2 Tools and Setup

Before we started the thesis project, we assumed the front-end environment would largely rely on web browsers. As a result, we expected to build it with JavaScript. For this reason, we found AutobahnJS[1], an open source JavaScript networking library.

AutobahnJS supports WAMP at a higher level, while supports the WebSocket protocol at lower level. Thus, it is feasible to create a real-time enabled HTML5 applications with the library. Moreover, AutobahnJS has - another advantage of AutobahnJS - clear tutorials and documents.

As for the backend deployment, it relatively has more options where a list of existing library has been being appended to support WAMP. From the perspective of programming languages, there are implementation on python, php, JavaScript, Java and others. Our implementation is based on WAMP.IO, a NodeJS library supporting WAMP. WAMP.IO does not handle the WebSocket protocol, but its dependency does.

At the beginning, there were some technical problems in WAMP.IO, leading an initial connection failure with a WAMP client. Thus, we moved to Ratchet, a backend implementation using php, until the developer fixed those problems. Compared WAMP.IO with Ratchet, we found out Ratchet is more verbose, from the perspective of development. There are some features we did not need. WAMP.IO, however, targets the desired features - Pub/Sub and RPC - straightforward. With its simple architecture, we could easily customise the library to match demands more closely. Moreover, its devel-

[1] http://autobahn.ws/js

oper, behind the library, actively identified and fixed bugs, making the whole development reliable.

7.3 Simulation

We mentioned about the light, the switch/dimmer, the sensor and the sun in the thesis. Considering the thesis scope, we decided to focus on the software level, rather than the hardware part. Thus, we, at the current stage, simulated the light, the switch, the sensor and the sun.

For the simulation, we ported the most relevant features - directly related to the implementation - to the software level. Meanwhile, we analysed its possibility bringing those features and the results back to physical objects.

For example, a light bulb emits light, where the light is the key link connecting the virtuality to the reality. We mapped light luminance into digital numbers at the software level. The luminance can thus be controlled by adjusting digital numbers. The light status - on and off - similarly, can be simulated by e.g. boolean value.

As for the switch/dimmer, a switch turns on or off the light, while a dimmer adjusts the luminance of the light at a certain range. The action of turning/adjusting is the key part that we simulated. The simulation of turning/adjusting, is, being the case, connecting the light component and the switch/dimmer component at the software level, and making the light response to the action of the switch/dimmer. This is, actually, how a switch/dimmer does in the real world.

A sensor, generally, converts physical quantity into digital signal which can be further read or analysed by a observer or an instrument. Thus, a sensor is, essentially, a programme that provides data at the software level. Consequently, we simulated the sensor through a programme piping the output of the sun to the light.

As for the sun, we also ported it to the software level. To do so, we made a programme that generated a sequence of numbers periodically increasing to a peak then recurring.

7.4 Measurement

We measured about overheads and Roundtrip Delayed Time (RTT) comparing WAMP with MQTT. Figure 7.1 illustrates the message overheads on WAMP and MQTT.

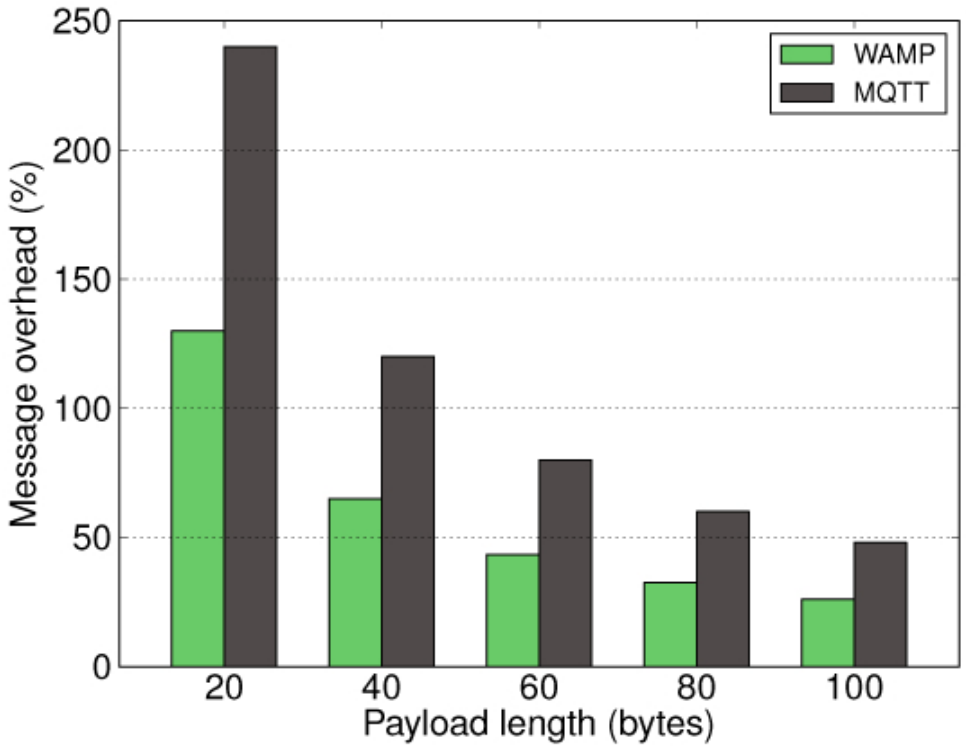

Figure 7.1: Message Overhead

WAMP shows its lightweight design better than that of MQTT. For each 20 bytes of data sent, WAMP will have 1.25 times overheads larger than the message itself, while MQTT will have around 2.4 times overheads generated.

Figure 7.2 illustrates the message RTT on WAMP and MQTT.

We measured the RTT from a local workstation to an Amazon micro instance in Ireland. And the result we got is, generally, MQTT has a longer RTT than that of WAMP; WAMP RTT starts to increase after sending 10k bytes, while MQTT stays constantly. The sudden increment could be caused by the programme design, the workstation or the Amazon instance we used. As the package size increased, the processing and queueing time might consequently enlarge.

7.5 Scalability

In Pub/Sub and RPC Patterns for Web-based IoT applications, the broker(s) plays a key role managing the interconnection of clients. Each client differs

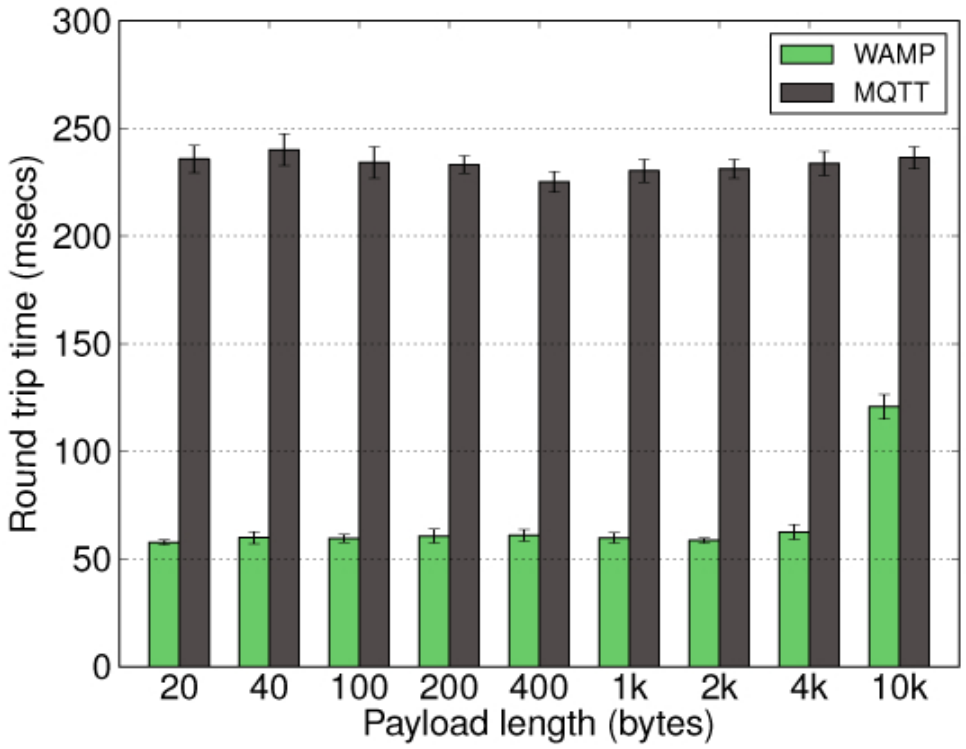

Figure 7.2: Message RTT

from others by its ID. Thus, the nature of scaling in these patterns is the implementation of addressing clients. Further communications and interactions, among users and backends, rely on the successful navigation of the broker(s).

In the scenario of the smart lighting and controlling application, the scaling issue generated by the enlarging amount of lights, switches and light sensors. Considering the limited resources that an embedded web server are assigned, the architecture might not suit for a large scale without other resources support such as extra brokers with power sources.

As for the database-centric pattern, the CouchDB solution naturally provides a reliable and flexible way to scale horizontally and to replicate, with the growing amount of data. The database acts as a hub between IoT devices and end users, while it can be further developed into cloud-based architecture which maximises its scalability.

7.6 Security

The WebSocket protocol supports connections over Transport Layer Security (TLS). TSL provides communications security over the Internet [12]. Furthermore, the WebSocket protocol providers applicable security considerations as the followings[33]:

- Non-browser Client resistant. The WebSocket protocol prevents malicious running inside a web browser.
- Origin Considerations. The WebSocket protocol provides a way for servers that do not intend to process input from web pages from other origins.
- Masking. All data from clients to servers are masked, so that the attacker does not have knowledge about how the data being sent.

The security considerations on database-centric pattern implemented by CouchDB has been discussed in [16]. As for database, one of the most important security issue is authentication and access control. It is notable that CouchDB, by default, does not turn on access control for different users. This means, everybody has the same permission as the administrator. CouchDB authenticates users in many ways, such as HTTP basic authentication, cookie authentication and OAuth [19]. CouchDB has started to support Secure Sockets Layer (SSL) since CouchDB version 1.1, and thus provides communication security over the Internet.

Chapter 8

Future and Conclusion

At the current stage, the implementation is done at software level: hardware part are simulated. Therefore, the development process can be trimmed to fit in the scope of the thesis. However, we, carefully, preserved the essential features of each hardware component and simulated at the software level. This means, there should not be many different behaviours for further deployment on hardware components.

One example, which could link all the features back to hardware, is Raspberry Pi[1]. Raspberry Pi supports Python as its official programming language by default, but any language which will compile for ARMv6 can be used. This implies that NodeJS can be ported on Raspberry Pi. Moreover, the embedded web sever and its upper layers, which we implemented, are not restricted by NodeJS, meaning the architecture can be built with other programming languages, such as Python. A light, a switch and a sensor can, subsequently, be connected to Raspberry Pi. We can, furthermore, use a phone's sensors as a data source, and transmit the data from the phone to Raspberry Pi.

The thesis is about applying web technology on Internet of Things. We focused on the WebSocket protocol as the part of web technology. The WebSocket creates a bi-directional connection between a server and a web browser. We highlighted its advantage by comparing it with the traditional method, HTTP, according to the behaviours of fetching resources from a web server. Furthermore, we chose WAMP as a subprotocol to support WebSocket complementarily, enabling WebSocket unprecedentedly functionality to interact with Internet of Things services.

Moving onto the part of Internet of Things, We integrated a sensor (light sensor) and an actuator (light switch/dimmer) into the application, blending

[1] http://www.raspberrypi.org/

with the web technology. The major feature of the sensor (sending data) and that of the actuator (commanding and configuration) perfectly match the massage patterns, Pub/Sub and RPC, which WAMP provides. Consequently, the smart lighting and controlling Internet of Things service emerged.

Additionally, we compared the smart lighting and controlling application with social network IoT sensing application, an application interacting with IoT services with traditional method (HTTP), from the perspective of architectures, implementation, scalability and security.

Moreover, We compared WAMP with other registered WebSocket subprotocols: MBWS, SOAP and STOMP from the viewpoint of features; and with other potential protocols: CoAP and MQTT, from the viewpoint of practicability. We further evaluated WAMP and MQTT with communication overheads and RTT.

There are, surely, many potential room for the future research on the blending of web technology with Internet of Things. The upsurge in bleeding-edge web technology might be supportive to the future Internet of Things services research, e.g., for the efficiency, maintainability and scalability.

Bibliography

[1] acobs, Ian; Walsh, N. Architecture of the world wide web, volume one, 2004.

[2] Ashton, K. That 'internet of things' thing. *RFiD Journal 22* (2009), 97–114.

[3] Atzori, L., Iera, A., and Morabito, G. The internet of things: A survey. *Computer Networks 54*, 15 (2010), 2787–2805.

[4] Booth, D., and Liu, C. K. Web services description language (wsdl) version 2.0 part 0: Primer. *W3C Recommendation 26* (2007).

[5] Christin, D., Reinhardt, A., Mogre, P. S., and Steinmetz, R. Wireless sensor networks and the internet of things: Selected challenges. *Proceedings of the 8th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze* (2009), 31–34.

[6] Chui, M., Löffler, M., and Roberts, R. The internet of things. *McKinsey Quarterly 2* (2010), 1–9.

[7] Colitti, W., Steenhaut, K., and De Caro, N. Integrating wireless sensor networks with the web. *Extending the Internet to Low powerand Lossy Networks (IP+ SN 2011)* (2011).

[8] Comer, D. E. *Computer networks and internets*. Prentice Hall Press, 2008.

[9] Crockford, D. The application/json media type for javascript object notation (json).

[10] Culler, D., Estrin, D., and Srivastava, M. Guest editors' introduction: overview of sensor networks. *Computer* (2004), 41–49.

[11] Deering, S. E. Internet protocol, version 6 (ipv6) specification.

[12] Dierks, T., and Rescorla, E. Rfc 5246: The transport layer security (tls) protocol. *The Internet Engineering Task Force* (2008).

[13] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T. Rfc 2068: Hypertext transfer protocol—http/1.1, january 1997. *Status: PROPOSED STANDARD*.

[14] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol–http/1.1, 1999.

[15] Fielding, R. T., and Taylor, R. N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT) 2*, 2 (2002), 115–150.

[16] Francesco, M. D., Li, N., Raj, M., and Das, S. K. A storage infrastructure for heterogeneous and multimedia data in the internet of things. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on* (2012), IEEE, pp. 26–33.

[17] Garrett, J. J., et al. Ajax: A new approach to web applications, 2005.

[18] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., and Lafon, Y. Soap version 1.2 part 1: Messaging framework. w3c recommendation, 2007.

[19] Hammer-Lahav, E. The oauth 1.0 protocol.

[20] Hapner, M., and Suconic, C. The messagebroker websocket subprotocol.

[21] Hickson, I. Server-sent events. *W3C Working Draft WD-eventsource-20091222, latest version available at¡ http://www. w3. org/TR/eventsource* (2009).

[22] INFSO, D. Networked enterprise & rfid infso g. 2 micro & nanosystems, in co-operation with the working group rfid of the etp eposs, internet of things in 2020, roadmap for the future [r]. *Information Society and Media, Tech. Rep* (2008).

[23] International Business Machines Corporation (IBM), E. Mqtt v3.1 protocol specification, 2010.

[24] Lewis, D. What is web 2.0? *Crossroads 13*, 1 (Sept. 2006), 3–3.

[25] Lubbers, P., and Greco, F. Html5 web sockets: A quantum leap in scalability for the web. *SOA World Magazine* (2010).

[26] Madlmayr, G., Langer, J., Kantner, C., and Scharinger, J. Nfc devices: Security and privacy. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on* (march 2008), pp. 642 –647.

[27] Masinter, L., Berners-Lee, T., and Fielding, R. T. Uniform resource identifier (uri): Generic syntax.

[28] Nielsen, H. F., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H. W., and Lilley, C. Network performance effects of http/1.1, css1, and png. In *ACM SIGCOMM Computer Communication Review* (1997), vol. 27, ACM, pp. 155–166.

[29] O'reilly, T. What is web 2.0, 2005.

[30] Postel, J. Internet protocol.

[31] Postel, J. Rfc 793: Transmission control protocol, september 1981. *Status: Standard* (2003).

[32] Reynolds, J., and Postel, J. Assigned numbers. Tech. rep., STD 2, RFC 1700, October, 1994.

[33] RFC6455, I. The web socket protocol, 2012.

[34] Rodriguez, A. Restful web services: The basics. *Online article in IBM DeveloperWorks Technical Library 36* (2008).

[35] Shelby, Z., Hartke, K., and Bormann, C. Constrained application protocol (coap).

[36] Smith, L., and Lipner, I. Free pool of ipv4 address space depleted. *Online (February 2011), February* (2011).

[37] Snell, J., Tidwell, D., and Kulchenko, P. *Programming Web services with SOAP*. O'Reilly Media, 2009.

[38] Spero, S. Analysis of http performance problems, 1994.

[39] Srinivasan, R. Rpc: Remote procedure call protocol specification version 2.

[40] Stankovic, J. A. When sensor and actuator networks cover the world. *ETRI journal 30*, 5 (2008), 627–633.

[41] Sterling, B. Shaping things.

[42] Vasseur, J.-P., and Dunkels, A. *Interconnecting smart objects with ip: The next internet*. Morgan Kaufmann, 2010.

[43] Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I. S., Mazura, M., Harrison, M., Eisenhauer, M., et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends* (2011), 9–52.

[44] Wang, V., Salim, F., and Moskovits, P. *The Definitive Guide to HTML5 WebSocket*. Apress, 2012.

[45] Wu, M., Lu, T.-J., Ling, F.-Y., Sun, J., and Du, H.-Y. Research on the architecture of internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on* (2010), vol. 5, IEEE, pp. V5–484.