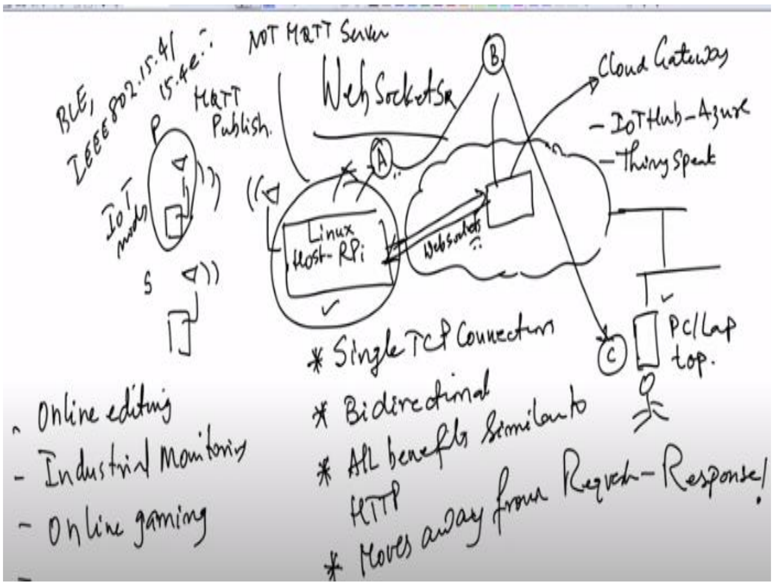
Design for Internet of Things
Prof. T V Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science-Bengaluru

Lecture - 38
WebSockets

Folks, let us look at another very exciting protocol which is called WebSockets, fine? And why should we study this WebSocket may be something that you may want to ask yourself. I will show you a demonstration. As usual Vasanth and Abishek have set up a demonstration. But you should understand the philosophy for WebSockets, okay.

(Refer Slide Time: 00:53)

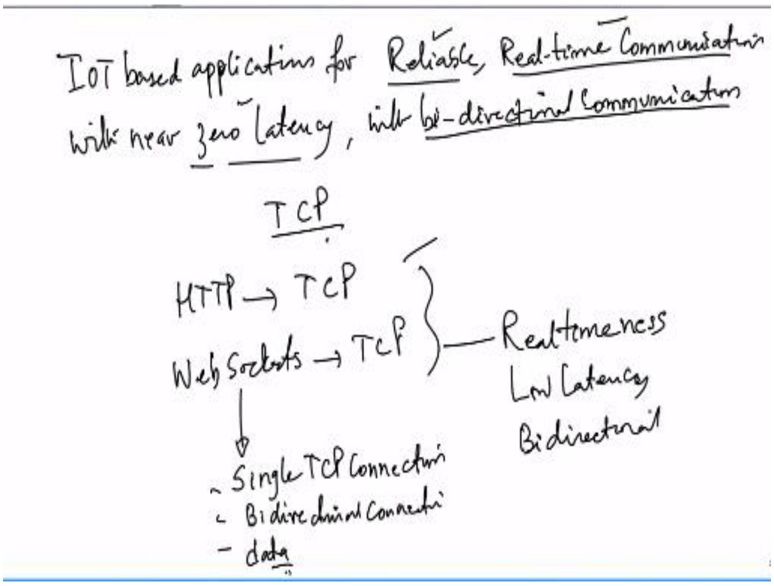


Let me put, I put down this very small picture for you which is talking of MQTT publish, right? These are small IoT nodes. These are very small IoT nodes and they have the capability, they are perhaps running a lightweighted Paho client, okay. And they are able to publish their data. This may be publishing, this can be publishing, this can be subscribing.

And there can be many publishing nodes and there can be several, you know subscribing nodes. All that is fine within this network, okay. But think of a situation where there is a human who is sitting on the other side of the cloud, other side of the internet. He is in front of some PC or a laptop, and he is there physically. That means he is sitting here physically, okay.

And he wants every time this node publishes, every time this node publishes, he wants to see this on the screen here in QuickTime. He has to see it and it should be in a low latency situation, extremely low latency, okay.

(Refer Slide Time: 02:02)



So it should be reliable. It should have real time communication with near zero latency and it should allow you to do bidirectional communication. These requirements have to be met, okay. Now how to do it? These are very small nodes, which perhaps do not even have the ability to transmit data directly to the cloud. They may have a very small radio like Bluetooth low energy or they may have a small radio like IEEE 802.15.4 and 15.4e, okay.

These kind of radios are what these nodes have; low power, small range and they are able to communicate. Let us say such nodes while they can communicate successfully, within themselves in a distance of let us say 50 meters, 75 meters and so on. They obviously do not have the capability to directly give it to the cloud. So you obviously need some sort of a gateway device.

Let us say the gateway device of interest is something like this Linux, Raspberry Pi, okay, either running Linux or running Raspbian or one of them. Now this system has the ability to aggregate all the published data and also pass on all the subscribed data, okay. Assume that you are still looking at, this is not offering the broker. This is not really a broker, not a broker. That is the point.

In other words, it is not the MQTT server. It is not the MQTT server. The MQTT server is still in the cloud. This Raspberry Pi is just aggregating data, MQTT data from several publishers and so on and uploading this MQTT data directly. Now as I said, you want to meet this requirement. What is that requirement we have in mind? It should meet, it should be reliable, it should be real time, near zero latency should be there.

And the way to achieve that is this host okay, this host establishes a connection to the cloud system, I will call that B, A to B uses WebSockets. This is a protocol that they may use. Why, is the question? If you use WebSocket between A and B and from B to this laptop, I will call this C. B to C is another connection. A to B is one, B to C is one more, okay?

If you do like this okay this also can be WebSocket connection, okay. You cannot do A to C that is not possible. Because A has no clue who is asking for data. Please note A to C if it communicates that means you have broken the idea of MQTT, which is not allowed. What is the idea of MQTT? Publishers and subscribers are not known to each other. There is no end-to-end knowledge.

Publishers do not know who are all the subscribers. Subscribers do not know anything about the publishers. Go back to the ThingSpeak example. You pick data from San Diego some website, right? How do they know that you are sucking the data from them? They do not know.

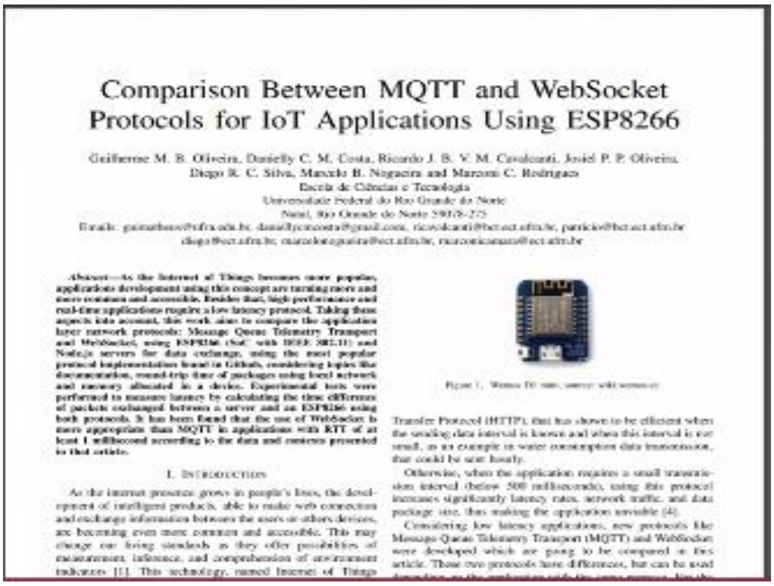
Similarly, ThingSpeak also gave you other publicly available things you could download by an Excel file and you could actually you know build your own application. That is one way of building the application. What about live data that is coming into ThingSpeak? You want to get the data live into you. One way is you set up your own channel, you upload the data and do it.

The other is you subscribe to certain channels and expect them to give you data every time you are published. That is another way, right? We will not get into that detail. So therefore A to C is not going to work. A has to go to B. B has to give it to C. And this should happen in extremely low latency condition, okay. Zero latency, it should be real time and it should ensure reliability.

The one mantra for all this is TCP. TCP provides you reliable communication and it does not have real timeness though, right? Because there is a certain amount of overhead associated with if you look at applications which use TCP, applications like HTTP and so on, they are not really amenable for real timeness. Also uses TCP. This can give you real timeness.

It can give you low latency, okay. It can give you bidirectionality. None of these things are possible with HTTP, although both of them use TCP. The question is why? Firstly, this is using a single connection, single TCP connection unlike HTTP and it provides bidirectional connection and it uses extremely small amount of, small amounts of data. And I will point you to one paper which I was reading which will help you to understand that. So let us go and look up that paper.

(Refer Slide Time: 08:13)



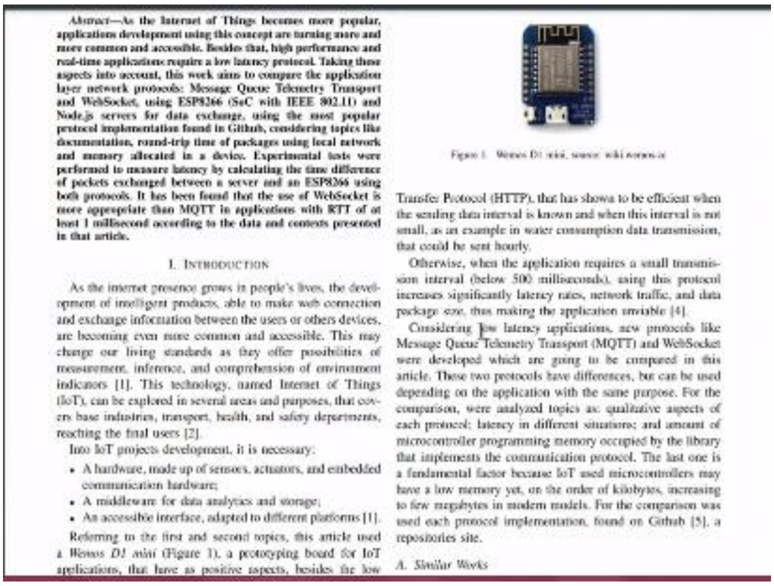
Comparison Between MQTT and WebSocket Protocols for IoT Applications Using ESP8266

Guilherme M. B. Oliveira, Danielly C. M. Costa, Ricardo J. B. V. M. Cavalcanti, Josiel P. P. Oliveira, Diego R. C. Silva, Marcelo B. Nogueira and Marconi C. Rodrigues
Escola de Ciências e Tecnologia
Universidade Federal do Rio Grande do Norte
Natal, Rio Grande do Norte 59078-275

In this paper, this is a paper which I tried to download. And it will tell you about comparison between MQTT protocol and WebSocket protocols for IoT applications using a very popular chip called ESP8266. If you did not know about 8266, you better know it. It is one of the most popular platform node which you can buy. And you can experiment heavily with ESP8266, okay.

That is not the point. I am also not going to read this paper in great detail, okay. But I am going to tell you what exactly WebSocket is in comparison with MQTT, okay.

(Refer Slide Time: 08:50)



Abstract—As the Internet of Things becomes more popular, applications development using this concept are turning more and more common and accessible. Besides that, high performance and real-time applications require a low latency protocol. Taking these aspects into account, this work aims to compare the application layer network protocols: Message Queue Telemetry Transport and WebSocket, using ESP8266 (SoC with IEEE 802.11) and Node.js servers for data exchange, using the most popular protocol implementation found in Github, considering topics like documentation, round-trip time of packages using local network and memory allocated in a device. Experimental tests were performed to measure latency by calculating the time difference of packets exchanged between a server and an ESP8266 using both protocols. It has been found that the use of WebSocket is more appropriate than MQTT in applications with RTT of at least 1 millisecond according to the data and contexts presented in that article.

I. Introduction

As the internet presence grows in people's lives, the development of intelligent products, able to make web connection and exchange information between the users or others devices, are becoming even more common and accessible. This may change our living standards as they offer possibilities of measurement, inference, and comprehension of environment indicators [1]. This technology, named Internet of Things (IoT), can be explored in several areas and purposes, that covers base industries, transport, health, and safety departments, reaching the final users [2].

Into IoT projects development, it is necessary:

- A hardware, made up of sensors, actuators, and embedded communication hardware;
- A middleware for data analytics and storage;
- An accessible interface, adapted to different platforms [1].

Referring to the first and second topics, this article used a *Wemos D1 mini* (Figure 1), a prototyping board for IoT applications, that have as positive aspects, besides the low

Figure 1. Wemos D1 mini, source: wiki.wemos.cc

Transfer Protocol (HTTP), that has shown to be efficient when the sending data interval is known and when this interval is not small, as an example in water consumption data transmission, that could be sent hourly.

Otherwise, when the application requires a small transmission interval (below 500 milliseconds), using this protocol increases significantly latency rates, network traffic, and data package size, thus making the application unviable [4].

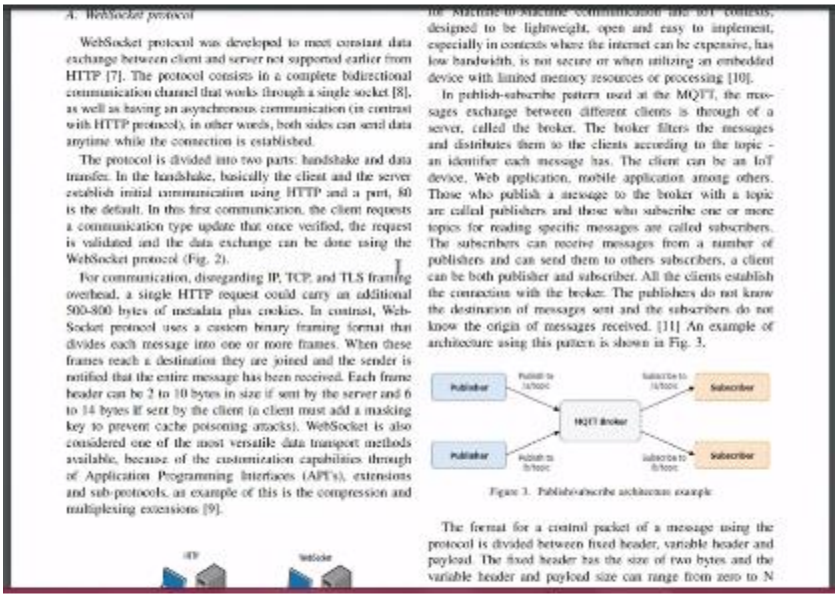
Considering low latency applications, new protocols like Message Queue Telemetry Transport (MQTT) and WebSocket were developed which are going to be compared in this article. These two protocols have differences, but can be used depending on the application with the same purpose. For the comparison, were analyzed topics as: qualitative aspects of each protocol; latency in different situations; and amount of microcontroller programming memory occupied by the library that implements the communication protocol. The last one is a fundamental factor because IoT used microcontrollers may have a low memory yet, on the order of kilobytes, increasing to few megabytes in modern models. For the comparison was used each protocol implementation, found on Github [5], a repositories site.

A. Similar Works

Now if you look at low latency, this paper is saying if you look at low latency applications, MQTT of course offers you certain amount of low latency, and WebSocket also offers you latencies. These protocols have differences. You may think that they are low latency applications, but there are differences between them.

And this paper is actually making comparison and presenting some nice interesting results about the latency. But that was not the point we started off, right? We wanted to understand a little bit about WebSocket itself.

(Refer Slide Time: 09:30)



A. WebSocket protocol

WebSocket protocol was developed to meet constant data exchange between client and server not supported earlier from HTTP [7]. The protocol consists in a complete bidirectional communication channel that works through a single socket [8], as well as having an asynchronous communication (in contrast with HTTP protocol), in other words, both sides can send data anytime while the connection is established.

The protocol is divided into two parts: handshake and data transfer. In the handshake, basically the client and the server establish initial communication using HTTP and a port, 80 is the default. In this first communication, the client requests a communication type update that once verified, the request is validated and the data exchange can be done using the WebSocket protocol (Fig. 2).

For communication, disregarding IP, TCP, and TLS framing overhead, a single HTTP request could carry an additional 500-800 bytes of metadata plus cookies. In contrast, WebSocket protocol uses a custom binary framing format that divides each message into one or more frames. When these frames reach a destination they are joined and the sender is notified that the entire message has been received. Each frame header can be 2 to 10 bytes in size if sent by the server and 6 to 14 bytes if sent by the client (a client must add a masking key to prevent cache poisoning attacks). WebSocket is also considered one of the most versatile data transport methods available, because of the customization capabilities through of Application Programming Interfaces (API's), extensions and sub-protocols, an example of this is the compression and multiplexing extensions [9].

designed to be lightweight, open and easy to implement, especially in contexts where the internet can be expensive, has low bandwidth, is not secure or when utilizing an embedded device with limited memory resources or processing [10].

In publish-subscribe pattern used at the MQTT, the messages exchange between different clients is through of a server, called the broker. The broker filters the messages and distributes them to the clients according to the topic - an identifier each message has. The client can be an IoT device, Web application, mobile application among others. Those who publish a message to the broker with a topic are called publishers and those who subscribe one or more topics for reading specific messages are called subscribers. The subscribers can receive messages from a number of publishers and can send them to others subscribers, a client can be both publisher and subscriber. All the clients establish the connection with the broker. The publishers do not know the destination of messages sent and the subscribers do not know the origin of messages received. [11] An example of architecture using this pattern is shown in Fig. 3.

Figure 3. Publish-subscribe architecture example

The format for a control packet of a message using the protocol is divided between fixed header, variable header and payload. The fixed header has the size of two bytes and the variable header and payload size can range from zero to N

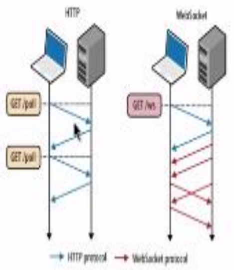
Let me show you what exactly I mean by the WebSockets advantage. Look at this paragraph folks. It is beautifully written, okay. It says as far as communication, disregarding IP, TCP and TLS framing overhead a single HTTP request, could carry an additional 500 to 800 bytes of metadata, plus cookies.

This is a huge amount of additional 500 to 800 bytes of metadata which it will have to carry with every HTTP request, okay. In contrast WebSocket protocol uses a custom binary framing format that divides each message into one or two more frames. It divides it into one or two more frames. When these frames reach a destination, they are joined together, these frames are all put together.

And the sender is notified that the entire message has been received. Each frame header can be 2 to 10 bytes in size if sent by the server, and 6 to 14 bytes if sent by the client, okay. A client must add so many other related things related to cache poisoning and all that he writes that. So WebSocket is also considered one of the most versatile data transport methods available, because of the customization capabilities, through of application programming interfaces, and all that.

So the point is, look at the size reduction in terms of the additional data bytes which are carried in HTTP, as compared to that of WebSockets. No comparison at all, folks. Where is 500 to 800, as compared to each frame carrying only just 2 to 10 bytes, the header, right? So this is what is the key point. Because HTTP is in other words is very heavy and it is request response. And I will show you what I mean by that.

(Refer Slide Time: 11:33)



The format for a control packet of a message using the protocol is divided between fixed header, variable header and payload. The fixed header has the size of two bytes and the variable header and payload size can range from zero to N bytes.

MQTT has three Quality of Services (QoS) levels to message delivery:

- QoS 0 (At most once delivery) Messages are delivered at most once or no, is not stored by either party and there is no acknowledgment of delivery on the network.
- QoS 1 (At least once delivery) Messages must be delivered at least once, the sender stores and tries to send a message until it receives a confirmation, so the receiver can receive and process a message several times.
- QoS 2 (Exactly once delivery) Messages are delivered exactly once, the message is stored in the sender and

Figure 2. HTTP and WebSocket protocols communication flow, source: hpbn.co (modified)

237

Look at this picture. This picture is telling you some fantastic insights into why WebSocket is very important. Look at the arrows. You asked for something, you got something. You asked for something, you got something. You can never get something without asking. In other words, you have to request response, request response. But look at WebSocket, you say GET WebSocket. One arrow goes in one direction.

Now look at the number of arrows coming back. With just one GET, you are getting data continuously in your own real time. You may ask for something else and then you may ask for something else, and so on. So it is essentially telling you that the protocol is very different from that of HTTP. Although it starts with HTTP, as the prime protocol, it switches to WebSockets, after connection is established by what is known as a option, which is called update, okay.

You send an update, and then it switches to WebSockets. That is the key point. Anyway, we will come to that as we go along. Continuing this story here, the cloud services like IoT Hub, or ThingSpeak, all of them are servers, also called brokers. They actually support WebSockets. So you can use WebSockets, directly from this Raspberry Pi System to this cloud based server, you can run it over WebSockets.

That is all I wanted to say. This can run over WebSockets. Therefore, any MQTT message published by this node automatically shows up here in QuickTime under low latency because this Linux, Raspberry Pi is actually running WebSockets to the cloud based system. Maybe it is IoT Hub, maybe it is ThingSpeak or whatever, right? And that guy in turn, has the ability to push the data from B directly to C and essentially provide you data in QuickTime.

So this is what we should be looking at. There are several applications for WebSockets. One of them is online editing of a file. Two people are on either side, they are editing a document, and you want to maintain real timeness, when you are editing the document, then you use WebSockets. Industrial monitoring is another application. Online gaming is another application.

So all these things put together, folks, we will make the protocol very exciting for you to work on, particularly when you are looking for low latency and providing your real timeness and bidirectionality. Let us now focus back on the demo. In the demo, what I wanted to show you is the following.

(Refer Slide Time: 14:33)

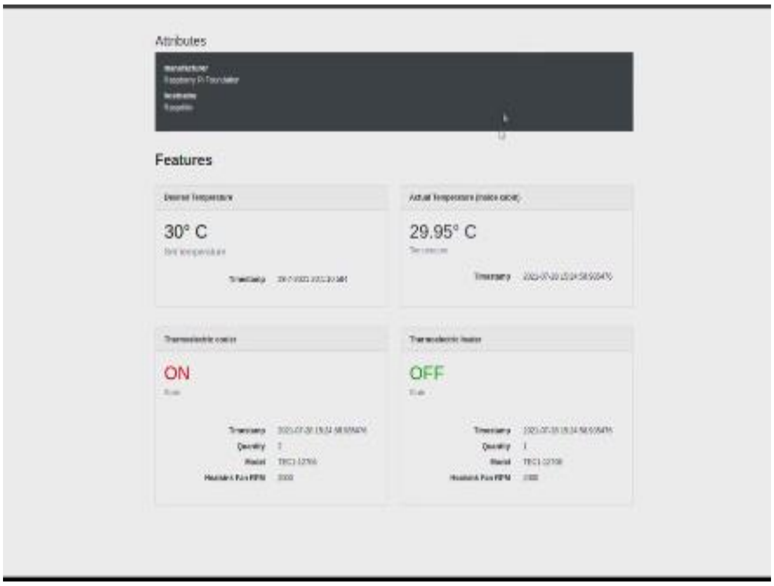
So let me show you the demo here, directly here. This what we have here is a chamber. There are two fans on top. These are what below the fan what you see those is essentially like a heat sink. What you see below is essentially a Peltier coolers. There are two coolers there. There are two fans and therefore there are two coolers.

There is a heater inside the box, semiconductor based heater, which is nothing but a Peltier junction which is placed inside. I cannot open and show you, but do not worry about it the idea of the experiment that I want to show you is we want to maintain certain fixed temperature for inside this chamber, okay. Now every time the chamber heats up because of the heater inside these fans will rotate and make the system cool.

Now every time you want to heat you stop these two fans essentially switch off the Peltier and switch on the heater which is inside and that will heat up the chamber and maintain a certain temperature inside the chamber. Think of this kind of chambers where you need to maintain extremely low temperatures typically for carrying vaccines and so on. This may be a useful portable unit.

What you see here is Raspberry Pi. And what you see here this part with these lights switching on and switching off essentially is a PLC board okay. This is programmable logic controller board, which is essentially ensuring that the chamber of interest is maintaining a certain temperature. Now this board actually is transmitting data to a controller system which is shown here on this screen.

(Refer Slide Time: 16:13)

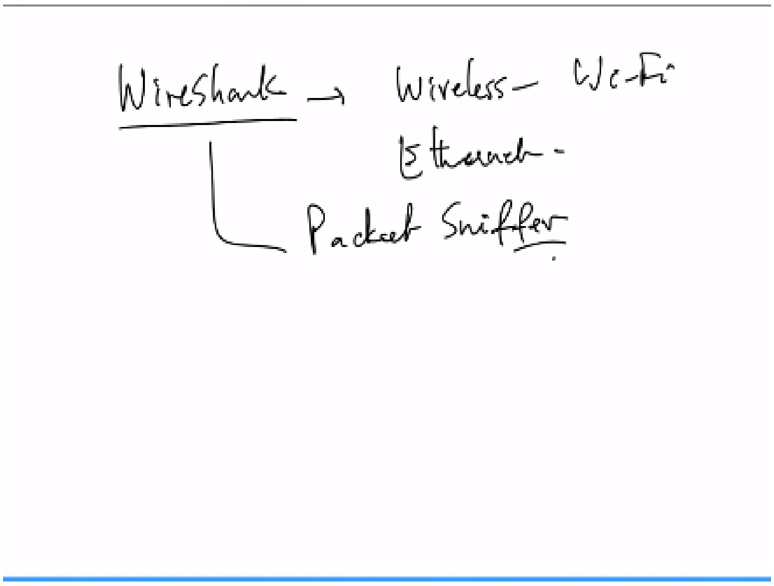


You can see now what you see on the left side is the desired temperature which is 30 degrees it has been set. The actual temperature is a little above that 30.05, it is actually changing dynamically as you can see. And then the thermoelectric cooler is ON why because the temperature has gone a little above and therefore it has to switch ON the cooler. And then the thermoelectric heater is OFF, okay.

So these heaters can go ON, OFF, ON, OFF depending on where you set your temperature. Right now it is at 30 degrees, right. And you want to see this data in real time, is it not. You want to see this data in real time as it is happening. Basically this is an application, what you see is an application. This application actually uses WebSockets. Now how do you know it is using WebSockets?

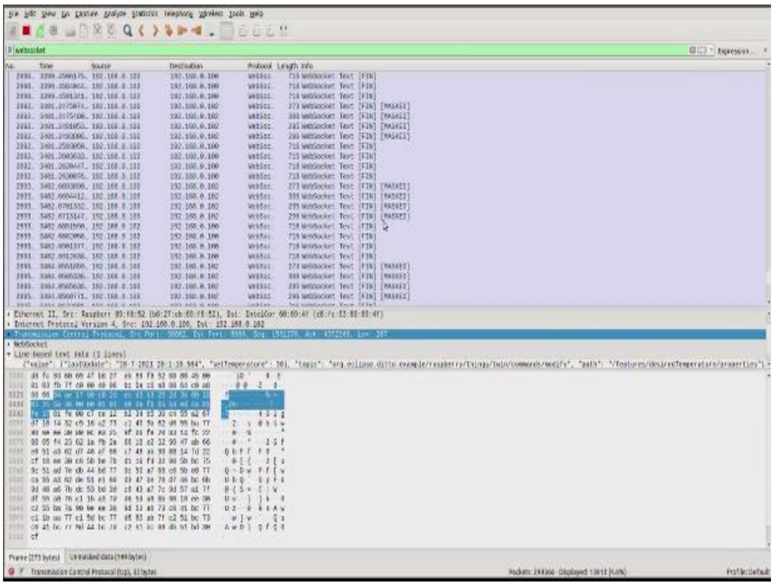
I will explain to you a very important tool which you must use if you are working on networks, IoT and so on and that tool name is called WireShark, okay.

(Refer Slide Time: 17:20)



This tool will examine anything that is going on a wireless link like a Wi-Fi link or Ethernet link. It is a sniffer, it is a packet sniffer, okay. And I will show you that indeed the packet sniffer is able to sniff and capture the WebSocket packets. And how does it look? Now we will go and look up that.

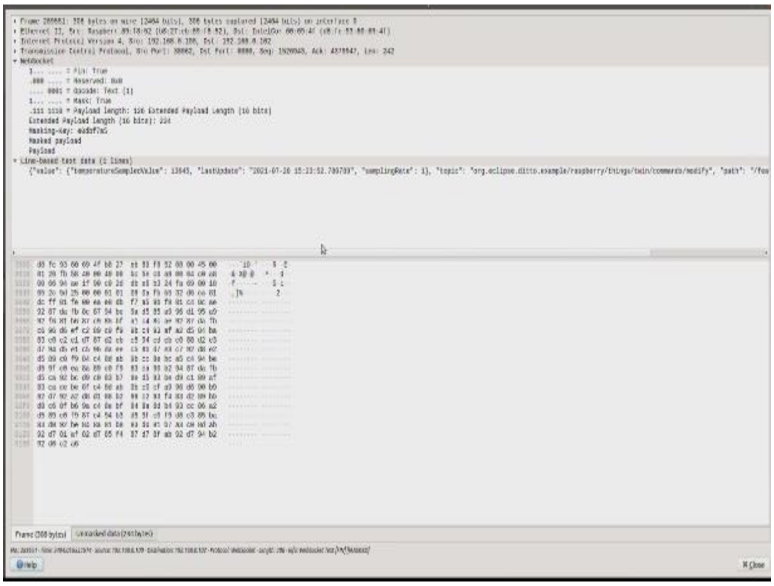
(Refer Slide Time: 17:56)



So this is a WireShark window as you can see right on top is written WireShark. And what Vasanth is actually doing is he has picked the WebSocket based packet. You can see on top is written WebSocket. You can see there, maybe you should click on that WebSocket. Right, there you see you click it there you will see that it is indeed a WebSocket packet.

And you can see that there is information about the WebSocket right there. There are four masked packets and all four of them are indeed the WebSocket packets. Recall what I mentioned to you about chunking the whole packet into smaller sizes and that is where the real timeness comes up. You can see that it is actually taken into four parts, okay.

(Refer Slide Time: 18:46)



Now click on that you see some data there. You have temperature sample value, you have last update, then you have the sampling rate, then some information related to the topic and so on, path and so on. If you take the next packet, it has other information which essentially is the state of the cooler, the fan rpm, the last updated value, the topic of interest, path and so on.

If you take the next packet, you will see information related to fan rpm, heater state and the last update topic and so on. In other words, this whole application which is, which we are demonstrating with this system here, you will see now, right? You know that the rpm, fan rpm means essentially this is the rpm of the fan. Heater state, heater is inside I mentioned to you.

These are the coolers, there are two coolers here, this is the cooler state. All this information which is captured by these IoT sensors is actually getting uploaded. And we are showing you that this information is actually coming via WebSockets. And essentially that is the information that you are conveying. You may be wondering what is this screen here, where is it coming from?

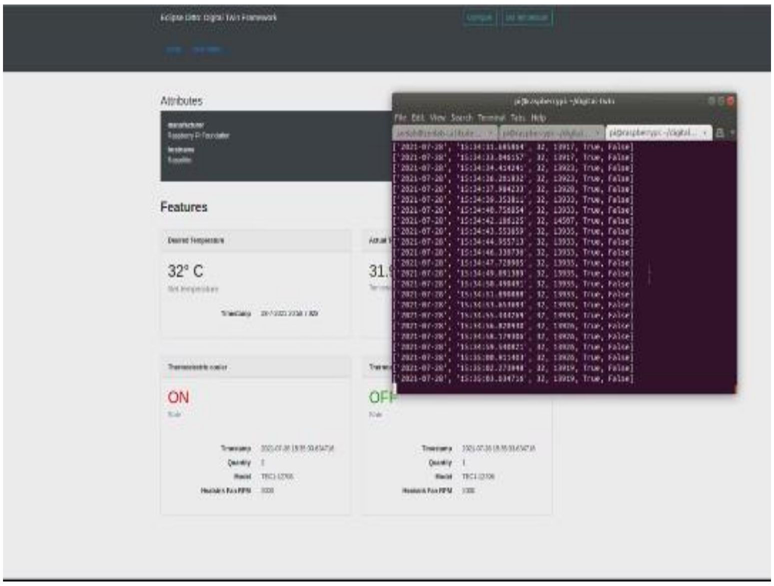
Well, what I have actually done is, this is the Raspberry Pi that you see. And this Raspberry Pi has the ability to run WebSocket connection to a laptop, okay. And the screen that I am showing you is a laptop. The laptop and this Raspberry Pi are connected over a wireless link and they are in the same subnet.

So essentially what you are seeing is the WebSocket transmission over a wireless link between the aggregation. Maybe you should understand this from this picture. These are the IoT nodes. This could be as I mentioned, 802.15.4e or 802.15.4. Both of them are IEEE and also it can be Bluetooth link. Now this Linux host R pi, which I show you here, right here, this one, this one here, this R pi is nothing but this R pi, okay.

This one is transmitting over Wi-Fi to a laptop. And there is a human here who is sitting and watching the screen. And what is the screen he is watching? The screen that he is watching is this screen. This is the screen, this exact screen where WireShark is running. So this screen, this is nothing but the screen, laptop, laptop screen. And what are the tools running on this?

One tool is WireShark. And the other tool that he is running is this application, which is essentially using WebSocket application. And how does the WebSocket application look? Let us go back to the screen.

(Refer Slide Time: 22:02)



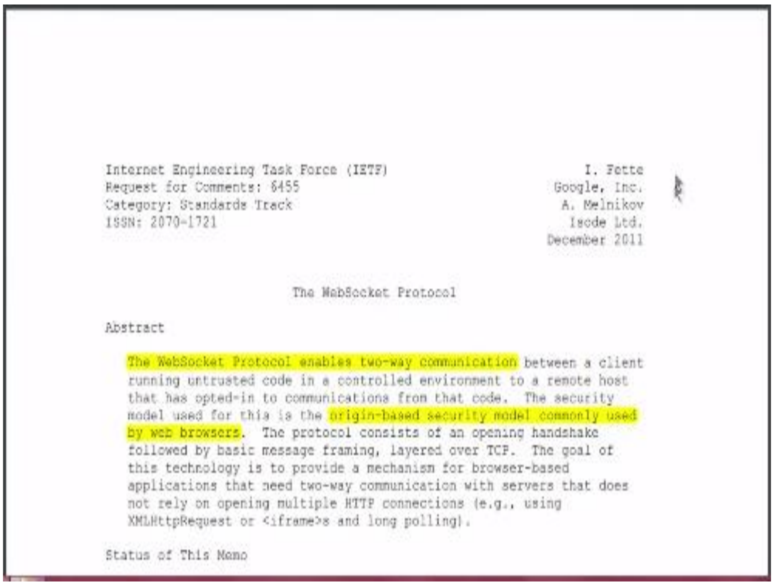
Okay, this is actually a web page. But what you see inside is the WebSocket termination, okay. The data is received over WebSockets that is what I am showing you here. And then put it onto this web page, which is easy for you to connect. So essentially, the data is arriving on WebSockets.

The screen that you, smaller screen that you see this arrow screen is indeed the WebSocket screen, reception of packets, and then we posted back on the create a web page and then put the data back there. So this is something which will allow you in WebSockets will allow you to control, to monitor, to actuate because it is bidirectional.

It will allow you to monitor you know, basically it will be able to monitor, you will be able to actuate, you will be able to do communication, bidirectional communication in real time, with low latency between your IoT nodes, actuators and sensing nodes and applications that are out there on the internet.

I hope this demonstration was useful so that you can also use WebSockets wherever these requirements have to be met. But before I close, as usual, I always want to tell you that if you want to learn more about WebSockets the most authentic source is indeed the RFC.

(Refer Slide Time: 23:24)



Internet Engineering Task Force (IETF) I. Fette
Request for Comments: 6455 Google, Inc.
Category: Standards Track A. Melnikov
ISSN: 2070-1721 Isode Ltd.
December 2011

The WebSocket Protocol

Abstract

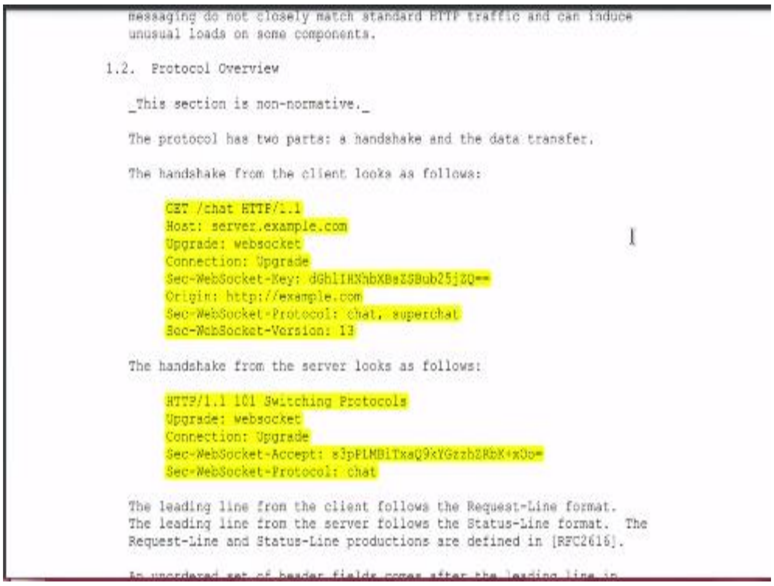
The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. 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 (e.g., using XMLHttpRequest or <iframe>s and long polling).

Status of This Memo

So let me take you to the RFC and directly show you the RFC. The RFC is here. And this is an important RFC. You can see this is RFC 6455. It is published in 2011. I am not going to go through this RFC because it is going to take time. But the most authentic source, if you want to know more about WebSockets read this RFC.

And the ones that are highlighted are just to give you an idea of the fact that whatever I explained is it is a two way communication and so on. And what is important is it also tells you why HTTP is not a good idea because of the problem related to request response. And the fact that HTTP is not really two way communication protocol. WebSocket protocol is designed to supersede existing bidirectional communication technologies but yet take benefit from existing infrastructure, okay.

(Refer Slide Time: 24:33)



messaging do not closely match standard HTTP traffic and can induce unusual loads on some components.

1.2. Protocol Overview

This section is non-normative.

The protocol has two parts: a handshake and the data transfer.

The handshake from the client looks as follows:

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

The handshake from the server looks as follows:

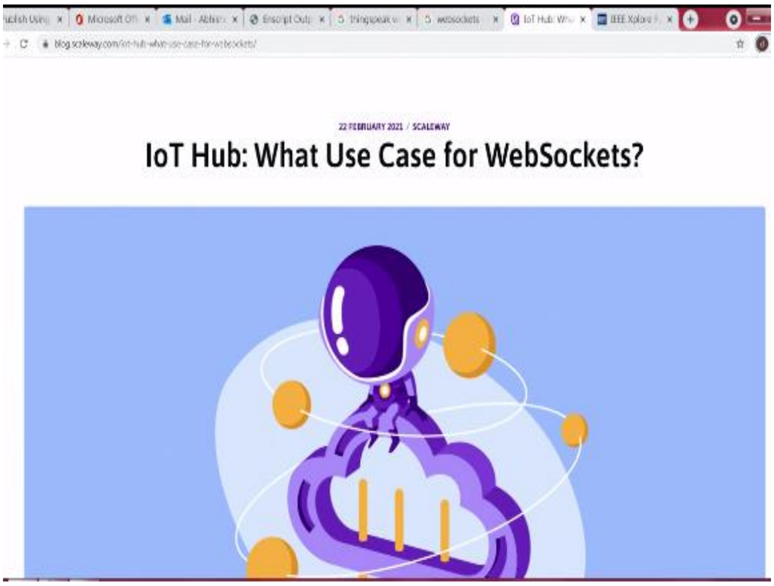
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616].

So now you see, as I said, you begin with HTTP. I highlighted that here, and then you upgrade to WebSocket. This is very important. That is why I highlighted this. Same here. The client asked for an upgrade, the server will upgrade and also tell you that yeah, I am fine, I can upgrade. So let us talk WebSocket and then everything becomes real time, low latency, single TCP connection and so on and so forth.

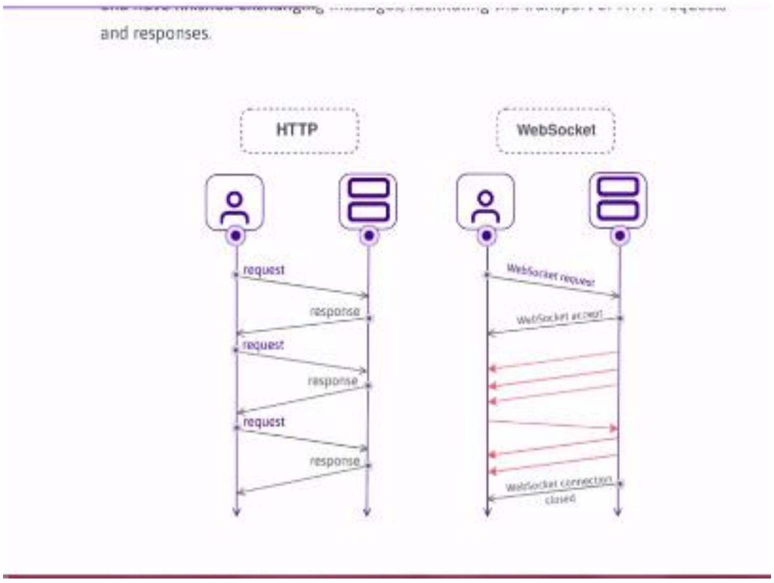
So please read this RFC for very authentic information about the protocol itself. As I mentioned to you, it is indeed a TCP based protocol. Yeah, so that is about what I wanted to tell you about the WebSockets.

(Refer Slide Time: 25:22)



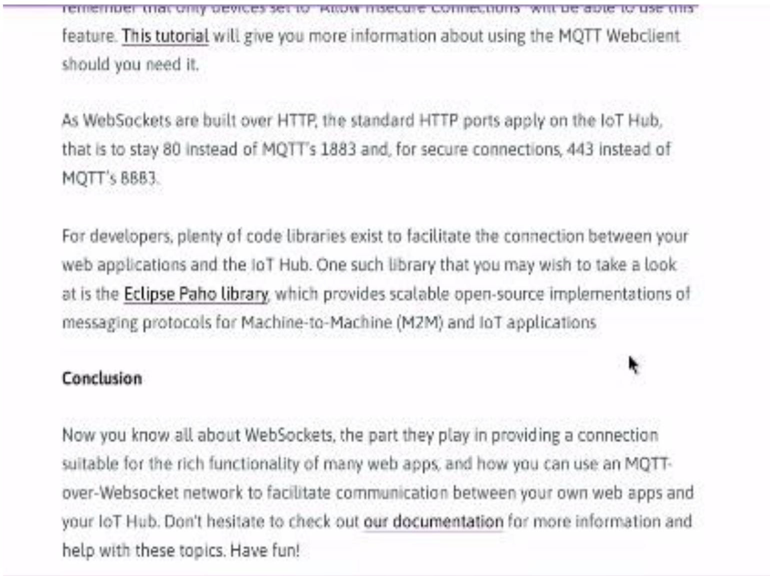
There is also this nice little article on IoT Hub, which is from Azure. Azure also supports WebSockets as much as ThingSpeak also supports WebSockets and IoT Hub has published a small article, what are the use cases for WebSockets? Please do read this protocol.

(Refer Slide Time: 25:43)



You will see the nice things that they write here, how HTTP and WebSockets are different. You can see the arrows and understand it lot better. It is actually, if you request you will get a response in HTTP, which is unlike in the case of WebSocket and is a highly scalable solution.

(Refer Slide Time: 26:00)



And WebSockets are built over HTTP and standard, you know the standard HTTP ports applied to IoT Hub, with the stay as 80, HTTP uses 80 instead of MQTTs 1883. And for secure connections that is 443 port number. Instead of MQTTs 8883, right. What is also interesting is what you can do with Eclipse Paho. For developers plenty of code libraries exists to facilitate connection between your web applications and IoT Hub.

One such library that you may wish to look up is the eclipse Paho library, which provides scalable open source implementation of messaging protocols for machine-to-machine communication and IoT applications. Thank you very much.