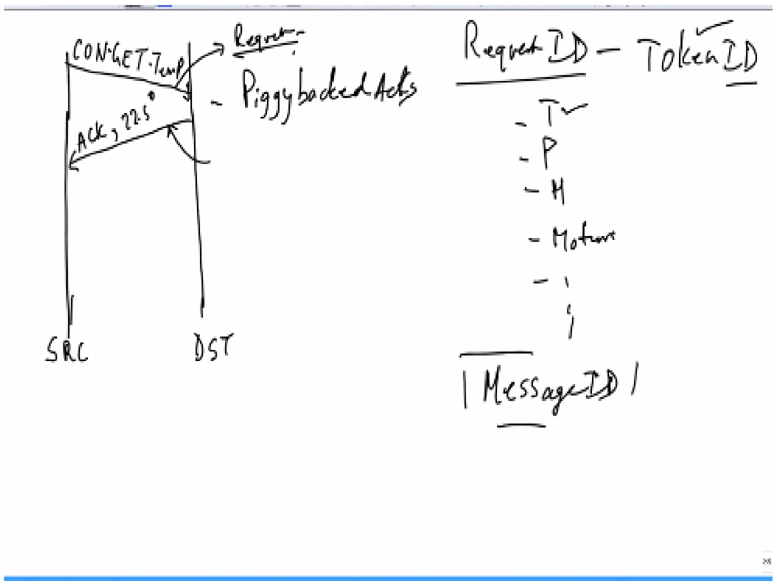
Design for Internet of Things
Prof. T V Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science-Bengaluru

Lecture - 37
COAP – 02

Alright, so let us look at some messages in the CoAP protocol. Excellent messages, easy to understand if you follow the simple logic that I am going to explain here.

(Refer Slide Time: 00:43)

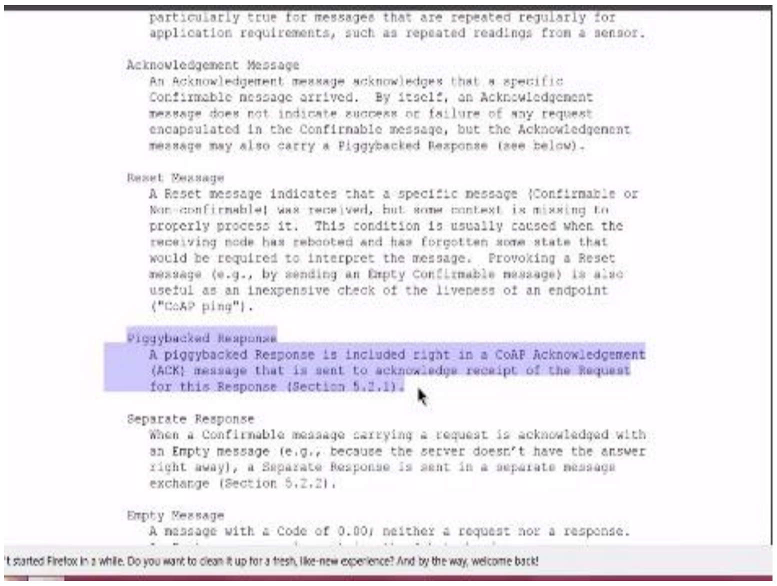


You have a source, you have a destination. You have to send the messages from the source to the destination. So you want to do it either confirmable mode, or non-confirmable mode. This is how a message goes. This is source, this is a destination. I send out a confirmable message. I am asking for a message. So I will use GET. And I am asking for a resource, which is temp let us say, which is temperature.

Now two possibilities exist. The destination may give an ACK and actually supply back the temperature or might, so l will say or, or might simply give an ACK and later send back the message acknowledgement with the temperature value. That is also possible, okay. So many ways by which you can do. So in other words, if you do it this way, there is a name for it. And that name is called piggybacked acknowledgments.

Beautiful, right? You are getting back the value here. And how do I know about it? Is there a way by which one can understand this? Yes, indeed, it is right here, you see this.

(Refer Slide Time: 02:14)



```
      particularly true for messages that are repeated regularly for
      application requirements, such as repeated readings from a sensor.

   Acknowledgement Message
      An Acknowledgement message acknowledges that a specific
      Confirmable message arrived.  By itself, an Acknowledgement
      message does not indicate success or failure of any request
      encapsulated in the Confirmable message, but the Acknowledgement
      message may also carry a Piggybacked Response (see below).

   Reset Message
      A Reset message indicates that a specific message (Confirmable or
      Non-confirmable) was received, but some context is missing to
      properly process it.  This condition is usually caused when the
      receiving node has rebooted and has forgotten some state that
      would be required to interpret the message.  Provoking a Reset
      message (e.g., by sending an Empty Confirmable message) is also
      useful as an inexpensive check of the liveness of an endpoint
      ("CoAP ping").

   Piggybacked Response
      A piggybacked Response is included right in a CoAP Acknowledgement
      (ACK) message that is sent to acknowledge receipt of the Request
      for this Response (Section 5.2.1).

   Separate Response
      When a Confirmable message carrying a request is acknowledged with
      an Empty message (e.g., because the server doesn't have the answer
      right away), a Separate Response is sent in a separate message
      exchange (Section 5.2.2).

   Empty Message
      A message with a Code of 0.00; neither a request nor a response.
```

't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Piggybacked response is included right in the CoAP acknowledgement message that is sent to acknowledge receipt of the request for this response. That was actually a request that was sent out. This was actually a request, okay. And that request was given with a response. This is actually a response, saying that I received your request. It is not actually a response, it is actually a acknowledging that it received the request.

And along with that, you are actually giving the response of the data, right, which is 22.5. That is really powerful. Energy wise, you can save, you can save one transmission from destination to source. So it is quite energy efficient, right? Now whenever you talk about requests, you want to match a request that I make to a client, and the client should respond back to that request.

So I may have multiple requests, right? I may request for temperature, I may request for pressure, I may request for humidity, I may request for motion data, so on and so forth. So each one of these requests, I can ask the same destination because it is equipped with all those sensors. So how do I differentiate one request from the other, okay?

So how do you differentiate between from one request to the other, you essentially have a request ID. In the parlance of CoAP, nobody talks about request ID, but it is there in the RFC, if you look carefully. It is actually called token ID, token ID, okay. So request ID is nothing but the token ID. I am requesting for some data from a given sensor and I am getting a response back.

So as I said, it can be temperature, it can be pressure, request for pressure data, request for humidity data, requests for motion sensing data, request for anything, right. Light information, indoor light, outdoor light, anything. All these are requests, which are making for different sensors, which are connected, and each one can be associated with a request ID.

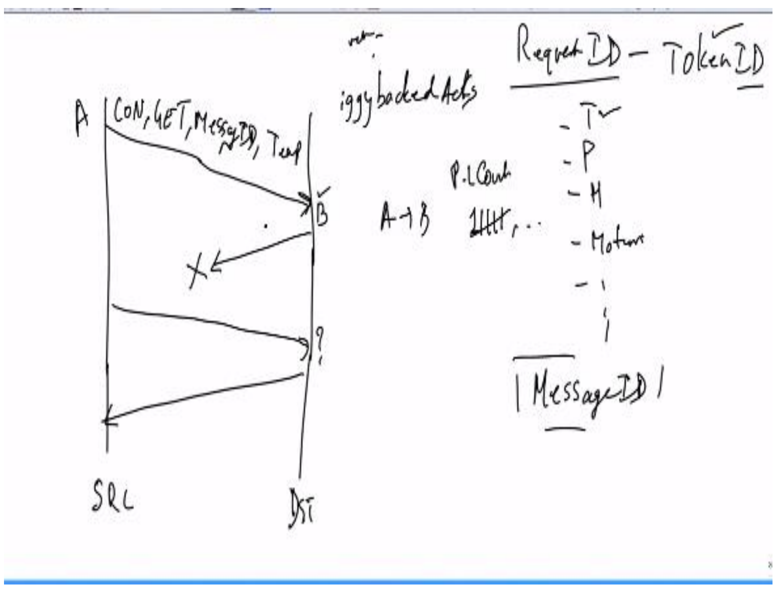
And that in the parlance of CoAP is actually called token ID. Now if I asked for temperature, and you did not have the temperature with you, you may say I am acknowledging your request. I will give you the temperature later. So what will you do? How will I know that you are actually giving me the temperature value for the request I sent? You return back the request ID. That is all.

I sent you with a request. You send it back with a request ID. I know that you have, we are matching the request and the response. So token IDs are used for purposes of matching the requests that we make. Request IDs, right? Request ID is given with a request response. By returning it with the same token ID, you are saying that was your request, this is the value, which I am going to give you. So that is about tokens.

Then there is also this notion of duplicates, that is with respect to packets, forget about requests, request is a slightly higher layer, right? Packets are the ones that are actually going out on the link. Those packets can get lost. And they have to perhaps be retransmitted. That time, what they do, they associate each message with an ID. And that is called the message ID.

So you have message ID and you have token ID. Token ID I think is clear in your mind. So I will not discuss that. Message ID is what you should know about, okay.

(Refer Slide Time: 06:11)



Now what you do, in this CON-GET-Temp, you normally I will modify this whole thing now, okay. I will modify this whole thing. And I will send CON, GET, message ID, okay. What is the value? What are you looking for? You are trying to get Temp. You send. Now this guy will give an ACK with this message ID and he will give back the value. This is how it actually works, okay.

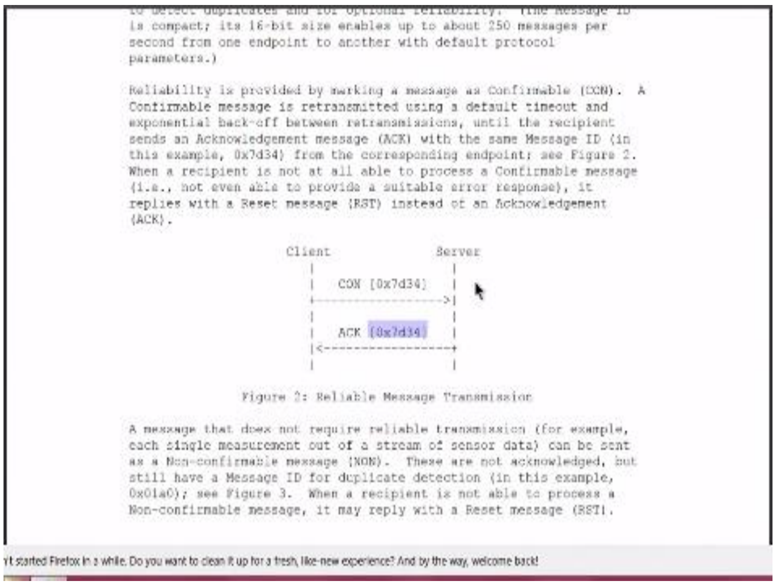
So you see, he is returning back the message ID to tell you that this is what you sent me. I got it. And I am responding back. It is possible of this situation also. When this was sent, this was dropped, right? It did not reach. Then what will happen? Or you can also think about the following, right? This may have reached, but it is on its way back, it may have got dropped, right? Then this guy what he will do?

He will perhaps again ask with the same message ID, he might ask with the same message ID. If he asked with the same message ID, this guy knows, hey, I got this here. Now he is asking me again here. So let me respond back. And now I actually keep account that A, this is source B. This is source, this is destination. Looks like A to B packet loss count I will increment by 1.

Second time it happens I will increment by 1. Third, fourth, fifth, I will increment by 1. So I actually know how good my link is with my destination. So that is how you use the message ID. All of this is nicely captured, perhaps even better explained in the RFC. So piggybacked response, separate response.

When a conformable message carrying a request is **is** acknowledged, with an empty message, a separate response is sent in a separate message exchange. You can also have an empty exchange. So let us see some pictures that will give you a better understanding of the whole thing.

(Refer Slide Time: 08:40)



```
   is compact; its 16-bit size enables up to about 250 messages per
   second from one endpoint to another with default protocol
   parameters.)

   Reliability is provided by marking a message as Confirmable (CON).  A
   Confirmable message is retransmitted using a default timeout and
   exponential back-off between retransmissions, until the recipient
   sends an Acknowledgement message (ACK) with the same Message ID (in
   this example, 0x7d34) from the corresponding endpoint; see Figure 2.
   When a recipient is not at all able to process a Confirmable message
   (i.e., not even able to provide a suitable error response), it
   replies with a Reset message (RST) instead of an Acknowledgement
   (ACK).

                        Client              Server
                           |                  |
                           |   CON [0x7d34]   |
                           +----------------->|
                           |                  |
                           |   ACK [0x7d34]   |
                           |<-----------------+
                           |                  |

                Figure 2: Reliable Message Transmission

   A message that does not require reliable transmission (for example,
   each single measurement out of a stream of sensor data) can be sent
   as a Non-confirmable message (NON).  These are not acknowledged, but
   still have a Message ID for duplicate detection (in this example,
   0x01a0); see Figure 3.  When a recipient is not able to process a
   Non-confirmable message, it may reply with a Reset message (RST).
```

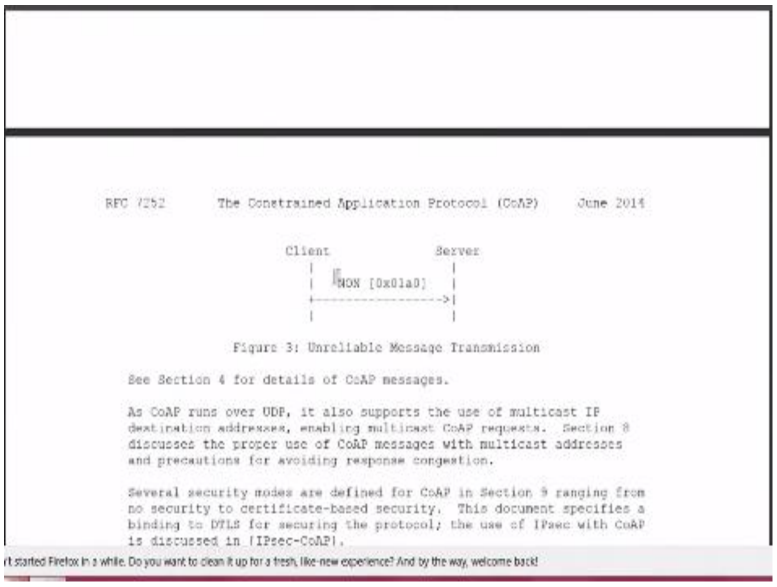
't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Look at this client, server. Client is asking for some data. CON, what is this? This is the message ID. What happened here? He acknowledged back the message ID, reliable message transmission. Okay, now you see. Reliability is provided by marking a message as confirmable.

A confirmable message is retransmitted using a default timeout and exponential back off between retransmissions until the recipient sends an acknowledgment with the same message ID from the corresponding endpoint. When a recipient is not able to process a confirmable message, it replies with a reset message instead of an acknowledgment ACK.

That means if you send a CON, and I am not able to honor it, I will give you back RST reset message. You can also do non-confirmable. When you do non-confirmable, also you will have a message ID. It is again to assist receivers to detect losses, duplicates and so on.

(Refer Slide Time: 09:46)

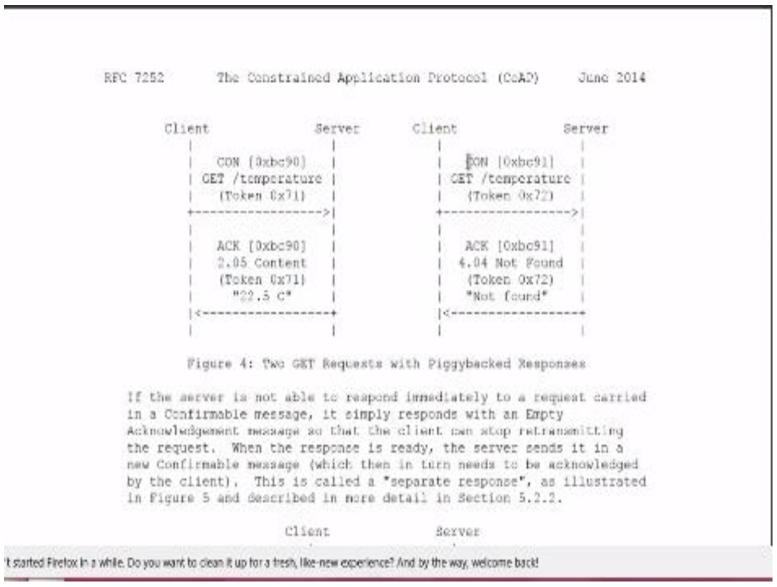


```
RFC 7252       The Constrained Application Protocol (CoAP)     June 2014

                        Client              Server
                           |                  |
                           |   NON [0x01a0]   |
                           +----------------->|
                           |                  |

                 Figure 3: Unreliable Message Transmission

   See Section 4 for details of CoAP messages.

   As CoAP runs over UDP, it also supports the use of multicast IP
   destination addresses, enabling multicast CoAP requests.  Section 8
   discusses the proper use of CoAP messages with multicast addresses
   and precautions for avoiding response congestion.

   Several security modes are defined for CoAP in Section 9 ranging from
   no security to certificate-based security.  This document specifies a
   binding to DTLS for securing the protocol; the use of IPsec with CoAP
   is discussed in [IPsec-CoAP].
```

't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

So again you see this is NON. Client, server, NON message ID to the server. And you are not expecting an acknowledgment, but you continue to put a message ID to detect duplicates to detect that there is a loss and so on. That is irrespective of whether you want a response or not, right?

(Refer Slide Time: 10:08)



```
RFC 7252       The Constrained Application Protocol (CoAP)     June 2014

        Client              Server       Client              Server
           |                  |            |                  |
           |   CON [0xbc90]   |            |   CON [0xbc91]   |
           | GET /temperature |            | GET /temperature |
           |   (Token 0x71)   |            |   (Token 0x72)   |
           +----------------->|            +----------------->|
           |                  |            |                  |
           |   ACK [0xbc90]   |            |   ACK [0xbc91]   |
           |   2.05 Content   |            |   4.04 Not Found |
           |   (Token 0x71)   |            |   (Token 0x72)   |
           |     "22.5 C"     |            |   "Not found"    |
           |<-----------------+            |<-----------------+
           |                  |            |                  |

           Figure 4: Two GET Requests with Piggybacked Responses

   If the server is not able to respond immediately to a request carried
   in a Confirmable message, it simply responds with an Empty
   Acknowledgement message so that the client can stop retransmitting
   the request.  When the response is ready, the server sends it in a
   new Confirmable message (which then in turn needs to be acknowledged
   by the client).  This is called a "separate response", as illustrated
   in Figure 5 and described in more detail in Section 5.2.2.

                        Client              Server
```

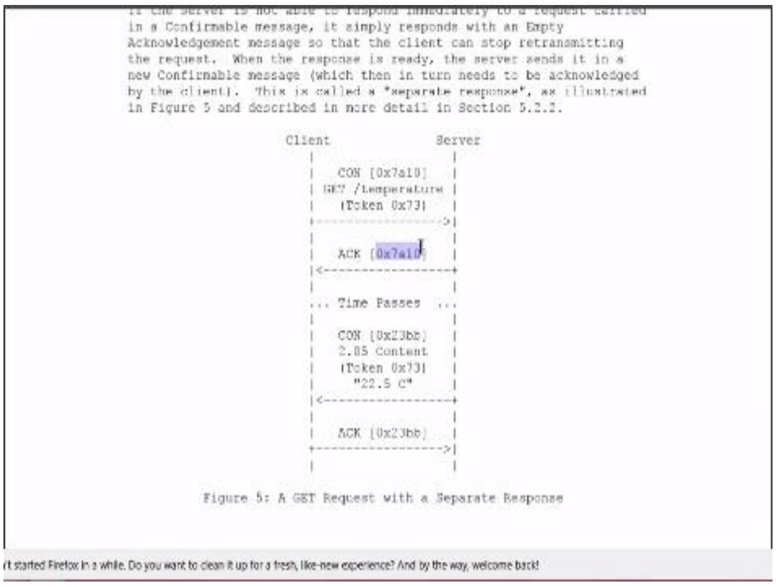
't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

So the whole thing goes like this. Look at this beautiful thing, CON. This is message ID GET, temperature, token. All of it is there here. What does this fellow say? I am asking you a GET, I am asking you to please give me your temperature value. This is my message ID. And this is my token, which I am going to give you, because I am asking this token for this temperature value, okay.

Now this guy says, great. I am acknowledging you. I am going to put back your message ID. I am giving you a response code, which is 2.05. And here is the content, which is 22.5. I am also returning back your request ID or nothing but the token ID. Great, this is done. Now let us look at this. CON, message ID, GET, please give me your temperature. I am giving you one token, which is 0x72.

Now the server says I am acknowledging that I got your message. Response code is 4.04. Because what you asked I do not have, but please note I received your message. That is why I am able to give you back with the same message ID and I am giving you back your token. 4.04 corresponds to not found. This is also possible, okay.

(Refer Slide Time: 11:53)



```
in a Confirmable message, it simply responds with an Empty
Acknowledgement message so that the client can stop retransmitting
the request.  When the response is ready, the server sends it in a
new Confirmable message (which then in turn needs to be acknowledged
by the client).  This is called a "separate response", as illustrated
in Figure 5 and described in more detail in Section 5.2.2.

                    Client              Server
                       |                  |
                       |   CON [0x7a10]   |
                       | GET /temperature |
                       |   (Token 0x73)   |
                       +----------------->|
                       |                  |
                       |   ACK [0x7a10]   |
                       |<-----------------+
                       |                  |
                       ... Time Passes  ...
                       |                  |
                       |   CON [0x23bb]   |
                       |   2.05 Content   |
                       |   (Token 0x73)   |
                       |     "22.5 C"     |
                       |<-----------------+
                       |                  |
                       |   ACK [0x23bb]   |
                       +----------------->|
                       |                  |

              Figure 5: A GET Request with a Separate Response
```

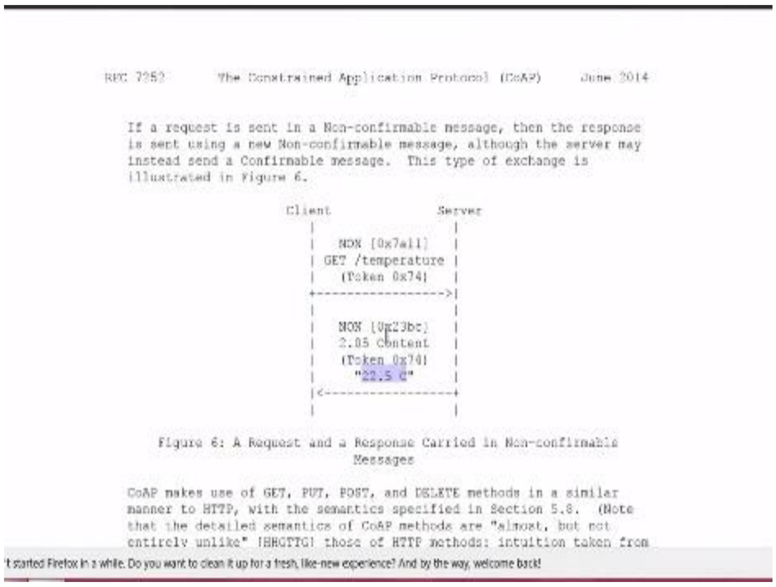
t started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Now look at other types of messages. CON, message ID. Get me your temperature. This is what the token I am going to give you. Acknowledgement comes. Pat comes the reply as they say, acknowledging user, this is your message ID. Then what happens? It is in low power. It is sleeping, fast asleep. Temperature sensor is down. ADC is down. You are in deep sleep.

Only CPU responded back to the client by saying I got your message. Now it is time for a timer interrupt to fire. When the timer interrupt fires on the server, the server now wakes up completely, switches on the ADC, acquires the data, processes the temperature data, packetize it and then tries to send it back. What does it do? It now sets up a fresh CON message, you can see this.

It sets up a fresh corn message. But remember, it has not forgotten the request ID. So it has put back the request ID. Everything else is new. Put back the CON. Put a new CON 2.05. That means response code is 2.05. And this is my temperature. For which the client says I am grateful to you. I am giving you my acknowledgement for the message ID you generated. And it says this is how it can do with a separate response.

(Refer Slide Time: 13:34)



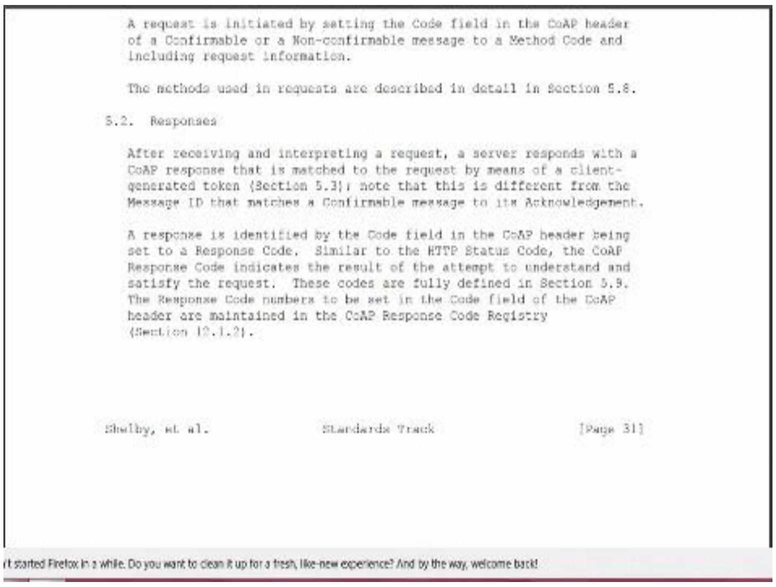
```
RFC 7252        The Constrained Application Protocol (CoAP)      June 2014

   If a request is sent in a Non-confirmable message, then the response
   is sent using a new Non-confirmable message, although the server may
   instead send a Confirmable message.  This type of exchange is
   illustrated in Figure 6.

                        Client              Server
                           |                  |
                           |   NON [0x7a11]   |
                           | GET /temperature |
                           |   (Token 0x74)   |
                           +----------------->|
                           |                  |
                           |   NON [0x23bc]   |
                           |   2.05 Content   |
                           |   (Token 0x74)   |
                           |     "22.5 C"     |
                           |<-----------------+
                           |                  |

       Figure 6: A Request and a Response Carried in Non-confirmable
                                 Messages

   CoAP makes use of GET, PUT, POST, and DELETE methods in a similar
   manner to HTTP, with the semantics specified in Section 5.8.  (Note
   that the detailed semantics of CoAP methods are "almost, but not
   entirely unlike" [HHGTTG] those of HTTP methods: intuition taken from
```

't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Now this is another message, non-confirmable message ID. Please get me your temperature. This is the token I am giving you. ACK will not come because this is non-confirmable but Providence is such that it has reached the destination server. So server says okay, fine no problem. I am sending you back a non-confirmable message. I have put a new message ID, response code is 2.05.

But I am giving you back your token and I am giving you my value. Beauty right? Send a NON you get back a NON. That is also possible. Let me also take you to other parts of this. So I will go a little more rapidly so that I can point you to some key points in this RFC about request and response.

(Refer Slide Time: 14:41)

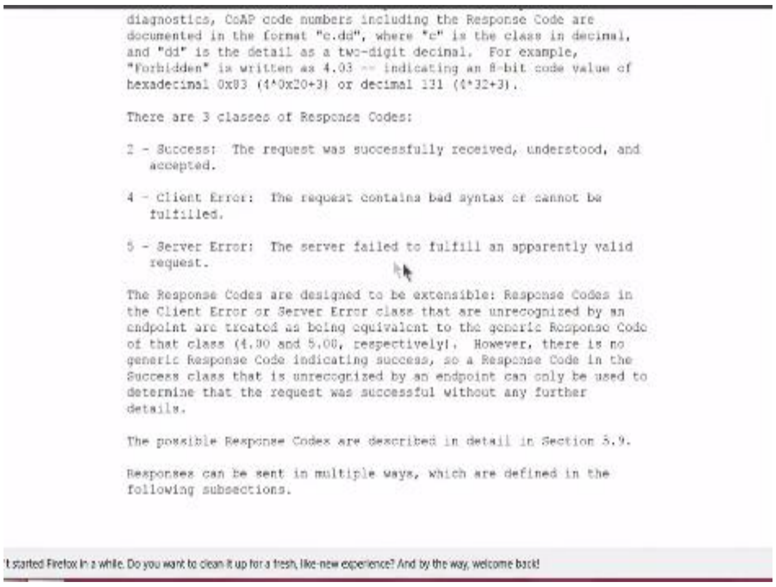


```
   A request is initiated by setting the Code field in the CoAP header
   of a Confirmable or a Non-confirmable message to a Method Code and
   including request information.

   The methods used in requests are described in detail in Section 5.8.

5.2.  Responses

   After receiving and interpreting a request, a server responds with a
   CoAP response that is matched to the request by means of a client-
   generated token (Section 5.3); note that this is different from the
   Message ID that matches a Confirmable message to its Acknowledgement.

   A response is identified by the Code field in the CoAP header being
   set to a Response Code.  Similar to the HTTP Status Code, the CoAP
   Response Code indicates the result of the attempt to understand and
   satisfy the request.  These codes are fully defined in Section 5.9.
   The Response Code numbers to be set in the Code field of the CoAP
   header are maintained in the CoAP Response Code Registry
   (Section 12.1.2).



Shelby, et al.              Standards Track                   [Page 31]
```

't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Yeah, so responses are important. So let me go into some details of this responses, okay. Now after receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request type by means of tokens. We discussed this already. There are some nice things about this.

(Refer Slide Time: 15:02)



```
   diagnostics, CoAP code numbers including the Response Code are
   documented in the format "c.dd", where "c" is the class in decimal,
   and "dd" is the detail as a two-digit decimal.  For example,
   "Forbidden" is written as 4.03 -- indicating an 8-bit code value of
   hexadecimal 0x83 (4*0x20+3) or decimal 131 (4*32+3).

   There are 3 classes of Response Codes:

   2 - Success:  The request was successfully received, understood, and
      accepted.

   4 - Client Error:  The request contains bad syntax or cannot be
      fulfilled.

   5 - Server Error:  The server failed to fulfill an apparently valid
      request.

   The Response Codes are designed to be extensible: Response Codes in
   the Client Error or Server Error class that are unrecognized by an
   endpoint are treated as being equivalent to the generic Response Code
   of that class (4.00 and 5.00, respectively).  However, there is no
   generic Response Code indicating success, so a Response Code in the
   Success class that is unrecognized by an endpoint can only be used to
   determine that the request was successful without any further
   details.

   The possible Response Codes are described in detail in Section 5.9.

   Responses can be sent in multiple ways, which are defined in the
   following subsections.
```

't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

If you see 2, that means it is success. 4 means client error. 5 means server error. 2.05, I showed you. 2.05 means what? 2 obviously means it is success. 2.05 that means the request was successfully received, understood and accepted. 4 is what? The request contains a bad syntax, but perhaps was successfully received, right? And therefore, it could not be fulfilled. 4.04 is the same as the response code 4.

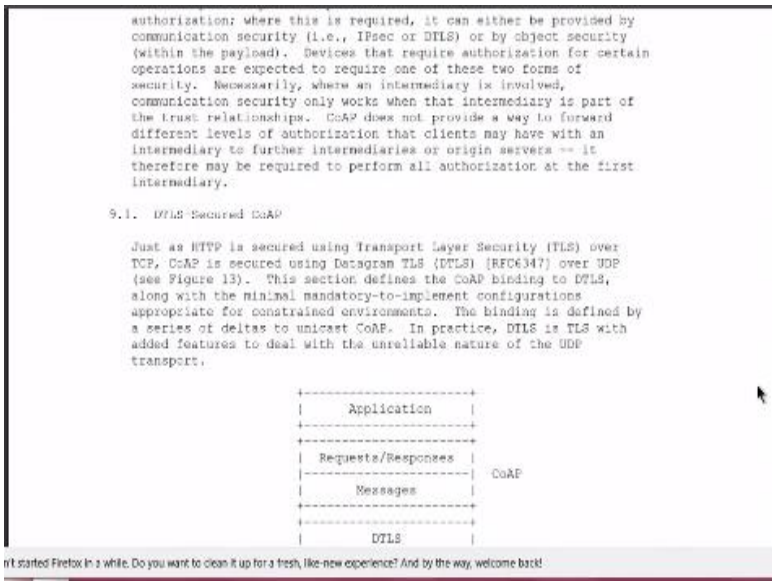
5 is server error, has failed completely. So that is what it is here, okay. And this goes into great detail of talking about message IDs and tokens and so on, which I think you must look up very carefully and understand this. Folks, so this is very critical. And I must tell you that there is one more superly nice option in CoAP which we must spend a little more time to understand.

And that comes to two important aspects. In CoAP, unlike MQTT, the CoAP systems support discovery of services, okay. What were you doing in MQTT? In MQTT, you knew that it is indoor light sensor, you knew it is an outdoor light sensor, which was publishing. So your application was built by subscribing to those topics. Because you knew the topics a priori. But if you did not know, how will you discover that indoor is there?

How do you know let us say, in a network, a new sensor is added. For one window, you may have had east window, west window may have had another set of sensors; indoor, outdoor, motion detection and so on. You just added it, okay. But if I did not know, if I am an application builder, and I did not know that is going to be an issue, right? That problem exists in MQTT. It is not an automatic discovery of resources.

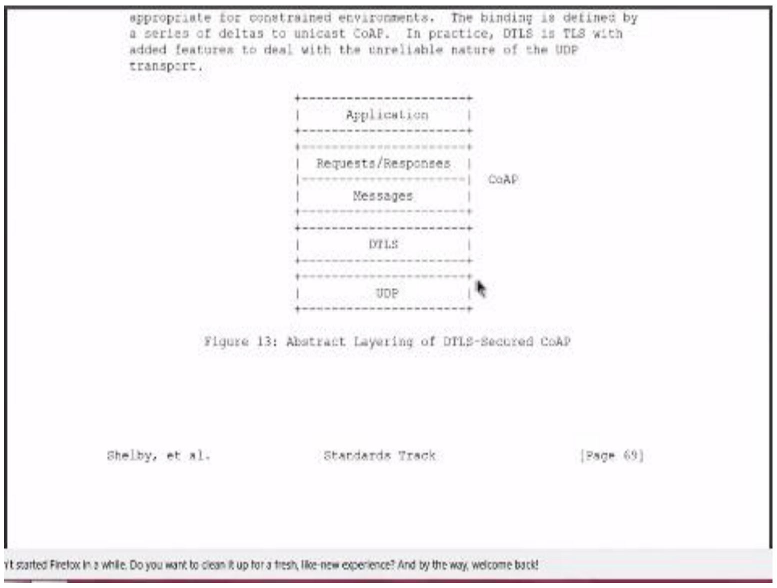
Whereas CoAP has that facility of discovery of resources. So if you do a GET to well-known core, then all the resources available, will be issued by the server. It will actually tell you what are all the services it is offering. So that is a beautiful option of resource discovery that is available in CoAP. So that is what they mean by service discovery. And I will show you how you can use that in an effective way. And also you know build some interesting applications.

(Refer Slide Time: 18:13)


```
   authorization; where this is required, it can either be provided by
   communication security (i.e., IPsec or DTLS) or by object security
   (within the payload).  Devices that require authorization for certain
   operations are expected to require one of these two forms of
   security.  Necessarily, where an intermediary is involved,
   communication security only works when that intermediary is part of
   the trust relationships.  CoAP does not provide a way to forward
   different levels of authorization that clients may have with an
   intermediary to further intermediaries or origin servers -- it
   therefore may be required to perform all authorization at the first
   intermediary.

9.1.  DTLS-Secured CoAP

   Just as HTTP is secured using Transport Layer Security (TLS) over
   TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP
   (see Figure 13).  This section defines the CoAP binding to DTLS,
   along with the minimal mandatory-to-implement configurations
   appropriate for constrained environments.  The binding is defined by
   a series of deltas to unicast CoAP.  In practice, DTLS is TLS with
   added features to deal with the unreliable nature of the UDP
   transport.

                      +----------------------+
                      |      Application     |
                      +----------------------+
                      +----------------------+
                      |  Requests/Responses  |
                      |----------------------|  CoAP
                      |       Messages       |
                      +----------------------+
                      +----------------------+
                      |         DTLS         |
```
n't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

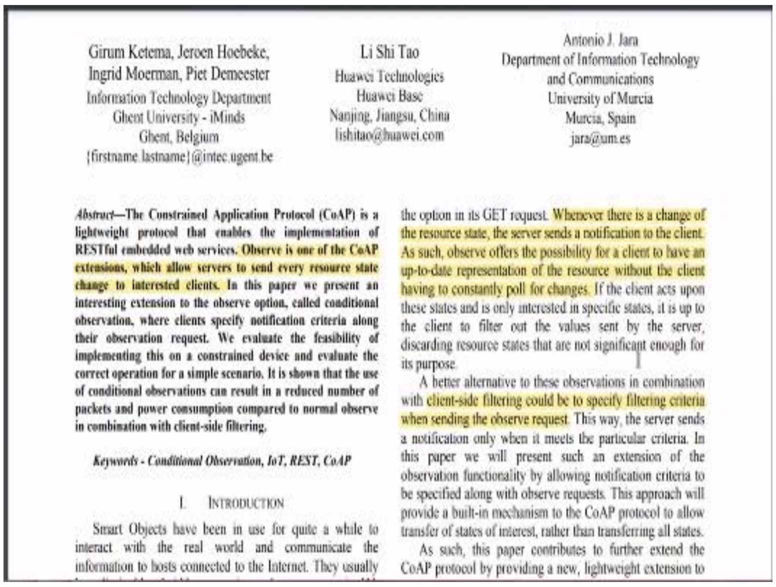
I mentioned to you that CoAP as much as MQTT is highly secure. Please do look up their DTLS stack, which is out there.

(Refer Slide Time: 18:23)


```
   appropriate for constrained environments.  The binding is defined by
   a series of deltas to unicast CoAP.  In practice, DTLS is TLS with
   added features to deal with the unreliable nature of the UDP
   transport.

                      +----------------------+
                      |      Application     |
                      +----------------------+
                      +----------------------+
                      |  Requests/Responses  |
                      |----------------------|  CoAP
                      |       Messages       |
                      +----------------------+
                      +----------------------+
                      |         DTLS         |
                      +----------------------+
                      +----------------------+
                      |          UDP         |
                      +----------------------+

              Figure 13: Abstract Layering of DTLS-Secured CoAP




Shelby, et al.             Standards Track                    [Page 69]
```
't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

You can see that messages pass through DTLS before they are passed to the UDP. Datagram Transport Layer Security, okay. So they get encrypted and they are sent out. So this is an important thing. So what I did not explain to you is a outstandingly good option for energy harvesting and battery operated devices if you use CoAP. And that comes to a nice option, which is called observe. Observe is a very powerful option.

(Refer Slide Time: 18:57)



Girum Ketema, Jeroen Hoebeke, Ingrid Moerman, Piet Demeester
Information Technology Department
Ghent University - iMinds
Ghent, Belgium
{firstname.lastname}@intec.ugent.be

Li Shi Tao
Huawei Technologies
Huawei Base
Nanjing, Jiangsu, China
lishitao@huawei.com

Antonio J. Jara
Department of Information Technology and Communications
University of Murcia
Murcia, Spain
jara@um.es

***Abstract*—The Constrained Application Protocol (CoAP) is a lightweight protocol that enables the implementation of RESTful embedded web services. Observe is one of the CoAP extensions, which allow servers to send every resource state change to interested clients. In this paper we present an interesting extension to the observe option, called conditional observation, where clients specify notification criteria along their observation request. We evaluate the feasibility of implementing this on a constrained device and evaluate the correct operation for a simple scenario. It is shown that the use of conditional observations can result in a reduced number of packets and power consumption compared to normal observe in combination with client-side filtering.**

Keywords - Conditional Observation, IoT, REST, CoAP

I. INTRODUCTION

Smart Objects have been in use for quite a while to interact with the real world and communicate the information to hosts connected to the Internet. They usually

the option in its GET request. Whenever there is a change of the resource state, the server sends a notification to the client. As such, observe offers the possibility for a client to have an up-to-date representation of the resource without the client having to constantly poll for changes. If the client acts upon these states and is only interested in specific states, it is up to the client to filter out the values sent by the server, discarding resource states that are not significant enough for its purpose.

A better alternative to these observations in combination with client-side filtering could be to specify filtering criteria when sending the observe request. This way, the server sends a notification only when it meets the particular criteria. In this paper we will present such an extension of the observation functionality by allowing notification criteria to be specified along with observe requests. This approach will provide a built-in mechanism to the CoAP protocol to allow transfer of states of interest, rather than transferring all states.

As such, this paper contributes to further extend the CoAP protocol by providing a new, lightweight extension to

This is a paper I was reading recently of how nicely researchers have used this observe option in CoAP in a very effective manner. You do not have to read so much about it. You just have to know what observer is. I highlighted it for you here. It says that observe is one of the CoAP extensions, which allows servers to send every resource state change to interested clients.

Supposing the temperature was set to 22.5 degrees, okay. You do not report back to requesting clients. But if it changed to 28 degrees, that is a huge change. Then you report, not otherwise. That is fantastic. So you are observing a resource. Only if that resource changes, you will actually intimate the client that has requested for that data. That is what is shown here.

(Refer Slide Time: 20:02)

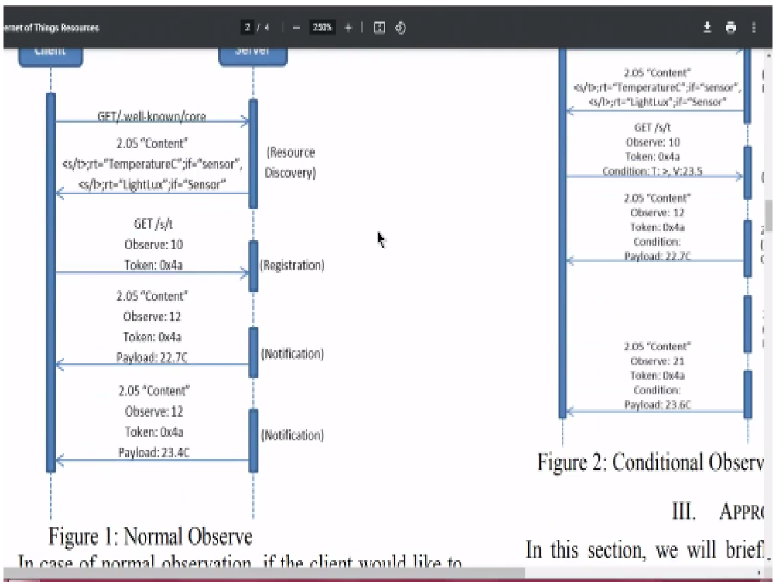


Figure 1: Normal Observe

In case of normal observation, if the client would like to

Figure 2: Conditional Observ

III. APPR

In this section, we will brief

You can see in this picture. What does it say? So let me expand it a little more so that you understand it better. GET.well-known core. That means you are discovering all resources. What happens? 2.05. That means 2 is for success. 2.05 content. What are all the sensors out there? Temperature and light lux both sensors are there. The client could easily discover what all the services that the system is able to provide, okay.

Now temperature and light lux both of them are there. Now using that it is saying that now get s, slash s slash t, because it has found out that it is a temperature sensor is of interest 28, you can see that. This is the s/t which is used here, okay. s/l might perhaps correspond to light sensor. So it has said get this thing. Now before saying a GET it made this nice option observe, okay.

And it gave a token, this is the token. It is a request ID. So for this request ID it put a token. Now see what happens. This guy says 2.05 that means it is success. Puts back the same token, puts back observe called 12. This guy says observe 10 that guy put back observe 12 and put back that payload as 22.7, okay. Then again, it sent back a notification.

(Refer Slide Time: 21:47)

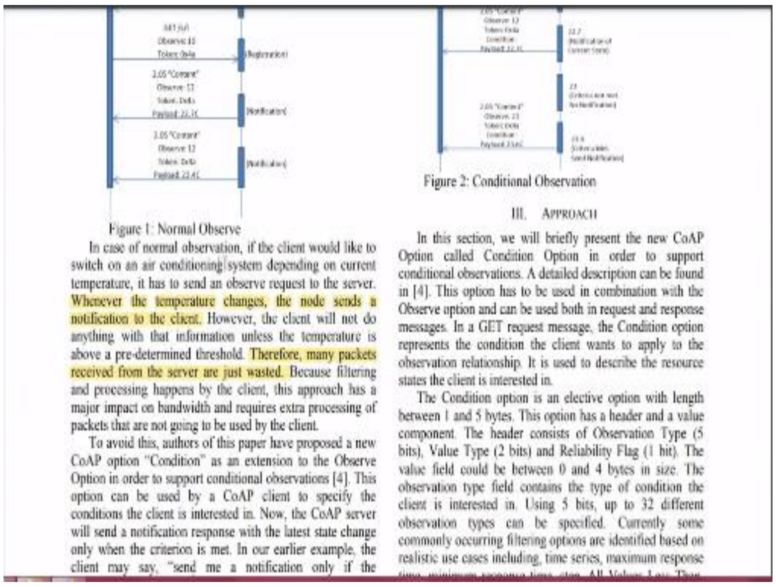


Figure 1: Normal Observe

In case of normal observation, if the client would like to switch on an air conditioning system depending on current temperature, it has to send an observe request to the server. Whenever the temperature changes, the node sends a notification to the client. However, the client will not do anything with that information unless the temperature is above a pre-determined threshold. Therefore, many packets received from the server are just wasted. Because filtering and processing happens by the client, this approach has a major impact on bandwidth and requires extra processing of packets that are not going to be used by the client.

To avoid this, authors of this paper have proposed a new CoAP option "Condition" as an extension to the Observe Option in order to support conditional observations [4]. This option can be used by a CoAP client to specify the conditions the client is interested in. Now, the CoAP server will send a notification response with the latest state change only when the criterion is met. In our earlier example, the client may say, "send me a notification only if the

Figure 2: Conditional Observation

III. APPROACH

In this section, we will briefly present the new CoAP Option called Condition Option in order to support conditional observations. A detailed description can be found in [4]. This option has to be used in combination with the Observe option and can be used both in request and response messages. In a GET request message, the Condition option represents the condition the client wants to apply to the observation relationship. It is used to describe the resource states the client is interested in.

The Condition option is an elective option with length between 1 and 5 bytes. This option has a header and a value component. The header consists of Observation Type (5 bits), Value Type (2 bits) and Reliability Flag (1 bit). The value field could be between 0 and 4 bytes in size. The observation type field contains the type of condition the client is interested in. Using 5 bits, up to 32 different observation types can be specified. Currently some commonly occurring filtering options are identified based on realistic use cases including, time series, maximum response

In the normal observation if the client would like to switch on the air conditioning system, depending on the current temperature, it has to send an observe request to the server. Whenever the temperature changes, the node sends a notification to the client. However, the client will not do anything with that information unless the temperature is above a predetermined threshold.

Therefore many packets received from the server are just wasted. This is how you have built the application. You are asking with observe, normal observation client would like to switch on the AC depending on the current temperature. You are asking the server to give it a temperature. It has to send an observe request to the server. Whenever there is a change in temperature, the node will send a notification, okay.

But your application is built to do something only if it is beyond a certain predetermined threshold, only then it will work. So what happens to all the packets that the system sent? Nothing it is going to do. It is just going to be wasted. Because filtering and processing happens by the client, okay. So this approach has a major impact on the bandwidth. A server is reporting a change, but application says I do not want that value.

So it has to discard. To avoid this what it does is they have proposed, the authors have proposed this one. See look at what is in this here. It has given a value 22.7 with the same token 0x4a. And again, the resource changed to another value 23.4. That time it put back the same token and the observe and sent it again.

So you see every time there is a change in the value of this resource it is intimating the application, which is nothing but the application running on the client side. This is real waste of resources because client is building an application perhaps only if it crosses 30 degrees, let us say. What is it going to do? Nothing. It is going to take this 22.7 data packet, throw it into the dustbin. Same with deleted.

Same with the other one. It may simply throw it out because it says it is not useful, thank you very much, but I am not going to use it kind of, right. How to solve that? You do a conditional observe. Well this is not there in the standard. So I encourage you to try this, okay. This is what was proposed by the new authors, by these authors of this paper.

You do GET well-known core. You find out that there is temperature and lux. Then you go after the temperature, supply it a token 0x4a and you supply now a condition. Amazing, right? You supply a condition t greater than so and so, only then you report me back the value, okay. But what actually happened was 23.5 but you got this value only for the first time. Because you sent a GET it is it is obliged to respond with a reply, right?

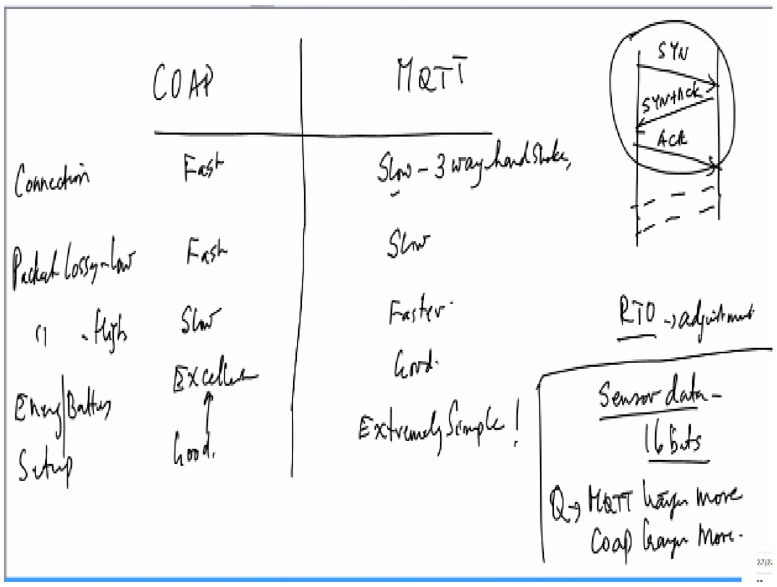
First time alone it responds by saying, okay thank you very much. I understand your condition. You want values greater than 23.5. No problem. Right now my temperature is 22.7. I am just giving you that value. Then lot of time passes and then maybe something else, 23.5 is far off from 22.7. So it has to wait for a significant amount of time for it to heat up, right? So some time passes.

Then what happens slowly from 22.7 perhaps it starts rising, and then it touches 23.6, which is greater than the threshold set by the condition. Moment it crosses that condition, again you get 2.05 success code, content, observe is 21. Same token, this is important request ID and the response IDs are matching. And the condition is payload is set to 23.6.

So this way, one can also improve on the basic stack by using this observe in a very interesting manner. And yet, sort of you know use this in energy battery constrained devices. And that is why CoAP indeed is a very powerful protocol, which one can depend upon. So that is about this efficiently observing IoT resources, I encourage you to read up and understand it better.

But before we close on this particular topic, I am sure you have one confusion in your mind. And that confusion is what I am going to try to tell you, okay.

(Refer Slide Time: 26:37)



	COAP	MQTT
Connection	Fast	Slow - 3 way handshake,
Packet Losses - low	Fast	Slow
" - High	Slow	Faster.
Energy/Battery	Excellent	Good.
Setup	Good.	Extremely Simple !

SYN
SYN+Ack
ACK
RTO → adjustment
Sensor data - 16 bits
Q → MQTT longer more
CoAP longer More.

So let us see, when should I use CoAP and when should I use MQTT? I am sure you have this problem. I am very sure, okay. First thing is CoAP is fast. Because it uses UDP. It is not heavy like TCP, SYN SYN+, ACK and so on. Whereas this is a bit I would say it is a bit slow, okay. Three way handshake is required. Three way handshake is required and then you can start transmitting data, right?

Three way handshake comes from TCP, TCP-SYN, SYN+ACK and then ACK. So this is SYN. This is SYN+ACK. And this is ACK. This three way handshake should happen. Only then data can start flowing from here, okay. So therefore, this delay is there. There can be a significant amount of delay, setup delay before it can actually start transmitting. So initial delay can be high.

So that is what I mean by fast and slow initial delay. But during passage of time, let us say you have packet losses. You have packet losses, okay. Low, low packet losses. You can have packet losses which are high, okay. Now I will remove it from here. This will be fast. Parameter here is I would say connection. Connection is fast, connection is slow. The second parameter is packet losses.

Packet losses, I will say low packet losses and packet losses high packet losses. See if there are low packet losses okay 1% or less than 1% and so on, you are better off by using this is fast here. You are much better off by using, this can again be slow, okay. But if the packet losses are high, then I think this gets slow. And this may be much faster because it knows how to work around large packet losses.

It does retransmissions and knows how to manage. Based on the packet loss it may know how to increment the timeout timers, RTO okay retransmission timeout timers, and so on. It is a sophisticated stack, right TCP. All that part RTO adjustment is very much possible by TCP. Whereas these things do not really exist in the UDP world. So folks, all I can say is the choice of CoAP and MQTT largely depends on the application.

But I had just given you some numbers on how it can be used. More than you know bandwidth and packet losses and so on, some of the features in CoAP are outstanding for energy and battery related options, okay. Energy battery I will call it here. Energy/battery, okay. This I would say is excellent. This is good I will say. I would not give it bad marks, I will say it is good.

Same thing is that energy bar slash battery or lifetime systems and all so many levers and hooks are available in CoAP, which is quite natural. But whereas with MQTT, there are not so many options, there are options also. But I think it is much more amenable in CoAP.

For example, if you are having a lot of sensor data, and you want to have timer based interrupts of sensor data, which we discussed in the last class, I think CoAP and timer based interrupts will work excellently well, they gel very well, right? And you look at the observe feature, you can do based on certain observations every time it increases. There is a change in a value, it can be reported in a trivial way.

And you can improve that by putting thresholds. You can not only put one, but you can put multiple thresholds and improve it even more. So lot of things you can do with CoAP. And of course, it is not that MQTT is very simple, extremely simple to set up, right? I would say if someone gives it some marks for how simple it is extremely simple setup, I will simply say setup.

This is extremely simple and amenable, available everywhere. This is okay. I mean, you can still use a browser, or you can build your own CoAP system. I would say it is simply it is good, okay. Setup is okay. It is not so bad either. So you can go on listing like this folks, and then arrive at your own decision of what would be an efficient protocol to run with.

For a sensor data, which is 16 bytes, no 16 bits let us say, bytes is too much, okay. For a sensor data of 16 bits question is, does MQTT transfer more or CoAP transfers more, it is not clear. So you may have to calculate based on the header length, and all that, and then put down a number. And then see for yourself how good is for sending 2 bytes of data, whether it is good to send over CoAP or over MQTT.

And if the wireless link is of varying of one type, whether this is better, or the other one is better, and so on. So I think this discussion will help you articulate it better. I urge you to read the RFC documents very carefully, because lot of stuff is out there. And we cannot cover everything there. Thank you very much.