Internet of Things
Computer Science and Engineering
Politecnico di Milano

Third Homework

Erfan Rahnemoon 10720184 - 943057

22-03-2020

Summary

[*Answering some questions related to an analysis of the sniffed packets by Wireshark.*]

Questions ans Answers

0.1 What's the difference between the message with MID: 3978 and the one with MID: 22636?

We filter both packets with filter,

$$coap.mid == 3978 || coap.mid == 22636$$

and then we could show that the main difference is that the packet with MID=3978 is a confirmable packet that will have acknowledgment, but the message with MID=22636 is non-confirmable, so there is no ACK for it. Also, the value of SZX is different in both packets, which will determine the size of the packet.

0.2 Does the client receive the response of message No.6949?

Yes, if we get to the message with NO.6949, then we could filter with the MID of this message.

$$coap.mid == 28357$$

So we find the ACK packet of this message in the NO.6953.

0.3 How many replies of type confirmable and result code "Content" are received by the server "localhost"?

By applying the following filter, we could see the number of packets is eight packets. The coap.type is set to 2 to find replies to confirmable type, and coap.code is set to 69 to find responses with a type of "content".

$$(coap.type == 2)\&\&(coap.code == 69)\&\&(ip.dst == 127.0.0.1)$$

No.	Time	Source	Destination	Protocol	Length	Info	srcport	dstport	mqtt_len
90	9.704166705	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:63229, 2.05 Content, TKN:11 fd c7 72, End of Block #0, /living…	5683	46292	
1047	23.625323733	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:4920, 2.05 Content, TKN:b6 3d cc 6d, End of Block #0, /living_…	5683	42585	
1337	36.639686319	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:23246, 2.05 Content, TKN:e8 cf 79 21, End of Block #0, /living…	5683	34439	
2124	41.646753209	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:13240, 2.05 Content, TKN:47 a9 45 65, End of Block #0, /living…	5683	52638	
2537	52.663764603	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:29961, 2.05 Content, TKN:6d cf 30 dd, End of Block #0, /living…	5683	50928	
2673	58.668864968	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:25273, 2.05 Content, TKN:b6 69 52 f1, End of Block #0, /living…	5683	34822	
2921	81.689542540	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:48882, 2.05 Content, TKN:b5 cc 8e 5b, End of Block #0, /living…	5683	34970	
3055	90.691363524	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:21099, 2.05 Content, TKN:f8 eb 78 0b, End of Block #0, /living…	5683	57330	

0.4 How many messages containing the topic “factory/department*/+” are published by a client with user name: “jane”? Where * replaces the dep. number, e.g. factory/department1/+, factory/department2/+ and so on. (btw, * is NOT an MQTT wildcard)

First, we find the packets that contain the username of "jane" to find the IP and port of the client used by "jane".

$$mqtt.username == "jane"$$

Then we use the ports that we found from the previous filter to find all the packets sent by "jane".

$$tcp.srcport == 42821 || tcp.srcport == 40989 || tcp.srcport == 40005 || tcp.srcport == 50985$$

Finally, we update the previous filter to have the packets with the topic of "factory/department" and the type of published data that we should set mqtt.msgtype to three. Accordingly, the number of messages is six.

$$mqtt.msgtype == 3 \&\& (tcp.srcport == 42821 || tcp.srcport == 40989 || tcp.srcport == 40005 ||$$

$$tcp.srcport == 50985) \&\& mqtt.topiccontains" factory/department"$$

No.	Time	Source	Destination	Protocol	Length	Info	srcport	dstport	mqtt_len
1584	39.396671378	127.0.0.1	127.0.0.1	MQTT	200	Publish Message (id=4) [factory/department2/section4/plc]	42821	1883	129
1629	39.401658935	127.0.0.1	127.0.0.1	MQTT	210	Publish Message (id=1) [factory/department2/section1/hydraulic_valve]	40989	1883	139
2540	53.418595851	127.0.0.1	127.0.0.1	MQTT	213	Publish Message (id=2) [factory/department2/section1/hydraulic_valve]	40989	1883	142
2863	78.402142281	127.0.0.1	127.0.0.1	MQTT	194	Publish Message (id=5) [factory/department2/section4/plc]	42821	1883	124
3132	98.453437832	127.0.0.1	127.0.0.1	MQTT	207	Publish Message (id=3) [factory/department2/section1/hydraulic_valve]	40989	1883	136
5830	122.397452291	127.0.0.1	127.0.0.1	MQTT	209	Publish Message (id=2) [factory/department1/section3/hydraulic_valve]	50985	1883	138

0.5 How many clients connected to the broker “hivemq” have specified a will message?

First, we filter packets with the following filter to find the DNS records related to “hivemq”.

$$frame\ contains\ hivemq$$

Then, we use these IP addresses to find all the massages send for this broker.

$$ip.dst == 18.185.199.22 || ip.dst == 3.120.68.56$$

Next, we update the filter to find the messages with active Will flag; additionally, for being sure we have will_mesaage in all of the messages, we could add mqtt.willmsg to the filter, but generally, the Will flag is enough.

$$(ip.dst == 18.185.199.22 || ip.dst == 3.120.68.56) \&\& mqtt.conflag.willflag == 1 \&\& mqtt.willmsg"$$

Eventually, if we assume the client as one physical device with one IP address in the network, the number of connected clients is one. However, if we assume the client a program that is running on the physical machine, we will have 16 connected clients to the broker because we have 16 different source ports for the messages.

0.6 How many publishes with QoS 1 don't receive the ACK?

To get the number of published messages with QoS set to one, we use the following filter that mqtt.msgtype is set to three, which means published packets. So the number of all published massages with QoS of one is 124.

$$(mqtt.msgtype == 3)\&\&(mqtt.qos == 1)$$

Then we use the following filter to get the number of PUBACK (acknowledgment of publication packets) of publication with QoS of one, which is equal to 74 messages. The mqtt.msgtype is set to four because in MQTT specification for acknowledgment of Publication packets with QoS of one, the number 4 is used.

$$mqtt.msgtype == 4$$

As we know, the QoS of one means at least once delivery so that some ACK may be duplicated, but by looking at port and ID of ACKs, we know that all of them are unique. Consequently, 124-74=50 publication packets are without ACK.

0.7 How many last will messages with QoS set to 0 are actually delivered?

In the beginning, we find messages with active Will flag and the Qos of zero by the following filter, which the number of them is equal to 19.

$$mqtt.conflag.willflag == 1\&\&mqtt.conflag.qos == 0$$

Now, by knowing the port and IP of the packets, we found in the first part, we filter ACK Connect messages by considering the QoS of zero means at most once delivery. We apply the next filter, and by the help of vertical line In the No. column, IPs, and ports, we find the ACK of each one and mark them. Also, the mqtt.msgtype equal to one means connect command and equal to two means the ACK of the command.

$$mqtt.msgtype == 2||mqtt.msgtype == 1$$

Finally, if we assume the three malformed ACK Connect packets are not acceptable, so we are sure 16 of these messages are delivered.

mqtt.msgtype==1 && frame.marked==1

No.	Time	Source	Destination	Protocol	Length	Info	srcport	dstport	mqtt_len
538	21.167779680	127.0.0.1	127.0.0.1	MQTT	181	Connect Command	43907	1883	111
544	21.167980418	10.0.2.15	3.120.68.56	MQTT	133	Connect Command	47047	1883	75
693	21.199355426	10.0.2.15	5.196.95.208	MQTT	133	Connect Command	43559	1883	75
879	21.291103737	10.0.2.15	137.135.83.217	MQTT	127	Connect Command	43003	1883	69
1515	39.383836950	127.0.0.1	127.0.0.1	MQTT	169	Connect Command	60395	1883	99
2304	44.459288018	10.0.2.15	3.120.68.56	MQTT	125	Connect Command	49919	1883	67
2481	49.447970967	10.0.2.15	18.185.199.22	MQTT	127	Connect Command	40971	1883	69
3395	106.660752524	127.0.0.1	127.0.0.1	MQTT	181	Connect Command	36795	1883	111
3508	106.670878457	127.0.0.1	127.0.0.1	MQTT	147	Connect Command	55633	1883	77
3769	106.696299808	127.0.0.1	127.0.0.1	MQTT	170	Connect Command	33201	1883	100
3784	106.696833547	127.0.0.1	127.0.0.1	MQTT	153	Connect Command	56313	1883	83
4045	106.817119274	10.0.2.15	137.135.83.217	MQTT	133	Connect Command	36463	1883	75
4409	109.301171949	127.0.0.1	127.0.0.1	MQTT	151	Connect Command	60419	1883	81
4608	109.324085356	127.0.0.1	127.0.0.1	MQTT	148	Connect Command	49041	1883	78
4681	109.334623694	10.0.2.15	18.185.199.22	MQTT	121	Connect Command	36453	1883	63
4774	109.360199401	10.0.2.15	3.120.68.56	MQTT	157	Connect Command	50573	1883	99
5916	122.405482925	127.0.0.1	127.0.0.1	MQTT	153	Connect Command	56927	1883	83
6169	122.501895148	10.0.2.15	137.135.83.217	MQTT	149	Connect Command	58167	1883	91
6551	127.474044621	10.0.2.15	5.196.95.208	MQTT	133	Connect Command	34841	1883	75

mqtt.msgtype==2 && frame.marked==1

No.	Time	Source	Destination	Protocol	Length	Info	srcport	dstport	mqtt_len
540	21.167880560	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack	1883	43907	2
671	21.193396238	3.120.68.56	10.0.2.15	MQTT	62	Connect Ack	1883	47047	2
798	21.226481199	5.196.95.208	10.0.2.15	MQTT	62	Connect Ack	1883	43559	2
1022	21.401251730	137.135.83.217	10.0.2.15	MQTT	113	Connect Ack	1883	43003	55
1517	39.383904896	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack	1883	60395	2
2363	44.478912453	3.120.68.56	10.0.2.15	MQTT	62	Connect Ack	1883	49919	2
2501	49.475368032	18.185.199.22	10.0.2.15	MQTT	114	Connect Ack	1883	40971	56
3397	106.660783845	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack	1883	36795	2
3510	106.670940511	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack[Malformed Packet]	1883	55633	2
3771	106.696420309	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack	1883	33201	2
3794	106.697079222	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack	1883	56313	2
4079	106.926559171	137.135.83.217	10.0.2.15	MQTT	62	Connect Ack	1883	36463	2
4411	109.301247325	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack	1883	60419	2
4610	109.324157164	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack[Malformed Packet]	1883	49041	2
4747	109.355170307	18.185.199.22	10.0.2.15	MQTT	62	Connect Ack	1883	36453	2
4792	109.406276455	3.120.68.56	10.0.2.15	MQTT	62	Connect Ack	1883	50573	2
5918	122.405564022	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack[Malformed Packet]	1883	56927	2
6223	122.615527301	137.135.83.217	10.0.2.15	MQTT	62	Connect Ack	1883	58167	2
6581	127.500079366	5.196.95.208	10.0.2.15	MQTT	62	Connect Ack	1883	34841	2

0.8 Are all the messages with QoS > 0 published by the client "4m3DWYzWr40pce6OaBQAfk" correctly delivered to the subscribers?

First, we find the client's IP and ports by the next filter by using the client ID.

$$mqtt.clientid == 4m3DWYzWr40pce6OaBQAfk$$

Then we filter the messages send by the client which have the QoS of bigger than zero and the port and IP we found in the first part.

$$ip.src == 10.0.2.15\&\&tcp.srcport == 58313\&\&mqtt.qos > 0$$

After, we apply the following filter.(the mqtt.msgtype is set to five to find the Publish Received packets)

$$mqtt.msgtype == 5\&\&tcp.dstport == 58313$$

We find only one Publish Received packet, so the subscriber receives the packet with No. 2423(Distinguishing packets by ID), but the receiver does not receive the packet with No. 968 correctly because there was no ACK for it even though the QoS is bigger than zero.

ip.src==10.0.2.15 && tcp.srcport==58313 && mqtt.qos > 0

No.	Time	Source	Destination	Protocol	Length	Info	srcport	dstport	mqtt_len
968	21.367816800	10.0.2.15	5.196.95.208	MQTT	99	Subscribe Request (id=1) [factory/department2/section3/deposit]	58313	1883	41
2423	45.369441197	10.0.2.15	5.196.95.208	MQTT	193	Publish Message (id=3) [factory/department1/section1/deposit]	58313	1883	134

mqtt.msgtype == 5 && tcp.dstport==58313

No.	Time	Source	Destination	Protocol	Length	Info	srcport	dstport	mqtt_len
2425	45.395228239	5.196.95.208	10.0.2.15	MQTT	62	Publish Received (id=3)	1883	58313	2

0.9 What is the average message length of a connect msg using mqttv5 protocol? Why messages have different size?

We use the following filter to find all the MQTT Connect messages that the version is five.

$$(mqtt.ver == 5)\&\&(mqtt.msgtype == 1)$$

Then we use the mqtt.len as one column of Wireshark to find the massage length of each one and then calculate the average message length, which is 30.22. (In each multiplication, the first number is the number of the packet, and the second number is the length of the massage.)

$$((35 * 13) + (1 * 20) + (5 * 25) + (1 * 27) + (1 * 29) + (2 * 30)+$$

$$(2*32)+(1*33)+(4*65)+(4*69)+(4*77)+(1*78)+$$
$$(1*83)+(1*86))/63=30.222$$

The size of the message, depending on which flags are set, will vary a lot. For example, by setting the flag of Will the size of the packet could increase significantly by considering the size of the Will Topic and Will message. Also, the username and password flag could increase the size of the packet depending on the size of the username and password field, and client ID also increases the message length.

0.10 Why there aren't any REQ/RESP pings in the pcap?

MQTT works on the TCP, and one of the TCP problems is the half-open connection to get around this problem. MQTT defines a field, which called keep-alive. Keep alive ensures that the connection between the broker and client is still open and that the broker and the client are aware of being connected.

The keep-alive is the duration in which the client and broker can send no message from the last transmitted message, and connection will be assumed open by both sides. In the absence of any message, the client and broker need control procedure to keep each other in check. This procedure is done by REQ/RESP pings.

For no REQ/RESP, there are two possibilities; first, the procedure is deactivated by setting the keep-alive to zero, which by checking the messages, we know that the keep-alive value is not zero. Second, As long as messages are exchanged frequently, and the keep-alive interval is not exceeded, there is no need to send an extra message to establish whether the connection is still open. The second possibility is the scenario that is happing in this sniffing, especially the duration of sniffing, is less than most of the keep-alive durations. Also, of the frequent exchange of messages between broker and clients, we did not have the REQ/RESP.