ISSN 2395-1621

Comparative Study of Messaging Protocols for The Internet of Things

Mr. Rushikesh Shrikant Vathare
*Research Scholar, Assistant Professor, Department of E&TC Engineering, Arvind Gavali College of Engineering, Panmalewadi, Satara-*415015 *Maharashtra, India.*
Dr. Ganashree T.S.
Associate Professor, Department of E&TC Engineering Dayananda Sagar College of Engineering, Kumaraswamy, Bangalore- 560078 Karnataka, India.
Prof. Dr. ShrinivasA. Patil
Professor, HOD, Department of Electronics & Telecommunication Engineering D.K.T.E. Society's Textile & Engineering Institute, Ichalkaranji – 416115 Kolhapur, Maharashtra, India
Mr. Niraj Bhupal Kapase
Assistant Professor, Department of Electronics Engineering D.K.T.E. Society's Textile & Engineering Institute, Ichalkaranji – 416115 Kolhapur, Maharashtra, India
Ms. Sayali Sadashiv Parve
*Assistant Professor, Department of E&TC Engineering, Arvind Gavali College of Engineering, Panmalewadi, Satara-*415015 *Maharashtra, India.*

ABSTRACT

This Internet of Things (IoT) consist of lightweight devices with low power, and short-ranged wireless communication. The things can communicate with each other to form a network. In IoT, broadcast transmission is widely used along with the maximum usage of wireless networks and their applications. Hence, it has become crucial to authenticate broadcast messages. Several key management schemes have been introduced, and their benefits are not recognized in a specific IOT application. Security services are vital for ensuring the integrity, authenticity, and confidentiality of the critical information. Therefore, the authentication mechanisms are required to support these security services and to be resilient to distinct attacks. Various authentication protocols such as MQTT[6], CoAP[11], HTTP[8] and XMPP[7] lightweight authentication protocols, and broadcast authentication protocols are compared and analyzed for all secure transmission applications. The major goal of this survey is to compare and find out the appropriate protocol such as MQTT[6], CoAP[11], HTTP[8] and XMPP[7] for further research. Moreover, the comparisons between various authentication techniques are also illustrated.

Keywords: Internet of Things Protocol comparison, MQTT, CoAP, HTTP and XMPP.

ARTICLE INFO

Article History

Received: 25th March 2017

Received in revised form :

25th March 2017

Accepted: 25th March 2017

Published online :

4th May 2017

I. INTRODUCTION

We are not the only ones interested in comparing the IoT messaging protocols. Vasileios Karagiannis and his colleagues from Thessaloniki, Greece recently published a paper called "A Survey on Application Layer Protocols for the Internet of Things".
There is (at least) the choice of these popular protocols:

- HTTP/REST
- CoAP
- XMPP
- AMQP
- MQTT
- DDS

They nearly have the same list of protocols namely CoAP[11], MQTT[6], XMPP[7], AMQP[1] and HTTP/REST. The only difference is that they left out DDS and added Web sockets as another alternative.
They came out with the following difference between the protocols:

a. Transport: UDP-based CoAP is preferable in this category followed by all the other TCP-based alternatives and HTTP[8]-based REST at the end
b. QoS options: Only MQTT[6], AMQP[1] and CoAP[11] offer QoS options for the transport. For CoAP[11] this is questionable as the confirmable attribute is just used to do what TCP already does for all other protocols. So real QoS levels are only provided by AMQP[1] and MQTT[6] in my opinion.
c. Architecture: This is an important point. Do the protocols rely on request/response (REST, CoAP), or on publish/subscribe (MQTT[6], AMQP[1]) or do they support both (XMPP, Web sockets)?
d. Security: It is argued that DTLS (CoAP) offers lower security than TLS/SSL (all other protocols). Especially, DTLS breaks some of the benefits of UDP.
e. Extensibility: It is just mentioned shortly in the section on XMPP[7]. This is a quite important point in my opinion. Here, XMPP with its XML-based approach is the clear winner. This comes at the cost for high message overhead and XML parsing.

CoAP[11] also supports basic pub sub through its Observe option. Thus a client can issue a GET request to a smart object which ends up in a subscription to updates. MQTT[6] also has a UDP-based variant MQTT-SN[11] while I think it is not widely distributed. For XMPP[7] one should also mention EXI as an alternative encoding reducing the XML overhead. An important issue missing in the comparison is the discovery aspect. CoAP is the winner of this category as it provides a well-known URI to access meta information about sensors

MQTT -

MQ Telemetry Transport (MQTT) is an open source protocol for constrained devices and low-bandwidth, high-latency networks. It is a publish/subscribe messaging transport that is extremely lightweight and ideal for connecting small devices to constrained networks.

MQTT[6] is bandwidth efficient, data agnostic, and has continuous session awareness. It helps minimize the resource requirements for your IoT device, while also attempting to ensure reliability and some degree of assurance of delivery with grades of service.

MQTT targets large networks of small devices that need to be monitored or controlled from a back-end server on the Internet. It is not designed for device-to-device transfer. Nor is it designed to "multicast" data to many receivers. MQTT[6] is extremely simple, offering few control options.

A performance test using MQTT

The task was to implement a scenario for the MQTT[6] protocol to measure half round-trip times and overhead of the protocols involved to the data transmitted actually. The payload is synthetic being at sizes of 10bytes, 100bytes and 1000bytes.

The protocol transmits binary data, but I decided to transmit my data packed into JSON which involves another overhead that will not be part of the math (as I count it as information). Due to JSON the minimal payload I can send will be 28 (13 digits for the timestamp + 9 characters for "timestamp" + "{" + ":"). The message is published under a specific topic which will not be counted into the actual payload but as part of the MQTT[6] protocol ("data/synthetic" in this case, which are 14bytes). Furthermore, there are 5bytes that seems to be part of the message ID or so. I used Wireshark in order to inspect the localhost network interface to gather the values.

- 10bytes
  Full TCP packet: 113 bytes, TCP data: 47 bytes, message: 47-14-5 = 28 bytes
- 100bytes
  Full TCP packet: 198 bytes, TCP data: 132 bytes, message: 132-14-5 = 113 bytes
- 1000bytes
  Full TCP packet: 1098 bytes, TCP data: 1032 bytes, message: 1032-14-5 = 1013 bytes

This means, the overhead of

A. MQTT[6] to actual message,
B. TCP to the full MQTT packet,
C. The full TCP packet to the actual message

is shown in the following table:

Size	A	B	C
10bytes	40,4%	58,4%	75,2%
100bytes	14,4%	33,3%	42,9%
1000bytes	1,8%	6,0%	7,7%

Table 1: Overhead of protocols

This means, upon requesting 10bytes payload there will be 75,2% overhead of the whole packet arrived just for the message. I did not count ACK packages and packages involving keep-alive, connection setup and shutdown etc. which will add a lot more. Part of the task was measuring the actual messages that arrive per second (mps). I decided to measure messages when the client was running on the same computer as well as the client was running on the Raspberry Pi in the same network connected via LAN over a Fritzbox.

Size requested	Raspberry Pi (in LAN)	Local on same PC
10bytes	192mps	2902mps
100bytes	226mps	2850mps
1000bytes	202mps	2226mps

Table 2: Measurement of actual messages that arrive per second (mps)

As you can see the message rate degredates upon payload size increase. Somehow the amount of messages transmitted by 10bytes payload size is less than the other values on the Raspberry Pi. The overall amount of messages is much more less than I expected it to be. I profiled my Java client and looked for places where a lot of data is allocated. 59,3% of all allocation happened in the *org.eclipse.paho.client.mqttv3. MqttClientpublish* method which is used in order to publish a message as client to a given topic on a given server. Another 11.3% of all allocation happens in the javax.json implementation of *javax.json.JsoncreateWriter* which is created on every message sent in order to transform the Java objects into

JSON objects. So there seem to be problems upstream that need to be fixed in order to increase the application performance.

Characteristics of MQTT

- Very simple and light protocol on top of TCP.
- Good fit for wireless applications.
- Publish/Subscribe paradigm.
- Web socket support

MQTT On the wire messages:

- connect (with or without authentication)
- publish/puback
- subscribe/suback
- ping/pingack for keepalive
- disconnect

CoAP

Although Web protocols are available and usable for IoT devices, they are too heavy for the majority of IoT applications. The Constrained Application Protocol (CoAP) was designed by the IETF for use with low-power and constrained networks. CoAP[11] is a RESTful protocol. It is semantically aligned with HTTP[8], and even has a one-to-one mapping to and from HTTP[8].

CoAP[11] is a good choice of protocol for devices operating on battery or energy harvesting. Some features of CoAP[11]:

- CoAP uses UDP.
- Because CoAP[11] uses UDP, some of the TCP functions are reproduced in CoAP. For example, distinguishes between confirmable (requiring an acknowledgement) and non-confirmable messages.
- Requests and responses are exchanged asynchronously over CoAP[11] messages.
- All the headers, methods and status codes are binary encoded, which reduces the protocol overhead.
- Unlike HTTP[8], the ability to cache CoAP[11] responses does not depend on the request method, but the Response Code.

CoAP[11] fully addresses the need for an extremely lightweight protocol and the ability for a permanent connection. And if you have a Web background, using CoAP is relatively easy. The internet of things is developing quickly. Really quickly! After notebooks and smartphones, people now try to connect almost every device over the internet somehow. This brings great advantage in terms of features, but also leads us into some very big trouble: The known technologies do not fit for various situations any more. One of these problems is the communication from small devices over constrained networks. That's where CoAP[11] comes in place. The constrained application protocol is an application layer protocol, which is designed to be easy to use and to only have small overhead. The IETF designed CoAP[11] as a RESTful protocol with multicast and observe support. The core features are specified in the RFC 7252. CoAP[11] is using architecture similar to HTTP/REST, where you can access different resources using the GET/PUT/POST/DELETE/ methods on the coap-uri. An example would be: A smart thermometer is connected to your network and you could access the temperature using CoAP[11] like this: GET coap://localhost:5683/sensor.

Characteristics of CoAP

- Simple to encode: targets 8 bits MCU.
- UDP based, targets low power IP networks.
- Two level of QoS: confirmable message or not.
- Simple observation mechanism

Various libraries of CoAP

After reading a lot about CoAP, I wanted to try and build something with it. The first step was to go and look through the various libraries out there and decide which ones to use. Some of the most known libraries are Libcoap as a C-Library and Californium (JAVA), which is distributed by the Eclipse foundation. As parts of our project are done in NodeJS, I decided to have a look on a small library called node-coap as well. Apart from all libraries, there is a firefox plug-in which called Copper. It is super easy to use and perfect for developing and testing with CoAP[11], as it let's you send all kinds of requests "by hand", and directly shows you what kind of response you get

HTTP

HTTP[8] is the foundation of the client-server model used for the Web. The more secure method to implement HTTP is to include only a client in your IoT device, not a server. In other words, it is safer to build an IoT device that can only initiate connections, not receive. After all, you do not want to allow outside access to your local network

Before i tested the speed on the Raspberry Pi, i tried to run the client local on the same machine as the server. First i got around 2000 messages per secound and the information were buffert a long time on the server and arrived the webclient later and later. That endet after i stopped do give the chunk of each package on the console on the iot-server. After that my implementation got speeds around 20k messages per secound (with 20 Byte per message). 20 Bytes are my smalest kind of package, because i use "utf-8" encoding and so every Letter has a size of 1Byte. The message is only a json object with the timestamp: {"sT":1421139314965}. Selecting the 10byte on the webclient my code sending 20-byte-messages. On bigger choosen sizes i just add a "payload" with random letters to the json-object. Now i let the pi send the stream to the iot-server. There we found following speeds:

20B – 2223mps ~ 49ms – message to overhead: 0
100B – 1995mps ~ 46ms – message to overhead: 0,8
1000B – 1761mps ~ 45ms – message to overhead: 0,98

The biggest difference to simulating the problem on localhost the messagespersecound-rates are a kind of slower. In my oppinion that depends on the speed of the pi, because the messages are generated in a while-loop and can't be generated much faster. The times needn't be correct, because i haven't timesynced it correctly yet. The message to overhead i calculated in following way: Message to overhead = (size-20Byte)/size. I minimized the times from the server to the web client because i used them on the same machine and there wasn't a considerable value in delay.

Comparative Analysis and Summary

The table below contains a summary of the IoT protocol landscape

Protocol	**MQTT**	**CoAP**	**HTTP**
Transport	TCP	UDP	TCP
Messaging	Publish/Subscribe Request/Response	Request/Response	Request/Response
2G, 3G, 4G Suitability (1000s nodes)	Excellent	Excellent	Excellent
LLN Suitability (1000s nodes)	Fair	Excellent	Fair
Compute Resources	10Ks RAM/Flash	10Ks RAM/Flash	10Ks RAM/Flash
Success Stories	Extending enterprise messaging into IoT applications	Utility Field Area Networks	Smart Energy Profile 2 (premise energy management, home services)

Table 3: Comparative analysis of IoT protocols

How to choose protocol for IoT scenario
• How constrained are the devices?
• Reliable/unreliable network?
• What is the message rate?
• How is the data processed further?
• Push or Pull?

MQTT

No built in result command path support needed to define a result topic pattern (over) addressing result (req_id) in custom payload, if device is offline

- no TTL (Time To Live) for command
- old command could be delivered (if "retain" flag)
- new command could be lost (if not "retain" flag)
- commands are enqueued only if not "clean session"

CoAP

Message response after a while pattern addressing problem (mobile roaming, NAT, ...) QoS with "confirmable" message or not

HTTP

more verbose (ASCII, headers, ...) for few data addressing problem (mobile roaming, NAT, ...) no QoS (based on TCP)

Conclusion

Every processor and every application need to be configured, upgraded and monitored. Device management is not an option. Each protocol must be secured. And synchronized: You can't trigger an update with a protocol; while you are rebooting the device using another M2M/IoT is not a simple problem. Security and provisioning are really the hardest ones. Try hard to reduce the number of protocols to make your life easier. CoAP[11] with LWM2M can provide a light device management and application protocol to rule them all

But CoAP is still a newcomer in the field and not aone size fits all solution. Let's specify device management on top of MQTT[6]

Challenges for research in IoT Protocols

1. Constrained devices
2. Bi-directional communication
3. Scaling to 100.000s of devices
4. Unreliable networks
5. Push messaging
6. Security

References

[1] Benjamin Aziz, A Formal Model and Analysis of an IoT Protocol, Ad Hoc Networks ADHOC1234 S1570-8705(15)00118-3 May (2015)

[2] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645-1660.

[3] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A Survey, Comput. Netw. 54 (15) (2010) 2787-2805.23

[4] D. Bandyopadhyay, J. Sen, Internet of Things: Applications and Challenges in Technology and Standardization, Wireless Personal Communications 58 (1) (2011)

[5] O. Vermesan, P. Friess, Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, The River Publishers, 2013.

[6] O. Mazhelis, H. Warma, S. Leminen, P. Ahokangas, P. Pussinen, M. Rajahonka, R. Siuruainen, H. Okkonen, A. Shveykovskiy, J. Myllykoski, Internet-of-Things Market, Value Networks, and Business Models: State of the Art Report, Tech. Rep. TR-39 (2013).

[7] D. Locke, MQ Telemetry Transport (MQTT) V3.1 Protocol Speci_cation (2010).

[8] P. Saint-Andre, K. Smith, R. Tronon, XMPP: The De_nitive Guide Building Real-Time Applications with Jabber Technologies, O'Reilly Media, Inc., 2009.

[9] SenWang, Jun Bi, JianpingWu, Xu Yang, and Lingyuan Fan. On adapting HTTP protocol to content centric networking. In Proceedings of the 7th International Conference on Future Internet Technologies, pages 1–6.

[10] ACM, 2012. K. Birman, T. Joseph, Exploiting Virtual Synchrony in Distributed Systems, SIGOPS Oper. Syst. Rev. 21 (5) (1987) 123-138

[11] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, Information and Computation 100(1) (1992) 1-77.

[12] Dejana Ugrenovic, Gordana Gardasevic CoAP protocol for Web-based monitoring in IoT healthcare applications, 23rd Telecommunication Forum TELFOR 2015, Serbia, Belgrade, Nov 24-26,2015

[13] A. J. Stanford-Clark, G. R. Wightwick, The Application of Publish/Subscribe Messaging to Environmental, Monitoring, and Control Systems, IBM J. Res. Dev. 54 (4) (2010) 396-402.

[14] U. Hunkeler, H. L. Truong, A. Stanford-Clark, MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks, in: Proceedings of the Third International Conference on Communication System software and Middleware (COMSWARE 2008), IEEE, 2008, pp. 791-798.24

[15] R. Baldoni, M. Contenti, S. T. Piergiovanni, A. Virgillito, Modelling Publish/Subscribe Communication Systems: Towards a Formal Approach, in: 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003), IEEE Computer Society, 2003, pp. 304-311.

[16] L. Abidi, C. Cerin, S. Evangelista, A Petri-Net Model for the Publish-Subscribe Paradigm and Its Application for the Verication of the Bonjour Grid Middleware, in: Proceedings of the 2011 IEEE International Conference on Services Computing, SCC '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 496-503.

[17] C. Wang, A. Carzaniga, D. Evans, A. Wolf, Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems, in: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9 - Volume 9, HICSS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 303-307

[18] D. Garlan, S. Khersonsky, J. S. Kim, Model checking publish-subscribe systems, in: Proceedings of the 10th International Conference on Model Checking Software, SPIN'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp.166-180

[19] M. H. ter Beek, M. Massink, D. Latella, S. Gnesi, A. Forghieri, M. Sebastianis, Model checking publish/subscribe noti_cation for thinkteam &174;,Electron. Notes Theor. Comput. Sci. 133 (2005) 275-294.

[20] Y. Jia, E. L. Bodanese, C. I. Phillips, J. Bigham, R. Tao, Improved reliability of large scale publish/subscribe based moms using model checking, in: 2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014, IEEE, 2014, pp. 1-8.

[21] L. Baresi, C. Ghezzi, L. Mottola, On accurate automatic veri_cation of publish-subscribe architectures, in: Proceedings of the 29th International Conference on Software Engineering, ICSE '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 199-208.

[22] F. He, L. Baresi, C. Ghezzi, P. Spoletini, Formal analysis of publish/subscribe systems by probabilistic timed automata, in: Formal Techniques for Networked and Distributed Systems - FORTE 2007, 27th IFIP WG 6.1International Conference, Tallinn, Estonia, June 27-29, 2007, Proceedings, Vol. 4574, Springer, 2007, pp. 247-262.

[23] S. Gallotti, C. Ghezzi, R. Mirandola, G. Tamburrelli, Quality prediction of service compositions through probabilistic model checking, in: Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures, QoSA '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 119-134.

[24] A. Fehnker, L. V. Hoesel, A. Mader, Modelling and Veri_cation of the LMAC Protocol for Wireless Sensor Networks, in: Proceedings of the 6th International Conference on Integrated Formal Methods, IFM'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 253-272.

[25] A. Fehnker, P. Gao, Formal Veri_cation and Simulation for Performance Analysis for Probabilistic Broadcast Protocols, in: Proceedings of the 5th International Conference on Ad-Hoc, Mobile, and Wireless Networks, ADHOC-NOW'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 128-141

[26] F. Heidarian, J. Schmaltz, F. W. Vaandrager, Analysis of a clock synchronization protocol for wireless sensor networks, Theor. Comput. Sci. 413 (1)(2012) 87-105.

[27] B. Aziz, A formal model and analysis of the mq telemetry transport protocol, in:9th International Conference on Availability, Reliability and Security (ARES 2014), Fribourg, Switzerland, IEEE, 2014.

[28] A. Banks, R. Gupta, MQ Telemetry Transport (MQTT) V3.1.1 Protocol Speci_cation: Committee Speci_cation Draft 02 / Public Review Draft 02 (2014)

