Comparison of IoT Data Protocol Overhead

Vasil Sarafov
Advisor: M.Sc. Jan Seeger
Seminar Future Internet SS2017
Chair of Network Architectures and Services
Departments of Informatics, Technical University of Munich
Email: sarafov@cs.tum.edu

ABSTRACT

The Internet of Things is expanding at fast pace and every year constrained devices that rely on intercommunication are being deployed. Knowing how much overhead a communication mechanism adds to the system can be of huge importance for its optimal utilization and prevent performance degradation. In this paper we construct an abstract theoretical model for deriving and comparing the overhead of the WebSocket, CoAP and MQTT protocols when sending upstream an arbitrary number of data packets. We then validate the end results with an experiment and show that CoAP with non-confirmable messages demonstrates the least overhead when no datagrams are lost, followed by MQTT with QoS 0, which outperforms the WebSocket protocol by a tiny margin.

Keywords

Protocol overhead, Throughput, Performance Comparison, CoAP, MQTT, WebSocket, IoT, Internet of Things, Data Protocols

1. INTRODUCTION

For decades people have been involved in a technological revolution, which has opened a new chapter in human history. Starting from science and education and continuing with important industrial and medical applications, the Internet has been accelerating the world's progress for years. Even more, it has now become one of the most important communication media.

The tremendous advance in the fields of electronics, robotics and artificial intelligence has led to the next stage of this revolution - bringing the *Internet* to a new state where it can be used for interconnecting *Things* and *Machines* that are not operated by people and communicate autonomously with one another. This phase was given the name *The Internet of Things* (IoT).

One of the core concepts of IoT is exactly the communication between interconnected devices. In most cases the connected nodes are operating in constrained environments and have very limited resources such as CPU power, RAM and available energy. Therefore an optimal communication mechanism is a must. Calculating the application protocol overhead is a step towards such optimization and in this paper we compare the overhead of three data protocols that are widely used in various IoT services - WebSocket, Constrained Application Protocol (CoAP) and MQTT.

The WebSocket Protocol was designed as a solution to the problem which the Web was facing for many years - the lack of full duplex asynchronous communication between a client and a server. Since WebSocket allows that and is fairly easy to integrate in existing HTTP infrastructure, many IoT devices and platforms choose to implement it.

CoAP's aim is to be for the IoT world what HTTP is for the Web. It brings the well known REST model [8] to networks with constrained nodes by relying on UDP and very small protocol overhead.

MQTT is a protocol that follows the publish-subscribe communication pattern and provides a very convenient decoupling of the peers in large distributed systems that can be used to optimize complex business logic. It was designed to be light, highly scalable and easy to adjust and implement on the client side, which makes it a perfect fit for the IoT domain.

The paper is structured as follows. In Chapter 2 we present a very simple mathematical model that is used in Chapter 3 to compare WebSocket, CoAP and MQTT, where the protocols are described in a more detailed manner as well. We validate our theoretical estimations with an experiment in Chapter 4 and make the overall conclusion in Chapter 6. In addition, we explain how the proposed model can be extended so that it can be utilized for estimating the protocol overhead in more complex real-world applications.

Related Work

Existing work compares different lightweight internet protocols mainly based on the provided feature set. Performance evaluation is usually done only empirically and for a very specific use case. In this paper we provide a general theoretical overhead comparison that is in addition experimentally validated. Its results can be applied for an arbitrary scenario, independently of the physical connection.

In [20] Thangavel et al design and implement a middleware for wireless sensor networks. By using the common middleware they empirically evaluate the performance of CoAP and MQTT. [12] compares CoAP's and MQTT's performance in NBIoT networks, emphasizing the adaptation of both data protocols in regard to the limitations of the NBIoT physical properties. [7] provides a qualitative and quantitative comparison between MQTT and CoAP for smartphone-based sensor systems. The quantitative analysis is executed in a

doi: 10.2313/NET-2018-03-1_02

WiFi-based network for a publish/subscribe use case and therefore utilizes CoAP with its observe extension. Very similarly, [2] discusses the feature differences between the same two protocols and benchmarks them based on performance criteria such as overhead and energy consumption. In [21] Yokotami and Sasaki measure the performance differences between HTTP and MQTT in a publish/subscribe scenario. Their comparison is based on bandwidth and server resource allocation costs for up to 1000 connected IoT devices.

A very detailed performance examination of CoAP, MQTT and DDS (Data Distribution Service for real-time systems) is presented in [5] by Chen and Kunz. They quantitatively compare the protocols' bandwidth consumption, experienced latency, packet loss and operation in low quality wireless networks in terms of a medical IoT use case.

2. MATHEMATICAL MODEL

In this section we present a very short and simplified mathematical model for deriving the overhead of an arbitrary data communication protocol in an abstract form. Our goal is to apply it directly to the protocols described in Chapter 3 and compare the end results.

Considering the fact that our aim is to compare the overhead of data communication protocols for the *IoT application domain*, we make the following observations and assumptions, which will help us simplify our model:

- We are interested only in the size overhead of the protocol (how much additional control information is needed to send x amount of application data). This is of huge importance for constrained devices, such as IoT nodes, because every additional byte being processed implies a higher energy consumption.
- Latency is not taken into consideration since it is strongly dependent on the physical properties of the underlying network. We focus on *application data protocols* which can be utilized in networks with different physical attributes.
- Following the OSI layering model [4], data protocols have the same overhead for the Physical (L1), Data Link (L2) and Network Layers (L3) when they are used in the same environment. Therefore we are interested in comparing the overhead differences starting with the Transport Layer (L4).
- For the sake of simplicity we assume that *no* IP fragmentation is taking place which holds true in many constrained networks [19, Section 4.6]. This assumption implies a payload size of less than 1024 Bytes for the data protocols that were chosen, which is further justified in Chapter 3.2.
- Whether or not the application data is compressed is completely irrelevant to the model. This is because the payload is represented by its total size in the model calculations.
- A typical IoT use case is when a device sends data (e.g. sensor values) to a server or a gateway in a particular time slot (also known as a communication slot) after it has established connection. In the remaining time it would normally process time critical tasks or simply enter a deep sleep mode to save energy. For the sake of simplicity but still evaluating a realistic use case, we restrict ourselves only to upstream communication and examine each protocol in a scenario where the client (an IoT device) is the data source and the server is the data collector.

Parameter	Description	Constraints
p	Sum of the size of the headers from Layer 1, 2 and 3 that are present in every sent packet.	$p \in \mathbb{N}$
b	Indicates whether the data protocol has an opening and closing handshake ($b = 1$) or not ($b = 0$).	$b \in \{0, 1\}$
H_o	Size of the opening handshake in Bytes (Layer 4 upwards).	$H_o \in \mathbb{N}_0$
H_c	Size of the closing handshake in Bytes (Layer 4 upwards).	$H_c \in \mathbb{N}_0$
n	Number of the available communication slots until the connection is closed.	$n \in \mathbb{N}$
x_i	Size of the application data in Bytes that is to be sent in communication slot i.	$x_i \in \mathbb{N}_0, \leq 1024$
h_i	Size of the data protocol header in Bytes that is needed to wrap x_i.	$h_i \in \mathbb{N}$

Table 1: Description of the parameters used in the mathematical model for evaluating the overhead of an arbitrary data protocol.

In Table 1 are presented the parameters which describe our abstract data protocol model.

An important observation is that for CoAP [19, Section 3] and MQTT [1, Section 3.3] the headers that wrap the application data have a constant size. Hence for both protocols holds:

$$h_i = h_j, \forall i, j \in \{1, 2...n\}$$

For the WebSocket protocol the headers that wrap the application data differ with at most 2 Bytes in size [13, Section 5.2]. Therefore for WebSocket holds:

$$|h_i - h_j| \leq 2, \forall i, j \in \{1, 2...n\}$$

Considering the above observations and for simplification reasons, with negligible error we assume a constant header size:

$$h := h_i = h_j, \forall i, j \in \{1, 2...n\}$$

Furthermore, we define D_{app} as the total amount of appli-

doi: 10.2313/NET-2018-03-1_02

cation data that will be sent in a single connection:

$$D_{app} := \sum_{i=1}^{n} x_i = n\widetilde{x} \tag{1}$$

where $\widetilde{x}$ is the average of all x_i.

Analogously we obtain the total amount of data (including all L1-L7 overheads) that is sent when D_{app} is dispatched as D_{total}:

$$\begin{aligned} D_{total} := & \underbrace{b(c_1 p + H_o)}_{\text{Opening Handshake}} + \underbrace{\sum_{i=1}^{n}(p + h_i + x_i)}_{\text{Sending of } D_{app}} + \underbrace{b(c_2 p + H_c)}_{\text{Closing Handshake}} \\ & = \underbrace{bp(c_1 + c_2) + np}_{\text{L1-L3 overhead}} + \underbrace{bH_o + bH_c + nh}_{\text{L4-L7 overhead}} + \underbrace{n\widetilde{x}}_{D_{app}} \end{aligned} \tag{2}$$

where c_1 and c_2 are the number of packets that are needed to complete the opening and closing handshake respectively.

Hence the actual data protocol overhead for n communication slots is given by the overhead sum between Layer 4 and Layer 7:

$$\omega(n) := bH_o + bH_c + nh \tag{3}$$

Furthermore, for the protocol's throughput τ we obtain:

$$\begin{aligned} \tau(n, \widetilde{x}) &= \frac{D_{app}}{D_{total}} \\ &= \frac{n\widetilde{x}}{\omega(n) + n\widetilde{x} + bp(c_1 + c_2) + np} \end{aligned} \tag{4}$$

Obviously $\forall n, x \in \mathbb{N}.\tau(n, \widetilde{x}) < 1$ and $\tau(n, \widetilde{x})$ is a strongly monotone increasing function. We use this observation in Chapter 4.

3. THEORETICAL PROTOCOL OVERHEAD ESTIMATION

In this section we apply the mathematical model described in Chapter 2 for the WebSocket, CoAP and MQTT protocols respectively. We do this by giving the exact Layer 4 to Layer 7 costs and approximating where necessary. Encryption layering with Transport Layer Security (TLS) [15] for WebSocket and MQTT and Datagram Transport Layer Security (DTLS) [16] for CoAP is omitted for the sake of simplicity.

Furthermore proxy and cache optimizations, which can in many cases boost the performance of the communication system, are also not taken into consideration. The reason for this assumption is that proxy usage is very often tightly coupled to the application's logic and therefore is not suitable for our abstract evaluation framework.

3.1 The WebSocket Protocol

WebSocket is a data protocol that exposes TCP on a higher abstraction level so that it can be used almost directly by Web browser applications. It was standardized by the IETF in 2011 with RFC 6455 [13].

The motivation behind its design was to solve the lack of asynchronous communication from server to client in HTTP and provide a long living full duplex connection that can be integrated in existing HTTP infrastructure [18], making it suitable for IoT use cases where a real time bidirectional interaction with the device is desired. The compatibility in terms of deployment between the two completely different protocols is achieved via the HTTP upgrade header which notifies the server to change the protocol from HTTP to WebSocket.

A WebSocket-based communication stack is layered on top of TCP. Therefore the client and server should not take care of data fragmentation or packet acknowledgement. Similarly to HTTP, a WebSocket connection can be secured with TLS. In most cases the real application protocol (defined as subprotocol in [13, Section 1.9]) is layered directly over the WebSocket, which means that WebSocket is used only for the connection and does not create any constraints to the business logic of the system.

The life cycle of a non-TLS secured WebSocket connection is shown on Figure 1. It consists of:

- A 3-way TCP opening handshake that costs approximately 60 Bytes[1].
- A 2-way WebSocket opening handshake. Its size can vary based on the connection meta information such as the endpoint and server hostname [13, Section 1.3]. A good approximation is $\approx$ 310 Bytes (Client side $\approx$ 170 Bytes, Server side $\approx$ 140 Bytes).
- A routine for sending the actual application data over WebSocket. Each packet that is not buffered by the sender (i.e it is sent directly) is wrapped in a frame of total size 12 or 14 Bytes. Frames are used instead of a direct streaming approach in order to prevent mandatory buffering and to allow dynamically adjustable multiplexing of the duplex communication in future versions of the protocol [13, Section 5.2]. Furthermore, very often resource-constrained IoT devices cannot support a buffering mechanism because of lack of enough memory.
- A 2-way WebSocket closing handshake, which consists of two empty frames, 14 Bytes each [13, Section 1.4]. Thus it has a total size of 28 Bytes.
- A 3-way TCP closing handshake that costs approximately 60 Bytes.

Hence we obtain together with the TCP message and acknowledgement wrapping of the WebSocket messages the approximated values for our overhead estimation model in Table 2.

From that we directly derive the final representation of the WebSocket data protocol overhead, using Equation 3:

$$\omega_{WebSocket}(n) \approx 600 + 54n \tag{5}$$

[1]We assume that the TCP header size is 20 Bytes (TCP header options are not considered)

doi: 10.2313/NET-2018-03-1_02

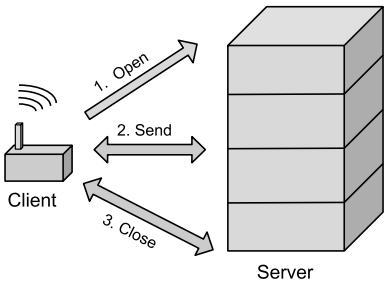


Figure 1: Life cycle of a bidirectional WebSocket connection

Parameter	Value
b	1
H_o	≈ 430
H_c	168
h	54

Table 2: Model values for the WebSocket protocol

3.2 Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a web transfer protocol that follows the request-response communication pattern (Figure 2) and is intended for use with constrained nodes and networks in machine-to-machine (M2M) applications. The protocol was officially standardized by the IETF in June 2014 with RFC 7257 [19].

CoAP's general purpose is to allow a subset of the convenient REST architecture that HTTP provides for the Web [8] to be utilized by applications running on microcontrollers. Typically, microcontrollers have limited hardware resources such as memory, CPU power and energy and often communicate through constrained networks such as 6LoWPAN [14]. This makes the direct utilization of REST with the full HTTP stack hard. Nevertheless, CoAP can be further extended to support the observe communication pattern by executing augmented GET requests [9].

To provide flexibility and address the constrained nature of the nodes, CoAP is layered over UDP. Using TCP instead of UDP is theoretically possible but not standardized. Additional layering over DTLS secures the connection [16].

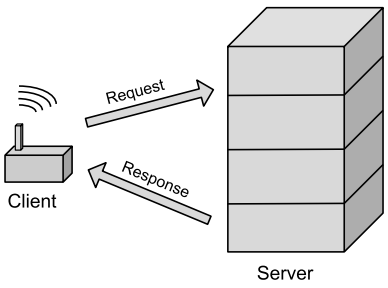


Figure 2: The CoAP client requests resources from the server by specifying a Request Verb and an endpoint. The server responds accordingly with the requested resource.

Parameter	Value	Note
b	0	No handshake
H_o	0	No handshake
H_c	0	No handshake
h	38	non-confirmable request with non-confirmable response
h	50	non-confirmable request with confirmable response
h	38	confirmable request with piggybacked response
h	50	confirmable request with separate non-confirmable response
h	62	confirmable request, separate confirmable response

Table 3: Model values for the CoAP protocol and all request/response message types (upstream communication only)

CoAP deals with the lack of order and unreliable transmission of UDP datagrams by utilizing a very simple messaging layer over the actual request/response. Thus important and time-critical packets can be acknowledged which is a very simple form of an adjustable quality of service (QoS) [19, Section 4]. A CoAP request can be in either confirmable or in non-confirmable message. In case of a confirmable request, the server must send an acknowledgement message. This message can either contain the response (called piggybacked response) or be empty (the response is sent separately). Following the same strategy, separate responses can be wrapped inside either confirmable or non-confirmable messages [19, Section 4.2].

CoAP is intended to be used without IP fragmentation. Hence problems can occur when larger amounts of data need to be transferred (example: Over-the-Air firmware updates). Therefore RFC 7959 proposes a blockwise transfer technique for CoAP messages [3]. However, a payload size of 1024 Bytes can be assumed to be transported without any fragmentation through a normal not constrained network [19, Section 4.6].

The CoAP request/response header has a fixed size of 11 Bytes when cache is disabled. For our estimation we use a 4 Bytes long token to match requests with responses which is the half of the allowed length [19, Section 3]. An acknowledgement message costs exactly 4 Bytes and the underlying UDP header is 8 Bytes long. With this information and the assumption that the application data is transferred by the client in an upstream manner (for example via POST requests) and the server's response does not contain any payload but only control information such as status code, we obtain the protocol overhead for all 5 different possibilities of the request/response life cycle (Table 3).

Thus we can conclude:

$$38n \leq \omega_{CoAP}(n) \leq 62n \tag{6}$$

doi: 10.2313/NET-2018-03-1_02

3.3 MQTT

The MQTT protocol is a client-server publish-subscribe messaging transport protocol. It was created in 1999 because of a use case where a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over satellite connection was needed [10]. The current version is 3.1.1, which as of 2014 is an OASIS and as of 2016 an ISO standard [1, 11].

MQTT is designed with simplicity in mind. It is lightweight and similarly to CoAP is suitable for machine-to-machine communication in constrained environments, where the code footprint and network bandwidth are scarce. It is layered over TCP and can be secured with TLS. In general, MQTT can function over an arbitrary transport protocol that provides ordered, lossless bi-directional connections. Hence it can be even layered on top of The WebSocket Protocol (Chapter 3.1), which makes possible writing browser-based MQTT clients.

Although the protocol is TCP based, it provides three different levels for Quality of Service (QoS) for the message delivery [1, Section 4.3]:

- QoS 0 where messages are assured to arrive at most once, hence can be lost when connection problems occur. In most cases QoS 0 is enough since MQTT can take advantage of TCP's connection reliability mechanisms.
- QoS 1 where messages are assured to arrive but duplicates can occur. This level requires a 2-way handshake for each sent message.
- QoS 2 where messages are assured to arrive exactly once. This level and QoS 1 are intended for systems where TCP's mechanisms are not enough - for example transaction systems or services that are unreliably interconnected on the physical level such as satellites. QoS 2 requires a 4-way handshake for each sent message.

QoS is important because it allows controlling the protocol overhead and thus the network can manage its quality alone when bandwidth or connection problems occur. Furthermore, QoS levels 1 and 2 imply a session which makes MQTT stateful when needed.

An MQTT connection is *exactly one* of the following two types:

1. Publishing Connection - A client connects to the server (also known as message broker) to publish multiple messages during the lifetime of the connection. (Figure 3).
2. Subscription Connection - A client connects to the broker and subscribes to message topics. During the lifetime of the connection it receives the publications of other clients which are forwarded by the broker.

Generally speaking, the message broker acts as an intermediary between the publishers and subscribers and makes

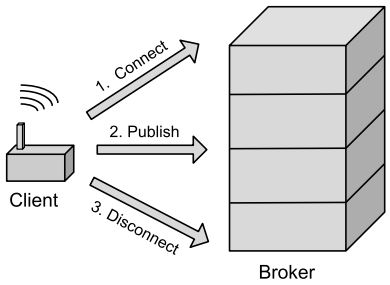


Figure 3: Connection life cycle of a client publishing information to an MQTT broker

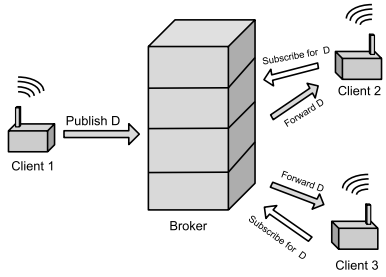


Figure 4: MQTT based publish-subscribe communication between peers

sure that the messages are delivered, based on the QoS (Figure 4). Without loss of generality, we evaluate the *publishing connection* since the *subscription connection* is identical and provides exactly the same overhead. Moreover, we are not taking into consideration the in-built username/password mechanism that MQTT provides because TLS layering is omitted. From a security point of view sending the authentication credentials in plain text makes no sense and in Chapters 3.1 and 3.2 we have evaluated the WebSocket and CoAP protocols respectively without any authentication mechanism.

We take the TCP overhead into consideration as described in Chapter 3.1. The MQTT fixed header is exactly 2 Bytes long [1, Section 2.2] and considering the message exchange algorithms described in [1, Section 3], we present in Table 4 the values for our overhead estimation model. With that said we conclude the lower (QoS 0) and upper (QoS 2)

Parameter	Value	Note
b	1	Handshake present
H_o	≈ 200	TCP and MQTT opening handshake with Will message
H_c	102	TCP and MQTT closing handshake
h	42	QoS 0, no authentication
h	86	QoS 1, no authentication
h	174	QoS 2, no authentication

Table 4: Model values for the MQTT protocol for all QoS message types (publishing connection only)

doi: 10.2313/NET-2018-03-1_02

bounds for the overhead of an MQTT-based communication:

$$300 + 42n \leq \omega_{MQTT}(n) \leq 300 + 174n \quad (7)$$

4. EXPERIMENTAL PROTOCOL OVERHEAD COMPARISON

We now present an empirical validation of the protocol overhead estimations that were made in Chapter 3.

4.1 Structure of the Experiment

The experiment was conducted in a local WiFi network with IPv4-based addressing and the following hardware:

- Client - Raspberry PI model B (Quad Core 1.2GHz Broadcom BCM283, 1GB RAM) running Ubuntu Core Linux 16 with kernel version 4.4.0-72-generic.
- Server - Laptop with Quad Core CPU Intel i7-2620M 3.4GHz, 8GB RAM, running Fedora Linux with kernel version 4.12.14-300.fc26.x86_64.

The total amount of bytes for each execution was measured with Wireshark [6]. When needed, packet loss was simulated on the client side using the Linux kernel module netem[2]. Further information about the used software components and the code for the clients and servers can be found in Appendix A.

4.2 Experiment Results

We use the throughput equation (Equation 4) we derived in Chapter 2 to compare the protocols. Moreover we also know from the same equation that the following statement holds for arbitrary protocols A and B:

$$\omega_A(n) < \omega_B(n) \Rightarrow \tau_A(n, \widetilde{x}) > \tau_B(n, \widetilde{x})$$

Thus from equations 5, 6 and 7 by comparing the estimated constants, we expect that CoAP with non-confirmable requests and responses will perform at best, then MQTT with QoS 0 and the WebSocket protocol will share the second and third place.

On Figure 5 we can see the experiment results for up to 100 communication slots, i.e. up to 100 data packets were sent through the lifetime of a single connection, and average application data size of 128 Bytes with no packet loss. For the MQTT and CoAP protocols we have used those protocol configurations, which showed the lower and upper overhead bounds respectively according to equations 6 and 7. Those are QoS 0 and QoS 2 for MQTT (Chapter 3.3) and non-confirmable requests/responses and confirmable requests with separate confirmable responses for CoAP (Chapter 3.2).

On Figure 6 we can see the results for the same experiment configuration but with a simulated 20% packet loss on the client side. Important to note is that packet losses are independently distributed, i.e. networking burst was not considered. Furthermore, non-confirmable CoAP messages are not retransmitted and thus completely lost when datagrams are dropped. This explains the unusually higher throughput values for the non-confirmable CoAP scenario.

Figure 5: $\tau(n, \widetilde{x})$ for $\widetilde{x} = 128$ and no packet loss

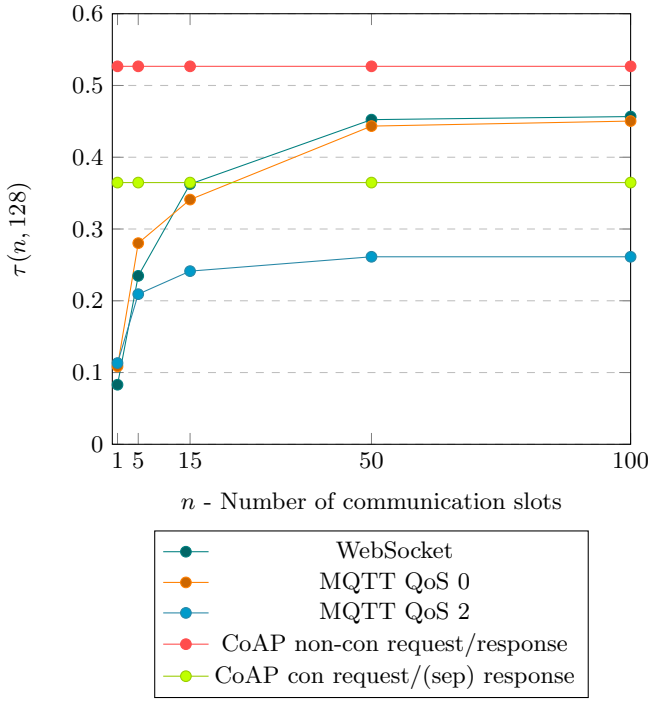
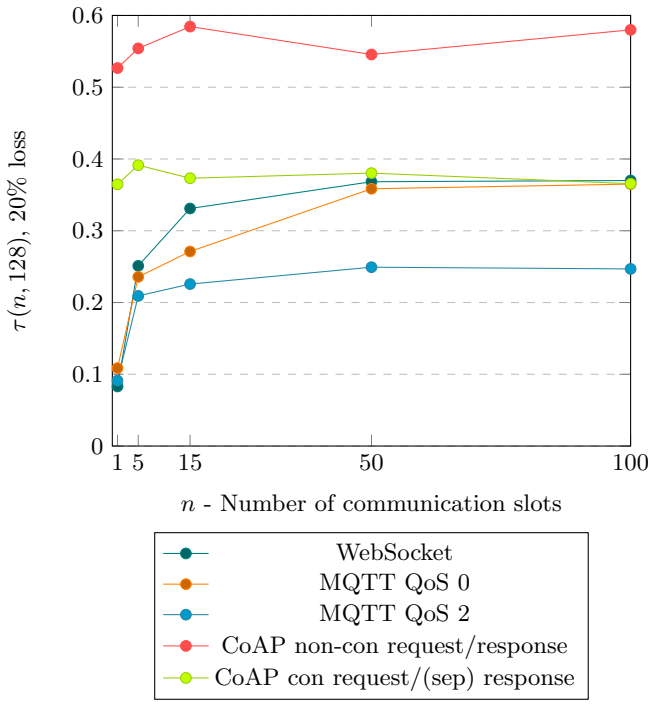


Figure 6: $\tau(n, \widetilde{x})$ for $\widetilde{x} = 128$ with 20% packet loss



doi: 10.2313/NET-2018-03-1_02

As expected, CoAP has a constant throughput when no datagrams are lost since it lacks an opening/closing handshakes. On the other hand, MQTT and WebSocket start with a poor throughput because of the handshake costs but outperform CoAP with confirmable requests and separate confirmable responses after approximately 15 communication slots (Figure 5). However, this tendency does not hold true when packets are lost. In that case CoAP with confirmable messages outperforms both TCP data protocols although all three of them converge to the same throughput rate for larger number of communication slots (Figure 6). In the general case, MQTT with QoS 2 performs at worst because of the large number of control messages that are exchanged between the client and the server. As a second result of that its overall throughput grows much slower than the one of its QoS 0 companion but remains almost the same when packets are lost.

The experiment was executed two more times with the same amounts of communication slots and payload sizes of 64 and 512 Bytes respectively, without packet loss. An important observation is that MQTT QoS 0 outperforms WebSocket faster with bigger payloads and all protocols achieve a more than two times better throughput when the average application data size is increased from 64 to 512 Bytes. In the latter case CoAP's best throughput jumps from 38% to 83%. MQTT with QoS 0 improves its throughput from 29% to 77%.

As it can be observed in all cases of the experiment execution, our theoretical estimations in Chapter 3 are correct and CoAP with non-confirmable requests/responses demonstrates the best throughput, followed by WebSocket and MQTT with QoS 0, which share the second place.

5. FURTHER WORK

Important assumptions were made in Chapter 3 in order to simplify the theoretical overhead estimation model. Not considering TLS securing for WebSocket and MQTT and DTLS securing for CoAP is a huge drawback, although the same results as in Chpater 4.2 are expected. Additionally taking into account cache and proxy optimization best-practices will surely complicate the model but also allow its direct utilization in real-world applications.

An important application layer protocol for the IoT domain which we did not discuss is the Extensible Messaging and Presence Protocol (XMPP) [17]. Comparing the many different IoT extensions for XMPP to CoAP and MQTT will be of a huge benefit, considering the fact that XMPP is heavily used in IoT platforms and services.

6. CONCLUSION

Knowing how much overhead a data protocol generates is of huge importance in cases where the networking nodes and the network itself are resource-constrained. IoT devices face many hardware limitations and as a result of that they should implement suitable communication protocols.

We showed that the theoretical overhead of CoAP with non-confirmable requests and responses is the least when compared to MQTT and WebSocket for the same amount of requests and data sent upstream, using an identical physical connection layer. Utilizing CoAP in this configuration, however, comes with the disadvantage of a higher probability of loosing packets. Furthermore, CoAP provides a constant throughput when no datagrams are lost, which does not always hold true in constrained networks.

MQTT with QoS 0 demonstrates the second best overhead. Compared against CoAP, it relies on TCP and hence can handle lost packets on the Transport Layer. Moreover, MQTT QoS 0 has a better throughput than CoAP with confirmable requests and separate confirmable responses, when no packet loss occurs. Otherwise CoAP's reliable configuration performs better because of the lighter UDP overhead. Using MQTT QoS 2 on the other hand generates too much overhead and causes the protocol to perform at worst. This is albeit understandable, considering the fact that MQTT QoS 2 was designed for satellite networks where TCP is unreliable.

WebSocket demonstrates almost the same throughput as MQTT with QoS 0. What both of these protocols share in common is the additional cost that the peers must pay to open the connection. As the connection is kept alive and data is being sent, the initialization costs start paying off and the maximum achievable throughput is reached.

All three protocols increase their throughput when more application data is sent in a single dispatch. Rising the average data packet size from 64 to 512 Bytes improves the overall throughput more than two times.

The above results compare the three protocols *only based on their overhead*. This, however, is not applicable for use cases where the application's business logic is tightly coupled to a very specific type of connection. CoAP provides a request/response mechanism which makes it suitable for general use cases that do not rely on a long living connection. On the other hand achieving a full duplex asynchronous communication between the client and the server with CoAP is impossible. If this is desired, WebSocket is to be considered. When an efficient mass distribution of data to interested parties in the system is wanted, MQTT should be chosen.

7. REFERENCES

[1] MQTT Version 3.1.1 Plus Errata 01, Dec. 2010.

[2] S. Bandyopadhyay and A. Bhattacharyya. Lightweight internet protocols for web enablement of sensors using constrained gateway devices. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 334–340. IEEE, 2013.

[3] C. Bormann and Z. Shelby. Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959, Aug. 2016.

[4] N. Briscoe. Understanding the OSI 7-layer model. *PC Network Advisor*, 120(2), 2000.

[5] Y. Chen and T. Kunz. Performance evaluation of iot protocols under a constrained wireless access network. In *Selected Topics in Mobile & Wireless Networking*

[2]https://wiki.linuxfoundation.org/networking/netem

(MoWNeT), 2016 International Conference on, pages 1–7. IEEE, 2016.

[6] G. Combs et al. Wireshark. *Web page: http://www. wireshark. org/last modified*, pages 12–02, 2007.

[7] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali. Comparison of two lightweight protocols for smartphone-based sensing. In *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on*, pages 1–6. IEEE, 2013.

[8] R. T. Fielding and R. N. Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[9] K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, Sept. 2015.

[10] E. B. HiveMQ. MQTT Essentials: Part 1 Introducing MQTT. http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt, 2015.

[11] Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1. Standard, International Organization for Standardization, Geneva, CH, June 2016.

[12] T. Maksymyuk, M. Brych, S. Dumych, and H. Al-Zayadi. Comparison of the iot transport protocols performance over narrowband-iot networks. *Internet of Things and Ubiquitous Communications*, 1(1):25–28, 2017.

[13] A. Melnikov and I. Fette. The WebSocket Protocol. RFC 6455, Dec. 2011.

[14] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, Sept. 2007.

[15] E. Rescorla and T. Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008.

[16] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, Jan. 2012.

[17] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120, Mar. 2011.

[18] P. Saint-Andre, S. Loreto, S. Salsano, and G. Wilkins. Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. RFC 6202, Apr. 2011.

[19] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[20] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan. Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE, 2014.

[21] T. Yokotani and Y. Sasaki. Comparison with http and mqtt on required network resources for iot. In *Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016 International Conference on*, pages 1–6. IEEE, 2016.

APPENDIX

A. ADDITIONAL MATERIALS

The experiment discussed in Chapter 4 is fully reproducible. All code used to perform it together with the latest version of the paper itself are available at https://gitlab.com/v45k0/iot-data-protocols.