Article

Dynamic Session Layer Selection in IoT Actuation

Rolando Herrero

College of Engineering, Northeastern University, Boston, MA 02115-5000, USA
E-mail: rolando.herrero@northeastern.edu

Received: 18 April 2023; **Revised:** 2 July 2023; **Accepted:** 13 July 2023

Abstract: While most Internet of Things (IoT) solutions involve sensing, some of them also introduce actuation mechanisms. Specifically, devices interact with assets in the environment and transmit sensor readouts to applications that perform analytics. These applications typically reside on the network core and, in turn, process the readouts that trigger the transmission of actuation commands to the device. One important issue in these schemes is the nature of the communication channels. Most devices are wireless and therefore they are affected by the effects of signal multipath fading that results in application loss. More importantly, these impairments may cause actuation commands to be lost or to be critically delayed. In this context, several standard mechanisms have been proposed for the transmission of traffic from the application to the devices. They fall under two main architectural categories: (1) Representational State Transfer (REST) and (2) Event Driven Architecture (EDA). In this paper, we analyze two protocols associated with each of these two architectures by comparing them in order to assess their efficiency in IoT actuation solutions. This analysis leads to the development of a mathematical model that enables the dynamic selection of the right technology based on network impairments.

Keywords: actuation, MQTT, CoAP, EDA, REST, IoT, broker

1. Introduction

IoT architectures follow a fix topology that involves devices interacting with applications. The devices support sensing and actuation while the applications perform analytics, generically through Artificial Intelligence (AI) algorithms. Machine Learning (ML) classification is a field of AI that provides several algorithms to support predictions that can be used to make actuation decisions [1].

While devices reside on the access side of the network, the application is, in most cases, on the core side in the context of a cloud computing scenario. Between devices and applications, there is a network edge where an edge device like a router or gateway translates protocols. Access networks are characterized by low transmission rates associated with constrained devices and physical layer technologies. Core networks, on the other hand, rely on traditional Internet protocols and, therefore, support higher transmission rates. Devices observe the environment through sensing and affect the environment by means of actuation. To minimize latency and improve the Round Trip Time (RTT) between sensing and actuation, applications sometime reside on the edge or in the access network itself. Edge and access applications are respectively representative of fog and mist computing scenarios.

REST scenarios follow the Create, Read, Update, Delete (CRUD) paradigm where devices are created and deleted, sensor readouts are read and actuation commands are updated [2]. Under REST an endpoint transmits a request that triggers a response on the far end. In the context of this research work, the application transmits an actuation request to the device such that the device performs the actuation and transmits back a response. EDA scenarios rely on a broker that reliably forwards messages between actuator drivers and applications. In this scenario, the application sends an actuation command to the broker, the broker then forwards the command to the device. EDA as opposed to REST, does not require the device to transmit an end-to-end message to acknowledge the reception of the command. EDA relies on lower layer protocols to support reliable date transmission between endpoints.

In this paper we examine the performance of actuation mechanisms under REST and EDA scenarios. The focus on actuation has to do with the fact that actuation is associated with transmission of commands from the application down to the devices. Compare this with sensing where readouts are transmitted from devices to the application. In the context of this paper, we look at two different protocols: Hyper Text Transfer Protocol (HTTP) to support REST and Message Queue Telemetry Transport (MQTT) to support EDA [3-5]. First, we present the different session flows and then we introduce a mathematical model to evaluate the performance of each mechanism. This model is then used as the driver of an algorithm that enables the determination of the best protocol given network conditions. The algorithm is then evaluated in an experimental framework.

The following is a list of the contributions of this research work:

- A mathematical model that provides a way to estimate session layer impairments for REST and EDA architecture.
- An algorithm that relies on the mathematical model to enable the dynamic switching between different session layer protocols to improve the overall system performance.
- An experimental framework to evaluate the proposed algorithm.

The remaining sections of this paper are the following: Motivation and a literature review are presented in Section 2. In Section 3, we look at the message flows associated with actuation in the context of HTTP and MQTT protocols. A mathematical model and an algorithm for dynamic protocol selection are presented in Section 4. In Section 5, an experimental framework to evaluate the aforementioned algorithm. Finally, in Section 6 conclusions and future work are discussed.

2. Motivation and Literature Review

The main motivation for the research presented in this paper is to find a mechanism that optimizes the propagation of actuation commands from applications to devices. The biggest challenge is minimizing the end-to-end latency to expedite the transmission of such commands. This is critical in Real-Time Communication (RTC) IoT scenarios where high latency implies low Quality of Service (QoS). For example, consider a case where an application transmits a flight path to an Unmanned Aerial Vehicle (UAV) that results from processing through ML readouts coming from on-board sensors. In this context, any excessive latency becomes a liability that can lead not only to failure but also to physical and personal damage [6]. Because most IoT devices are constrained in nature, another very important requirement is for the algorithm to be computationally simple and flexible enough that can be deployed in as many scenarios as possible.

The use of IoT networking protocols to propagate messages between applications and devices is a topic of extensive research. When considering REST architectures, in addition to HTTP, the Constrained Application Protocol (CoAP) is one of the preferred mechanisms to support the transmission of device messages. In this context, the performance and the effects of latency, loss, throughput and other impairments on the application under both CoAP and HTTP are evaluated in constrained IoT networks in [7]. In [8], CoAP, HTTP, IEC 61850 and other protocols are compared from a perspective of performance. Similarly, several congestion control mechanisms are introduced in [9] to improve end-to-end latency in the context of network layer impairments. In [10], HTTP session management drives the support of REST notifications to enable an IoT motion detection application. The performance of web enabled actuators that rely on HTTP sessions is explored in [11]. REST HTTP actuation in the context of industrial IoT architectures is analyzed in [12]. Because REST protocols are independent of the lower layers, in [13] the support of HTTP and CoAP is taken into account in the context of different transport layers.

From the perspective of the EDA architecture, several applications of MQTT have been studied. In [14] the authors introduce an indoor localization solution in an EDA scheme. Similarly, an architecture to support automated traffic light control in Smart City EDA topologies is presented in [15]. A detailed list of interop issues associated with MQTT, as well as common applications are explored in [16]. Loss and latency in MQTT scenarios are analyzed in [7]. The performance of MQTT with respect to other protocols, not only EDA but also REST ones, is presented in [17]. A voice controlled IoT system that relies on MQTT messaging is introduced in [18]. An MQTT-based robotic system that is used to mimic human hand movement is presented in [19]. An aquaponics system that supports actuation and sensing in MQTT sessions is analyzed in [20]. In [21], a building automation architecture that relies on a LoRa based MQTT protocol stack is introduced. The effects on network resources in MQTT and HTTP scenarios is presented in [22]. This analysis, however, does not address the performance differences of these two mechanisms. In [23], the authors propose an architecture that allows MQTT brokers to cooperate and share their data with other interested MQTT brokers. Similarly, in [24], the authors evaluate the resource consumption of state-of-the-art fuzzing frameworks, thereby understanding the degree to which brokers are tested before deployment. A comparison of the characteristics of MQTT, HTTP and secure HTTP (HTTPS) is presented in [25]. Table 1 summarizes each paper and list of protocols that each reference analysis.

Table 1. Summary of Related Work

Reference	Protocols under analysis
Reusing Web-Enabled Actuators from a Semantic Space-Based Perspective [11]	HTTP
Internet of Things application layer protocol analysis over error and delay prone links [9]	CoAP, MQTT
Performance evaluation of IoT protocols under a constrained wireless access network [7]	CoAP, HTTP
Validation of a CoAP to IEC 61850 Mapping and Benchmarking vs HTTP-REST and WS-SOAP [8]	CoAP, HTTP
RESTful Motion Detection and Notification using IoT [10]	HTTP
A REST and HTTP-based Service Architecture for Industrial Facilities [12]	HTTP
Analysis of the constrained application protocol over quick UDP internet connection transport [13]	CoAP, HTTP
Application of Fingerprint-based Indoor Localization System Using IEEE 802.15.4 to Two-Floors Environment [14]	EDA generic
IoT-Based Urban Traffic-Light Control: Modeling, Prototyping and Evaluation of MQTT Protocol [15]	MQTT
Internet of Things: Survey and open issues of MQTT protocol [16]	MQTT
Performance evaluation of IoT protocols under a constrained wireless access network [7]	MQTT
Analysis of the constrained application protocol over quick UDP internet connection transport [17]	MQTT
The voice controlled Internet of Things system [18]	MQTT
An IoT based wireless robotic-hand actuation system for mimicking human hand movement [19]	MQTT
Data Acquisition and Actuation for Aquaponics using IoT [20]	MQTT
A LoRa enabled building automation architecture based on MQTT [21]	MQTT
Comparison with HTTP and MQTT on required network resources for IoT [22]	MQTT, HTTP
Secure Data Distribution Architecture in IoT Using MQTT [23]	MQTT
Resource-Intensive Fuzzing for MQTT Brokers: State of the Art, Performance Evaluation, and Open Issues [24]	MQTT
Performance Comparison of HTTP, HTTPS, and MQTT for IoT Applications [25]	MQTT, HTTP

Note that none of these papers addresses the difference between REST and EDA topologies to enable the transmission of actuation commands. Moreover, none of them provides an algorithm, like the one introduced in Section 4, that enables to dynamically select between REST and EDA architectures to maximize the system performance.

3. Actuation Message Flows

This Section introduces the different message flows associated with the actuation in the context of IoT that are relevant to this research work.

The framework presented in this paper follows the topology shown in Figure 1 that is derived from ETSI IoT reference architecture document [26]. Devices observe assets and send readouts to the application. The application, in turn, transmits back actuation commands to the devices. This leads to REST and EDA topologies, illustrated in Figure 2, that support protocols stacks made of layers that comply with the Internet Engineering Task Force (IETF) layered architecture. These layers are (1) the physical layer that enables signal modulation over the channel, (2) the link layer that supports access to the network core, (3) the network layer that provides end-to-end routing, (4) the transport layer that supports application traffic multiplexing and (5) the application layers that enables security and session management along with the transmission of application traffic itself. Note that Figure 2 shows the flow of messages in the context of both architectures and does not take into account the effects of lower layer protocols. Moreover, these flows are representative of a large number of REST and EDA protocols but not all of them. For example, certain event-driven protocols like the Data Distribution Service (DDS) do not rely on brokers.

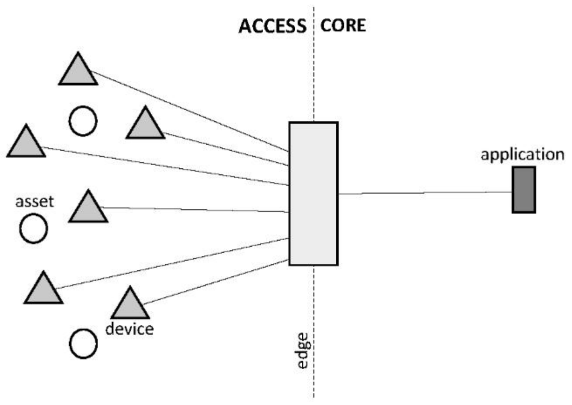


Figure 1. IoT Topology

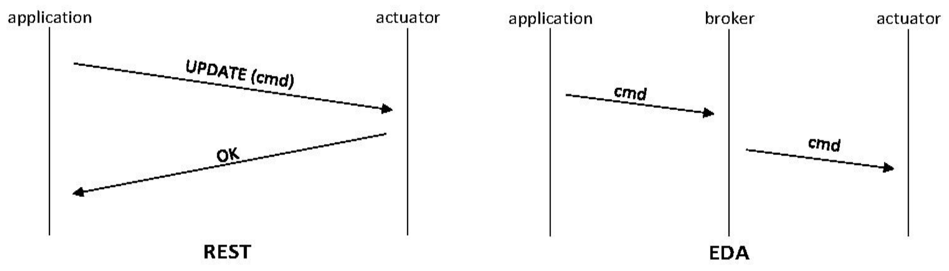


Figure 2. REST vs EDA

Figure 3 shows two stacks associated with HTTP and MQTT mechanisms. They both rely on traditional IEEE 802.11 (WiFi) physical and link layers that support Internet Protocol (IP) networking. Similarly, in both case the transport relies on the Transport Control Protocol (TCP). For each topologies the application layer is carried out by the corresponding HTTP and MQTT protocols. On top of the application layer is the actual actuation traffic that results from the transmission of actuation commands.

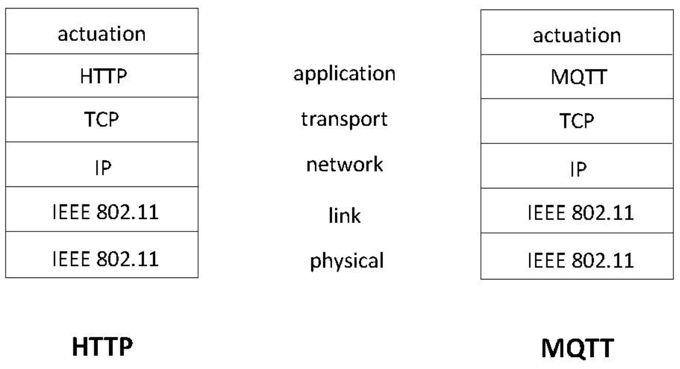


Figure 3. Protocol Stacks

Figure 4 shows the message flows for both HTTP and MQTT. Note that this Figure corresponds to a subset of the architectures shown in Figure 2. Specifically, HTTP and MQTT respectively fit the REST and EDA paradigm. These flows only indicate application layer exchanges and do not take into account the lower layers. Under HTTP, a POST request is directly transmitted from the application to the actuator driver. The request has a body that carries the actual actuation command. Once the request is received, the actuator driver transmits a 200 OK message to confirm. Note that 200 is the response code that represents the OK response. Under MQTT, a PUBLISH message is directly transmitted to the broker. The broker, in turn, forwards the message, as is, to the actuator driver. The message holds the actuation command.

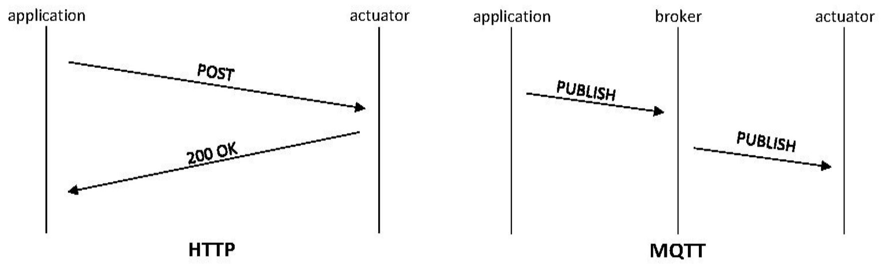


Figure 4. Flows: HTTP vs MQTT

4. Dynamic Protocol Selection Algorithm

This Section introduces the mathematical fundamentals that serve as the basis of the dynamic protocol selection algorithm introduced in this paper.

Network impairments like packet loss and latency result from the transmission of actuation commands in scenarios of multipath fading associated with wireless communication. In this context, the Gilbert-Elliot channel model shown in Figure 5 is representative of these scenarios [27]. The assumption is the existence of two states: (1) good and (2) bad respectively linked to low and high network loss. In the good state, the packet loss probability is e_G, while in the bad state, the packet loss probability is e_B. In addition, the model has two additional parameters: the channel good-to-bad transition probability p and the channel bad-to-bad transition probability α. Network packet loss and loss burstiness are controlled by the parameters p and α respectively. Note that under normal communication between two endpoints, loss is typically different for each direction. Because this model only supports half-duplex communication, two models (with different parameters) are typically needed to represent full duplex scenarios. Also note that for HTTP and MQTT sessions loss and latency are tied together because retransmissions due to loss cause messages to take longer to arrive. Channels with low loss are associated to channels with low latency. This channel model is used from this point on to derive all the following expressions.

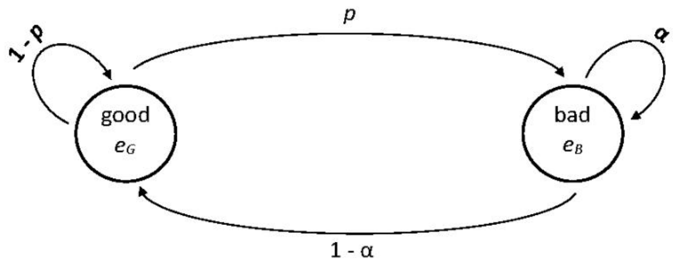


Figure 5. Gilbert-Elliot Channel Model

For MQTT, that relies on the transmission of readouts from the device to the application, only one channel model is needed. This model is known as the forward channel model. In this context, the overall probability that the forward channel is in a good ($P_{f,g}$) or bad ($P_{f,b}$) state is respectively given by

$$P_{b_f} = \frac{p_f}{1-\alpha+p_f} \tag{1}$$

and

$$P_{g_f} = \frac{1-\alpha_f}{1-\alpha+\alpha_f} \tag{2}$$

where p_f and α_f are the p and α parameters of the forward channel respectively.

The probability of a successful transaction is given by

$$P_{M_1,M_2} = P_{M_2|M_1,b_f} P_{M_1|b_f} P_{b_f} + P_{M_2|M_1,g_f} P_{M_1|g_f} P_{g_f} \tag{3}$$

where $P_{M_2|M_1,b_f}$ and $P_{M_2|M_1,g_f}$ are the probabilities of a successful message M_2 transmission given that a message M_1 has been transmitted when the channel is in bad and good states respectively. M_1 represents the message sent from the device to the broker and, similarly, M_2 represents the message sent from the broker to the application. In addition, $P_{M_1|b_f} = 1-e_{f,B}$ and $P_{M_1|g_f} = 1-e_{f,G}$ are the probabilities of successful transmission for the same bad and good channel states. Note that $e_{f,B}$ and $e_{f,G}$ are the e_B and e_G parameters of the forward channel. Assuming messages are independent, the $P_{M_2|M_1,b_f}$ and $P_{M_2|M_1,g_f}$ probabilities are respectively given by

$$P_{M_2|M_1,b_f} = P_{M_1|b_f} P_{b_f\to b_f} + P_{M_1|g_f} P_{b_f\to g_f} \tag{4}$$

$$P_{M_2|M_1,g_f} = P_{M_1|g_f} P_{g_f\to g_f} + P_{M_1|b_f} P_{g_f\to b_f} \tag{5}$$

where $P_{b_f\to b_f} = \alpha_f$, $P_{b_f\to g_f} = 1-\alpha_f$, $P_{g_f\to b_f} = p_f$ and $P_{g_f\to g_f} = 1-p_f$ are the transition probabilities between the different channel states. In addition, $P_{M_1|b_f} = 1-e_f$ and $P_{M_1|g_f} = 1-e_f$ leading to

$$P_{M_2|M_1,b_f} = \left(1-e_{f,B}\right)\alpha_f + \left(1-e_{f,G}\right)\left(1-\alpha_f\right) \tag{6}$$

and

$$P_{M_2|M_1,g_f} = \left(1-e_{f,G}\right)\left(1-p_f\right) + \left(1-e_{f,B}\right)p_f \tag{7}$$

that, in turn, imply

$$\begin{aligned} P_{M_2,M_1} = {} & \left[\left(1-e_{f,B}\right)\alpha_f + \left(1-e_{f,G}\right)\left(1-\alpha_f\right)\right]\left(1-e_{f,B}\right)\frac{p_f}{1-\alpha_f+p_f} \\ & + \left[\left(1-e_{f,G}\right)\left(1-p_f\right) + \left(1-e_{f,B}\right)p_f\right]\left(1-e_{f,G}\right)\frac{\alpha_f}{1-\alpha_f+p_f} \end{aligned} \tag{8}$$

with $e_{f,B}$ and $e_{f,G}$ defined based on the channel model.

For HTTP, as opposed to MQTT, the communication is bidirectional, so two channel models are needed: (1) a forward one and (2) a backward one. For those two models the channels probabilities are therefore $P_{f,g}$, $P_{f,b}$, $P_{b,g}$ and $P_{b,b}$. In this case the probability of successful transmission, derived similarly to equation 9 above, is

$$P_{R_s,R_q} = \left[P_{R_s|b_b} P_{b_b} + P_{R_s|g_b} P_{g_b} \right] \left[P_{R_q|b_f} P_{b_f} + P_{R_q|g_f} P_{g_f} \right] \quad (9)$$

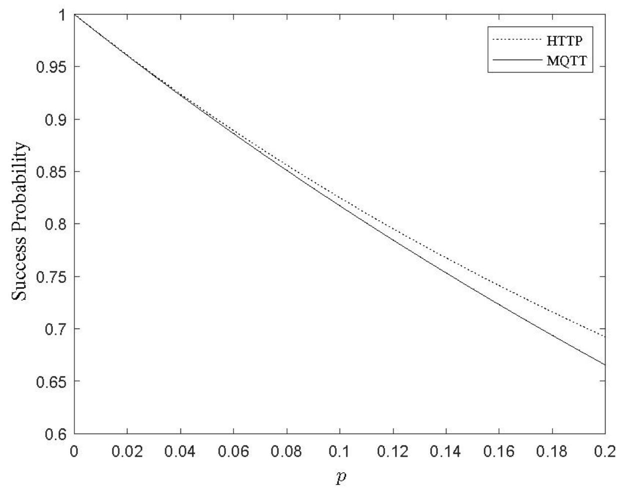
where $P_{R_s|b_b}$ and $P_{R_s|g_b}$ are the probabilities of a successful response when the backward channel is in bad and good states respectively and, similarly, $P_{R_q|b_f}$ and $P_{R_q|g_f}$ are the probabilities of a successful request when the forward channel is in bad and good states respectively. R_q represents the request sent from the device to the application and, similarly, R_s represents the response sent from the application to the device.

In addition, because $P_{R_q|b_f} = 1 - e_{f,B}$, $P_{R_q|g_f} = 1 - e_{f,G}$, $P_{R_s|b_b} = 1 - e_{b,B}$ and $P_{R_s|g_b} = 1 - e_{b,G}$, P_{R_s,R_q} can be obtained as

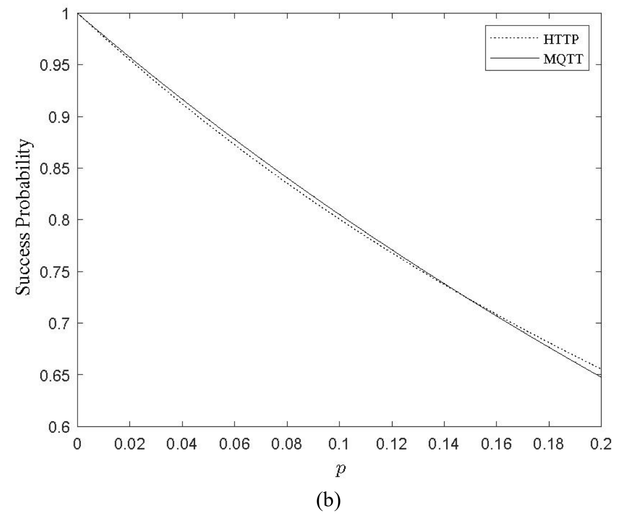
$$P_{R_s,R_q} = \left[\left(1 - e_{b,B}\right) \frac{p_b}{1 - \alpha + p_b} + \left(1 - e_{b,G}\right) \frac{1 - \alpha_b}{1 - \alpha_b + p_b} \right] \left[\left(1 - e_{f,B}\right) \frac{p_f}{1 - \alpha + p_f} + \left(1 - e_{f,G}\right) \frac{1 - \alpha_f}{1 - \alpha_f + p_f} \right] \quad (10)$$

with $e_{f,B}$, $e_{f,G}$, $e_{g,B}$ and $e_{g,G}$ defined based on the channel models. For simplicity, it can be assumed that all packets are lost when the channel is in a bad state and it can be also assumed that no packets are lost when the channel is in a good state. Specifically, $e_{f,B} = 1$, $e_{f,G} = 0$, $e_{g,B} = 1$ and $e_{g,G} = 0$.

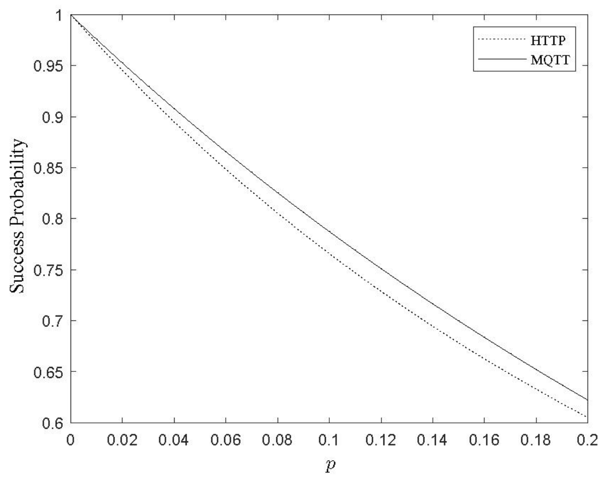
Figure 6 shows the probability of successful transmission of HTTP and MQTT messages, PRs,Rq and PM2,M1 respectively, for different values of the network layer packet loss (p) and packet loss burstiness (α). Note that depending on the values of the latter parameter, HTTP is sometimes more efficient than MQTT and, of course, some other times is not. For example, for $\alpha = 0.01$ HTTP is more efficient and, similarly, for $\alpha = 0.1$ MQTT is more efficient. Note that for $\alpha = 0.05$ the performance of both protocols is quite similar.



(a)



(b)



(c)

Figure 6. P_{R_s,R_q} (HTTP) vs P_{M_2,M_1} (MQTT). (a) $\alpha = 0.01$; (b) $\alpha = 0.05$; (c) $\alpha = 0.10$

Equations 8 and 10 lead to the following algorithm that dynamically determines whether HTTP or MQTT should be used to forward actuation commands from the application to the device:

(1) Compute the p_f, p_b, α_f and α_b estimators using Maximum Likelihood Estimation (MLE) as indicted below

$$\hat{p}_f = \frac{J_{f,L,H}}{J_{f,L,H} + J_{f,L,L}}$$

$$\hat{p}_b = \frac{J_{b,L,H}}{J_{b,L,H} + J_{b,L,L}}$$

$$\hat{\alpha}_f = \frac{J_{f,H,H}}{J_{f,H,H} + J_{f,H,L}}$$

and

$$\hat{\alpha}_b = \frac{J_{b,H,H}}{J_{b,H,H} + J_{b,H,L}}$$

where $J_{f,H,H}$, $J_{f,L,H}$, $J_{f,H,L}$ and $J_{f,L,L}$ respectively, reflect the transition counts between low and high loss probability states throughout the analysis period in the forward channel. Similarly, $J_{b,H,H}$, $J_{b,L,H}$, $J_{b,H,L}$ and $J_{b,L,L}$ are the same parameters for the backward channel. Note that these parameters are the outcome of a TCP segment analysis. In essence, TCP protocol information can be used to obtain the p and α estimators.

(2) Compute $\hat{P}_{M_2,M_1}$ as

$$\hat{P}_{M_2,M_1} = \frac{\left(1-p_f\right)\left(1-\alpha_f\right)}{1-\alpha_f + p_f} \tag{11}$$

(3) Compute $\hat{P}_{R_s,R_q}$ as

$$\hat{P}_{R_s,R_q} = \frac{\left(1-\alpha_b\right)\left(1-\alpha_f\right)}{\left(1-\alpha_b + p_b\right)\left(1-\alpha_f + p_f\right)} \tag{12}$$

(4) If $\hat{P}_{M_2,M_1} > \hat{P}_{R_s,R_q}$ select MQTT otherwise select HTTP.

To summarize, this algorithm takes into account the network layer impairments to select the best application layer session protocol. This selection is based on information that can be obtained from the TCP layer that both HTTP and MQTT rely on. Based on this, one of these two protocols can be dynamically selected to propagate application actuation commands to the device. An advantage of this algorithm is its low complexity of order $O(1)$ that enables it to be deployed on super low end IoT devices. Because, as indicated in Figure 6, the performance of HTTP and MQTT are highly dependent on network layer impairments, any application can use this algorithm to dynamically decide the protocol to select in order to transmit actuation commands to a device.

5. Experimental Framework

This Section introduces the experimental framework that is utilized to evaluate the performance of the algorithm introduced in Section 4.

To evaluate the efficiency of the algorithm presented in Section 4, the experimental framework topology shown in Figure 7 is used. The goal is to measure the latency between the application and the device when an actuation command is transmitted. In order to do so, Netualizer can be used to deploy emulated protocol stacks. Specifically, the protocol stacks shown in Figure 3 are integrated to build the topologies in Figure 7. Netualizer is a framework that supports Protocol Stack Virtualization (PSV) and enables the deployment of protocol stacks and topologies. Figure 8 shows this experimental topology on Netualizer. The functionality provided by an algorithm implemented (as a Lua script) that runs on Netualizer. Lua is a popular lightweight scripting language that is characterized by a very small fingerprint that is ideal for the deployment of applications on low end constrained embedded devices. In the context of the algorithm above, the script enables the retrieval of TCP layer parameters that support the estimation of the $J_{f,H,H}$, $J_{f,L,H}$, $J_{f,H,L}$ and $J_{f,L,L}$ metrics.

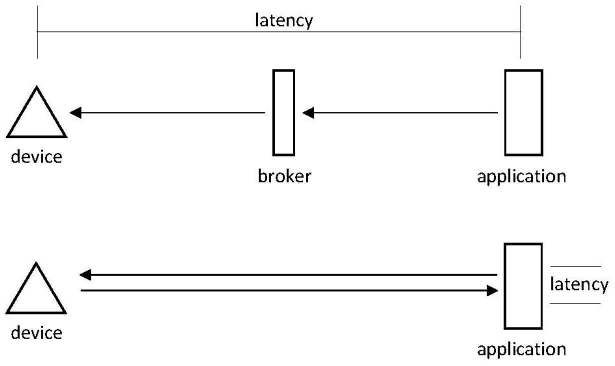


Figure 7. Experimental Framework

For different conditions given by levels of packet loss (parameter p) and packet loss burstiness (parameter α) for the forward and backward channels, the latency for HTTP and MQTT are estimated. Specifically, 300 actuation commands are transmitted at a rate of one command per second. It is assumed that in this scenario devices are sparsely deployed so for all intent and purpose they are isolated in their own subnets where there is little media access contention. The packet loss and packet loss burstiness follow a uniform distribution with $p \in (0.1, 0.4)$ and $\alpha \in (0.1, 0.9)$. Three scenarios are considered: (1) using HTTP, (2) using MQTT, and (3) dynamically selecting between HTTP and MQTT following the algorithm introduced in Section 4. Note that the algorithm runs on the application layer of the application stack shown in Figure 8 and dynamically selects the best transports to support the transmission of actuation commands. In this context, latency is measured in the device stack for MQTT, as the traffic is unidirectional. Similarly, latency is measured in the application stack for HTTP as the traffic is bidirectional.

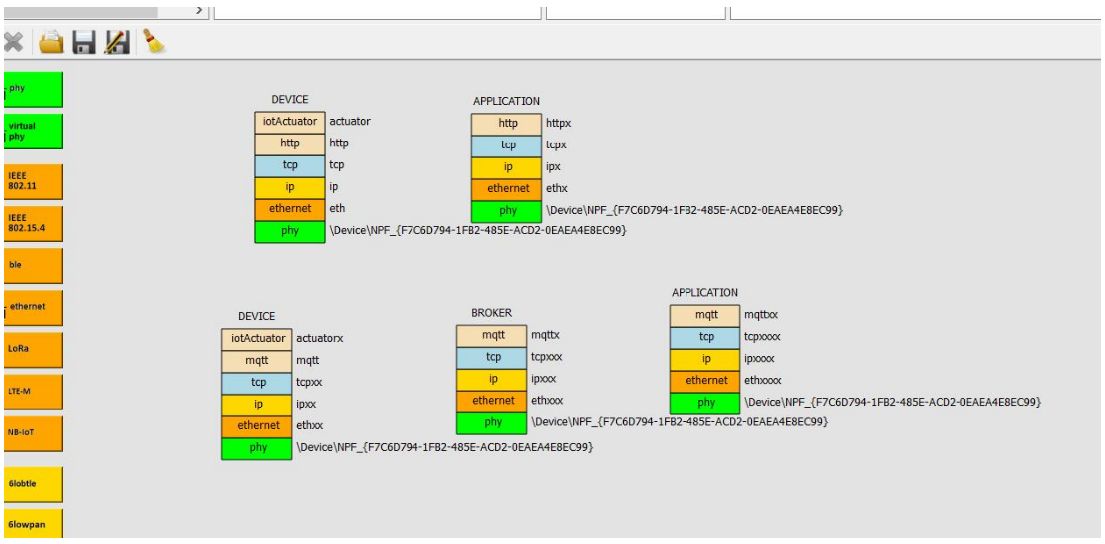


Figure 8. Experimental Topology on Netualizer

Figure 9 shows the actual end-to-end latency obtained for each of the three aforementioned mechanisms. It can be seen that HTTP and MQTT exhibit latency levels that vary in the 50 to 120 milliseconds range. For some samples, latency associated with HTTP is larger than that associated with MQTT, for other samples, the opposite is true, that is, latency associated with MQTT is larger than that associated with HTTP. The end-to-end latency that results from applying the algorithm results in an overall latency that combines the best case scenarios of HTTP and MQTT. Figure 10 shows the actual transitions between HTTP and MQTT that result for this particular scenario. Note that the transitions are quite uniform, it can be seen that around 53.28% of the time, the algorithm selects HTTP while for the remaining 46.72% it selects MQTT.

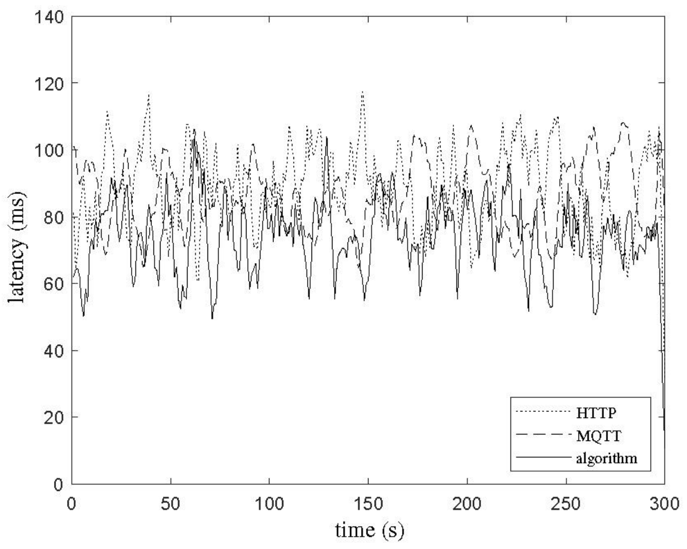


Figure 9. End-to-End Latency

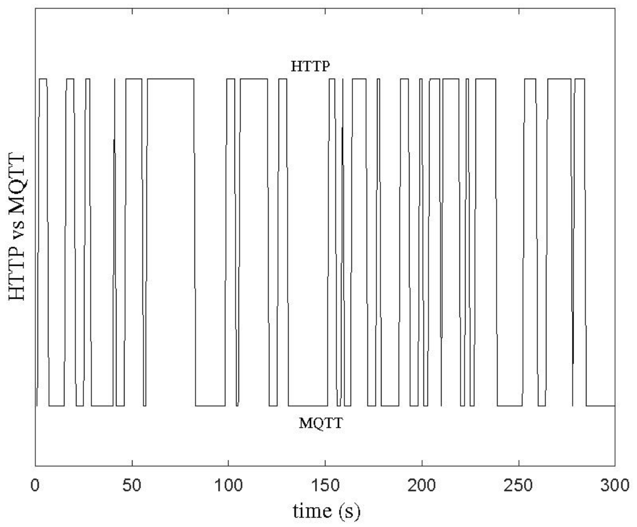


Figure 10. Mode Selection

Table 2 shows the experimental metrics that are measured as a result of the algorithm. Specifically, the mean latency and standard deviation in units of milliseconds is shown. Again, note that the proposed algorithm outperforms both HTTP and MQTT session management mechanisms to propagate actuation commands from the application to the device. The Table also shows an alternating scenario where HTTP and MQTT are sequentially applied 50% of the time. Specifically, the algorithm exhibits an overall latency that is 12.84%, 14.01% and 21.65% lower than that resulting from HTTP, MQTT and alternating session management respectively.

Table 2. Measured Experimental Metrics

Mechanism	Mean (ms)	STD deviation (ms)	HTTP (%)	MQTT (%)
HTTP	86.88	13.60	100	0
MQTT	88.05	11.17	0	100
alternating	92.15	19.26	50	50
algorithm	75.72	10.26	53.28	46.72

Similarly, the standard deviation of the latency is 24.56% and 8.15% lower than that of plain HTTP and plain MQTT respectively. Note that the Table also shows, in its last two columns, the ratio between HTTP and MQTT activity under each scenario. Because there are no other mechanisms that support dynamic session layer protocol switching, the proposed algorithm is only compared against static use of HTTP and MQTT.

6. Conclusions and Future Work

In this paper, we addressed the transmission of actuation commands from applications to IoT devices. We introduced a scenario where two session layer protocols, HTTP and MQTT, respectively support REST and EDA architectures. One issue with these two architectures is that, depending on network and communication conditions, the latency from the application to the device may be different depending on the session layer protocol under consideration. For certain actuation commands, the latency may be smaller under HTTP, for some other commands, MQTT may provide better results. In this context, this paper introduces a novel algorithm that dynamically selects the best session layer mechanism given network metrics that are collected real time from analyzing the transport layer. The algorithm, by conveniently combining both mechanisms, is shown to lower end-to-end latency by around ten percent when compared to the plain use of either HTTP or MQTT. The algorithm is based on a mathematical model of the communication channel, that produces to expressions to obtain the probability of successful transmission for both HTTP and MQTT. Depending on the burstiness characteristics of the network packet loss, sometimes HTTP is more efficient and some other times MQTT is. It can be seen that the algorithm uniformly switches between one protocol and the other for uniformly distributed channel network layer impairments. This algorithm is the most significant contribution of the research work presented in this paper. There are a few limitations associated with the mechanism though. It is designed to work with REST and EDA architectures that follow topologies shown in Figure 2. The algorithm does not take into account routing constrains like those associated with the Routing Protocol for Low-Power and Lossy Networks (RPL) standard. The algorithm also assumes no access media contention as devices are deployed in their own subnets.

Note that although REST and EDA are architectures and HTTP and MQTT are protocols respectively associated with these architectures, other session layer protocols can be also taken into account. Going forward, this research can be extended to take into account other IoT session layer protocols like CoAP. Moreover, because CoAP relies on the User Datagram Protocol (UDP) for transport, CoAP relies on a different traffic model that may be worth investigating. Specifically, dynamically choosing between HTTP, MQTT and CoAP can be the next step for this algorithm. Obviously, this requires some additional channel modeling. Alternatively another line of research can take the effect of routing and multi-device deployment scenarios.

Conflict of Interest

There is no conflict of interest for this study.

References

[1] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," IEEE Circuits and Systems Magazine, vol. 20, no. 3, pp. 4–18, 2020.

[2] R. Herrero, Fundamentals of IoT Com-munication Technologies, ser. Textbooks in Telecommunication En gineering. Springer International Publishing, 2021. [Online]. Available: https://books.google.com/books?id =k70rzgEACAAJ

[3] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, May 2015. [Online]. Available: https://rfceditor.org/rfc/rfc7540.txt

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Rfc 2616, hyperte xt transfer protocol – http/1.1,” 1999. [Online]. Available: http://www.rfc.net/rfc2616.html

[5] K. B. Andrew Banks, Ed Briggs and R. Gupta, “Mqtt version 3.1.1 oasis committee specification,” Apr. 20 14. [Online]. Available: http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[6] R. Herrero, M. Cadirola, and V. K. Ingle, “Preprocessing and compression of hyperspectral images capture d onboard uavs,” vol. 9647, 2015, pp. 9647 – 9647 – 9. [Online]. Available: http://dx.doi.org/10.1117/12.21 86169

[7] Y. Chen and T. Kunz, “Performance evaluation of iot protocols under a constrained wireless access networ k,” in 2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT), April 2016, pp.1–7.

[8] M. Iglesias-Urkia, D. C. Mansilla, S. Mayer, and A. Urbieta, “Validation of a coap to iec 61850 mapping an d benchmarking vs http-rest and ws-soap,” in 2018 IEEE 23rd International Conference on Emerging Techn ologies and Factory Automation (ETFA), vol. 1, 2018, pp. 1015–1022.

[9] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G. E. Corazza, “Internet of things application layer proto col analysis over error and delay prone links,” in 2014 7th Advanced Satellite Multimedia Systems Confere nce and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), Sept 2014, pp. 39 8–404.

[10] R. K. Kodali and V. S. K. Gorantla, “Restful motion detection and notification using iot,” in 2018 Internatio nal Conference on Computer Communication and Informatics (ICCCI), 2018, pp. 1–5.

[11] A. Gomez Goiri, I. Goiri, and D. Lopez de Ipina, “Reusing web-enabled actuators from a semantic spacebas ed perspective,” in 2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubi quitous Computing, 2014, pp. 370–375.

[12] M. S. Alam, U. D. Atmojo, J. O. Blech, and J. L. M. Lastra, “A rest and http-based service architecture for i ndustrial facilities,” in 2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS), vol. 1, 2020, pp. 398–401.

[13] R. Herrero, “Analysis of the constrained application protocol over quick udp internet connection transport,” Internet of Things, vol. 12, p.100328, 2020.

[14] P. Puspitaningayu, N. Funabiki, Y. Huo, K. Hamazaki, M. Kuribayashi, and W.-C. Kao, “Application of fin gerprint-based indoor localization system using ieee 802.15.4 to two-floors environment,” in 2022 IEEE 4th Global Conference on Life Sciences and Technologies (LifeTech), 2022, pp. 239–240.

[15] R. Zitouni, J. Petit, A. Djoudi, and L. George, “Iotbased urban traffic-light control: Modelling, prototyping and evaluation of mqtt protocol,” in 2019 International Conference on Internet of Things (iThings) and IEE E Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (C PSCom) and IEEE Smart Data (SmartData), 2019, pp. 182–189.

[16] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, “Internet of things: Survey and open issues of mqtt protocol,” in 2017 International Conference on Engineering & MIS (ICEMIS), 2017, pp. 1–6.

[17] R. Herrero, “Mqtt-sn, coap, and rtp in wireless iot realtime communications,” Multimedia Systems, Jul 202 0. [Online]. Available: https://doi.org/10.1007/s00530-020-00674-5

[18] C. Lai and Y. Hwang, “The voice controlled internet of things system,” in 2018 7th International Symposiu m on Next Generation Electronics (ISNE), May 2018, pp. 1–2.

[19] U. Rai, M. Patil, A. P. Singh, and W. Arif, “An iot based wireless robotic-hand actuation system for mimick ing human hand movement,” in 2020 International Conference for Emerging Technology (INCET), 2020, p p.1–6.

[20] A. Nichani, S. Saha, T. Upadhyay, A. Ramya, and M. Tolia, “Data acquisition and actuation for aquaponics using iot,” in 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Co mmunication Technology (RTEICT), 2018, pp. 46–51.

[21] S. Spinsante, G. Ciattaglia, A. Del Campo, D. Perla, D. Pigini, G. Cancellieri, and E. Gambi, “A lora enable d building automation architecture based on mqtt,” in 2017 AEIT International Annual Conference, 2017, p p. 1–5.

[22] T. Yokotani and Y. Sasaki, “Comparison with http and mqtt on required network resources for iot,” in 2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), 20 16, pp. 1–6.

[23] F. Azzedin and T. Alhazmi, “Secure data distribution architecture in iot using mqtt,” Applied Sciences, vol. 13, no. 4, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/4/2515

[24] L. G. A. Rodriguez and D. M. Batista, “Resourceintensive fuzzing for mqtt brokers: State of the art, perfor mance evaluation, and open issues,” IEEE Networking Letters, vol. 5, no. 2, pp. 100–104, 2023.
[25] S. K. Sukjun Hong, Jinkyu Kang, “Performance comparison of http, https, and mqtt for iot applications,” T he International Journal of Advanced Smart Convergence, vol. 12, no. 1, pp. 9–17, 2023.
[26] E. T. S. Institute, “Machine-to-machine communications (m2m); functional architecture,” Oct. 2013.
[27] O. Hohlfeld, R. Geib, and G. Hasslinger, “Packet loss in real-time services: Markovian models generating q oe impairments,” in 2008 16th Interntional Workshop on Quality of Service, June 2008, pp. 239–248.