Performance Evaluation of IoT Protocols under a Constrained Wireless Access Network

Yuang Chen
Department of Biology
Queens University
Kingston, Canada
13yc45@queensu.ca

Thomas Kunz
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
tkunz@sce.carleton.ca

***Abstract*—One of the challenges faced by today's Internet of Things (IoT) is to efficiently support machine-to-machine communication, given that the remote sensors and the gateway devices are connected through low bandwidth, unreliable, or intermittent wireless communication links. In this paper, we quantitatively compare the performance of IoT protocols, namely MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), DDS (Data Distribution Service) and a custom UDP-based protocol in a medical setting. The performance of the protocols was evaluated using a network emulator, allowing us to emulate a low bandwidth, high system latency, and high packet loss wireless access network. This paper reports the observed performance of the protocols and arrives at the conclusion that although DDS results in higher bandwidth usage than MQTT, its superior performance with regard to data latency and reliability makes it an attractive choice for medical IoT applications and beyond.**

Keywords—Internet of Things; Constrained Wireless Access Network; Protocols; Medical Applications; CoAP; MQTT; DDS

I. INTRODUCTION

As the Internet of Things (IoT) expands to a plethora of applications through the increasing minimization of hardware, availability of versatile sensors, and "smart objects" [1], many potential protocols are emerging for M2M communications. From this, the question of which protocol to use for the Internet of Things becomes a topic of high interest. Due to the remote nature and need for wireless networking of smart objects, IoT systems must be able to cope with potentially unreliable, intermittent, and low bandwidth connections for its access network. Several popular protocols at the application layer available today are geared towards the M2M communication role, namely MQTT (Message Queuing Telemetry Transport) [2], CoAP (Constrained Application Protocol) [3], DDS (Data Distribution Service) [4] and XMPP [5]. Today, the quantitative comparison of IoT M2M communication protocols within an unreliable, low bandwidth access network is still largely unexplored. The objective of this research is to obtain quantitative metrics that give insight into the actual performance of the previously mentioned IoT protocols within a constrained wireless access network under an applied medical scenario. From this, conclusions can be drawn when deciding upon which protocol(s) is(are) optimal for certain required performance metrics, both specific to this medical scenario and in general.

This research is funded by an NSERC USRA from Carleton University

IoT is a concept that has the versatility to be applied to almost any application. For this specific experiment, a medical monitoring application was chosen due to the immediate and significant benefits that could potentially result. Medical sensors worn by patients stream measured data into a local patient gateway, which in turn transfers this data to the central server. That server, in turn, can provide access to the data to caregivers, either by having caregivers retrieve/pull the patients' information from it, or by having the server push information to the caregiver, for example, when critical thresholds are breached (Figure 1). This IoT application will potentially increase the amount of patients that a single caregiver can monitor effectively.

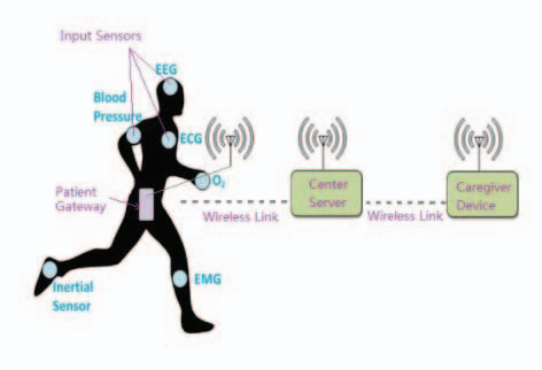


Fig. 1. The patient health data monitoring system used in this study

The remainder of this paper is organized as follows: First, the existing protocols are summarized in Section 2, followed by a short survey of related work in Section 3. In Section 4, the experimental environment is described. Section 5 presents and analyzes the experiment results. Finally, conclusions are drawn and future work is identified in Section 6.

II. PROTOCOL FEATURES

This section provides a summary of the main features for the IoT protocols CoAP, MQTT, DDS and Custom UDP, which are compared in this paper. We also initially explored the use of XMPP, but had to drop this. XMPP was initially intended for messaging/chat applications, the available protocol implementations are tightly coupled with the chat client GUI, and active development of the protocol seems to have ceased.

978-1-5090-1743-0/16/$31.00 ©2016 IEEE

A. CoAP

CoAP is a stateless protocol developed by the IETF to replace HTTP in resource-constrained devices. Being a UDP-based RESTful protocol, it uses a request/reply structure and has low overhead and a low degree of optional QoS. In order to receive telemetry, a client must constantly request the server to send the information. CoAP primarily supports a peer-to-peer style of communication but can be expanded to support one-to-many functions via the use of IP multicast.

B. MQTT

MQTT is a TCP-based publish-subscribe protocol developed by IBM and then open-sourced for messaging applications. In a publish-subscribe format, clients can either "publish" data on a specific topic to the server or "subscribe" to a topic where the server will automatically send new data on the topic to the subscriber once registered. MQTT combines the relatively high overhead and high QoS of TCP with the one-to-one, one-to-many, and many-to-one capabilities of a publish-subscribe format. Additionally, this protocol also allows clients to specify which telemetry topics are of interest and receive only data published through those topics.

C. DDS

Created as a networking middleware to circumvent the disadvantages of centralized publish-subscribe architecture, DDS is a TCP-based protocol that features decentralized nodes of clients across a system and allows these nodes to identify themselves as subscribers or publishers through a localization server. The use of this system negates the need for users to identify where other potential nodes are or which topics they are interested in, as the DDS nodes self-discover across a network and send/receive telemetry anonymously based only on topics. After linking publishers and subscribers, the connection between these clients bypass the server and are peer-to-peer (Figure 4).

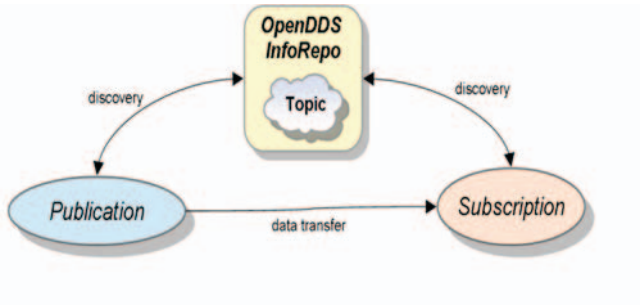


Fig. 2. DDS communication architecture

D. Custom UDP

The Internet also provides a variety of application protocols, using either UDP or TCP. As most of the communication in the "traditional" Internet involves humans, such protocols are not necessarily optimized for/design for M2M communication. Nevertheless, for comparison purposes, we designed our own application-layer protocol over UDP. This protocol exchanges custom-designed JSON strings over UDP server/client in publish-subscribe form. Because publish-subscribe protocols lower overhead by removing the need for constant client requests to obtain telemetry, the potential of a UDP-based publish-subscribe protocol having very low overhead combined with its one-to-many/many-to-one capabilities is an attractive prospect for an ideal IoT protocol; thus a custom protocol was created to match these specifications and will be used in this study to determine the performance of such an architecture. In the following, we simply refer to this protocol as "custom UDP".

III. RELATED WORK

There have been numerous qualitative reviews of various communication protocols that could potentially be applicable to IoT [6][7][8][9]. However, fewer papers related to the quantitative comparison of IoT protocols have been published to date. D. Thangavel et al. [10] compared the performance of CoAP and MQTT under a common middleware with different rates of packet loss and its effect on latency as well as overhead vs. packet size. However, the influence of other network conditions such as latency and bandwidth cap were not considered in this research. S. Bandyopadhyay et al. [11] compared the performance of CoAP's request-response mode and resource-observe mode with MQTT in terms of overhead vs. various packet sizes among two different packet loss (0% and 20%) conditions; additionally, power consumption vs. bytes of data communicated was also assessed in this research, however with packet loss being considered as the only factor characterizing the network condition. N.De Caro et al. [12] utilized smartphones as sensing platforms, then compared the performance of CoAP vs MQTT in terms of per-layer bandwidth usage, round trip time (for delay), and packet received ratio based on a 20% packet loss; this research compared the performance difference between a TCP-based protocol and a UDP-based protocol; however, a comparison between two different TCP-based IoT protocols such as MQTT and DDS was lacking.

To the best of our knowledge, this study is the only quantitative comparison including a wide set of IoT protocols such as DDS and a custom UDP protocol. The comparison of DDS' performance against other IoT protocols in this study in particular is novel. This paper aims to provide a comprehensive understanding of the actual performance for various IoT protocol candidates across a low quality network with low reliability, high latency, and narrow bandwidth. Through varying each of network bandwidth, network average packet loss rate, and network latency as an independent variable, trials were performed to assess its impact on bandwidth consumed, actual packet loss, and experienced telemetry latency; these metrics will be used as indicators of various aspects of performance for all protocols. Representative test results will be also be presented visually in this paper. Furthermore, a set of trials with a combination of narrow bandwidth, high latency, and high packet loss will also be performed to allow better gauging of protocol performance across a realistic low quality network.

IV. EXPERIMENTAL SETUP

This section introduces both hardware setup and software setup in the trials.

A. Hardware Setup

The testbed is shown in Figure 3 and Figure 4.

Patient Gateway/Caregiver APP

- 1x Raspberry Pi model 2 running Raspbian Linux Sensors.
- 1x Cooking Hacks eHealth sensor kit revision 2 including the following items:
  - eHealth sensor system shield
  - Heart rate/blood oxygen sensor
  - Temperature sensor
  - Skin resistance sensor
  - Skin conductivity voltage sensor
  - Patient accelerometer/orientation sensor
- 1x Arduino Uno revision 3 Central Server
- 1x Windows laptop ASUS Zenbook running Virtual Box Ubuntu

Links

- 1x Linksys network router used to set up the wireless network between the patient gateway, the server, and caregiver application.

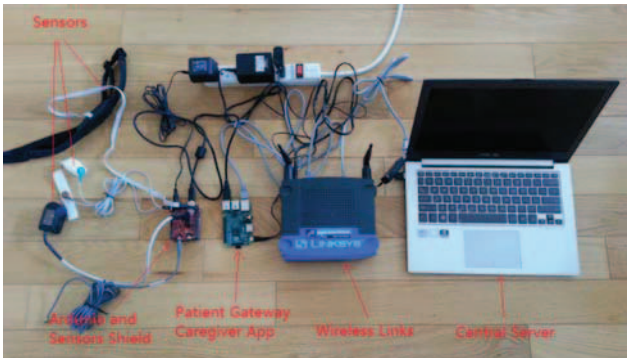


Fig. 3. Testbed

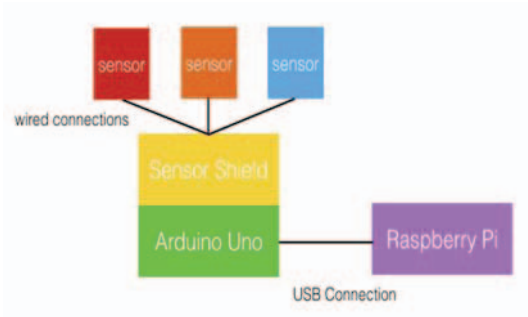


Fig. 4. Sensor devices and patient gateway

The eHealth sensor system by Libelium is only compatible with Arduino systems and not directly compatible with the Raspberry Pi. We therefore attached the sensors to an Arduino board and connected it via USB to the Raspberry Pi board which acts as a "patient gateway" (Figure 4). The patient gateway is connected to the central server through a wireless network. The caregiver device in turn connects to the central server to fetch patient information. The connections that will be tested are the links from the patient gateway to the server and from the server to a caregiver device.

B. Software Setup

It was necessary to develop server and client applications to support the sending and reception of medical data for each protocol using existing implementations. A reader application was also developed to allow the telemetry received on the Arduino to be available to Raspberry Pi.

To simulate a constrained low reliability network, NetEM [13] is used to set packet loss and latency of outgoing packets; TBF [14] is used to set bandwidth on both server and client. NetEM is installed in locations where the patient gateway publishes data to the central server and where the server transmits telemetry to clients. In order to obtain precise latency measurements (latency from time when data is sent from patient gateway to the time received by a caregiver), a roundtrip setting was used; this is achieved by having the same patient gateway sending the telemetry additionally running the caregiver application so that the round trip delay can be accurately calculated without time synchronization among the patient gateway, the central server, and the caregiver device. The developed medical telemetry application over all protocols features a basic statistics function showing packet loss and latency. Wireshark [15] was used for measuring bandwidth consumed.

All trials for each independent variable setting are set to run for 10 minutes and 3 repetitions with the final reported result being the average of the three trials. The format of communication is a JSON string. Every packet contains 409 bytes of heart rate, blood oxygen, skin moisture, and accelerometer measurement data. Every second, 4 packets are sent. In 10 minutes, 409 bytes * 4 packets/second * 600 seconds = 981600 bytes total are sent.

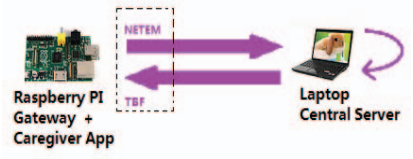
Setting / Protocol	
COAP	COAP server ----------> COAP client COAP client <----------COAP server
MQTT	MQTT publisher ---------->MQTT Broker MQTT subscriber <---------- MQTT Broker
DDS	DDS publisher ----------> DDS subscriber DDS subscriber <----------DDS publisher
Customized UDP	UDP client (publisher)----------> UDP server UDP client (subscriber)<----------UDP server

Fig. 5. Software setup for each protocol

The next paragraphs discuss in more detail how we set up the various software components for each protocol. As mentioned earlier, we are running both the patient and caregiver application on the same device to facilitate measurement of round-trip times. As neither of these two applications is particularly CPU-intensive, they do not impact each other. Some contention may be expected as we are

sending data wirelessly over the same bandwidth-limited channel. However, for almost all wireless technologies, the patient-server and server-caregiver communication would interfere with each other, even if caregiver and patient application were to execute on separate devices.

CoAP: we used the CCoAP [16] implementation. To establish a roundtrip message pathway, both CoAP server and client applications are run on the gateway while a "stitched" CoAP server and client application runs on the server. The pathway starts with CoAP server on raspberry Pi sending telemetry to the PC side CoAP client; as the PC side client receives this telemetry, this data is transferred to the PC side server and is sent back to the Raspberry Pi (Figure 5). It is important to note that open source implementations of CoAP are less readily available when compared to MQTT and do not feature nearly as complete of a support environment (forums, online communities); the source codes for these implementations are poorly commentated and require additional effort to be adapted for other uses outside of their original examples. A popular CoAP implementation called CCoAP is the "Californium" series by the Eclipse Foundation. However, like the Paho MQTT implementation by Eclipse, they require mandatory use of their own cloud servers and do not provide any source code or information in regards to their servers which greatly reduces the flexibility of its use.

MQTT: we used the HiveMQ server [17] implementation and the Mosquitto MQTT clients [18] (both subscriber and publisher). To create a roundtrip for accurate latency readings, both the publisher and subscriber were run on the gateway. The broker was installed on the central PC server (Figure 5).There are many open source MQTT server and client implementations available and the support for some of these implementations are thorough such as HiveMQ server, Mosquitto clients, Paho clients, ActiveMQ servers and clients etc. However Paho requires the use of its own cloud servers to function.

DDS: In our experiements, we used the OpenDDS server and the OpenDDS client [19]. To establish a roundtrip message pathway, both OpenDDS server and client applications are run on the patient gateway while a "stitched" OpenDDS server and client application runs on the server (Figure 5). DDS is supported by Linux as well as Windows; the source code provided is very complete and well supported/documented by the openDDS website. It was noted that the time required for DDS to be compiled on raspberry pi 2 was 18 hours while all other protocols took less than 30 minutes.

Custom UDP: Custom designed UDP client/server in publish-subscribe form with JSON string as payload, implemented based on the Socket API. Similar to MQTT, both the publisher and the subscriber runs on raspberry Pi and a broker runs on the central PC server (Figure 5).

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

The three quantitative metrics that are measured to indicate protocol performance are bandwidth consumed, experienced latency and experienced packet loss. By varying system latency, system packet loss, and network bandwidth cap (i.e., capping the wireless link throughput) independently through network emulation tools NetEM and TBF, the quantitative metrics are measured with several independent variable settings. The following are some analysis and selected visuals to rationalize and show our results.

A. Bandwidth Consumption

In all tests, 981600 bytes of application data was transmitted by the patient application in a 10 minute interval. With network packet loss rate changing from 0% to 25%, or network system latency changing from 0 to 400 ms, TCP-based protocols and UDP-based protocols responded differently in terms of bandwidth consumption, as shown in Figures 6 and 7. Note that, for latency, we did not simply added a fixed amount of latency but delayed packets based on a uniform distribution, ranging from 80% to 120% of the target latency. CoAP and Custom UDP do not consume additional bandwidth with increased network packet loss or increased network latency because no re-transmission is involved. MQTT and DDS increase their bandwidth consumption with higher rates of network packet loss, as expected of TCP-based protocols because of the inherent re-transmission mechanisms to guarantee packet delivery. An interesting observation is that MQTT showed little change in bandwidth consumed until the 200 ms system latency where it begins to decrease while DDS shows a stable level of consumption through all settings. The overall decreasing trend of MQTT as system latency increases could be explained by the nature of TCP handshakes; as the client is waiting for the acknowledgment of the server, the high latency of the network creates standby time in the system where no telemetry is sent or received. Under all network conditions of increasing network packet loss rate and increasing network latency, DDS always consumes approximately twice the bandwidth when compared against MQTT; obviously, DDS then must generate at least twice the number of control packets as MQTT does.

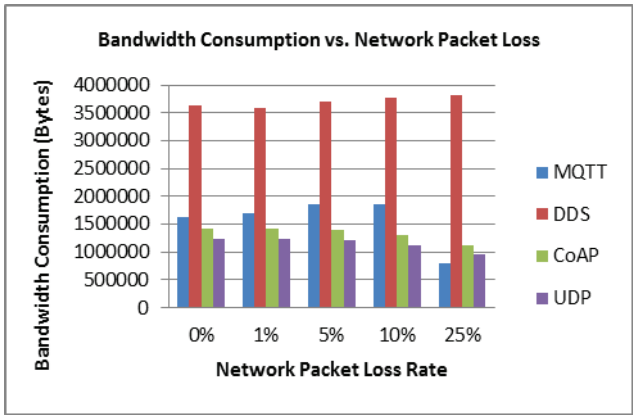


Fig. 6. Bandwidth consumption vs. network packet loss rate

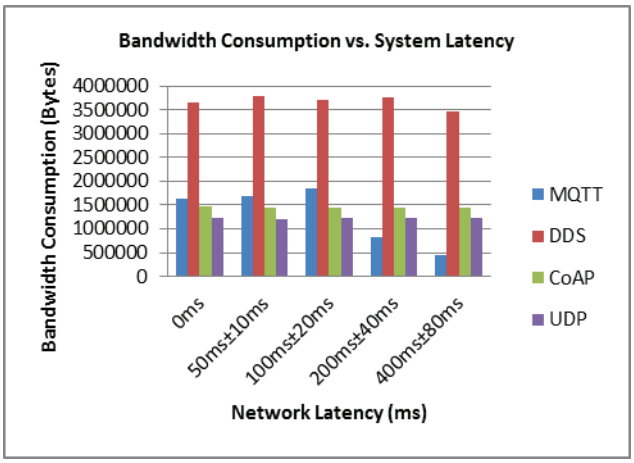


Fig. 7. Bandwidth consumption vs. system latency

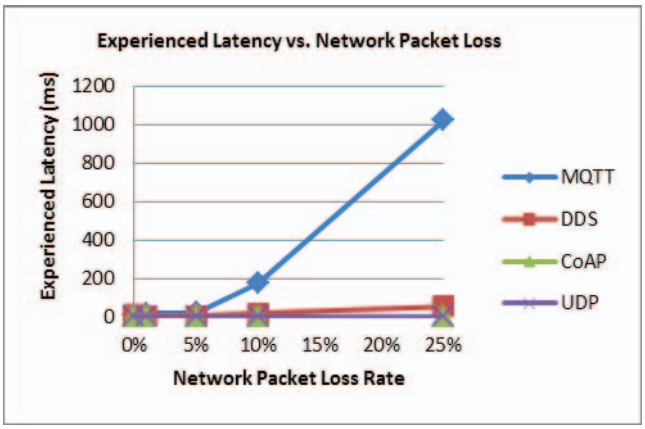


Fig. 9. Experienced telemetry latency vs. network packet loss rate

B. *Experienced Latency*

Experienced telemetry latency was measured by varying system latency, network packet loss rate, and bandwidth cap. The results are shown in Figures 8, 9 and 10. When network latency varies, the experienced latency of CoAP and custom UDP is very close to the system latency. DDS outperforms MQTT significantly when the network latency rises from 100ms to 400ms; it is observed that in these conditions, where DDS has a latency close to system latency, MQTT shows much higher experienced latency. When network packet loss rate increases from 5% to 25%, both DDS and MQTT show no packet loss but come with very different costs in terms of experienced telemetry latency; DDS experiences relatively little additional latency while MQTT reaches more than 1000 ms of experienced latency. Additionally DDS shows very steady, minimal latency without significant changes while network cap bandwidth decreases. In contrast, MQTT experienced increasing latency when the network cap bandwidth decreased. DDS outperforms MQTT in terms of latency under all degraded network conditions.

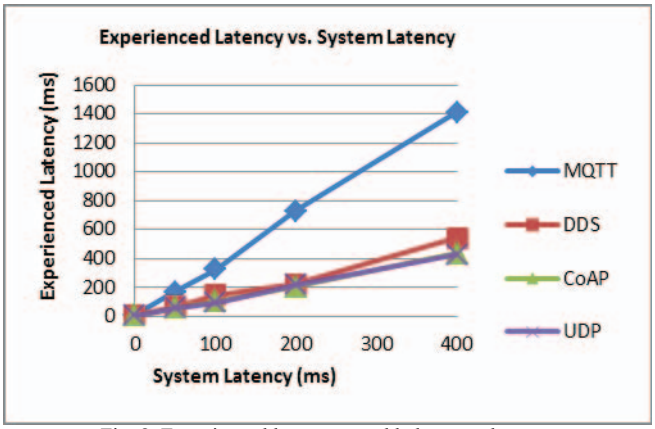


Fig. 8. Experienced latency vs. added system latency

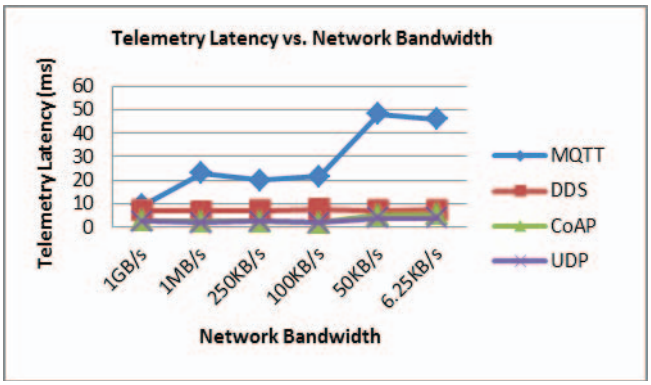


Fig. 10. Telemetry latency vs. network bandwidth

C. *Experienced Packet Loss*

It was observed in Figure 11 that given the network has a consistent rate of packet loss, the experienced packet loss of CoAP and custom UDP will take to a level very close to the system rate. On the other hand, both TCP-based protocols MQTT and DDS have no experienced packet loss regardless of network packet loss rate.

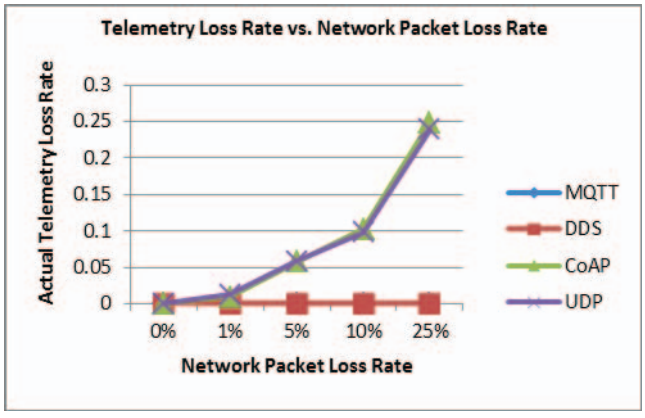


Fig. 11. Actual telemetry loss vs. network packet loss

D. *Low Quality Wireless Network*

In order to approximate the performance of all protocols in an application under a constrained, low quality wireless network, network packet loss was set to 10%, system latency to 100ms ± 20ms, and bandwidth cap to 6.25kB/s to simulate such an environment; the trial results are shown in Figures 12 and 13. In order to evaluate the protocols' control overhead proportional to the application data, the overhead was normalized by subtracting the amount of application data from the total amount of transferred bytes to obtain the number of control packets; then the number of control packets was divided by the amount of application data. It is observed that DDS generates the most telemetry overhead; in order to transfer 100 bytes of user data, DDS generates ~289 bytes of control data. MQTT generates ~76 bytes of control data for transferring 100 bytes of user data. However, by utilizing more bandwidth than MQTT, DDS subsequently shows significantly better performance in terms of telemetry latency averaging at 234 ms, less than half of MQTT's 578ms latency.

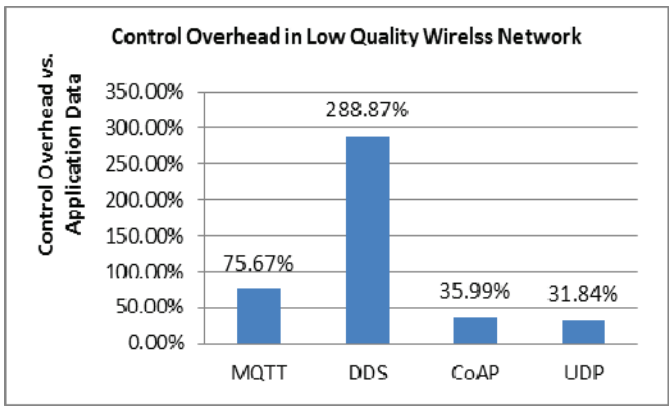


Fig. 12. Control overhead proportioned to application data in low quality wireless network

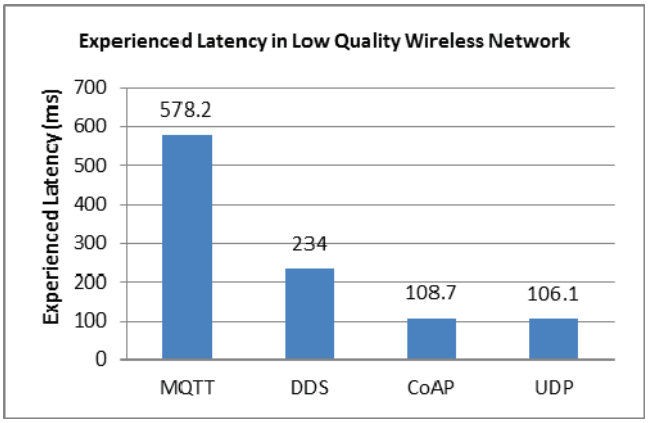


Fig. 13. Experienced latency in low quality wireless network

VI. CONCLUSIONS AND FUTURE WORK

Two types of conclusions are drawn from this study.

In terms of software availability, implementation readiness, and support environments of the protocols, the source code of OpenDDS as an implementation of DDS is provided in full on its website and is well documented; this is also true for MQTT. Additionally, MQTT has a versatile variety of implementations and possesses many online resources of forums and communities. In contrast, the open source implementation of CoAP is less readily available and poorly supported or commentated.

In terms of protocol performance, this study has revealed that both TCP-based protocols, DDS and MQTT, experience zero packet loss under degraded network conditions of up to 25% average network packet loss rate and 400 ms system latency. However, DDS significantly outperforms MQTT in terms of experienced telemetry latency in various poor network conditions such as high system latency, high network packet loss rate, and narrow network bandwidth cap. Despite DDS showing significantly higher bandwidth consumption than MQTT, its superior performance on data latency and reliability makes it the more attractive candidate for not only medical IoT applications, but any IoT system that requires loss-less, low latency performance. The UDP based CoAP and custom UDP are potentially viable for certain applications that require low bandwidth consumption and low latency; however as with any UDP-based protocol, the significant caveat of unpredictable packet loss will likely prevent it from being used in many IoT applications.

It is recommendable to establish more comparisons of basic performance metrics and appropriate supporting qualitative observations (human readability, ease of implementation) for other M2M protocols that can potentially be applied in an IoT scenario. Possible candidates include AMQP, JMS, and other protocols that become of interest as IoT develops and becomes the ubiquitous focus of technology.

REFERENCES

[1] D. Giusto, A. Lera, G. Morabito, L. Atzori, The Internet of Things, Springer, 2010. ISBN 978-1-4419-1673-0

[2] MQTT protocol specification (http://mqtt.org)

[3] CoAP protocol specification (http://coap.technology)

[4] DDS protocol specification (http://www.omg.org/spec/DDS)

[5] XMPP protocol specification (http://xmpp.org)

[6] R. Sutaria, and R. Govindachari, "Making sense of interoperability: Protocols and Standardization initiatives in IOT." ComNet 2013, Hyderabad, India, 8-9 Nov 2013

[7] Z. Sheng, S. Yang, Y. Yu, A. V. Vasilakos, J. A. McCann and K.K. Leung. "A Survey on the IETF Protocol Suite For The Internet of Things: Standards, Challengs, and Opportunities", IEEE Wireless Communications, pp. 91-98, December 2013

[8] L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A survey", Computer Networks 54 (2010) pp. 2787-2805

[9] A. Asensio, A. Marco, R. Blasco, and R. Casas, "Protocol and Architecture to Bring Things into Internet of Things", International Journal of Distributed Sensor Networks, Volume 2014, April 2014.

[10] D. Thangavel, X. Ma, A. Valera, H. Tan, and C. K. TAN, "Performance evaluation of MQTT and CoAP via a common middleware" 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, pp.1-6, 21-24 April 2014

[11] S. Bandyopadhyay, and A. Bhattacharyya, "Lightweight Internet Protocols for Web Enablement of Sensors using Constrained Gateway Devices" 2013 International Conference on Computing, Networking and Communications (ICNC), pp.334-340, San Diego, CA, USA, 28-31 Jan 2013

[12] N. D. Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, ″Comparison of two lightweight protocols for smartphone-based sensing", 2013 IEEE 20th Symposium on Communications and Vehicular Technology in Benelux (SCVT), Namur, Belgium, 21Nov -2 Dec 2013

[13] NetEM: Software suite provides Network Emulation functionality (http://www.linuxfoundation.org/collaborate/workgroups/networking/netem)

[14] TBF: Token Bucket Filter used to control network bandwidth (http://linux.die.net/man/8/tc-tbf)

[15] Wireshark: network protocol analyzer (http://www.wireshark.org)

[16] CCoAP (Californium CoAP) project for CoAP server and client implementation (https://www.eclipse.org/californium/)

[17] HiveMQ project for MQTT server (broker) implementation (http://www.hivemq.com/)

[18] Mosquitto project for MQTT clients (both publisher and subscriber) implementation (http://mosquitto.org/)

[19] OpenDDS project for DDS server and client implementation (http://www.opendds.org/)