LWM2M for Cellular IoT: Protocol Implementation and Performance Evaluation

Radim Dvorak[1], Lukas Jabloncik[1], Michal Mikulasek[1], Martin Stusek[1], Pavel Masek[1], Radek Mozny[1,2], Aleksandr Ometov[2], Petr Mlynek[1], Petr Cika[1], and Jiri Hosek[1]

[1]Department of Telecommunications, Brno University of Technology, Brno, Czech Republic
[2]Tampere University, Unit of Electrical Engineering, Tampere, Finland
✉Contact author's e-mail: masekpavel@vut.cz

***Abstract*—With the growing number of IoT devices, numerous IoT protocols are being developed to provide the market with options based on product requirements. With the release of cellular IoT (CIoT) technologies in the Czech Republic, recently LTE Cat-M, the focus shifts towards not only satisfying the requirements of the device application but also minimizing the generated traffic and overall co-existence of many devices under one cell. It led to the modification of the existing and the creation of new IoT protocols designed to generate as little overhead traffic as possible while adopting the existing well-known communication schemes. Namely, the new UDP-based protocol for IoT, LWM2M strives to be the alternative to the well-established IoT protocols used outside of CIoT. This paper explains the communication principles and capabilities of the well-established IoT protocols with a heavy focus on the LWM2M protocol. Further, it describes difficulties during user implementation of the LWM2M protocol. Furthermore, two carefully designed scenarios are used to compare the differences among these protocols, focusing on generated overhead as it is subject to a fee paid to the network operators within the CIoT networks.**

***Index Terms*—CIoT, Application Protocols, Overhead, LWM2M, Library, NB-IoT**

I. Introduction

Over the last decade, the data transmission model from sensor to server has evolved rapidly, which led to the creation of application protocols designed specifically for these scenarios for the Internet of Things (IoT). These protocols are designed to enable easy data transmission while keeping the complexity low and, therefore, are often easy to implement with minimum requirements for hardware (additional memory) or software changes [1].

The majority of the application protocols for IoT build upon existing L4 transport protocols, TCP or UDP. However, some protocols can utilize alternative lower layer mechanisms to transport the data in addition to the standard TCP, UDP scenario (e.g., the SMS protocol) [2]. This approach negates the requirement to define new security systems built into the protocol, as widely used security standards for TCP and UDP are designed to protect the user and protocol data. Therefore, most of the TCP/UDP-based protocols rely on the well-known (D)TLS ((Datagram) Transport Layer Security), which not only allows for data encryption but is often used as an authentication protocol utilizing the X.509 certificates [3], [4]. If additional security is required, IPSec can be used to secure the lower layer communication.

As previously stated, most of the protocols are simple to implement. Due to the simplicity of these protocols, most modem manufacturers choose to incorporate selected IoT protocol stacks into their products, making them simple to use. As a result, product development and implementation can be sped up dramatically as the modem's in-built CPU takes care of the protocol itself, and the user can fully focus on the raw data being transmitted. What is even more, the modem in-built CPUs, often based on the ARM architecture, are becoming powerful enough to incorporate the user application within itself, thus eliminating the requirement of an external CPU from the bill of materials completely (e.g., Quectel's QuecOpen platform) [5].

Besides well-known protocols such as MQTT and CoAP, the new protocol LWM2M has appeared on the list of supported protocols for modules of different manufacturers. The LWM2M, a relatively new protocol standard, builds on top of CoAP and promises enhanced remote device management with inbuilt OTA support on top of well-established CoAP data transfer capabilities [6]. To this end, in this part of the paper, we describe the key functionalities of the protocols as mentioned above and provide a comparison of the main features related to the cellular IoT use cases.

The key contribution can be summarized as follows:

- LWM2M protocol user software implementation with description of inner workings as well as the difficulties encountered during development.
- Protocol comparison in terms of data overhead in CIoT.

The rest of the article is organized as follows. In the Section II basic description of the protocols for CIoT is provided with focus key mechanisms and security mechanisms. Next, in Section III, we will focus on the detailed overview of the LWM2M protocol as well as the user library created for this paper. In Section IV, the description of measurement cases will be provided as well as the results and a discussion on how different protocols perform in terms of data overhead. Lastly, in Section V, we conclude the paper.

II. Communication Protocols for IoT

In this section, multiple well-known application protocols for IoT will be described together with the LWM2M, the representative of the protocol for data transmissions and device management. The comparison of the protocols in question is given in Table I.

A. Message Queuing Telemetry Transport – MQTT

MQTT is a lightweight text-based protocol for IoT appliances released in 1999 and standardized in 2013 by OASIS (Organisation for the Advancement of Structured Information Standards). MQTT uses TCP as its transport layer protocol, which ensures reliable data transmission from client to server on Layer 3 of the TCP/IP model [7] [8].

Communication within the MQTT protocol is a classic client-server scheme based on so-called topics. The client can send data to the topic (publish), or it can "subscribe" to it to read any incoming data published to the topic by other clients (the Publish-Subscribe model) [8]. Security for MQTT can be implemented utilizing the username-password authentication scheme and/or additionally via Transport Layer Security (TLS) with X.509 certificates, which also ensures safe data exchange via commonly used encryption algorithms [9].

B. MQTT for Sensor Networks – MQTT-SN

MQTT for Sensor Networks is a protocol introduced by IBM in 2008 and standardized in 2013 by OASIS. MQTT-SN is a variant of the MQTT protocol, which is modified to suit better the mMTC applications of wireless sensor units in the IoT world. The MQTT-SN retains the main communication scheme from the MQTT (the publish-subscribe model) but uses more lightweight transport layer protocols such as UDP to limit the amount of data transmitted [9], [10]. Also, by design, it considers battery-powered and constrained devices and reduces the overall computation power required to implement the protocol. MQTT-SN also allows devices to use discontinuous transmission if the device is operated from a battery and is required to save power by turning off its radio [11]. In the communication chain, four entities are defined by the MQTT-SN standard:

- MQTT Broker – Standard MQTT server (Broker).
- MQTT-SN Gateway – Receives messages from clients or forwarders via MQTT-SN and relays them to the Broker via standard MQTT protocol.
- MQTT-SN Forwarder – Serves as a forwarding entity in case the client can reach no MQTT-SN Gateway.
- MQTT-SN Client – IoT Device [10], [12].

C. Constrained Application Protocol – CoAP

The Constrained Application Protocol (CoAP) is a lightweight application layer protocol introduced in the RFC7252 by the Internet Engineering Task Force (IETF). As its name implies, CoAP is designed for constrained devices limited by their computation power, memory size, and data volume and/or are powered from a battery [2].

The data model of CoAP is based on a well-known Hypertext Transfer Protocol (HTTP) with a similar request-response model and implementation of representational state transfer (REST) architecture, enabling basic web service, which allows CoAP and HTTP to coexist within one infrastructure and enables simple translation from CoAP to HTTP and vice versa via a simple proxy. Like HTTP, CoAP is also natively implemented in modern web browsers. While HTTP is a text-based protocol, the CoAP protocol's creators tried to simplify the data model and reduce the number of bytes needed to transfer a simple message, which also led to the utilization of UDP as the underlying transport protocol [2].

Due to the usage of an unreliable UDP, CoAP has to compensate for unreliable transmission on the application layer. To compensate, CoAP defines two types of messages, Non-Confirmable (NON) and Confirmable (CON), which, together with Acknowledgement (ACK) and Reset (RST), form the 4 basic types of messages within the CoAP protocol. It makes CoAP a universal protocol, as confirmation of received messages is mostly unnecessary in some applications. However, in some scenarios, confirmations are needed (e.g., alerts, alarm messages, warnings, etc.) [2].

TABLE I: Comparison of application protocols for CIoT.

	MQTT	**MQTT-SN**	**CoAP**	**LWM2M**
L4	TCP	UDP, Other	UDP, SMS	UDP
Model	Synchronous	Asynchronous	Asynchronous	Asynchronous
Pattern	Publish-subscribe	Publish-subscribe	Both	Both
Restful	No	No	Yes	Yes
QoS layer	Transport	Application	Application	Application
Additional QoS	0,1,2	0,1,2	CON/NON	CON/NON
Dev. management	No	No	No	Yes
Data format	Any	Any	Any	TLV, JSON
Security	TLS	DTLS	DTLS	DTLS, OSCORE
Reg. ports	1883, 8883 (DLTS)	1883, 8883 (DTLS)	5683, 5684 (CoAPs)	5683, 5684 (CoAPs)
LPWA module supp.	Yes	No	Yes	No
Use-cases	Simple/advanced sensors, metering	Simple sensors	Simple sensors	Advanced sensors, remotely managed devices
Examples	Smart meters	Temperature, humidity	Temperature, humidity	Smart meters

Similarly to HTTP, DTLS, the UDP version of the TLS protocol, can be used with CoAP protocol to authenticate a given entity and encrypt data traffic (CoAPs – CoAP Secure) [2].

D. Lightweight Machine to Machine – LWM2M

LWM2M strives to be a new standard for M2M communication for constrained devices. LWM2M was introduced in 2017 by OMA (Open Mobile Alliance) to provide a universal protocol for all possible applications and unify the data model among different vendors under one protocol.

LWM2M protocol is built around the CoAP protocol, utilizing an in-built CoAP messaging scheme to interact with a target device. On top of CoAP, LWM2M builds an object-based data model where all objects are well-defined, non-interchangeable, and designed to allow data transmission and device management. In this regard, the LWM2M beats its competitors as none of the IoT protocols offers this feature embedded within its specification [6]. For security, Datagram TLS (DTLS) can be used for older versions of the protocol and a relatively new security scheme OSCORE with the newer versions [6].

III. LWM2M

This section contains an overview of the LWM2M protocol and the created LWM2M library. It focuses on the protocol's key features and communication principles and describes the library's design and functionality as well as difficulties encountered during development.

LWM2M protocol defines three entities in its communication chain.

- LWM2M Client – Device responsible for data collecting and object management.
- LWM2M Server – Responsible for managing the clients and data aggregation.
- LWM2M Bootstrap Server – Server responsible for initial client setup and providing information about connectivity-related matters [6].

Different from other conventional protocols, LWM2M introduces the Bootstrap server. Before a connection to a server can be attempted, the client has to undergo the bootstrap procedure, where initial configuration is loaded onto the device. This procedure can happen internally (the bootstrap procedure is integrated into the device's software), or the device can contact the Bootstrap server, which will then push the desired configuration onto it. This feature can be beneficial in case of the existence of multiple servers and load balancing [6].

A. LWM2M Data Model

LWM2M utilizes an object-based model for communication. Each object is represented by an integer ID which is well-defined by the OMA. An object contains multiple mandatory resources based on the object's purpose. Each resource also has its own well-defined integer ID. Some objects can have multiple instances, depending on the purpose of the object, each instance with its own unique ID [6].

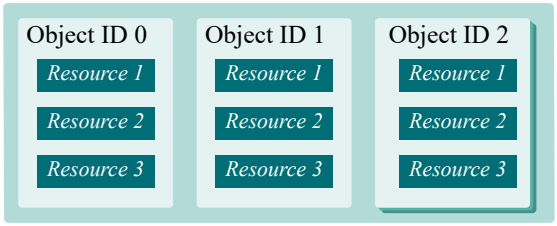

Fig. 1: LWM2M Data Model [6].

With this type of architecture, the server can access individual resources, instances or whole objects. To access the data, the server utilizes well-defined URI strings with IDs in the following format:

$$/ < Object\_ID > / < Instance\_ID > / < Resource\_ID >$$

where the instance or resource IDs can be omitted if requesting the whole object or instance of an object respectively.

B. Communication model

For communication among clients and servers, the LWM2M defines 4 interfaces. Depending on the interface, CoAP methods are mapped to different functionality within the LWM2M protocol.

- Bootstrap Interface.
- Registration Interface.
- Device Management and Service Enablement Interface.
- Information Reporting Interface [6].

After performing the bootstrap procedure, the client has to register to the server via the Registration interface. In the registration message, the client has to specify its endpoint name, which serves as an access token, and the device can be denied service if the token is invalid. After successful registration, the device has to perform periodical updates communicated during the registration procedure. If the update is not received within the specified time, the device is considered de-registered from the server and has to perform the registration again [6].

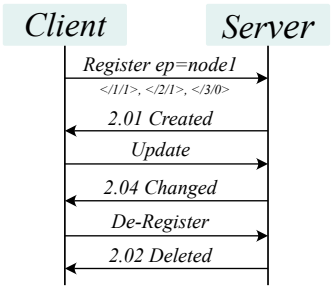

Fig. 2: LWM2M Registration Interface communication model [6].

After the successful server-client connection establishment, the server can start reading, writing data, and executing tasks via Device Management and Service Enablement Interface. Additionally, it can request a device to inform the server about

the status of the object, instance, or resource via the Information Reporting Interfaces Observe function. In the case of the Observe function, the client can be set to periodically inform (Notify) the server about any change of value or when the value reaches a defined threshold. To enable this functionality, the LWM2M specifies so-called Attributes that apply to all data structures (e.g., objects, instances, resources) [6].

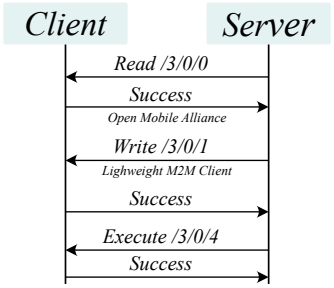

Fig. 3: LWM2M Device Management and Service Enablement communication model [6].

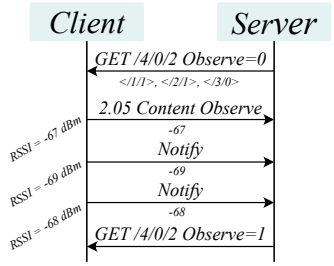

Fig. 4: LWM2M Information Reporting Interface communication model [6].

C. Library Architecture

While commercial LWM2M libraries are available on the market today, most of them are targeted for something other than constrained devices with limited memory and computation power. Therefore, we decided to create our solution to enable the protocol on constrained devices and also to assess the difficulty of the protocol implementation.

The library was made to be portable independently of the chip's architecture, its manufacturer, or the connectivity used to transmit and receive messages. This model brings some quality-of-life disadvantages as it is dependent on custom-made functions to interact with the library and ensure proper functionality of the system (for example, the device reboot function, which is mandatory for the LWM2M implementation and cannot be universally specified for all devices on the market) [13].

Furthermore, the library can also be set up to automatically handle the communication with the server via provided user-defined send and receive functions or simply send messages to a TX buffer where the user has to send and receive messages manually. It could be beneficial in time-sensitive single-threaded applications where sending the message directly could lead to some issues [13].

D. Implementation

The library consists of three main components.

- LWM2M Client – Main interaction point between the user and the library. It is designed to handle communication with the server, parsing CoAP messages, and data presentation. It also holds a reference to each Object created within the system.
- LWM2M Object – Representation of the instance of an object. It holds references to all the resources within the object as well as object and instance-specific identifiers.
- LWM2M Resource – Representation of the resource. Holds data that are reported to the server on demand.

Another important component is also an additional file, LWM2M Defines, where important constants for the functionality of the library are listed and where the overall behavior of the library could be specified (e.g., message format JSON, TLV, etc.) [13].

To keep track of the different library states and act in compliance with the specification, the library implements a state machine. After initialization, the library is in *NOT REGISTERED* state. Upon successful registration, the state becomes *REGISTERED IDLE*, which indicates the library is ready to process any incoming message from the serverv [13].

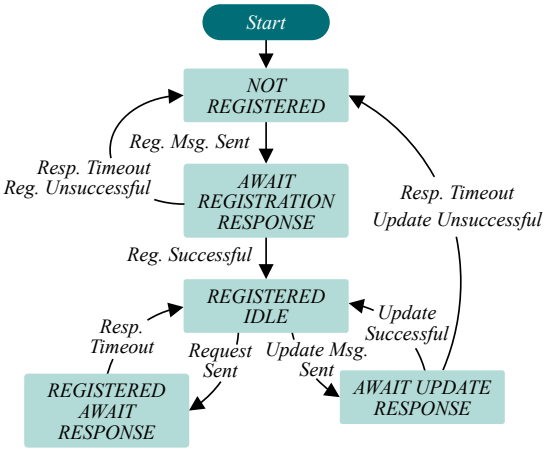

Fig. 5: Library state machine diagram [13].

When in *REGISTERED IDLE* state, the library will process an incoming message from the server. The message processing is built to be a tree-like structure where, at first, the message is handed over to the corresponding communication interface handler. The message is further processed in the interface handler to determine the requested LWM2M method. After the method is determined, the message is further passed down to the respective method handler, which processes the request and fulfills it by interacting with objects, resources, and the internal library structure. Also, at this stage, a response is generated if required.

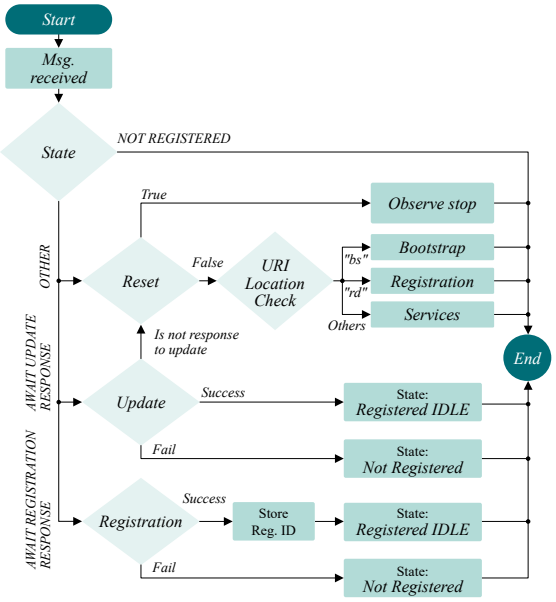

Fig. 6: Example of message handover to interfaces [13].

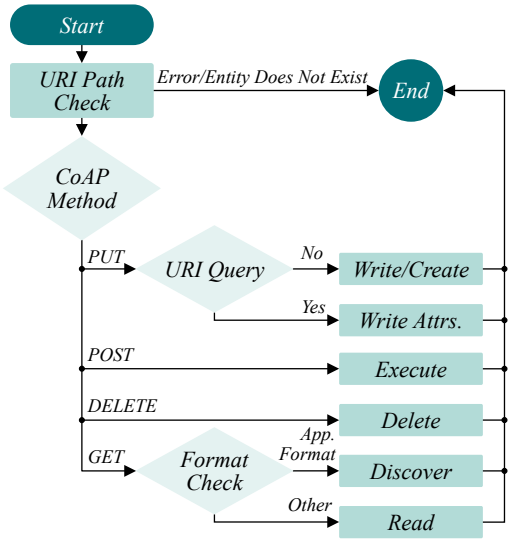

Fig. 7: Example of message handling for Device Management and Service Enablement interface [13].

E. Verification

During the development, the library was heavily tested against the widely available open-source LWM2M server Leshan by Eclipse. The final library implementation was also tested with multiple commercial servers that are open to the public as well as with private servers. This step was important to ensure library compatibility with other platforms as the protocol implementation might vary slightly from implementation to implementation. To name a few, the library was tested with IoT aggregators such as ThingsBoard or private solutions such as the LWM2M server of a local connectivity provider.

To test the library's portability, it was also tested with different transmission technologies such as Ethernet, Wi-Fi, NB-IoT, and LTE Cat-M [13].

IV. Performance Evaluation of the LWM2M

To evaluate the LWM2M protocol performance, test scenarios were created utilizing the Quectel BC660K-GL module, which is an NB-IoT module equipped with *Qualcomm 212 LTE IoT Modem* chipset. The selection of the BC660K-GL was based on in-built support for all of the well-known application protocols in this article. The test was performed in the public Vodafone Czech Republic NB-IoT network.

For each scenario, protocol initialization and termination handshake messages (if required), including lower layer handshakes and procedures, were recorded and included in the evaluation as they are part of the protocol's real-world implementation and are equally charged by the network operators. Furthermore, a server counterpart was selected from the available open-source alternatives for each protocol included in the test. TCP/UDP – Custom scripts, MQTT – Mosquitto broker, CoAP – Waakama, LWM2M – Leshan server.

To compare the protocols equally, scenarios were designed to mimic real-world utilization scenarios of telemetry reporting and also to minimize the amount of data overhead. For example, for CoAP-based protocols, *NON* messages were utilized whenever possible (LWM2M Observe), while the MQTT protocol utilized *PUBLISH* messages.

The test consists of 2 main scenarios.

- Short-term overhead evaluation.
- Long-term overhead evaluation.

For the short-term evaluation, multiple messages of sizes 12, 64, 128 and 256 bytes of user data were sent in the 30-second interval. The communication for each protocol was captured and evaluated.

Results can be seen in Fig. 8 where the total amount of data for each protocol TX and RX is depicted in kB. The transmitted user data can be seen above the 0 line and remain constant for each message size. What differs is the amount of overhead data depicted under the 0 line.

As expected, the UDP-based protocols show very little overhead in the RX direction, while TCP-based protocols show a significant amount of data transmitted in the RX direction due to the TCP reliability mechanisms on layer 3 and also its minimum header size of 20 bytes. This behavior could become an issue for the CIoT applications where small messages are transmitted as the protocol overhead could play a significant part in the total amount of bytes transmitted while the customer is charged based on the total generated traffic. It becomes evident in Fig. 8 where the total amount of overhead vs. the total amount of data transmitted reached up to 80% for the TCP protocol and up to 85% for the MQTT protocol with 12 bytes data size as shown in Table II.

In Fig. 9, the total amount of overhead data required is depicted for a sample of 20 messages including protocol setup, data transmission and termination. Here, as well as in the Fig. 8, the TCP trace is evident on the TCP-based protocols.

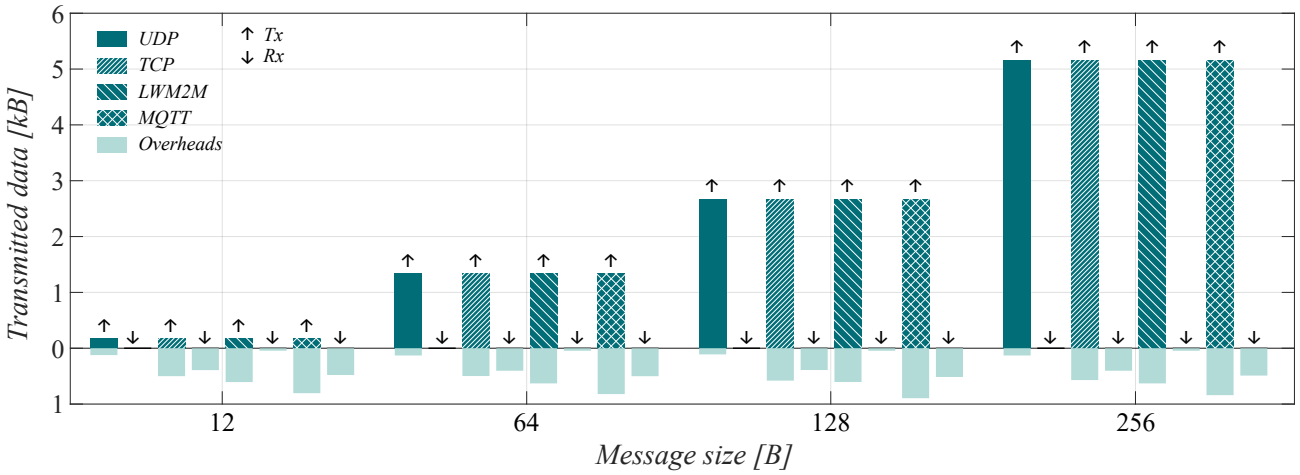

Fig. 8: Protocols data vs. overhead comparison.

TABLE II: Overhead % vs data size for 20 messages.

-	Overhead %			
Protocol	12B	64B	128B	256B
UDP	40	11,11	5,88	3,03
TCP	79,59	42,23	26,77	15,45
CoAP + LWM2M	75,35	36,44	22,28	12,53
MQTT	84,37	50,31	33,95	20,44

While the UDP protocol has the lowest overhead, it does not implement any mechanisms to ensure reliable data delivery or any device management features.

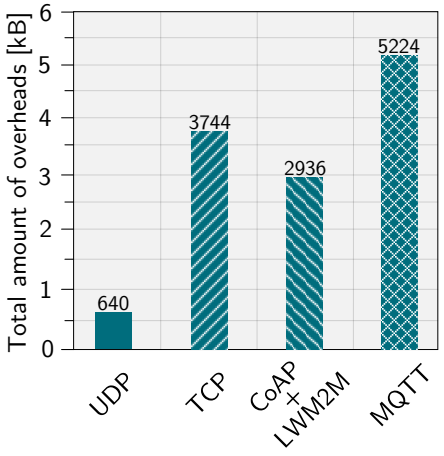

Fig. 9: Total overheads for 20 messages.

The LWM2M protocol, on the other hand, was able to achieve less overhead, utilizing the *NON* messages scheme for data reporting, than standalone TCP protocol while enabling complex device management and reliable data transmission when required due to the CoAPs in-built *CON and ACK* messages scheme. While the LWM2M protocol is quite difficult to implement from zero-ground and is quite memory-heavy, it makes it the most universal protocol in our list due to the small overhead and divisibility of applications that it offers. For the long-term evaluation, protocols were tested regarding the number of messages per 24 hours ranging from 1 to 25 (˜1 – 24 hours of the period). Two scenarios of the implementation were tested. First, the scenario where the user remains permanently connected to the server via keep-alive/heartbeat mechanisms. The second case where the connection is being released after every message sent. For this test, the data size was chosen to be 128 bytes.

As the results show in Fig. 10, the amount of data varies quite dramatically. The chosen application design approach can have a significant influence on the total amount of bytes transmitted, where the keep-alive approach starts to be preferable for applications with the requirement of data exchange greater than 10 per day for TCP (or 8 per day for MQTT).

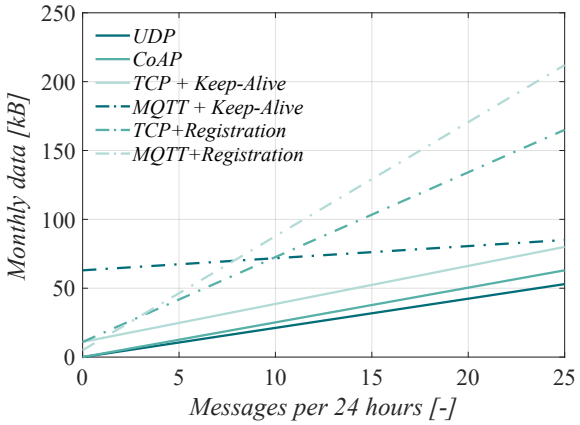

Fig. 10: Amount of data transmitted per month.

On the other hand, for applications requiring data exchange less than 10 times a day, the registration / de-registration process is the preferred method for total data transmission. While the previous statement is true for TCP-based proto-

cols, the UDP-based do not implement such features on the Transport Layer. In the case of LWM2M, periodic updates must be performed to achieve the same behavior. To this end, the lifetime variable describing the time of the next required update can be arbitrarily set to any value, and, therefore, its usage is always preferred to the full registration process.

V. Conclusion

As the IoT market evolves, different requirements for IoT devices give birth to different lightweight, easy-to-implement IoT protocols. With the introduction of the IoT to the cellular infrastructure, the focus shifted to not only be lightweight but also to utilize the network as little as possible to accommodate many other devices connected to the same cell at the same time and also to save costs for data, which have to be paid to the network operators for. This paper provides a comparison of the well-known protocols as well as the newly added LWM2M protocols. Furthermore, it builds on the LWM2M protocol implementation and evaluation against its competitors.

Firstly, the overview of the well-known protocols for IoT was laid out to provide an overview of the inner workings of such protocols. Further, the LWM2M protocol was described in detail, as well as the user library created in the context of this paper. Lastly, scenarios were created to compare the protocols and evaluated in terms of utilization in the CIoT ecosystem. The scenarios created are based on our knowledge and experience with real-world IoT protocol utilization as well and were designed to compare the protocols as equally as possible in different scenarios.

While the TCP-based protocols are user and traffic-flow friendly to an extent, their utilization is not recommended in CIoT networks as their overhead data is significantly higher than their UDP-based counterparts. This additional overhead causes the network to become more stressed by the amount of additional data, which, in edge cases, could lead to the denial of service and is also charged for by the network operators.

To this end, the UDP-based protocols are the protocols to go at this moment in time. In addition to the less overhead generated, they are on par with the functionality they offer, including the application layers' reliability features and QoS, which in most cases is not mandatory, making them the most suitable candidates for applications where reliable transmission is not required or required in specific cases.

While offering the least overhead in our test, the simple UDP protocol does not offer any traffic management or QoS. The CoAP protocol offers a great balance between the amount of overhead generated and the functionality it offers. In combination with the LWM2M protocol, not only casual data transfers but also complex device management can be achieved while still keeping the data overhead lower than the TCP counterparts. To this end, based on the author's experience, the CoAP and LWM2M seem to be the best protocols most suitable for CIoT applications.

Acknowledgment

This research was financially supported by the Technology Agency of the Czech Republic under the TREND Programme, project no. FW07010004. The work of the 7th author is supported through European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska Curie grant agreement No. 956090 (APROPOS: Approximate Computing for Power and Energy Optimisation, http://www.apropos-itn.eu/).

References

[1] P. Masek, J. Hosek, K. Zeman, M. Stusek, D. Kovac, P. Cika, J. Masek, S. Andreev, and F. Kröpfl, "Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-on Experience," *International Journal of Distributed Sensor Networks*, vol. 12, no. 4, p. 8160282, 2016.

[2] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7252

[3] E. Rescorla, "RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3," 2018.

[4] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, Apr. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9147

[5] Quectel Ltd., "QuecOpen," https://www.quectel.com/masterclass-library/quecopen-qualcomm, 2023.

[6] O. M. Alliance, "Lightweight Machine to Machine Technical Specification," *Approved Version*, vol. 1, no. 1, 2017.

[7] OASIS, "MQTT Version 5.0 OASIS Standard," https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf, 2019, Accessed: 2023-07-01.

[8] B. Mishra and A. Kertesz, "The use of mqtt in m2m and iot systems: A survey," *IEEE Access*, vol. 8, pp. 201 071–201 086, 2020.

[9] O. Sadio, I. Ngom, and C. Lishou, "Lightweight Security Scheme for MQTT/MQTT-SN Protocol," in *Proc. of Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019, pp. 119–123.

[10] A. Stanford-Clark and H. L. Truong, "MQTT for Sensor Networks (MQTT-SN) Protocol Specification," *International Business Machines (IBM) Corporation Version*, vol. 1, no. 2, pp. 1–28, 2013.

[11] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-SN — A publish/subscribe Protocol for Wireless Sensor Networks," in *Proc. of 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, 2008, pp. 791–798.

[12] 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 *Proc. of IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 2014, pp. 1–6.

[13] Dvorak Radim, "Laboratory Demonstrator for LPWA Technologies Enabling the Communication via the Lightweight M2M (LWM2M) Protocol," https://www.vut.cz/en/students/final-thesis?zp_id=141376, 2021.