T··Mobile·

NARROWBAND IoT SOLUTION DEVELOPER PROTOCOLS GUIDE

INTRODUCTION

To deliver on its massive transformative potential, the Internet of Things needs to interconnect an enormous number of devices, services, and applications over data networks as efficiently as possible. Given the ever-growing diversity of devices—from resource-constrained sensors to sophisticated smart phones—broad range of environments, and myriad evolving use cases, building viable IoT solutions would be practically impossible if not for the decades of groundwork that have gone into creating messaging protocols for the web and mobile communications.

Built upon standards established by the 3rd Generation Partnership Project (3GPP), Narrowband IoT (NB-IoT) cellular networks now provide cost-efficient access to a massive number of resource-constrained IoT devices via mobile radio access networks. To take full advantage of NB-IoT, IoT solution developers need to understand the capabilities and constraints of the NB-IoT data communication.

Most IoT solution providers have a background in designing web or enterprise applications and are intimately familiar with the web protocol stack. Optimized IoT applications, however, require a specialized application-level data transport protocol and an IoT-centric communications stack.

For devices operating in remote locations, Message Queuing Telemetry Transport (MQTT) is often suggested as the preferred messaging protocol. MQTT, however, relies on the TCP/IP stack as the underlying transport, and TCP is suboptimal for connecting over mobile networks.

The Internet Engineering Task Force (IETF) developed the Constrained Application Protocol (CoAP) as an extremely lightweight communications protocol stack suitable for resource-constrained, remote devices. CoAP evolves and streamlines the web Representational State Transfer (REST)-ful model inherent in the HTTP to make it as efficient and lightweight as possible.

The Open Mobile Alliance (OMA) uses CoAP in the lightweight machine-to-machine (LWM2M) IoT device management standard. LWM2M enables remote management and control of IoT devices using a streamlined managed objects model and provides interfaces for securely monitoring and administering devices.

In this document, we highlight the main characteristics of IoT protocols and assess their suitability for the deployment over NB-IoT mobile networks both over IP and without IP using Non-IP Data Delivery (NIDD).

IoT DEVICES AND ECOSYSTEM

The Internet of Things is growing to encompass an enormous range of applications and devices. It is critical for the success of an IoT solution that not only the correct devices are used with the appropriate communications technologies, but also that the characteristics and patterns of the device data reporting, such as frequency, volume of data, and required upstream and downstream bandwidth, are accounted for. Access to power and the amount of electricity consumed by devices are also crucial considerations.

Fig. 1 presents an example of such classification mapping the typical areas of IoT applications, 3GPP-defined device classes, and main characteristics of the related mobile data technologies. Successful IoT solutions require that the correct devices are used with the appropriate communications technologies, and that the characteristics and patterns of the device data reporting, such as frequency and volume of data are accounted and planned for in the solution design phase.

NB-IoT is a recent evolution of the 3GPP LTE standard that provides efficient deployment of resource-constrained, low-cost IoT devices on contemporary mobile networks. NB-IoT defines communication channels with limited characteristics suitable for constrained, low-maintenance devices that offer long battery life and cost-efficiency. The constrained nature of NB-IoT devices and networks calls for careful planning of the communications patterns and higher-level application data communication protocol.

Fig. 2 presents characteristics of the NB-IoT data communication and recommendations for appropriate use of devices, communication patterns, and suitable classes of IoT applications. The rest of this paper explores the **MQTT** and **CoAP/LWM2M** messaging protocols and provides analysis and recommendations on the appropriate uses for each.

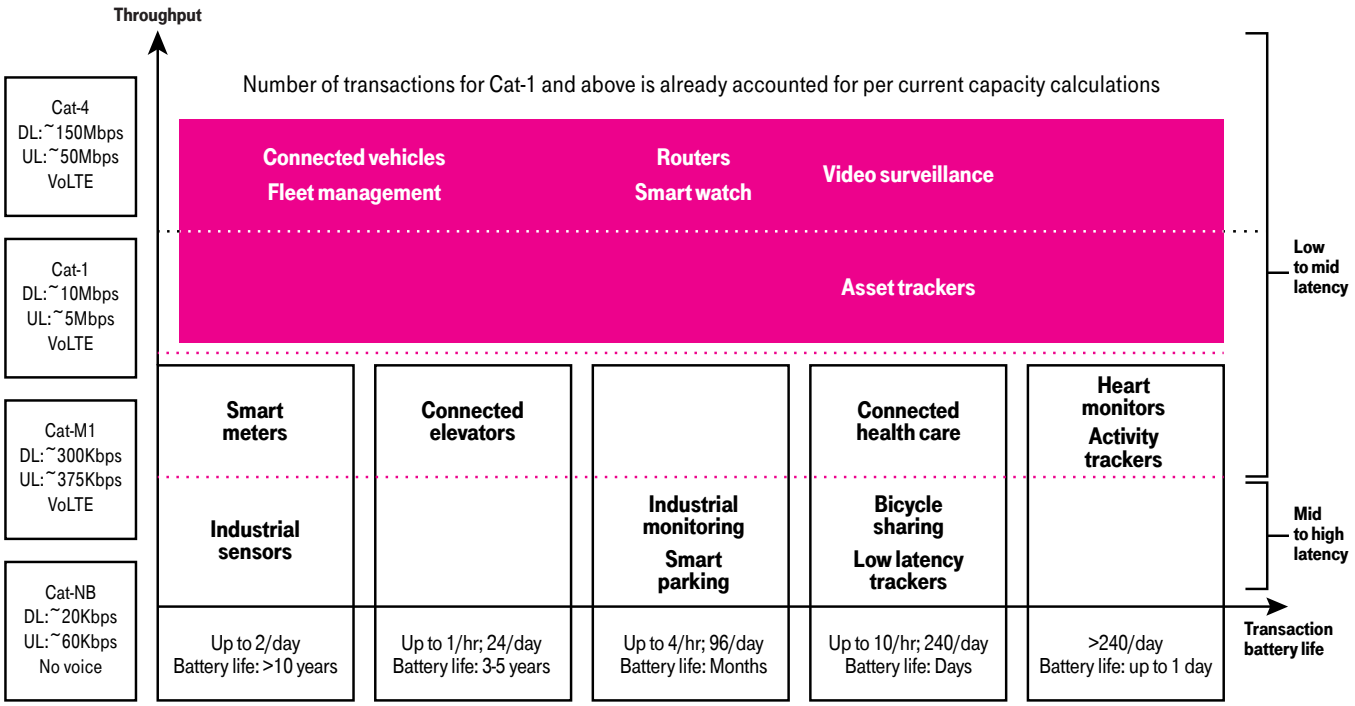

Fig. 1: Emerging IoT solutions and devices

Functionality	NB-IoT Non-IP/IP			LTE
Peak Data Rates	DN: 15-20 Kbps; Up: 20-25 Kbps			~ 300 Kbps
SMS	Yes			Yes
Mobility	Idle mobility			Yes
MQTT-Data	No			Yes
CoAP	Yes			Yes
LwM2M - Device Mgmt	Non-IP: Device Management, IP: SW updates			Yes
LwM2M - Data	Yes			Yes
Advantage	Deeper in-building coverage			Higher data rates, mobility voice
Device Profile	**Low**	**Med**	**High**	
Connections per day	Up to 2	Up to 48 (every 30 min)	Up to 240 (every 6 min)	No limit
Proj. Message size, bytes	50	150	350	1500
Size of user data Per day (bytes) max	100 bytes per day 0.036 bytes per day	4800 bytes per day 1.7 Mb per year	36000 bytes per day ~13 Mb per year	750,000
Acceptable latency, s	20 to 60	10 to 20	< 10	Low
Location	No	Yes	Yes	Yes
Mobility	Stationary	Mobile	Mobile	Mobile
Typical Applications	**Sensors, Metering:** Water, Gas. Air	**Smart Infrastructure:** City Lighting, Parking	**Limited Trackers:** Smart Bicycle, Livestock	**Real Time Trackers:** Kids, Pets **Wearables, Connected Transport**

Fig. 2: Recommended bandwidth and payload characteristics for NB-IoT and LTE IoT devices

WHAT IS NB-IoT?

NB-IoT is a mobile access technology targeted at low-cost support for the massive deployment of lightweight and constrained IoT devices. As such, it has the characteristics of the constrained network; specifically, its effective bandwidth in one sector is limited to 226.7 Kbps peak, and the overlaying protocols must efficiently address the low-level fragmentation. The protocol targets latency-insensitive applications. While NB-IoT is designed to allow less than 10 sec latency, typical NB-IoT devices and applications tolerate from 10 sec to more than 100 sec of latency. The NB-IoT devices suitable for applications that require long battery life (e.g., 10+ years) and where the devices are expected to transmit an average 200 bytes per day.[1]

NB-IoT can be used in both IP and NIDD modes. The data transmission cost is essentially the same as it is applied to gross transmitted data, not to the effective payload. Application developers need to be very efficient in their data transmission protocols and minimize the overhead of the data to be transmitted over cellular networks.

The new OMA LWM2M 1.1 protocol provides the necessary transport binding for deploying over the NB-IoT access networks. It is based on the streamlined CoAP and contains necessary semantics for performing device management and ensuring data transmission via the information-reporting interface.

The New NB-IoT Hourglass

NB-IoT presents developers with the challenge of fitting the IoT communication from constrained devices into a constrained communication channel with:

- Narrow bandwidth
- Low occurrence of data transactions
- Very small payloads
- High to very high latency
- Devices in sleep mode most of the time

Developers need to eliminate an unnecessary overhead introduced by the IP, TCP, and TLS. The IoT device classes targeted by cellular IoT and NB-IoT technologies will typically transmit from a single byte to tens of bytes in one payload. The devices will be in sleep mode most of the time, and there are strong limitations imposed by the network on this highly streamlined data transmission channel. Typically, the bandwidth will be around 10 kbps, and the payload has to fit in the radio frame. Devices are supposed to transmit very infrequently—at most, several times per hour, and the transmission latency may be very high, from 10 sec. to 100 sec.

These limitations are challenging for an industry used to the relative freedom of modern high-end communications channels. Cellular LTE networks deliver 10+ Mbps bandwidth with latency measured in milliseconds. Plus, reliable, long-lived TCP connections are always available. With the

effective payload on the order of magnitude of bytes, the overhead introduced by the TCP/IP stack becomes significant and can be 10x to 100x relative to actual payload.

NB-IoT allows constrained devices to overcome this limitation by introducing NIDD and eliminating the TCP/IP stack completely. However, this poses a new challenge for application developers, requiring them to address the need to manage payload fragmentation, in-order delivery, and flow control.

The combination of NB-IoT with the technologies in the LWM2M 1.1 stack effectively addresses those challenges and allows smooth transition to the constrained NB-IoT stack. Moreover, not only does it deliver data efficiently, but LWM2M's device management functionality enables the devices themselves to be managed remotely—allowing for low-maintenance, long-battery-life device deployments and solutions.

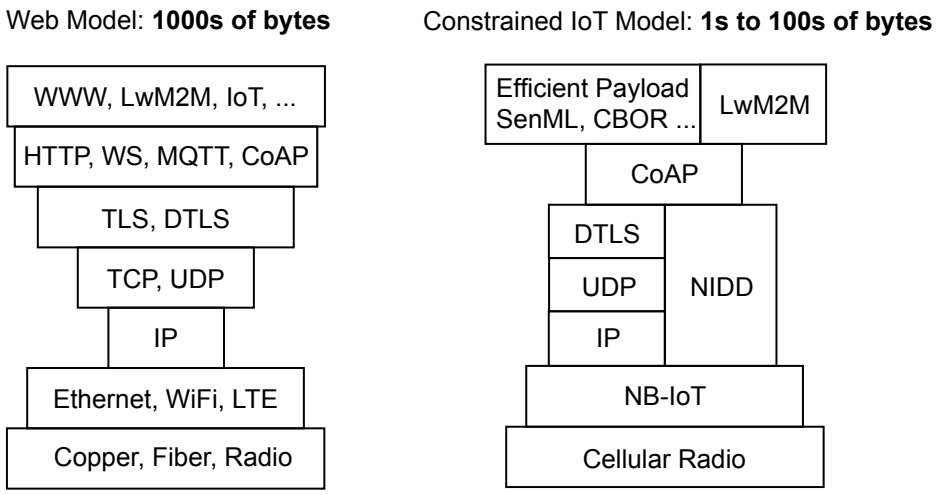

Fig. 3: Narrowband IoT hourglass model

In this model, the CoAP becomes the new spanning technology—the waist of the new NB-IoT hourglass model. With built-in congestion control, block-wise transfer fragmentation handling mechanism, efficient coding of the headers (options) and payload, and the ability to effectively transport agnostically, CoAP can be reused for both IP- and non-IP-based NB-IoT deployments.

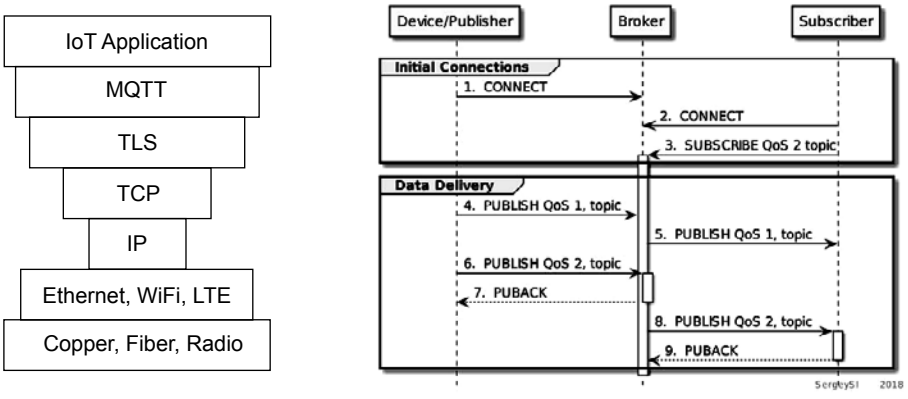

Fig. 4: MQTT protocol stack and baseline sequence, QoS 1 and QoS 2 messages

DATA COMMUNICATION PROTOCOLS FOR THE INTERNET OF THINGS

According to the Eclipse 2018 IoT Developer Survey, MQTT today is the leading messaging protocol in use today, followed by the group of web protocols, including HTTP 1.1, HTTP/2, and WebSocket. CoAP is the only leading UDP-based protocol.

It is worth noting that the popularity of the proprietary in-house protocol implementations is declining as developers realize that standardized, reliable data protocols with reasonable messaging or REST semantics are required to deliver interoperable performance for IoT solutions.

The web-based protocols are primarily used for communication on browser-based clients, which typically run on relatively high-capability devices, such as smartphones that use relatively high-bandwidth communications channels (e.g., LTE). However, the two protocols most suitable for mass IoT market are MQTT over TCP and LWM2M and CoAP over UDP. For all applications and devices deployed over NB-IoT, CoAP/LWM2M is recommended.

Both HTTP and MQTT require TCP/IP as the underlying transport and are not suitable for NB-IoT NIDD communication channels. In contrast, CoAP itself and CoAP-based LWM2M use UDP and can be deployed over NIDD since the only requirement is that the CoAP packets fit into a datagram. MQTT is the only protocol that fully supports the publish-subscribe messaging model, whereas HTTP and CoAP are REST based.

CoAP can be seen as the evolution of HTTP for resource-constrained devices and networks, and reuses some HTTP verbs, such as GET, POST, PUT, and DELETE. CoAP and LWM2M allow developers to use the observe/notify pattern modeled after the software observer design pattern. MQTT does not specify the payload format. While CoAP follows the web model similar to the HTTP content-type, LWM2M defines a specific format for representing objects and resources.

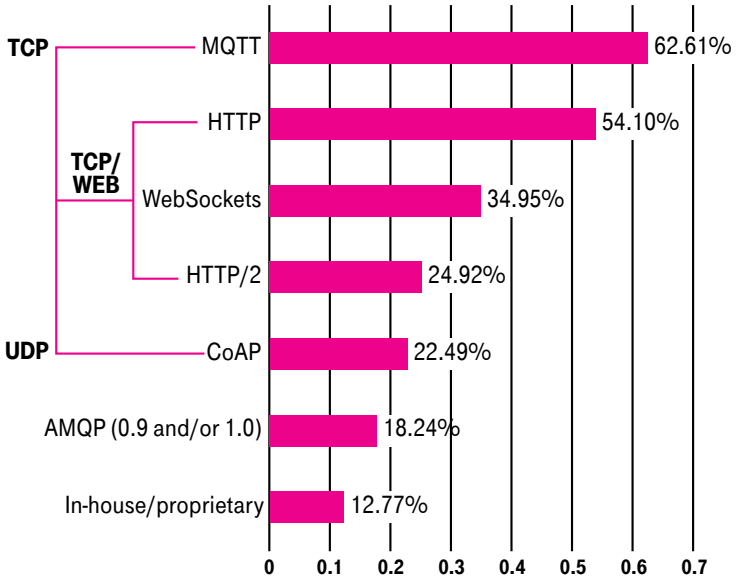

Fig. 5: Popularity of IoT data transport protocols in 2018[2]

Even though CoAP requires that a single CoAP/LWM2M packet must fit into a UDP datagram, a mechanism has been developed to handle fragmentation and transfer large data on the application level.[3]

As we will demonstrate further in this document, MQTT protocol is most suitable for relatively unconstrained applications where long-standing connections and large data payloads on the order of magnitude of hundreds to thousands of bytes are expected. CoAP and LWM2M are better suited for resource-constrained IoT applications and devices, where the payloads are very small.

MQTT

MQTT is the lightweight application layer messaging protocol designed for the relatively resource-constrained devices used for IoT applications.[2] It requires reliable, lossless, in-order delivery of packets, and relies on the TCP/IP-based reliability and ordered delivery. However, MQTT requires much less overhead than HTTP.

MQTT uses a topic-based, publish-subscribe model. Clients willing to transmit data connect to MQTT servers called message brokers by issuing the CONNECT command. If the connection is accepted, an MQTT broker replies with the CONNACK command. After the connection is established, data can flow in both directions.

Clients send data to servers in the PUBLISH messages associated to a particular topic. Clients willing to receive messages from message brokers indicate so by issuing a SUBSCRIBE command to the MQTT message broker, indicating a particular topic. As soon as publishing clients PUBLISH messages, all subscribers to the same topic start receiving notifications (PUBLISH messages) from the message broker.

MQTT message reliability (on top of the TCP) is assured by the three quality-of-service (QoS) levels:

QoS Level	Model	Description
0	At most once	Unreliable level – Message is delivered without duplication; no acknowledgment is required
1	At least once	Reliable level – Message is delivered at least once with possible duplications; acknowledgment is required
2	Exactly once	Reliable without duplications – MQTT uses four-way handshake mechanism to ensure that the message is delivered without duplications

Table 1: MQTT quality-of-service levels

CoAP

CoAP has been specifically designed by the IETF to meet the requirements of the constrained devices and communications channels. It is based on the REST architecture following the synchronous request-response HTTP model. CoAP uses the GET, PUT, POST, and DELETE methods and response codes similar to, but not exactly like, HTTP. Therefore, it is easy to map CoAP traffic from devices to the RESTful API logic.

CoAP uses the resource model mapped to the Universal Resource Identifier (URI) instead of MQTT topics. However, there is a similarity between the CoAP URIs and MQTT topics. For example, sensor devices publishing their sensor information to a server could be described in the following manner:

- CoAP sensor publishing to CoAP server: URI: coap://devices/sensors/temperature
- MQTT client publishing to a sensor queue on the broker: topic: "/devices/sensors/temperature"

The CoAP message format is compact and lightweight with only 4-byte fixed header encoding the GET, POST, PUT, DELETE methods and response codes, type of message (confirmable or nonconfirmable), and MessageID used to correlate requests with responses (ACKs).

Variable token is used for longer-term session

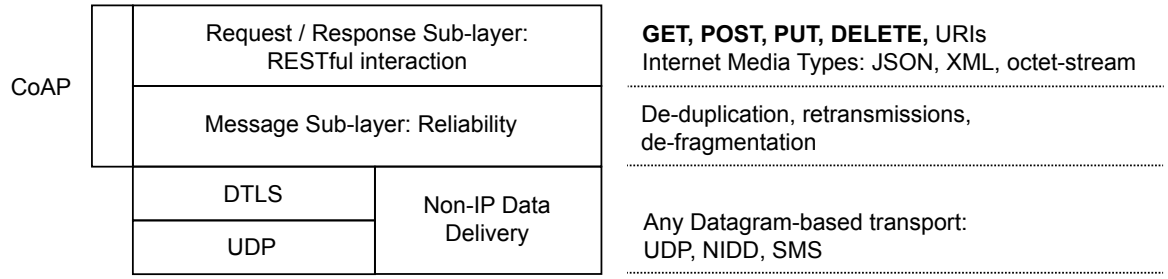

Fig. 6: CoAP messaging model and functional layers

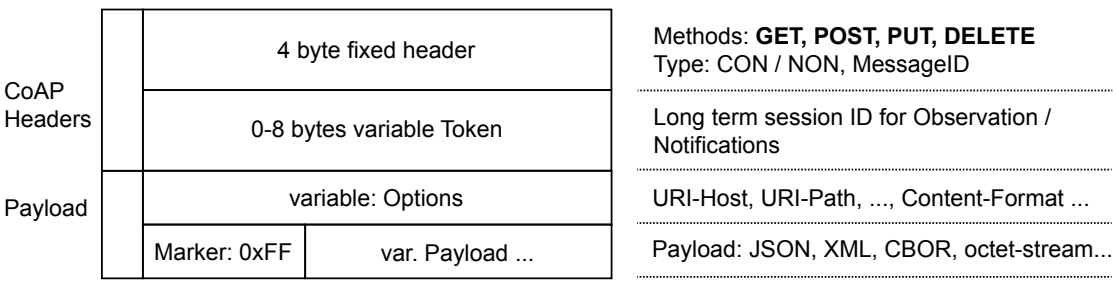

Fig. 7: CoAP message format

semantics (e.g., matching notifications to a particular observation request). Variable options encode URI, content-format, and information usually transmitted in the HTTP headers, followed by the variable payload, which can be any internet format specified by the content-format options, e.g., JSON, XML, or octet-stream. CoAP is extremely lightweight with only a 4-byte mandatory header. While MQTT specifies only 2 bytes for the fixed header, in reality, it has to always carry another 2-byte MessageID for any MQTT PUBLISH request.

CoAP's request-response mode is asynchronous and nonblocking. A client does not have to wait for the response to initiate another request, and CoAP responses can also carry data so that no round trips are wasted.

The main difference between CoAP and MQTT results from the MQTT requirement to run on the reliable, lossless, in-order, byte-stream delivery transport, which effectively mandates the TCP/IP stack.

CoAP, on the other hand, runs on top of UDP and provides its own reliability mechanism using confirmable and nonconfirmable message types, and the block-wise transfer fragmentation mechanism.

Transport layer security for CoAP is seamlessly provided in the Datagram Transport Layer Security (DTLS) protocol. CoAP provides its own reliability, flow control, and fragmentation handling layers, and the only hard requirement is that a CoAP message fits into an underlying protocol's datagram, meaning it can easily and seamlessly be deployed on any datagram-based transport, such as SMS or NB-IoT NIDD.

Another remarkable feature of CoAP is the ability to perform asynchronous push observe/notify requests, which are modeled after the observer software pattern. A server initiates a request to observe particular resources on the CoAP client. The client remembers the request and notifies the server each time the state of the observed resources change.

CoAP provides its own reliability mechanism using confirmable messages and nonconfirmable messages. Confirmable messages have the type "0" (CON), require an acknowledgment (message type 2 - ACK), and the client will continue resending it until it receives the ACK message from the server or it times out. Nonconfirmable messages have the type "v1" (NON) and the client will not wait for the ACK. CoAP ACK messages can carry useful payload data and themselves do not require separate ACKs. The CoAP block-wise transfer mechanism ensures reliable transmission of large-size payloads by splitting it in chunks that are guaranteed to fit into the lower-level datagrams (Fig. 9).

Whereas CoAP does not provide an explicit quality-of-service model (unlike MQTT), the confirmable/nonconfirmable message types allow it to loosely map to the MQTT QoS semantics as follows:

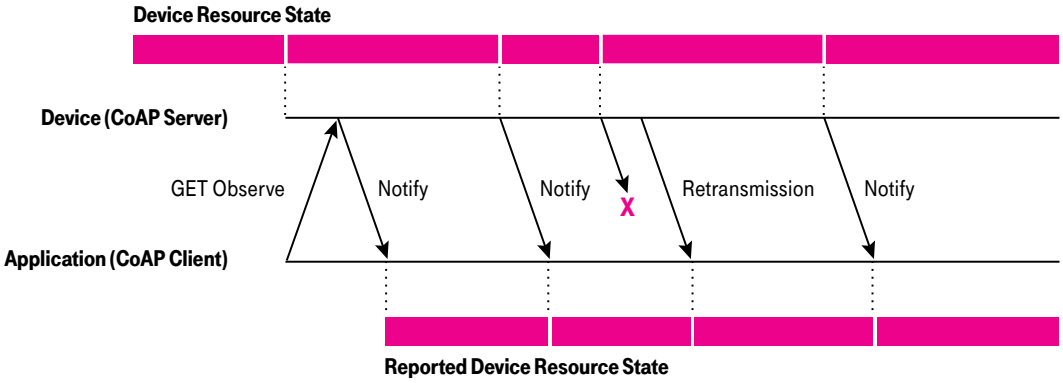

Fig. 8: CoAP resource observations with confirmable notifications. Illustration courtesy of K. Hartke, Universitaet Bremen TZI.

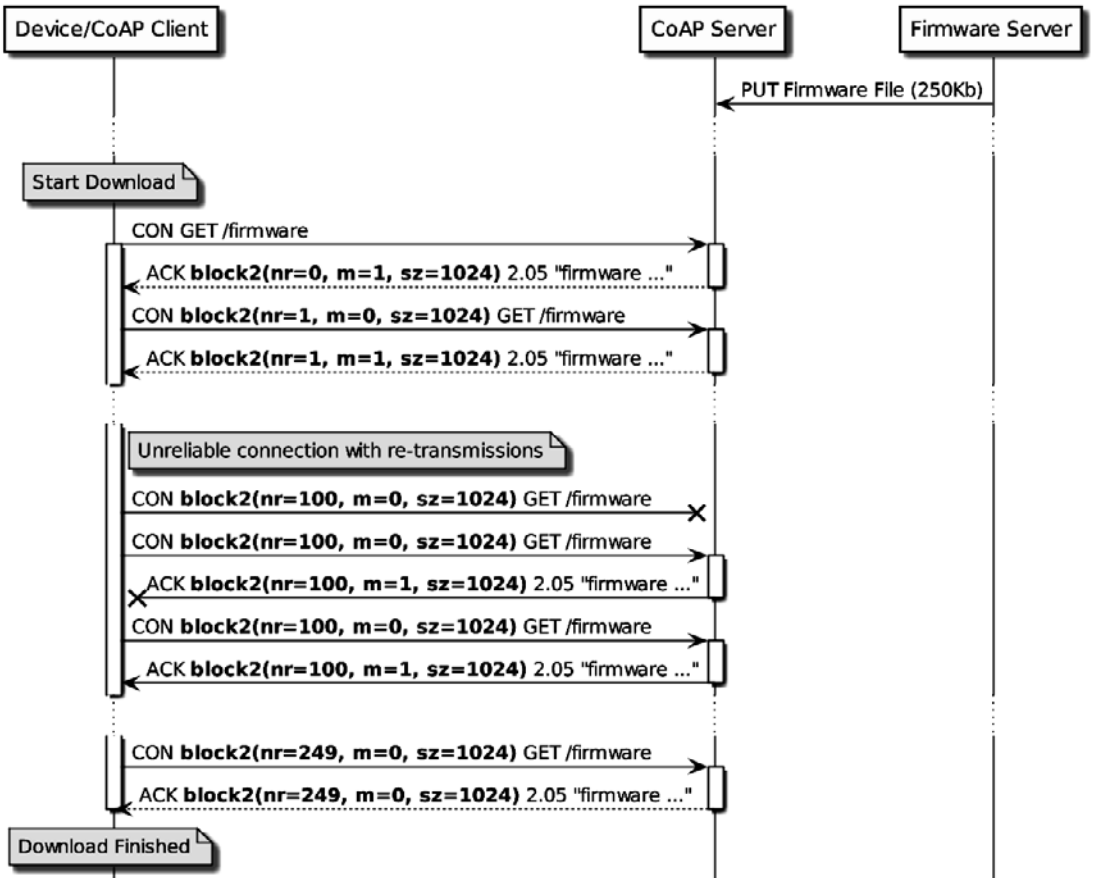

Fig. 9: CoAP fragmentation handling with block-wise transfer for large-size payload

CoAP message type	MQTT QoS level	Description
NON – Nonconfirmable	0 – At most once	Unreliable delivery; messages may be lost
CON – Confirmable	1 – At least once	Reliable delivery; messages will not be lost, but may be duplicated

Table 2: CoAP and MQTT QoS mapping

OMA Lightweight M2M (LWM2M)

Defined by the Open Mobile Alliance, LWM2M is built on the CoAP protocol, creates a specific CoAP profile, adds the object model for data representation on devices, and enables efficient binary payload.

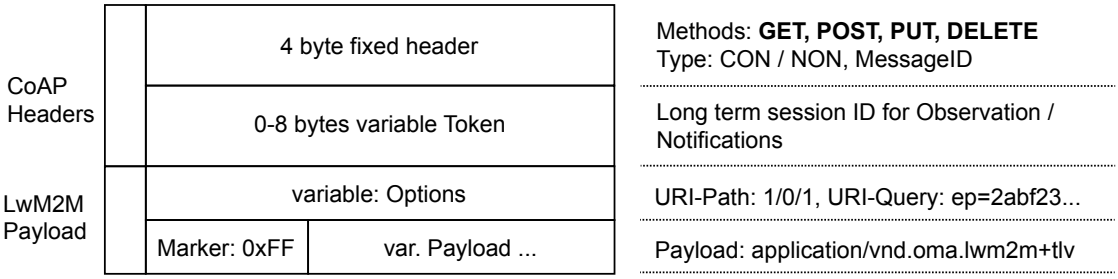

Fig 10: LWM2M fields mapped on CoAP message format

Fig. 10 shows how LWM2M reuses CoAP headers and maps all LWM2M-specific information into the LWM2M payload (URI parameters, specific LWM2M content formats, etc.). Everything else is encoded in the payload in an efficient binary manner.

LWM2M 1.1[5] mandates plain text, opaque binary, and CoRE Link data formats for client implementations to ensure on-the-wire data transmission efficiency and recommends implementing one of the two new data formats based on Sensor Markup Language (SenML) using either JSON or Concise Binary Object Representation (CBOR). Previously mandatory in LWM2M 1.0 type-length-value (TLV) and optional, JSON is not recommended for use in LWM2M 1.1.

Data format	IANA media type	CoAP content-format	Client mandatory
Plain text	Text/plain	0	1.0 optional, 1.1 mandatory
CoRE Link	Application/link-format	40	1.0 optional, 1.1 mandatory
Opaque	Application/octet-stream	42	1.0 mandatory for firmware downloads, 1.1 mandatory
TLV	Application/vnd.oma.lwn2m+tlv	11542	1.0 mandatory, 1.1 not recommended
JSON	Application/vnd.oma.lwm2m+json	11543	1.0 optional, 1.1 not recommended
SenML JSON	Application/senml+json	110	1.0 N/A, 1.1 recommended
SenML CBOR	Application/senml+cbor	112	1.0 N/A, 1.1 recommended

Table 3: LWM2M 1.1 data formats

LWM2M uses a flat object/resource model to describe sensor/actuator model for devices, where a client device contains data structures mapped to the device architecture and described by an object definition. Each object may have zero or more instances, and each instance contains resources, which can contain values and be readable, writable, or executable. LWM2M defines a set of objects; however, the objects may be defined outside of the LWM2M specification and used to model sensors and actuators of a device.

Below is a simple example of a smart lighting controller containing sensor and switch/dimmer objects:

ID	Name	Operations	Type	Range	Units	Description
5850	On/Off	RW	Boolean			Turn the light on or off
5851	Dimmer	RW	Integer	0–100	%	Light dimmer setting
5852	On Time	RW	Integer		sec	Time in seconds that the device has been on
5750	Application Type	RW	String			Application type, e.g., “Smart Dimmer”

Table 4: Example of the definition of a smart switch actuator object and resources (object URN: urn:oma:lwm2m:ext:3306)

Related LWM2M/CoAP payload could then be:

Light dimmer actuator LWM2M payload example (human readable contents of the binary TLV):

[URI-Path: /3306/0] // CoAP Path-Identifier of the Actuator object with the ID 3306 and instance 0

[Lightweight M2M TLV]:

- 5850: 1 // Light is on - value of resource 5850
- 5851: 73 // Dimmer is at 73% setting - value of resource 5851
- 5852: 1800 // Light has been on for 30 minutes - value of resource 5852
- 5750: “Smart Dimmer” // Application “Smart Dimmer” - value of resource 5750

ID	Name	Operations	Type	Range	Units	Description
5700	Sensor Value	R	Float		Defined by units resource	Current value of the luminosity sensor
5601	Min Measured Value	R	Float	0–100	%	Minimum measured value since port ON or reset
5602	Max Measured Value	R	Float		sec	Maximum measured value since port ON or reset
5605	Reset Min and Max Values	E	Opaque			Reset min and max measured values to current value
5701	Sensor Units	R	String			Type of sensor units, e.g., "lx" for Lux

Table 5: Example of the definition of a luminance sensor object and resources (object URN: urn:oma:lwm2m:ext:3301)

Light luminance sensor LWM2M payload example (human readable contents of the binary TLV):

[URI-Path: /3301/0] // CoAP Path-Identifier of the sensor object with the ID 3301 and instance 0

[Lightweight M2M TLV]:

- 5700: 250 // Luminance is 250 Lux - value of resource 5700
- 5601: 27 // Minimum value was 27 Lux - value of resource 5601
- 5602: 410 // Minimum value was 410 Lux - value of resource 5602
- 5701: “lx” // Measurement unit is Lux - value of resource 5701

The sensor/actuator LWM2M control flow (using the underlying CoAP protocol) may look like the following sequence:

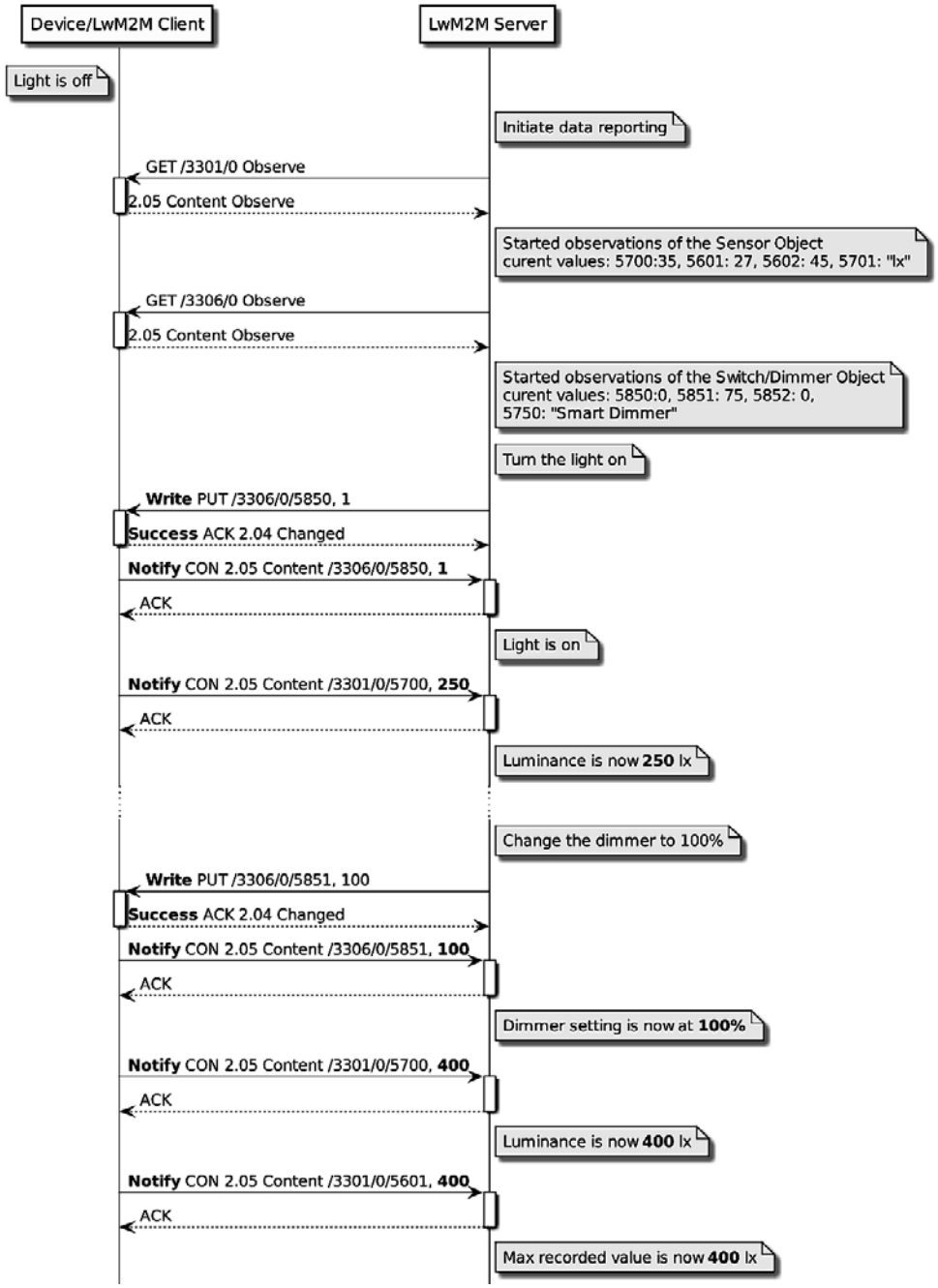

Fig. 11: Example of a sensor/actuator communication sequence managing a smart light device via OMA LWM2M/CoAP

In summary, it can easily be seen that:

- LWM2M does not introduce any header overhead to the underlying CoAP protocol.
- LWM2M-specific header and payload values are very lightweight and usually coded as binary (JSON and text payloads are optional).

Therefore, in the overhead and traffic analysis, we do not need to specifically separate CoAP and LWM2M.

Note, however, that whereas LWM2M uses mostly binary payload and headers (options), even for text payloads such as JSON, the resources and values are coded as integers. CoAP generally does not limit the formats, which can contain textual URI paths and text-based payloads, such as JSON and XML.

MQTT AND COAP TRAFFIC ANALYSIS

Packet overhead

The basic header structure and length for both MQTT and CoAP messages are very similar and lightweight. While CoAP requires a minimum of 4 bytes of headers, including the MessageID, the MQTT NOTIFY control packet also requires a minimum total of headers of 4 bytes (2 bytes for the fixed header and 2 bytes for the Packet Identifier). For all protocols, MQTT topic and CoAP URIs are considered to belong to the payload.

MQTT requires an ordered lossless packet delivery and relies on the TCP as the underlying transport.2 User Datagram Protocol (UDP) and other connectionless network transports, such as NB-IoT, are not suitable for MQTT due to the possibility of data loss and packet re-ordering. Therefore, using MQTT creates packet overhead from both IP and TCP protocols.

CoAP, on the other hand, is designed to use UDP transport, does not require lossless and ordered delivery, and provides higher protocol level fragmentation and flow controls. CoAP can be seamlessly deployed on any datagram transport, such as UDP, SMS, and NB-IoT (including NIDD).

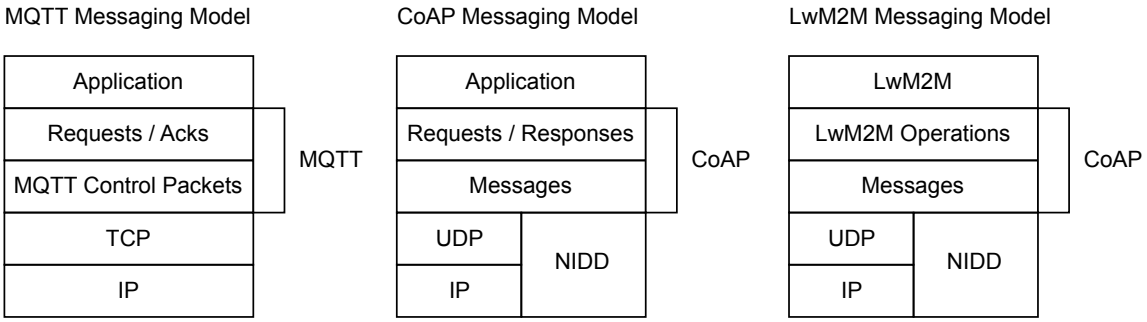

Fig. 12: Comparison of MQTT and CoAP messaging models

MQTT involves complete packet overhead of TCP/IP packets (20 byte minimum for IPv4 headers and 40 bytes for IPv6). The TCP layer requires a minimum of 24 bytes for the first two handshake messages and 20 bytes for the final ACK. Plus, teardown requires four messages with a minimum of 20 bytes each. Additionally, each TCP ACK will contribute an additional 20 bytes. In contrast, UDP packets only have a total of eight bytes each. When CoAP/LWM2M is deployed over the NIDD transport, the UDP/IP stack is completely eliminated, providing more space for the payload data (Fig. 13).

Considering that in most cases each data transmission (e.g., MQTT NOTIFY control packet) requires a TCP ACK (20 bytes), the CoAP/LWM2M savings will be even higher.

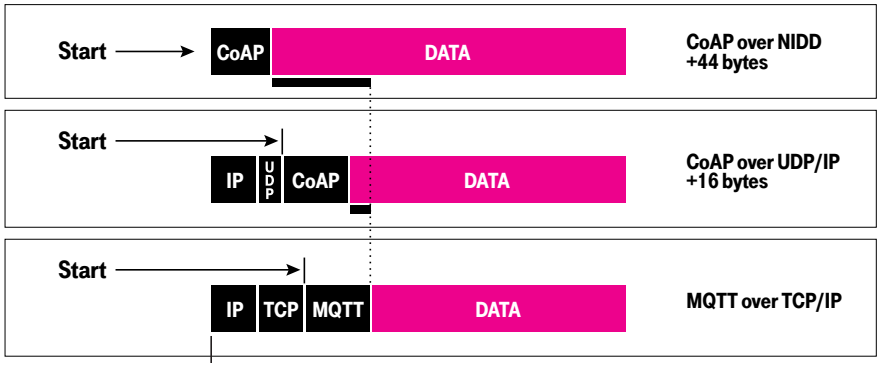

Fig. 13: Header overhead for IoT protocols over NB-IoT

Transmission overhead

Each TCP connection requires a three-way handshake with a total of three messages and approximately 130 bytes, whereas teardown requires four messages with a total of 160 bytes. Additionally, each MQTT packet will be followed by a TCP ACK (20 bytes each).

MQTT initially requires the client to initiate a connection to the MQTT broker with two messages: PUBLISH and PUBACK. After the MQTT connection has been established, the client can publish data by sending PUBLISH messages to the server. MQTT has three QoS levels, and the most typical QoS1 (at least once delivery) for reliable transmission will acknowledge each data publication with the PUBACK MQTT control packet.

The overall sequence with the overhead budget to send one piece of data is shown on Fig. 14. If data reporting is infrequent and requires TCP and MQTT to reestablish connections each time, then the relative overhead becomes huge for small data payload. For example, the estimated minimum overhead will be 352 bytes (not counting MQTT PUBLISH/PUBACK control packets), which would be more than 700 percent higher relative to the 50 byte payload. Clearly, this would be very expensive for applications where devices sleep most of the time, then wake up to report small payload, and go to sleep again. Additionally, MQTT provides a mechanism to keep the TCP connection alive. However, the keep-alive packets are to be initiated by the client, which will require the devices to wake up just for the sake of sending the keep-alive packet.

CoAP/LWM2M, on the other hand (Fig. 15), does not require any connection or session setup on the TCP/IP level. LWM2M provides a registration interface that requires the device client to register with the server. This will only require two CoAP messages (i.e., UDP datagrams). In addition, the server needs to send the first request to start observations, after which the client starts sending notifications asynchronously. LWM2M also mandates the acknowledgments (CoAP confirmable mode where a CoAP CON message is followed by the CoAP ACK message) for all LWM2M operations. It is not mandated for the Notify (LWM2M information reporting interface) operations but is highly recommended since NB-IoT does not prevent packet loss.

This leads to a very light transmission overhead for the CoAP data, even accounting for the registration and deregistration, for a total of approximately 20 bytes. Additionally, the client is not required to deregister as soon as it wakes up before the registration lifetime expires.

Experimental measurement of various application data transmission protocols over live NB-IoT network in various conditions is presented in Fig. 16. For good network conditions with low latency and low packet loss, all connectionless transports (raw UDP and CoAP) require only one message transmission, and the payload overhead is very low due to the CoAP headers. The overhead for the MQTT/TCP/IP stack would be approximately 10 times higher due to TCP and MQTT handshakes and ACKs.

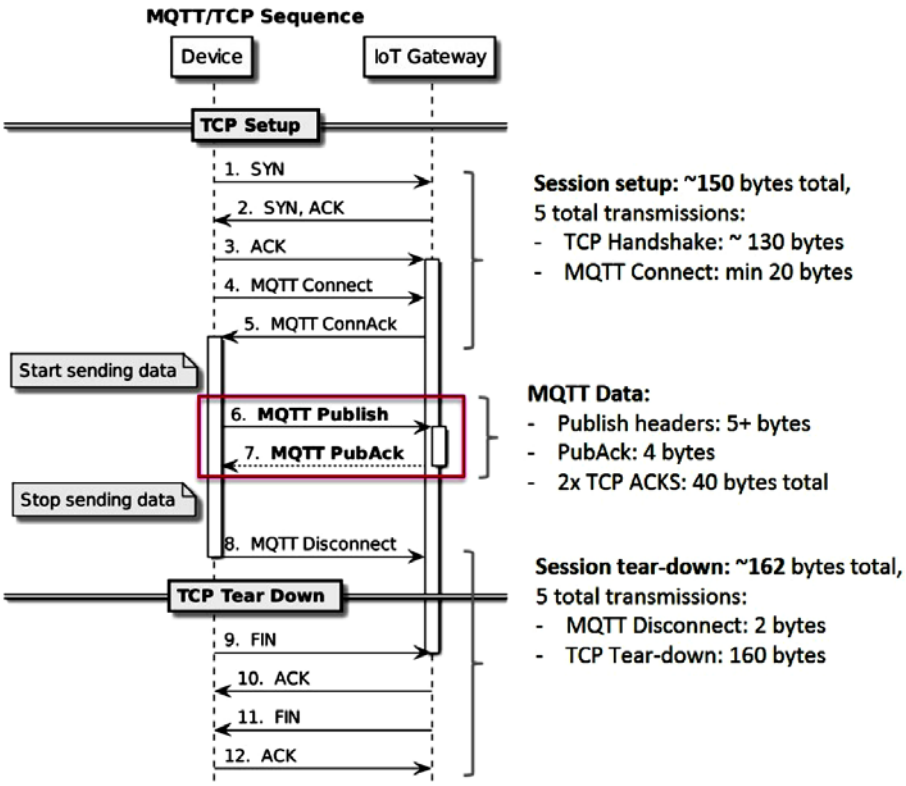

Fig. 14: MQTT/TCP single notification round-trip data, no TLS, no keep-alive, QoS1 messages: 312 bytes registration overhead

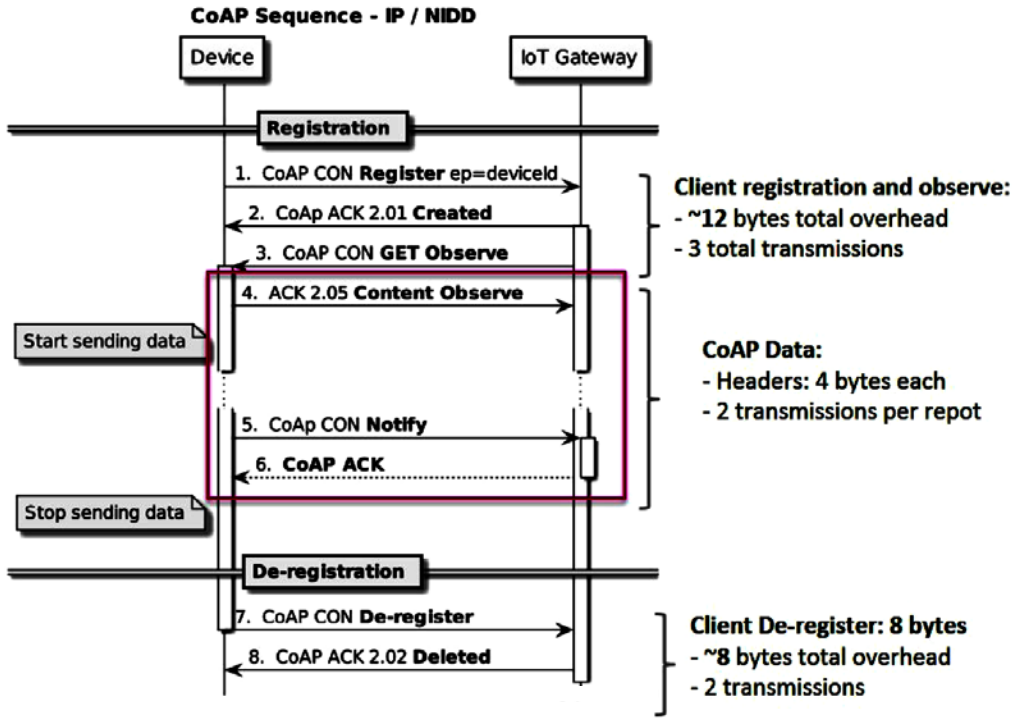

Fig. 15: CoAP single notification round-trip data, no TLS, CON messages: 20 bytes registration overhead

With degraded radio conditions and higher packet loss, connectionless transport overhead remains the same, but packets now start to get lost. CoAP overhead increases due to the increased number of retransmissions; however, the data does not get lost due to the CoAP confirmable message type mechanism. Overhead for MQTT/TCP would slightly increase due to the need to retransmit packets.

Overall, MQTT/TCP traffic requires approximately 10x more transactions and the data overhead is approximately 100x higher compared to the CoAP/UDP traffic for the 5-byte payload (Fig. 16).

Fig. 16: Application-layer payload and transaction summary in live NB-IoT network – 5-byte payload

CONCLUSIONS AND RECOMMENDATIONS

Both NB-IoT and CoAP have been designed to meet the requirements of resource-constrained IoT devices communicating over constrained transmission channels. They seamlessly complement each other for both IP- and non-IP data delivery.

LWM2M is based on CoAP and does not introduce significant overhead due to the LWM2M interface and data models. LWM2M inherits all the best characteristics of CoAP while adding lightweight efficient binary payload and path representation. It is ideally suited for NB-IoT, especially for applications where both data transmission and device management functionalities are required.

We offer the following recommendations when selecting application-level data transport protocols for IoT solutions:

- CoAP/LWM2M is the preferred method for IP data transmission over NB-IoT networks.
- CoAP/LWM2M over NIDD offers superior transmission performance for both network capacity and IoT solutions. It can provide from 224 to 1,157 percent higher payload efficiency compared to the TCP/IP-based MQTT.
- CoAP can be used as a standalone transport over NB-IoT if the requirements for data transmission are significantly different than the requirements for data management, and the application is trying to avoid the overhead of additional LWM2M features.
- We do not recommend using MQTT over NB-IoT networks due to very high overhead and increased number of data transactions.
- MQTT is appropriate for relatively long-standing MQTT/TCP/IP connections that transmit relatively high volumes of data (e.g., telematics applications). For IoT solutions that require MQTT transport, other types of radio technologies, such as LTE, should be considered.

Contact information

For any questions or comments, please reach out to IoTDeviceManagement@T-Mobile.com.

References

1. Y. P. E. Wang et al., "A Primer on 3GPP Narrowband Internet of Things," IEEE Communications Magazine, vol. 55, no. 3, pp. 117-123, Mar. 2017.
2. Eclipse IoT Developer Survey 2018.
3. IETF RFC 7959 – Block-Wise Transfers in the Constrained Application Protocol (CoAP), 2016.
4. MQTT version 3.1.1 OASIS Standard, Dec. 2015.
5. Lightweight Machine-to-Machine Technical Specification, Approved Version 1.1 – August 2018, Open Mobile Alliance.

Acknowledgments

Sergey Slovetskiy, Principal Engineer, Systems Design and Strategy
Jeff Ahmet, Principal Engineer, Technology Development & Strategy
Karthik Iyer, Principal Engineer, Device Technology

Battery life and cost savings compared to Cat-M modules and IoT plans. Coverage not available in some areas.

T-Mobile®