Expertise
and insight
for the future

Sy Hoang Mai

Developing Low-power Cellular IoT Solution with Narrowband IoT and Lightweight M2M

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

16 December 2019

Author	Sy Hoang Mai
Title	Developing Low-power Cellular IoT Solution with Narrowband IoT and Lightweight M2M
Number of Pages	60 pages + 2 appendices
Date	16 December 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Smart Systems
Instructors	Keijo Länsikunnas, Senior Lecturer Toni Rosendahl, Thesis Instructor

The rise of the IoT era has witnessed the emergence of new and disruptive Low Power Wide Area technologies. One of those innovations, Narrowband Internet of Things (NB-IoT), is a new standard specified in the 3GPP project to enable a wide range of IoT cellular applications by focusing on extended coverage, high-density deployment, low energy consumption, and low-cost end device. Furthermore, this wireless technology is a promising replacement candidate for legacy cellular M2M systems, which will be slowly phased out by the 2G and 3G sunset.

Lightweight Machine to Machine (LwM2M) is a new device management protocol aspiring to go beyond the current de facto IoT messaging MQTT. This new framework offers a standardisation in IoT device management and information reporting, thus promoting a high-level of interoperability among applications and cloud services. Furthermore, Lightweight M2M over UDP presents a significant advantage compared to MQTT over TCP for low power cellular devices.

This paper documents the process of developing firmware for an IoT device utilising the two technologies mentioned above, and at the same time, supplies relevant knowledge and analysis on subjects encountered throughout the project execution.

The outcome of this thesis is a functional proof-of-concept low power IoT device, which is capable of delivering sensor measurements to an LwM2M server securely via NB-IoT. The thesis is going to be used as a reference design at Etteplan Embedded Finland Oy to speed up future developments.

Keywords	IoT, NB-IoT, LwM2M, STM32, Mbed OS, Low power

Contents

List of Abbreviations

API	Application programming interface
BLE	Bluetooth Low Energy
CI	Continuous integration. A practice of merging all developers' works to a shared mainline, often automated nowadays.
DTLS	Datagram Transport Layer Security. A protocol providing security for datagram-based communications.
DUT	Device under test. Refers to the device that undergoes a testing procedure.
eDRX	Extended/Enhanced Discontinuous Reception. This feature in NB-IoT implies the mechanism of extending the cycle between paging attempts.
IDE	Integrated development environment. An application offering a set of comprehensive tools for programmers to develop software.
IPSO WG	"Internet Protocol for Smart Objects" Working Group. An organisation focusing on promoting global interoperability of IoT device based on open standards.
IWDG	Independent watchdog. A hardware element which resets the system if not refreshed after a determined period, often used as the last line of defence to rescue the system from unexpected software failure.
IoT	Internet of Things. Generally, this term identifies anything that has a direct or indirect connection to the Internet. In this thesis, this term addresses embedded devices with such characteristics.
LPWAN	Low power wide area network. A type of wireless communication technology designed for low-powered, long-range communications.
LTE	Long Term Evolution. A cellular standard specified by the 3GPP project, commonly known to consumers as 4G technology.

LoRa	A propriety spread spectrum modulation technique developed by Semtech
LoRaWAN	An LPWAN technology developed by Semtech based on LoRa technology.
LwM2M	Lightweight machine to machine. A protocol specified by the Open Mobile Alliance for Machine to Machine communications and IoT devices management.
MCU	Microcontroller unit
M2M	Machine to Machine
MQTT	MQ Telemetry Transport. A popular standard messaging protocol for Machine to Machine in IoT applications.
MTC	Machine-type communication. A synonym for Machine to Machine communication.
MTU	Maximum transmission unit. This value specifies the maximum size of an IP packet can be transported via a medium without fragmentation.
NAT	Network address translation. A method of mapping one IP address space into another by modifying the IP header, widely utilised as the way to conserve the address space from the IPv4 exhaustion.
NB-IoT	Narrowband Internet of Things. A low power wide area network technology specified by 3GPP, focusing on serving IoT applications.
OMA	Open Mobile Alliance. A standards body which develops open standards for the mobile phone industry.
PAT	Port Address Translation. An extension of NAT, enabling multiple devices in a local network to be mapped to a single public IP address.
PSM	Power Saving Mode. In the LTE context, the term indicates the sleep mode of the UE during which it exhibits the lowest current consumption.

RRC	Radio Resource Control
RTOS	Real-time operating system. An operating system intended to serve applications with real-time demand, widely used in embedded system projects.
TAU	Tracking area update
TCP	Transmission Control Protocol. A session-oriented communication model of the IP stack, providing an ordered and reliable communication scheme for Internet applications.
UART	Universal Asynchronous Receiver/Transmitter. A hardware component used for asynchronous serial communication, often integrated within the microcontroller.
UDP	User Datagram Protocol. A connectionless communication model of the IP protocol, mainly used by low-latency and loss-tolerating applications.
UE	User equipment. In the cellular context, this term implies any device used by an end-user to communicate with the network.
URC	Unsolicited Result Code. A message sent from the mobile equipment which is not an immediate result of an AT command, used for delivering an arbitrary event (e.g. modem has received a call) or result code of an asynchronous operation.
WWDG	Window watchdog. This component resets the system if not refreshed within a specific time window. Similar to the independent watchdog, it is used to rescue the system from an unexpected failure.

1 Introduction

The emergence of the Internet of Things (IoT) is considered to be the next revolution in data communication with a mission of forming an ecosystem in which each and every device is connected and able to make intelligent decisions. This newly emerged capability not only offers improvements to existing automation and manufacturing industries but also advances other fields such as agriculture, transportation, and healthcare by providing means for optimisation in efficiency and flexibility while cutting down excess expenses [1]. According to an Ericsson forecast [2, p. 8], by 2024, there will be approximately 22 billion IoT devices connected to the Internet. Consequently, this tremendous number of connections urges for new versatile and scalable wireless technologies that met the demand for future growth and changes in the IoT world.

Evidently, the characteristics of the connectivity technology have always played an essential role in deciding whether will it be adopted by the mass as commercial projects are driven by use-cases. For example, WiFi, a wireless protocol designed for low latency and high throughput communication, has become the preferred option for smart consumer devices thanks to its availability in virtually any modern home of the Western world. Unfortunately, this ubiquitous wireless protocol does not fit into applications where there is an appeal for low-power consumption and long radio range support along with a high degree of scalability. On the other hand, a large proportion of the IoT world is constrained devices infrequently send out a small amount of data and sleep most of the time to conserve energy, while considering low latency as a non-critical attribute. There have been many attempts to design wireless technologies to satisfy these expectations, and one of those promising pursuits is Narrowband IoT (NB-IoT) – a recent extension of the LTE standard which seeks to provide connections for billions of IoT devices worldwide over cellular. At the moment, network operators around the world are starting to sunset their 2G networks, leaving out a great opportunity for this new technology enter the machine to machine (M2M) market.

The goal of this thesis was to document, describe and explain the knowledge needed as well as the process of developing an IoT device which utilises NB-IoT and Lightweight

Machine to Machine (also known as Lightweight M2M or LwM2M) from a software developer perspective. The remainder of the thesis is organised as follows: Section 2 presents the theoretical knowledge of LwM2M and NB-IoT. Section 3 describes how the project was carried out with technical depth. Section 4 states the project outcome and provides additional discussions and analysis on different aspects that emerged during the implementation phase. Finally, Section 5 concludes the paper.

2 Theoretical Background

This theoretical background section provides the audience with the fundamentals of device management, Lightweight M2M protocol, and Narrowband IoT to prepare a solid ground for later discussions. Reference materials comprise official Lightweight M2M specification, articles collected from journal databases such as IEEE, MDPI, and development support documents presented by U-blox since their cellular module is used in the project.

2.1 Device Management System

As the number of connected devices soars, so are the demand for organisations to manage, configure and monitor their device fleets. Nevertheless, a device management system is not a new concept as this technique was adopted by mobile and Internet operators many years ago to supervise and provision devices. A device management framework generally offers a basic set of functionalities:

- Provision devices: concerns the bootstrapping process which setup identity and first configurations of the device.
- Configuration: allows administrators to remotely change device settings and parameters.
- Update: provides a software update delivery mechanism for deployed devices.
- Fault management: delivers fault report in case of a system failure, which helps maintenance to be carried out quickly and efficiently as possible to minimise loss.

These functionalities are expected to be performed securely under appropriate access right configurations and authentication scheme. A device management protocol, which defines operations between devices and its administrators, can be examined through three different aspects [3]:

- Protocol architecture: depicts how messages are packed and transported across the communication channel.
- Connection dynamics: characterises the communication paradigm between server and client.
- Standardised data model: describes the data model shared between server and client to perform operations defined by the protocol.

In subsequent sections, LwM2M protocol's characteristics, along with its features, are discussed in more details following these aforementioned criteria.

2.1.1 Lightweight M2M Protocol Architecture and Functionality

Lightweight M2M is a new device management protocol defined by Open Mobile Alliance (OMA), whose goal is to provide a holistic solution for remote management and service enablement in sensor networks and M2M environment. Figure 1 describes the architecture of Lightweight M2M protocol along with a simplified representation of its communication model.

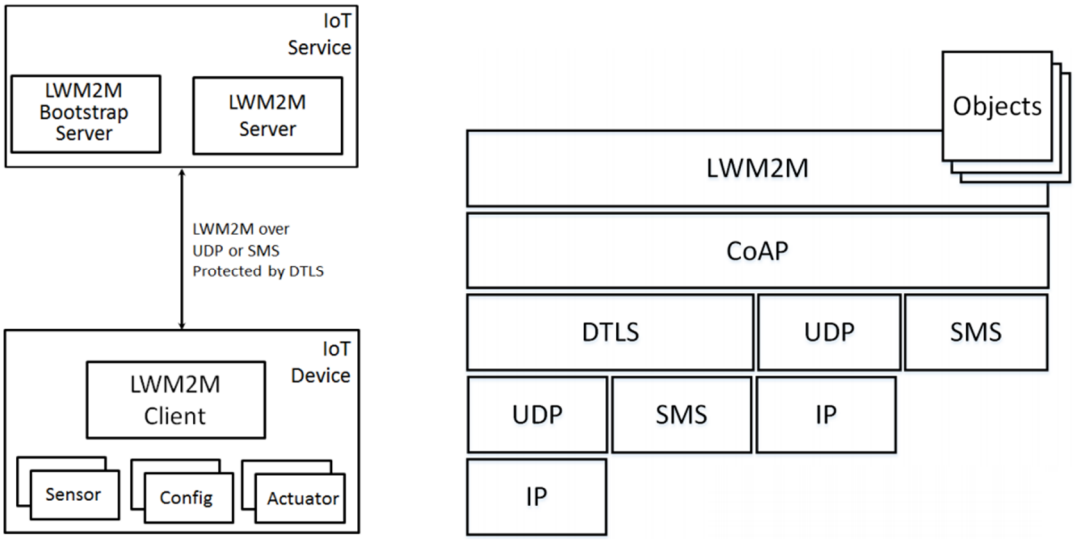

Figure 1. Lightweight M2M 1.0 architecture (right) and its simplified communication model (left). Copied from Mikko Saarnivala [4, p. 5]

As illustrated in Figure 1, the LwM2M layer lies on top of Constrained Application Protocol (CoAP), and below that are supported transport layers such as UDP, TCP, SMS, or LoRaWAN (TCP and LoRaWAN added in Revision 1.1) used for data transfer. This layering approach in the protocol stack separates responsibilities within the chain, making it transparent for developers who implement the LwM2M library as well as the library users.

Lightweight M2M protocol operates in a client-server model in which the server is the LwM2M server while clients are IoT devices. As the protocol adopts RESTful operations carried out over CoAP to perform its transactions, it is sensible to make the earlier clarification because from CoAP perspective LwM2M server is a CoAP client and managed device is a CoAP server. On the other hand, it is also worth noting that LwM2M protocol allows a device to register to and be managed by multiple servers, though this feature may introduce challenges for implementation and operation in practice. The LwM2M specification defines an object called Access Control Object, which specifies the permission each server has on a particular object or object instance within the client. For simplicity and conciseness purpose, the rest of this paper mostly refers LwM2M server in the singular form, assuming the management relationship consists of one server and one client. Figure 2 presents a high-level view of the data structured in a Lightweight M2M client.

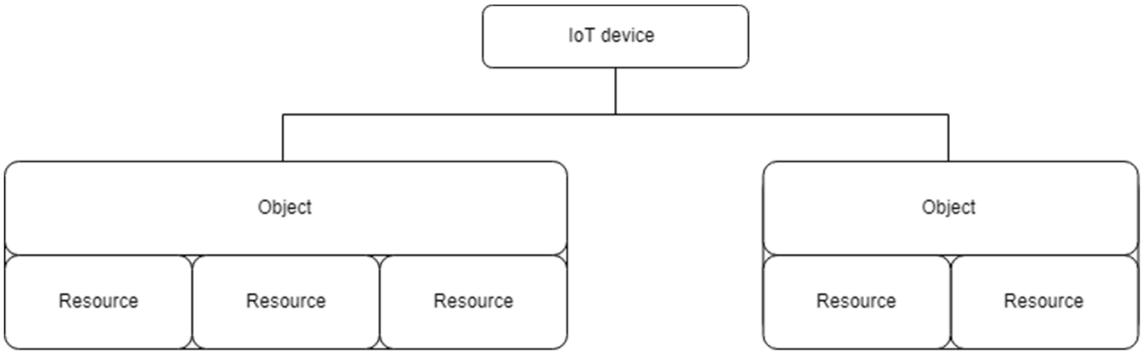

Figure 2. Data representation in lightweight M2M client data structure.

The LwM2M client, or the managed IoT device in this context, exposes its data to the server through a flat tree data structure. Each node of the tree is called an object, which can be single or multi-instantiable, consists of one or many resources. Each resource can take the form of a boolean, an integer, a string, an opaque or a method (action) that

may be read, written or executed. More depth on data representation and information reporting are covered in Chapter 2.1.4.

2.1.2 Device Management in Lightweight M2M

As a quick recap from earlier, LwM2M protocol operates in a client-server model, providing many IoT services including device bootstrapping as well as managing device state and collected data. An LwM2M server can serve multiple devices at the same time. On the other hand, a device can be managed by multiple servers, yet this is not a popular approach at the moment. This sub-section examines the bootstrap procedure, device management functionality and fault reporting process within the protocol.

2.1.2.1 Lightweight M2M Bootstrap Procedure

Bootstrapping is a term often refer to the process in which the first invoked program loads and executes more extensive programs, acting as a kickstart of the computer booting sequence. However, the bootstrap procedure in LwM2M does not imply this operation but instead denotes the act of which client retrieves useful information from a bootstrap server. This information contains the LwM2M server address and authentication credential needed for the device to later successfully perform a registration. The bootstrap server is, in fact, an ordinary LwM2M server but its sole duty is to distribute necessary information so that clients can connect and register to the right server. There are four different bootstrap modes: factory bootstrap, smart-card bootstrap, client-initiated bootstrap, and server-initiated bootstrap [5, pp. 19-23]. Factory and smart-card bootstrap are as descriptive as it sounds, suggesting the necessary information is provided during factory provisioning or coming from a smart-card. On the other hand, server-initiated bootstrap means the bootstrap server needs to push the required data to the client, but how does it knows the client need bootstrapping falls on implementation-specific. This subsection only explains the client-initiated bootstrap procedure as it is the most straightforward bootstrapping method. In this bootstrapping manner, essential information to connect to the bootstrap server is preloaded on the device, usually via factory provisioning or hard-coded inside the firmware. Figure 3 shows the process of a device-initiated bootstrap follows with registration to a device management server.

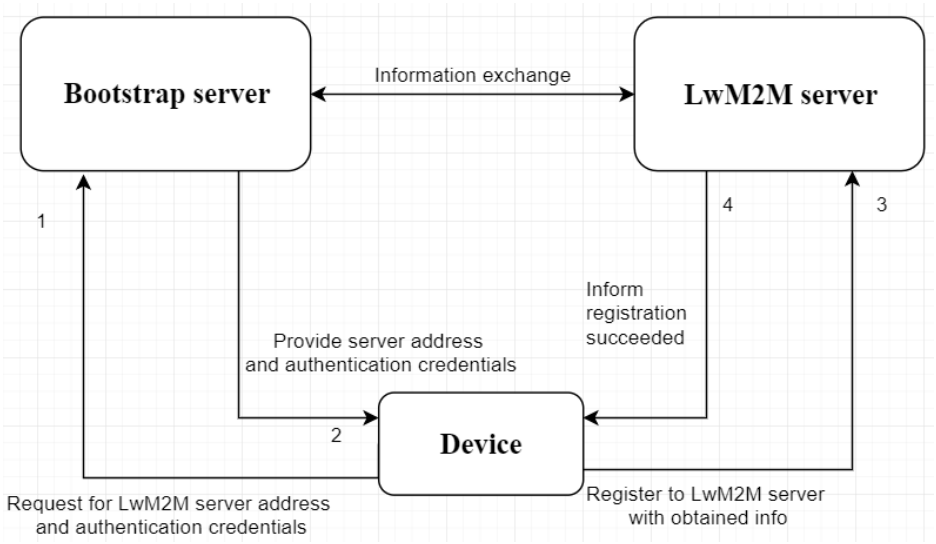

Figure 3. The process of a device-initiated bootstrap follows with registration to a device management server.

Figure 3 describes the bootstrapping process of an IoT LwM2M device as follows [6]:

1) The device uses its pre-provided information stored in non-volatile memory to contact the bootstrap server. This information contains the bootstrap server address and security credential needed for authentication.

2) After connected to the bootstrap server, the device receives new information about the upcoming LwM2M server and a new security credential for authentication.

3) The device disconnects from the bootstrap server and uses the newly obtained information to register to the LwM2M server.

4) Upon validated the client identity, the server sends an acknowledgement confirming that the registration has succeeded. The device is now considered registered and managed.

Even though this bootstrap procedure is not mandatory, there are many benefits to be derived from this feature, such as:

- The final bootstrap is executed after factory provisioning. As a result, the latest credentials in effect are not factory provisioned, thus lowering the risk of compromising device credentials during the manufacturing phase. In case devices failed to bootstrap due to their identities claimed, it might indicate that there is a security hole within the manufacturing process in which device credentials are leaked.

- The manufacturer has an additional opportunity to detect defective devices before shipping by checking that the bootstrap procedure succeeded, hence verifying basics operation and connectivity of devices.

- After the device has been deployed on the field, the bootstrapping mechanism becomes an effective way to do re-keying, or merely redirecting the device to a different server. This feature will come in handy when the device credential is known to be compromised, or current managing server is going down for maintenance, or an ownership change of devices is expected (meaning devices are to be managed by another organisation's server).

In summary, judging from the benefits coming from the bootstrapping feature, it is highly recommended to implement client-initiated bootstrapping functionality for the IoT device unless there are reasons to choose otherwise.

2.1.2.2 Device Monitor, Binding Mode and Fault Report in Lightweight M2M

Within the realm of LwM2M, a client needs to be registered to a management server to report its status and collected data. The specification defines a connection parameter called "Lifetime" which cites how long the client registration remains valid. In order to maintain a session, a client must renew its registration before this timer runs out, and such action is called a "Registration Update". When the server receives such registration update from a client, the Lifetime timer for that particular session is refreshed, and the client must do another update before the next deadline, and so on. In case the client misses the deadline, it is considered deregistered from the server and its session invalidated, thus, the client must redo the whole registration procedure again. Often this incident signals that the device needs maintenance from a power outage, connectivity issue, hardware malfunction or software defect. Consequently, developers would like to consider an appropriate lifetime value for their devices which is harmony between how quick a downtime can be detected and how high is registration update frequency, a trade-off among quality of service and power consumption. Figure 4 illustrates the agenda of a client registration and a registration update between a client and an LwM2M server.

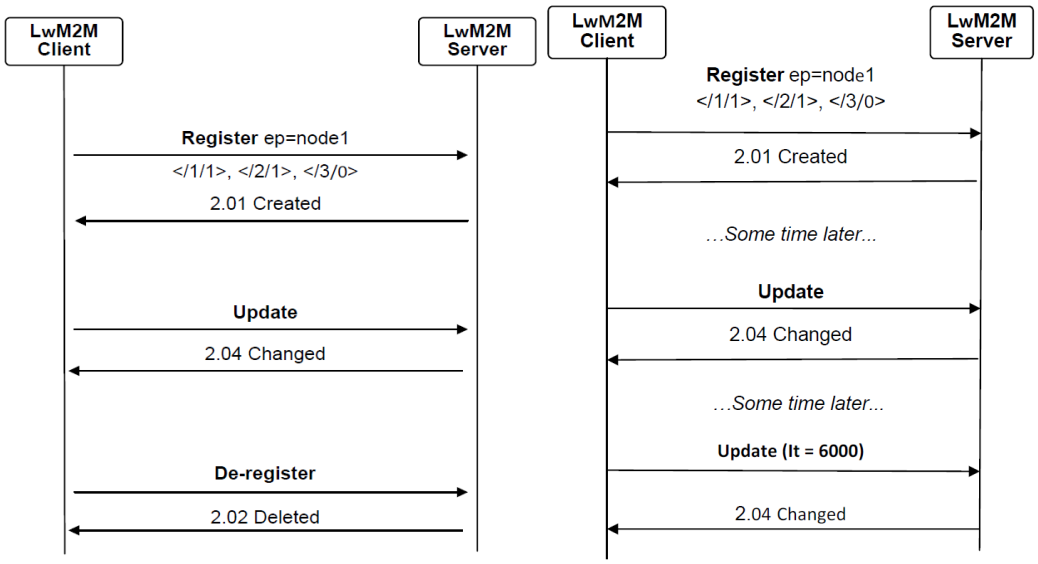

Figure 4. Client registration (left) and client registration update (right) flows. Copied from OMA Lightweight M2M specification [5].

At the beginning of a registration procedure, the client presents the server with its “End-point Name” (also known as client name), “Lifetime”, “LwM2M version”, “Binding Mode” (not mandatory), and “Object Instances List”. All these connection configurations are provided by the client. Commonly, the client attempts a registration update to renew its session, but such operation can also be used to change the current lifetime value, or signify an update on the object instances list in case objects or instances were added or removed.

The binding mode is an important parameter determining the behaviour of the connection between client and server, which certainly play a significant role in power-constrained applications. This parameter specifies the transport binding (e.g. UDP, SMS) along with whether the “Queue Mode” option is applied. Queue mode is an interesting feature which benefits low-powered IoT devices that sleep most of the time and may only be reached during a short time window. As stated in the LwM2M specification, Queue Mode requires the server to queue its requests when the client is unreachable and send them out as it is reachable again. The client may notify the server that it is now awake by sending a registration update, thus they can exchange messages for some amount of time before the client goes to sleep and again unreachable. As a result, the most appropriate binding mode for low powered IoT devices is likely to be UDP with Queue Mode.

Furthermore, the LwM2M specification defines a mandatory “Device” object with the ID of 3, which is designated to report a set of generic information of the device, including battery level, power-source voltage, memory-free, error code, to name a few. In case the predefined object is not sufficient for a particular reason, a private organisation can define its fault reporting object, or simply reuse a standard data reporting object. From a particular viewpoint, the device status is just another sensor value to be reported to the cloud, and it is up to the cloud server to make sense of it and determine the appropriate action, for example informing the operator about the low-battery state of the device.

2.1.3 CoAP in Lightweight M2M Service Enablement Interface

To grasp Lightweight M2M communication in-depth, one should know about CoAP characteristics as this protocol operates at one layer below LwM2M. In CoAP, messages are exchanged asynchronously between endpoints, often via unreliable transports like UDP in which data might fail to be delivered or arrive in an out-of-order manner. To counter this intrinsic drawback, this protocol defines two lightweight reliability ensuring methods:

- Stop-and-wait retransmission with back-off time for confirmable message.
- Duplicate detection with message ID.

Figure 5 gives an example of the differences between reliable and unreliable transmission.

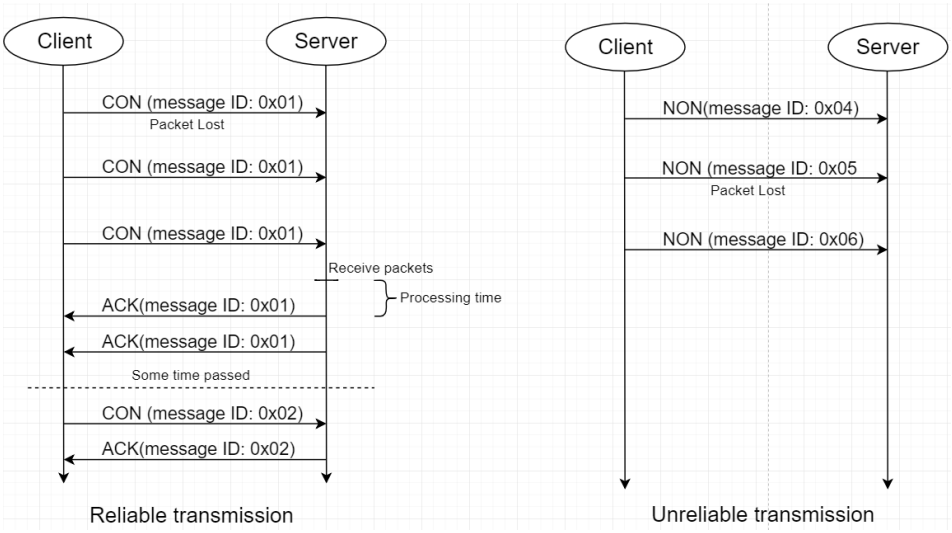

Figure 5. Illustration of reliable and unreliable transmission in CoAP.

There are four types of message in CoAP based on transport behaviour: confirmable (CON), non-confirmable (NON), acknowledgement (ACK), and Reset (RST).

- CON message requires an acknowledgement message (ACK) from the recipient. The sender will retransmit the same message at an exponential increase interval until the max number of attempts reached, or a matching reply is received, either it is a RST or an ACK.
- RST message is sent as a reply by the recipient in case it receives an empty, or unprocessable message due to lack of context. This behaviour holds in both cases where the orphan or invalid message is either CON or NON message.
- NON message does not require an acknowledgement from the recipient. This message type is particularly useful in case a particular piece of data (e.g. room temperature) needs to be sent at a regular interval. To increase the delivery rate, the sender can send out multiple copies of the same message. Though this approach indeed increases the reliability of messages, it also causes an increase in the network load.

Regarding the request/response semantics, CoAP operates with a client-server architecture and supports four basic methods: GET, POST, PUT and DELETE which is similar to HTTP's scheme. Furthermore, CoAP also supports the use of URI to enable access to associated information on the device. As a result, it is feasible to make a CoAP device operate as if it is a simple web application with the help of a CoAP-HTTP proxy.

2.1.4 Lightweight M2M Resource Model and Information Reporting

A Lightweight M2M client consists of a set of objects, each of which contains multiple resources identified by unique IDs. These resources together form an interface for LwM2M server to acquire data from its clients. Both client and server need to have a consensus on the data type (e.g. string or float) of resources to avoid misinterpretation. A resource within an object can be addressed via a URI in the {Object_ID}/{Object_Instance_ID}/{Resource_ID} format. The semantics of these IDs are determined in advance, thus the server can map or interpret the incoming data appropriately in the light of the circumstances.

The IPSO Smart Objects Working Group (IPSO WG), a joined force between OMA and IPSO alliance, proposes a list of LwM2M objects called IPSO smart objects as an attempt to standardise object models used for data reporting, providing a high level of interoperability for services and devices using LwM2M protocol [7] [8]. The data model of an LwM2M object comprises four parts:

- Object representation (Semantic)
- Data types
- Operations (Read/Write/Execute)
- Content format

Let us examine the IPSO temperature object definition as an example. Figure 6 presents the IPSO Temperature object definition.

Object definition

Name	Object ID	Object Version	LWM2M Version
Temperature	3303	1.0	1.0
Object URN		Instances	Mandatory
urn:oma:lwm2m:ext:3303		Multiple	Optional

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
5700	Sensor Value	R	Single	Mandatory	Float			Last or Current Measured Value from the Sensor
5601	Min Measured Value	R	Single	Optional	Float			The minimum value measured by the sensor since power ON or reset
5602	Max Measured Value	R	Single	Optional	Float			The maximum value measured by the sensor since power ON or reset
5603	Min Range Value	R	Single	Optional	Float			The minimum value that can be measured by the sensor
5604	Max Range Value	R	Single	Optional	Float			The maximum value that can be measured by the sensor
5701	Sensor Units	R	Single	Optional	String			Measurement Units Definition e.g. C for Temperature in Celsius.
5605	Reset Min and Max Measured Values	E	Single	Optional				Reset the Min and Max Measured Values to Current Value

Figure 6. IPSO Temperature object definition. Copied from openmobilealliance.org [9].

As presented in Figure 6, the temperature object is assigned ID 3303, and a client can contain multiple instances of this object type. As the case may be, different instances of the same object will represent different temperature measurements acquired by the system, such as room temperature, device internal temperature, or dew point temperature. These object instances can be interpreted differently according to the client context and are out-of-scope of the IPSO objects definition. For the sake of explanation, assuming the object instance ID has a value of 0, then its sensor value can be accessed via "3303/0/5700". According to IPSO object definition, performing a read on this URI from the server-side would return the latest temperature measurement as a floating number. On the other hand, the device has to reject other operations performed on the said URI with an error code. Failing to follow this compliance can cause data misinterpretation or security risk (e.g. compromising the private key in the server object via a supposedly illegal read operation). Figure 7 presents the Interaction between LwM2M client and server on read, write and execute operation.

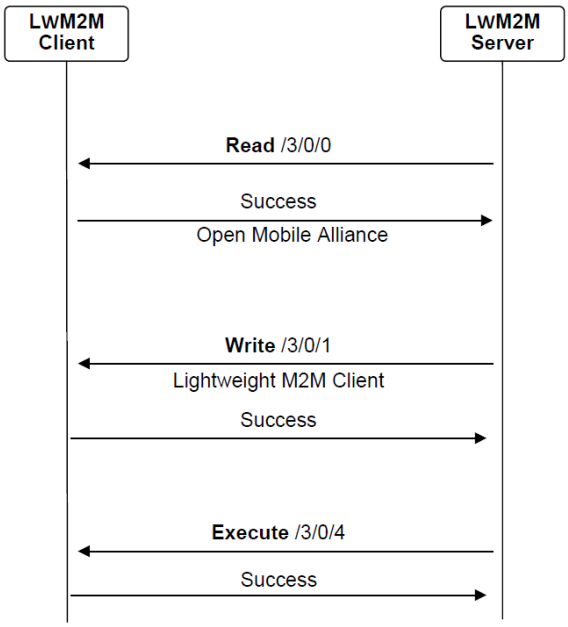

Figure 7. Interaction between LwM2M client and server on read, write and execute operation. Copied from OMA LwM2M Technical specification [5, p. 34].

Lightweight M2M defines three basic operations that can be performed on a resource: read, write, and execute. There are more sophisticated actions that can be performed, including create, delete, write-attribute, and discover which is not going to be discussed as they are advanced features which are not yet commonly used in embedded devices. As the name suggested, read operation allows reading the current value of resource or object, write operation changes a value on client-side, and execute operation will trigger pre-defined action on the client (e.g. device reboot). Figure 7 above describes how these operations are carried out in practice in an illustrative manner.

Lightweight M2M defines an information reporting mechanism to enable LwM2M server to keep track and get notified when new values are available on client-side, which is called “Observation”. The advantage of applying this observation pattern is once the server has subscribed to the client’s objects or object resources of interest, the client will voluntarily push changes to server when value update is available, thus eliminating the need for server polling for updates. The notification coming from client is an unreliable CoAP message; hence, it is not possible to detect if packets are failed to deliver in case the transport layer does not guarantee delivery (e.g. UDP). Despite the fact there might be no guarantee of delivery from the transport layer, the situation may not as disastrous as it sounds as the usual delivery rate is high enough for most IoT applications, and

CoAP does offer a reliability mechanism when needed as mentioned in Chapter 2.1.3. Section 4.5 discusses further the reliability of UDP and whether it is sufficient for IoT applications. Figure 8 provides an example of a (partial) LwM2M client consists of an Object of ID 0 and three instances of object ID 3303 (IPSO temperature object), each identified by its instance number.

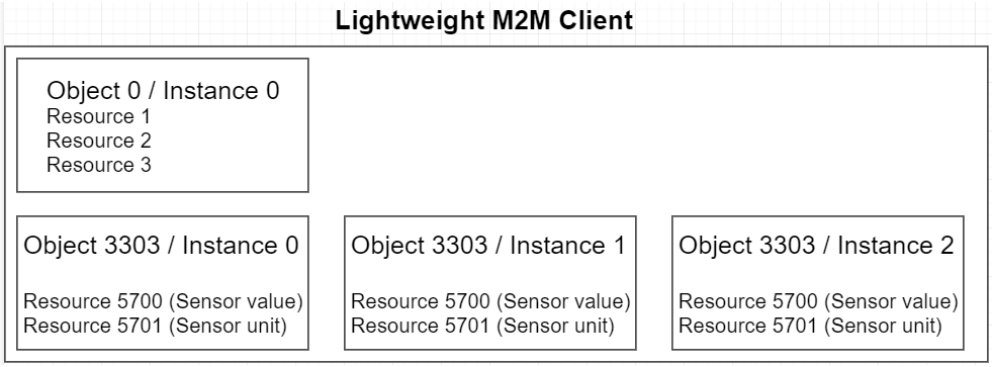

Figure 8. An example of a (partial) Lightweight M2M client.

To illustrate this observe and notify mechanism, assume a situation in which an LwM2M server would like to keep track of changes in an imaginary room temperature reported at URI "/3303/2/5700". Figure 9 describes the procedure of observing and notifying in LwM2M protocol.

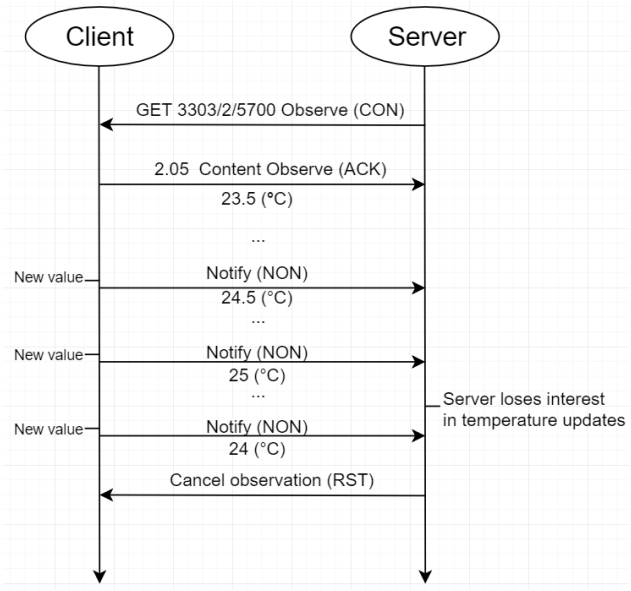

Figure 9. Information report via notification mechanism. Copied and modified from OMA LwM2M specification [5, p. 40].

First, the server initiates the observation by sending an observation request specifying the resource of interest, which will then be replied with an acknowledgement and the latest value of the resource. From that point, whenever the client has a new value update on the observed field, that value will be pushed to the server as a notification. Since there is no acknowledgement for a notification, it is straightforward to see that the notification should be idempotent to prevent unexpected complications. In the case of no longer having interest on the observation, the server can terminate this relationship by sending an “observation cancel” to the client after it receives a notification, indicating that it does not want to receive more notifications on that resource. Other existing observations are not affected by this cancellation.

Currently, there is no definition for notifying with acknowledgement in LwM2M, which are supposed to be useful on resources more important compared to others. As a limited workaround, the client can always send the same notification update more than once, maybe by pretending they are different consecutive updates with an identical value. As a matter of fact, a similar approach is used in Bluetooth Mesh to increase the probability for an unacknowledged message to reach its destination. However, this method should not be abused as it increases the device power consumption and may cause unnecessary load for the network.

2.2 Narrowband Internet of Things (NB-IoT)

This section first gives an introduction on NB-IoT, then later dives deeper into the fundamentals of this technology from an application developer perspective. Discussions focus on NB-IoT connectivity behaviour at a high level while describing the power consumption patterns associated with connection states. Hopefully, these analyses give an insight into how NB-IoT works as well as how it favours low power application. Furthermore, the last sub-section provides a list of best practices to minimise the device power consumption.

2.2.1 Cellular IoT in Internet of Things Landscape

When it comes to IoT connectivity options, traditional non-scalable wireless technology such as WiFi and BLE is not suitable because of their shortcomings in large scale deployment, which generally makes way for two other alternatives: mesh and cellular. These two approaches impose different advantages and disadvantages, therefore, the application requirements need to be taken into account when making decision on which technology to use.

Wireless mesh is the type of technology in which devices talk to their neighbours to form a network, making it feasible for outside-of-direct-radio-reach nodes to communicate. On the other hand, in the cellular world, network devices known as user entity (UE) have to talk with a base station, thus traffics will go through the network operator's system. Table 1 provides a concise comparison between wireless mesh and cellular.

Table 1. A brief comparison between wireless mesh and cellular. Modified from digi.com [10].

	Number of devices	Network characteristics	Device communication
Mesh	Many devices in the same location	Does not need cellular coverage Can be self-healing	Favours communication with network neighbours
Cellular	Only a few devices in the same location	Need network coverage	Devices communicate mostly with cloud server

As stated in Table 1, mesh networks are convenient for applications where many devices are at the same location, and mostly communicate locally. For example, Wi-SUN is a popular mesh solution for street lighting control as the lights can conveniently talk locally, while self-healing properties of mesh network keep the operation reliable. Many cities around the world, such as Miami and Paris, have deployed their wireless control lighting infrastructure, thus proving that this technology is indeed a practical solution [11].

On the other hand, the cellular approach is viable for devices prefer direct communication with the cloud. A few examples of applicable cellular systems are smart electricity meter and smoke detector. There are a lot of exciting developments in progress for NB-IoT and

LTE-M, both of which are subsets of the standard LTE. These two newly emerged technologies, especially at the sunset of 2G and 3G around the world, aim to satisfy the market demand for low power and long-range cellular.

Despite the fact of having different characteristics and area of usage, these two technologies can be complementary to each other. A good illustration for this statement is using a mesh network for sensor nodes and a few cellular nodes as network gateways. Consequently, this set up benefits from mesh scalability while keeping the cost in check as the product maker does not have to install a cellular module in every unit [12]. A few low-power mesh technologies to be named are Bluetooth mesh, Thread (6LoWPAN based), and Zigbee. Moving back to the main topic, this paper now focuses on NB-IoT – a cellular LPWAN technology.

2.2.2 Narrowband Internet of Things

Narrowband Internet of Things (NB-IoT) is a new low-power wide-area network technology introduced in the 3GPP Release 13 (2015) that aspires to enable a wide range of new IoT applications with improved power consumption, system capacity, spectrum efficiency, and support extended coverage. It has been estimated that a single NB-IoT base station can support 50,000 devices, while battery-powered NB-IoT devices can operate up to 10 years under specific conditions. Figure 10 highlights the landscape of IoT & Machine Type Communications (MTC) with two ends of the spectrum: massive MTC and critical MTC.

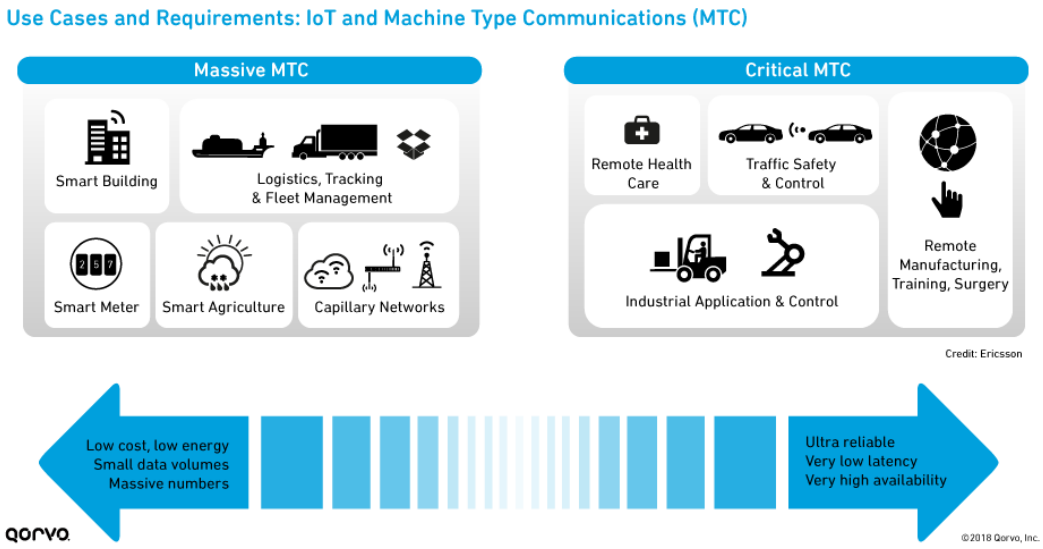

Figure 10. IoT applications and Machine Type Communications. Copied from: qorvo.com [13].

The massive MTC category consists of LPWAN technologies that can support a tremendous amount of devices, while critical MTC technologies offer reliable real-time communication. Within this outlook, NB-IoT inclines towards the massive MTC side as its characteristics favour a massive number of low-cost and low-power consumption devices targeting data-collecting applications. Figure 11 lists out three different spectrum deployment alternatives in deploying NB-IoT.

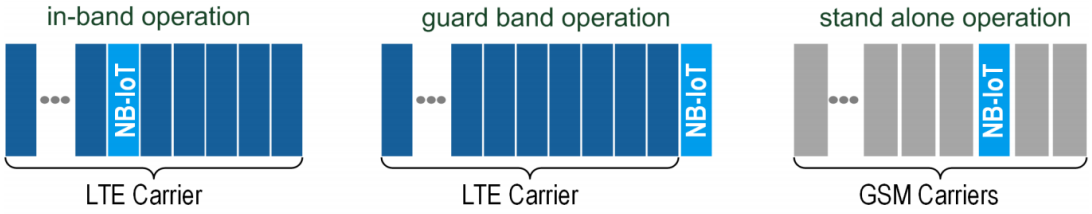

Figure 11. Operation modes in NB-IoT. Copied from "Narrowband Internet of Things whitepaper" [14, p. 9].

The first option shown in Figure 11 is standalone deployment in which NB-IoT carrier reuses an existing GSM band. The second option, in-band deployment, is occupying a part of the LTE carrier for NB-IoT usage. The last alternative, guard-band deployment, is deploying NB-IoT within in the LTE guard band. The physical layers of NB-IoT have been designed to operate in this option without hindering the existing LTE. Fortunately, an LTE base station, often built upon software-defined radio, requires only a software

upgrade to support NB-IoT, thus making it convenient to upgrade most of the existing LTE networks to support this new technology. Figure 12 shows the current deployment state of IoT cellular networks around the world by June 2019.

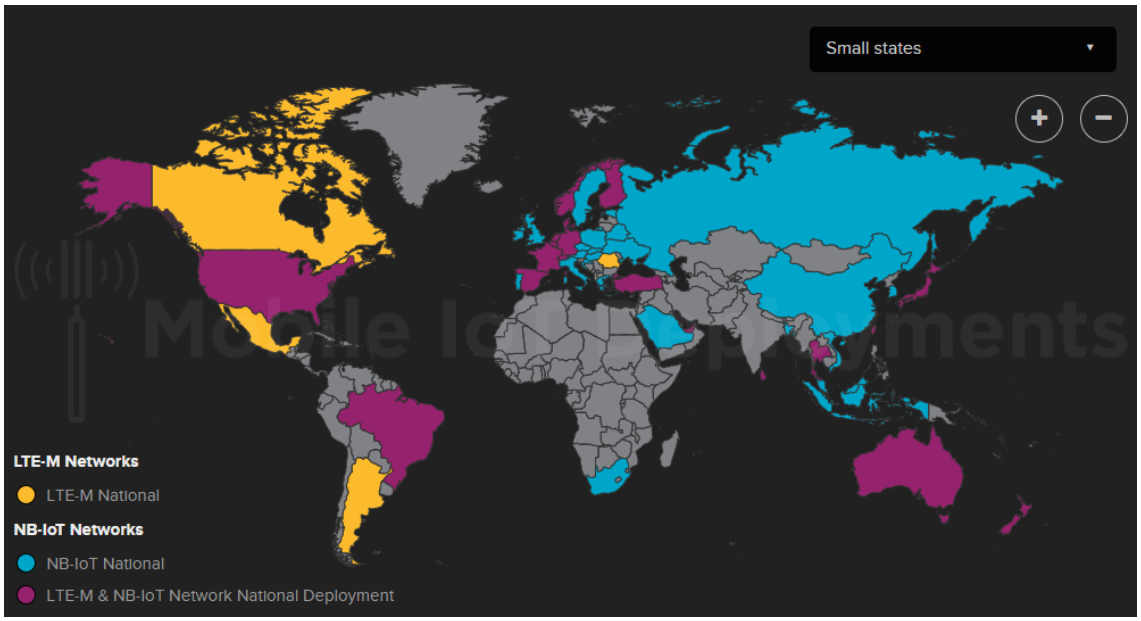

Figure 12. Mobile IoT cellular network deployment map (Jun 2019). Source: www.gsma.com [15].

According to the GSM Association, by May 2019, there have been 114 operators around the globe supporting NB-IoT and/or LTE-M [16]. At the dawn of cellular IoT, despite the fact LTE-M got more traction in the US, the rest of the world prefer to roll out NB-IoT first. Consequently, NB-IoT is the better option for IoT applications which does not target the US market. At the moment, all major network operators in Finland including DNA, Elisa and Telia provide support for NB-IoT, but only DNA offers LTE-M. Coming from the fact that Finland has excellent LTE coverage in residential areas [17], it would be reasonable to predict that NB-IoT devices are going to have a great opportunity within cities since they will receive strong signal, thus promise quality service.

2.2.3 NB-IoT Connection States and Low Power Features

Although NB-IoT is currently known as the most energy-friendly cellular technology in the licensed band, understanding what are the energy components in an NB-IoT connection is vital for developers to understand how to write proper firmware for low power

application. Figure 13 presents the state transfer diagram of user equipment (UE in terms of energy components.

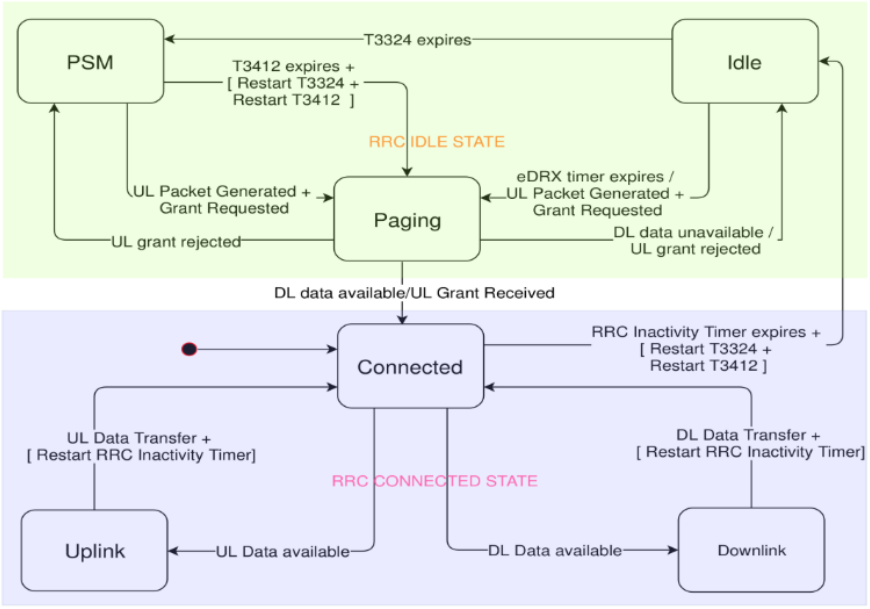

Figure 13. State transfer diagram of NB-IoT energy components. Copied from “Energy Modeling and Evaluation of NB-IoT with PSM and eDRX” [18].

The chart in Figure 13 comprises six different states:

- Connected (RRC-connected): The UE is in RRC-connected state and can exchange data – exhibits a high power consumption.
- Uplink (RRC-connected): UE radio sends data uplink when software sends new packets to the destination server – exhibits a high power consumption.
- Downlink (RRC-connected): UE receives downlink data when there is data sent to the device - exhibits a high power consumption.
- Paging (RRC Idle): UE is monitoring paging messages from the base station in its RRC-Idle state - exhibits small spikes in power consumption graph due to the radio reception.
- Idle (RRC Idle): UE waits for the next paging cycle or goes to power saving mode (PSM) if T3324 expires - exhibits a low power consumption.
- PSM (RRC Idle): UE turns off the radio for a long time and sleeps until T3412 expires or an uplink request came from the microcontroller - exhibits lowest power consumption.

Figure 14 shows a simpler illustration of NB-IoT operating modes from the application standpoint.

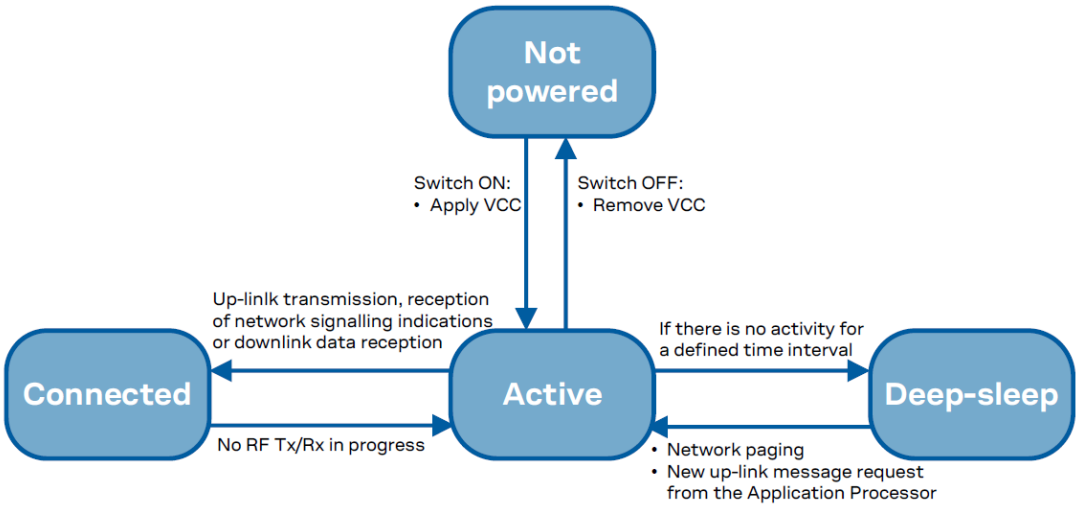

Figure 14. U-blox SARA-N211 module operating modes from an application perspective. Copied from “SARA-N2 series system integration manual” [19, p. 11]

Fortunate for developers, they do not need to know extensively about all the state transitions featured in the NB-IoT protocol as certified modules (e.g. SARA-N211) already managed these transitions, curtailing away a huge amount of responsibility from developers. The following Section 2.2.4 further explains two important timers T3324 and T3412 and gives a more descriptive example of the power consumption in a typical usage scenario.

2.2.4 Power Saving Techniques in NB-IoT

There are two features in NB-IoT to optimise the power consumption: connection release and resume; extended discontinuous reception (eDRX) in conjunction with power-saving mode (PSM). These features together help reduce the power consumption significantly, making it viable to create a battery-powered IoT device that may deliver the ten-years theoretical expectation.

Figure 15 describes UE operations in a typical NB-IoT usage scenario. On the top, there is the RRC connection state; the middle represents the activity between UE and base station in corresponding to power consumption; the bottom indicates whether the radio is enabled. Also, the power consumption in each event is presented accordingly, except for the PSM mode which actually is the state with the lowest power consumption.

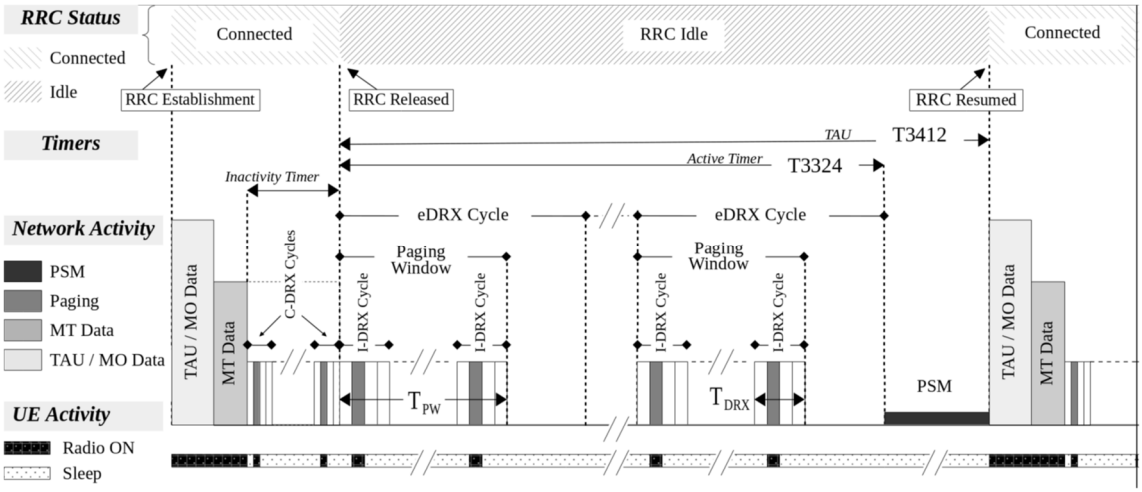

Figure 15. Summary of UE's behaviour in NB-IoT associated with power consumption. Copied from "Exploring the Performance Boundaries of NB-IoT" [20].

For power-constrained wireless application, the power consumption is in tight correlation with the radio activity. The prevalent strategy to conserve energy in such applications (e.g. BLE) is to schedule communication time window in advance for both sides, allowing the radio to turn off between these intervals. Without any surprise, this philosophy is indeed applied in NB-IoT.

At first, the UE goes to "RRC-connected" state as it sends out a mobile originated packet to the network, which could be triggered by the application code sending a UDP packet, or there is a tracking area update (TAU) to be performed. As a matter of fact, the UE can jump to this RRC-connected state at any point as it sends data to the base station. Next, the UE proceeds to wait for mobile terminated traffic and monitor Connected-eDRX (C-eDRX). Mobile terminated traffic is simply a term for a message sent from the network to the UE, as such message is terminated at mobile/UE side. After the "Inactivity Timer" expires (in fact this transition comprises few timers according to LTE specification), the module transits to "RRC-idle" state and starts monitoring Idle-eDRX (I-eDRX), at the same time it starts the "Active timer" (T3324) and "TAU timer" (T3412). During this state, the UE can sleep between paging occasions to reduce its power consumption. Paging occasions are time window in which the network can inform UE if there is a downlink packet for it, which will trigger UE to resume RRC-connected state to exchange data with the base station. Otherwise, when the "Active timer" (T3324) expires, UE will go to PSM

- the lowest power state and remains unreachable by the network until “TAU Timer” expires or UE’s application sends a packet uplink. At such events, the UE goes to RRC-connected state and is again ready to exchange data with the network. This connection release/resume mechanism is achievable as UE retains its network session context to avoid the overhead of renegotiation with the network. Figure 16 presents a typical power consumption pattern of a UE as described earlier.

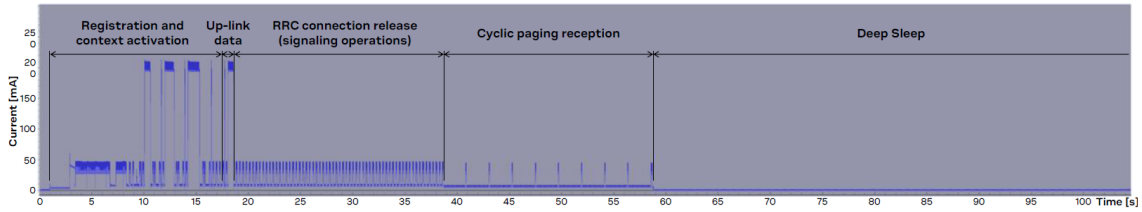

Figure 16. Modem current consumption from power-on to deep sleep mode visualised. Copied from SARA-N2 Series System Integration Manual [19, p. 13].

With the illustration in Figure 16, it would be simpler to recognise the power pattern without being distracted by connection states and radio activity information.

The purpose of eDRX and PSM in NB-IoT UE is to reduce receiver enabled time to save power at the cost of connection latency. In layman’s terms, the eDRX feature means the specifications now allows longer time duration between pagings. There are two eDRX types: Connected eDRX (C-eDRX) and Idle eDRX (I-eDRX). Even though the ultimate decision on connection parameters is up to the network, however, the UE can provide its preferred values for Active Timer (T3324), TAU Timer (T3412), paging window (denoted as T_{pw} in Figure 15), and eDRX cycle. Unfortunately, the “Inactivity Timer” and C-eDRX cycles are chosen by the network and UE cannot influence these parameters, but that also means there is less responsibility for the application developer. Developers should contact network operators to ask for supported network parameters choices since they may not allow all possible values listed in the 3GPP specifications. Also, it is worth noting that the network can change these connection parameters at any time. Appendix 1 provides a lookup table on how to encode eDRX and paging time window value. Appendix 2 supplies instruction on encoding T3324 and T3412. Figure 17 brings more details on eDRX regarding paging procedures in an illustrative manner.

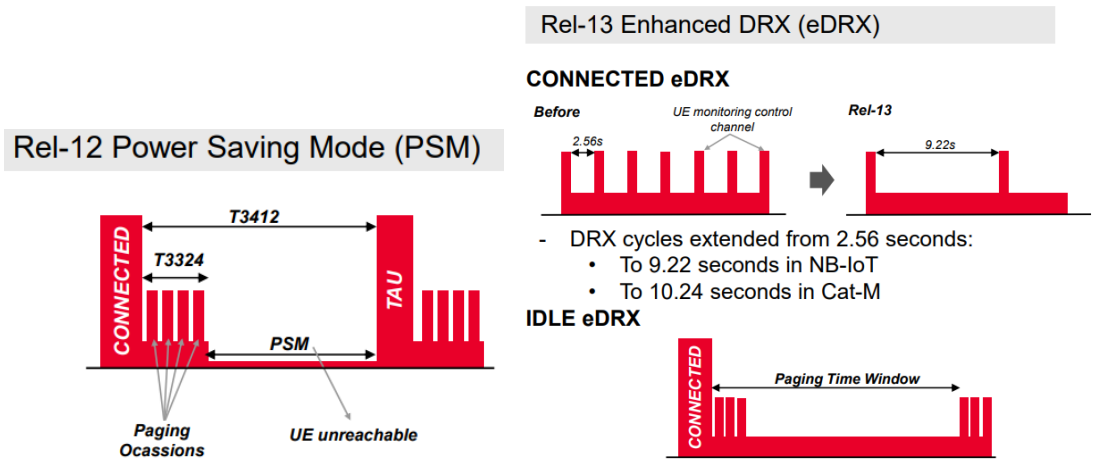

Figure 17. Magnified power consumption pattern of NB-IoT UE in paging procedures. Copied from Keysight NB-IoT Technical Fundamentals [21, p. 25].

Release assistant is a feature for UE to actively release the connection and go to RRC-Idle as soon as possible, which is especially beneficial for battery-powered devices. When the UE transmits a data packet uplink, it can use Release Assistant feature to notify the base station that either only one downlink response from the cloud is expected so the RRC will be released after the next downlink, or no further downlink is expected and RRC resource will be released right after the uplink transmission completed. If this feature is not used, the UE will be staying in RRC-connected state for a relatively long time (e.g.10-30s) depending on the network config before transiting to IDLE, which can be considered energy wasting. Figure 18 describes the power consumption patterns of an NB-IoT module sending a 512 bytes datagram under different network settings.

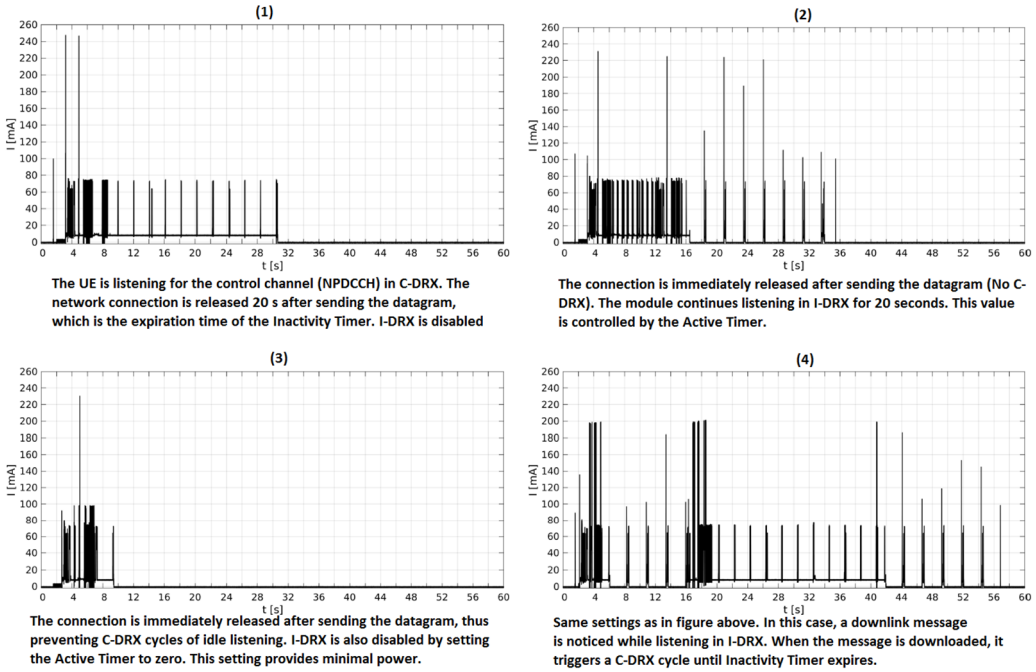

Figure 18. Power consumption of NB-IoT module sending a 512 bytes datagram under different network settings. Copied from “Exploring the Performance Boundaries of NB-IoT” [20].

For identifying operations of the UE, the module exhibits a deep sleep current of 3uA, 10 mA in Idle (which in fact differs from the observation made during this thesis as during the Idle state outside paging occasions the module has the same consumption as during deep sleep), 60 mA when radio is in reception mode and 200mA when radio transmits. In situation (1) and (4) UE disabled the I-DRX by setting T3324 to 0, while in (2) and (4) this timer is set to 20s. Situation (1) and (3) are nearly the same, and the only difference is (3) uses release assistant to conserve power by avoiding staying in RRC-connected state. On the other hand, (2) and (4) have the same settings, but in (4) there is a downlink during UE I-DRX monitoring process. This downlink message brings the UE to RRC-connected, and after receiving the message, the module spends some time monitoring C-DRX and I-DRX before going to deep sleep, meaning the Active timer (T3324) got reset by this downlink. This behaviour should be taken into account while developing applications. Figure 19 provides a summary of eDRX related connection parameters in NB-IoT.

Timer	Description	Configurable by UE?
Inactivity Timer	When the inactivity timer expires, it causes a transition from RRC Connected to the RRC Idle state. This timer is controlled by the eNB, not by the UE.	No
Active Timer (T3324)	The T3324 determines the duration during which the device remains reachable for the downlink through eDRX (RRC Idle mode). The device starts the Active Timer when it moves from RRC Connected to RRC Idle mode. When the Active Timer expires, the device moves to Power Saving Mode (PSM).	Yes
Paging Time Window (PTW)	This is the duration of a paging event composed of multiple DRX cycles. The paging event fits in the PTW; so the longer the PTW, the greater the number of DRX cycles.	Yes
DRX Cycle	The duration of a DRX cycle. It is a multiple of the Paging Occasions (PO) cycle (1280 ms). In a DRX Cycle, the UE listens for one PO and sleeps for the following POs	No
eDRX Cycle	The duration of an eDRX cycle. (the time between two PTWs).	Yes

Figure 19. Summary of eDRX related connection parameters in NB-IoT. Copied from “Exploring the Performance Boundaries of NB-IoT” [20, p. 4].

Figure 19 provides a table of summary of NB-IoT eDRX related timers mentioned in this section, coupled with information indicating whether the UE can suggest them. The TAU timer (T3412) is left out due to not related to the eDRX process, is suggestable by UE, making a total of six parameters to keep in mind during development.

2.2.5 NB-IoT Power Consumption Best Practices

The power usage of a device is determined by multiple factors, each of which if not appropriately engineered, could ruin the expected power efficiency. This section provides some tips to follow to optimise the device power consumption:

- Design a good PCB layout to reduce interference on the device. Be careful with antenna matching circuit.
- Carefully choose electronics components suitable for low-power operation to minimise quiescent current of the device. Sensors and peripherals need to be put in a low-power state while being unused.
- Select appropriate preferred configurations for T3324, T3412, eDRX cycle length and paging time window for the application requirements. Developers are recommended to ask the network operator if those configurations are accepted in their network. Experimenting by trial and errors is time-consuming.
- Use the NB-IoT release-assistant feature properly within the application.

  Set the device uplink power accordingly while avoid operating the device in coverage enhancement level 2.

NB-IoT currently only supports open-loop power control, meaning UE determines its transmitting power. There are two transmit power levels of 23 dBm and 20 dBm supported by CAT-NB1, and 14 dBm added in CAT-NB2. To enhance coverage for IoT device, UE is classified into Enhancement Coverage Level (ECL) ranging from 0 to 2 in

which 2 is the worst-case scenario according to the signal strength received and report by UE. This classification determines the number of repetition of the transmission to ensure the quality of service, but at the same time drives the power consumption of UE up with increased air time. As this issue depends on the physical deployment, the device is recommended to report its coverage class to the cloud server so maintainer can detect it to take appropriate actions such as deploying it at an alternative location.

2.3 Mbed OS

Mbed OS is an open-source operating system developed by ARM and its silicon partners, designed specifically for ARM Cortex-M microcontrollers. The operating system aims to simplify the device software development process by offering a common abstraction layer across multiple microcontroller series from different vendors including NXP, ST, Cypress, etc., hence reducing time-to-market for embedded devices in general and IoT devices in particular. Furthermore, this approach allows applications developed for Mbed OS to be migrated among Mbed compatible platforms with reasonable effort. Figure 20 gives a high level illustration of Mbed OS, presenting the framework's architecture and its main components.

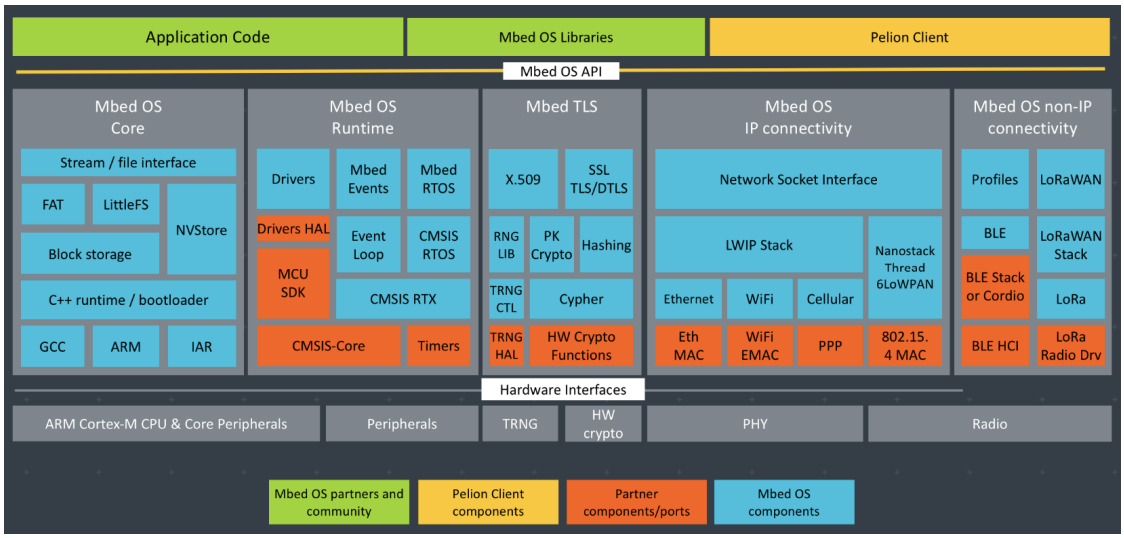

Figure 20. The architecture of Mbed OS. Copy from os.mbed.com [22].

First of all, Mbed OS attempts to unify commonly available functionalities in microcontrollers, for instance, UART, I2C, Timer, etc … under the same C++ application programming interface (API), making it virtually identical to configure and use a peripheral throughout the Mbed ecosystem, hence promoting code reusability. Apparently, this API unification also set up a good starting point for adding support to new targets or new features into existing targets.

Second, MBed OS comes with support for many software modules related to sensor drivers, data storage and connectivity. The availability of off-the-rack sensor drivers enables a quick and straightforward solution for integrating new sensors into the system. Besides, the OS also facilitate external data storage capability on SD card or SPI/QSPI flash. On the other hand, there are connectivity supports built-in in the OS to reduce the complexity of making an IoT device. Thanks to this flexibility offered by Mbed, developers can make their device supports IP based connectivity via Ethernet, WiFi, 6LoWPAN, cellular, or other forms of non-IP communication such as BLE, NFC.

Third, Mbed OS includes an RTOS for developing software with deterministic, multi-threaded, real-time execution. This component equips developers with RTOS primitives including threads, mutexes, semaphores, queues as well as other standard RTOS functionalities to accommodate the application requirements. The RTOS feature can be excluded if not needed in the program to save RAM and flash consumption.

Moreover, the ARM MBed team provides a list of comprehensive API documentation along with examples and tutorials on their website. As a result, these materials help developers to get familiar with MBed API as quickly as possible and help them to start developing their customised system.

Another perk offered by Mbed OS is Greentea, an automated testing tool. Tests are written in C++ as if it is a regular MBed based program, which will be executed directly on the microcontroller. On the one hand, this testing tool minimises the amount of labour needed since it handles all the device flashing as well as the test result collecting process from device-under-test (DUT). On the other hand, Greentea support “host-test” features which under the hood are Python scripts that run on a computer and communicate with the microcontroller. For example, a tester can write a test case in which the host machine

request DUT to send a specific piece of data to the cloud and then check if the same data is received on the cloud side, verifying the DUT connectivity capability. Since embedded applications are much less convenient to test compared to a pure software application, this automated tool is an excellent effort towards minimising the hassle of embedded testing, which will consequently promote better quality for IoT projects [23].

3 Design and Implementation

This section dives into the technical aspects of the project, starting with the goal and requirements of the project, then visits the system components selections and system architectural decisions along with relevant processes including testing and optimising the device power consumption. These contents should provide readers with an overall understanding of the system, and at the same time, give an outlook on how the project was carried out.

3.1 Project Goal and Requirements

This thesis was carried out as part of a client project at Etteplan Embedded Finland Oy to evaluate the capability of NB-IoT, LwM2M, and Mbed OS. Moreover, the artefacts of this project will be used as a reference design to shorten the execution time of future projects relying on the same technology stack and similar electronic components. This “Design and implementation” section focuses on the software aspect of the system as it aligns with the author’s duty throughout the project.

Regarding functionality specification, the device is expected to take data from a weather station via Modbus protocol and send reports on environmental measurements and its operating state to a cloud server over NB-IoT. Besides, the cloud server should be able to execute predetermined operations on the device, for example, rebooting. On the other hand, the whole system should operate as efficient and low power as possible.

3.2 System Components Selections

Choosing system components, one of the first and arguably the most crucial step, determine the foundation of the system. This subsection lists out hardware and software components along with commentaries why they are chosen.

3.2.1 Main Hardware Components Selections

One of the first steps in designing a constrained, low power IoT embedded device often is selecting the target microcontroller. After some research, an MCU from the STM32L4 family is selected for several reasons. First of all, ST Microelectronics is a well-known semiconductor provider in Europe offering a diverse portfolio of microcontrollers for a wide range of technical requirements. Second, Etteplan has delivered many successful projects which incorporates STM's components, including the selected MCU. Consequentially, software and hardware designers at the company are already familiar with the properties of this microcontroller as well as its development ecosystem. Third, the MCU offers a hefty amount of flash and RAM, along with multiple peripheral instances of UART, I2C, and SPI that presumably cover the expectations of the application. Though this pick might not be the ideal choice for large quantity production, it is sensible to prioritise creating a few working prototypes with as little hassle as possible at the project start. When it comes to an economic incentive for revising components in case of mass production, it is feasible to migrate the system to a less costly pin-compatible MCU within the same family, which would offer reduced ROM and/or RAM while still satisfying requirements [24]. Fourth, STM32L4 family is an ARM Cortex-M4F explicitly designed for low power applications [25], considering one of the key requirements of the project. Last but not least, the selected MCU is officially supported by Mbed OS, making the software development much more straightforward as there is no need to port the framework to the target.

Another essential physical component in this project is the NB-IoT modem, of which eventually U-blox SARA-N211 got selected. Just in case this brand name sounds unfamiliar, U-blox is a reputable wireless module provider known for offering high-quality precertified modules regarding WiFi, Bluetooth, GNSS and cellular [26]. Etteplan has previously conducted projects that use U-blox modem and feels confident in trying out this

NB-IoT module. Moreover, U-blox defines consistent footprint formats for their components, in this case, a form factor named SARA, making it convenient to migrate among modems with the same form as it reduces the effort needed for redesigning the schematic and PCB layout. On the other hand, this CAT-NB1 module is capable of operating in bands 8 and 20, compatible with networks in Finland. In future projects, it is possible to swap the modem to a SARA-N3 which support more frequency bands and CAT-NB2, or to SARA-N4/R4 if LTE-CATM1 or 2G fallback is requested. Another advantage of using U-blox products is that the company provides detailed materials necessary for software and hardware designing processes, and has always been responsive to customer support.

3.2.2 Software Component Selections

In the present project, Mbed OS was selected as the base for the firmware by the administration. While this may be true, this framework is, in fact, an appropriate choice for the system thanks to its ideology of unifying APIs to simplify development and attempts to provide proper support for external components. Mbed OS has been under active development by ARM and its partners for the last ten years, making it one of the most mature frameworks for microcontroller-based IoT device available. Equally important, the framework is open source and has been licensed under Apache 2.0, MIT, BSD along with a few royalty-free permissive binaries, thus making it applicable to commercial projects.

Another significant point of consideration was determining which IoT protocol to use. Again, Lightweight M2M was chosen by the administration and therefore used in this project. Genuinely, LwM2M is a good pick for several reasons. First, unlike the MQTT protocol operating on TCP, LwM2M can be used on UDP, which happens to be the only IP based protocol supported by SARA-N211. Second, the use of connectionless UDP favours low energy consumption as TCP protocol requires the device to stay awake and maintain the connection. Wakaama, an opensource lightweight M2M library backed by Eclipse, was selected due to practicality and financial reason. Interestingly, the built-in LwM2M client in U-blox N2 modem (not used in this project) is also based on the same library [27, p. 11]. From a subjective point of view, LwM2M is an unfamiliar name compared to MQTT as it is a latecomer of the IoT world and yet to be supported by major

cloud services. Fortunately, there are currently a few providers on the market support LwM2M such as ARM Pelion or Cumulocity IoT. Hopefully this promising protocol will gain more traction in the future as it is designed as a full-fledge device management protocol with interoperability in mind.

3.2.3 Guide on U-blox SARA-N211 NB-IoT Modem

This section presents a summary with tips for integrating SARA-N211 U-blox NB-IoT modem with the device. First of all, the microcontroller can communicate with the module through an asynchronous serial interface (UART) without flow control support. This serial interface supports 8N1 frame at four different baud rates of 4800, 9600, 57600 and 115200 [19, p. 17], but selecting a baud rate higher than 9600bps (fastest supported by the Low Power UART of the modem) will disable deep sleep operations [28, p. 48]. Even though there is no flow control support, the module activity can be detected via the V_INT pin (active high). This property acts as an excellent trigger to enable the UART of the MCU. The notifying mechanism of the V_INT pin is illustrated in Figure 21.

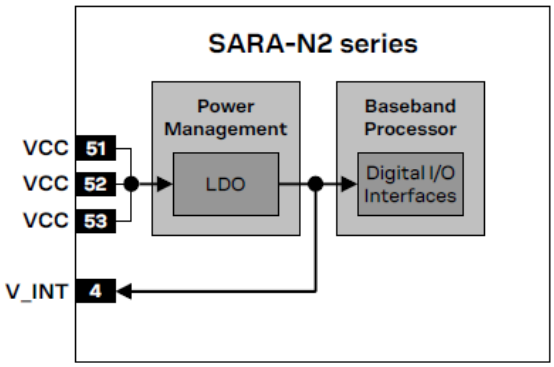

Figure 21. Interfaces supply output (V_INT) simplified block diagram in SARA-N2 series. Copied from SARA-N2 series System Integration Manual. [19, p. 13]

The microcontroller controls the module by issuing AT commands – commands used for modem controlling named after its “AT” (attention) prefix. Commands supported by the module can be lookup in the “SARA-N2 AT Commands Manual” [28]. Most of the commands coming from the MCU are acknowledged by the modem nearly instantly, however, the modem can issue an Unsolicited Result Code (URC) to the MCU at any time. URC serves as a way for the modem to actively notify the MCU, for example reporting a

change in network registration state or a newly arrived UDP packet. For this reason, the module's activity indication via V_INT pin comes in very handy, especially for low power device where the more power-downed peripheral, the better.

Coming back to the software perspective, whenever the application wants to send data to a cloud server, it first needs to open a socket with a specified destination IP address and port. Thanks to U-blox implementation, the N211 module comes with an embedded UDP stack, freeing the responsibility of having to accommodate a TCP/IP library (e.g. lwIP) for application developers. However, this pre-packaged convenience only supports UDP IPv4, thus binding the system with UDP and IPv4 issues. Fortunately, this limitation does not apply with other modem lines such as U-blox SARA R4 which supports both TCP and UDP on both IPv4 and IPv6. According to U-blox's manual, the module can only send and receive payload at a maximum of 512 bytes. However, based on observation during the module usage, this statement turns out to be inaccurate as the stated limitation only applies to uplink, but not to downlink. When there are more than 512 bytes of payload sent downlink, the module still manages to receive the data correctly as the real limit is close to 1500 bytes (1500 bytes is Ethernet MTU). This matter is revisited with more details in Section 3.7.

A minor issue with this U-blox module is that it does not give out any indication when an uplink packet is dropped. Even though this is not considered misbehaviour by the nature of UDP, it has been observed that the modem drops some packets when there is a relatively large amount of them sent uplink at once, decreasing the quality of service. Developers may want to safeguard their implementation with a self-regulating packet pace mechanism to avoid congestion on the module. This congestion is easy to reproduce by sending many packets consecutively (e.g. ten packets, each of 400 bytes) when the module connection is not yet in the RRC-connected state.

3.3 Development Environment and Team Collaboration Workflow

It is known that a well-established development environment and positive collaboration among team members play an essential role in working efficiency. With this wisdom in mind, team members discuss and settle on a common development environment as well as rules for the Git workflow. The integrated development environment (IDE) chosen for

this project is Eclipse CDT (also known as Eclipse for C/C++) with GNU MCU Eclipse plugin. The debugger used is ST-Link-V2 in conjunction with OpenOCD. The chosen compiler is GCC-ARM, and the build tool is Mbed CLI – a tool provided by the Mbed team to simplify the development process, including building and testing the application. Program traces are collected via UART and can be displayed on PC with an USB-to-TTL adapter. These setup decisions turn out to be cost-effective as these tools are quite easy to set up, at the same time, offering extensive yet convenient features at a minimal cost.

Eclipse is a well-known IDE maintained by the reputable Eclipse Foundation who houses over 350 open-source projects across a wide range of technologies [29]. According to the release log, Eclipse CDT receives a new update every three months, indicating the IDE is under active development and maintenance. On the other hand, GNU MCU Eclipse is a well-maintained plugin for Eclipse CDT which provides an extensive set of tools for ARM and RISC-V MCU at no cost. As a result, many silicon providers like ST and NXP provide customised Eclipse CDTs as recommended IDEs for their clients. From an embedded developer perspective, this toolset provides a functional text editor, a flexible way to configure the build process along with good integration with debugging utilities including GDB, OpenOCD, and JLink. Furthermore, the IDE offers a peripherals register view which enables developers to quickly inspect peripheral registers whenever the target is stopped, thus speeding up the debugging process, especially for low-level driver developments.

The project execution follows the Scrum methodology. About Git policy, the team decides to use interactive rebase instead of merging. Rebase before merging into master is not a problem within a small team, yet it makes the history on the master branch linear and simple to follow. Furthermore, a merge request must pass the CI pipeline and got approvals before getting accepted. As the CI process consumes time and requires starting up a physical machine, team members agree that pipelines are only required to run before merging and for every commit on the master branch.

3.4 System Design and Software Architecture

The primary responsibility of the program is similar to a generic IoT data collecting system, focusing on gathering sensor measurements and push them to the cloud for post-

processing. While this may be true, project steps are not as straightforward as they usually are because the design involves new technology stack with limited supporting materials. The first obstacle is porting the LwM2M Wakaama library from working with the POSIX interface to using Mbed OS APIs. In the beginning, this task was difficult and had no clear direction as the library does not come with any guide or instructions to achieve such a goal. However, the right path to the solution was soon revealed after the example code had been skimmed through and its execution flow followed by adding printing statement as well as using a debugger. As a matter of fact, the developers of the library has designed their code with portability in mind, defining wrapper facade functions for setting up and tearing down connections, at the same time designating a function to send data as well as a function to pipe received data into the library for processing. For this reason, even though the example code depends heavily on POSIX calls, the library is loosely coupled with this interface and can be ported to another platform by rewriting the mentioned functions appropriately and make minor changes on piping the received data into the library.

Regarding the architectural aspect, the firmware architecture strictly follows the gatekeeper design pattern. Gatekeeper pattern [30, p. 260] is a designed pattern in which only a task, the gatekeeper task, has sole ownership of a particular resource, and only it is allowed to use this resource directly, while other tasks can only use the said resource indirectly via the service offered by this gatekeeper. As a result, this pattern ensures mutual exclusion in accessing the resource while avoiding priority inversion or deadlock. Furthermore, as only one task has direct access to a resource, it will be easier to identify and resolve the issue in case one happens to arise. Figure 22 supplies a hardware block diagram, listing out components that physically comprise the system.

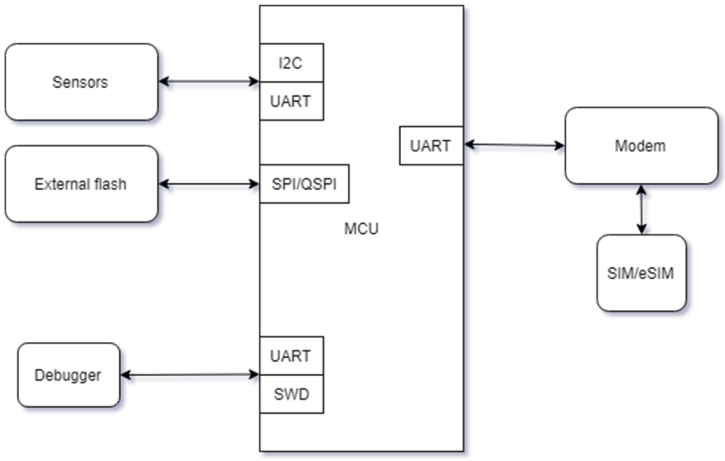

Figure 22. Hardware block diagram of the system.

As Figure 22 illustrated, there are multiple components within the system including sensors, external flash and cellular modem, which could be quite a challenge to manage in a single-threaded application. Figure 23 presents the structure of the software, focusing on threads' responsibilities and how do they interact with others following the gatekeeper pattern.

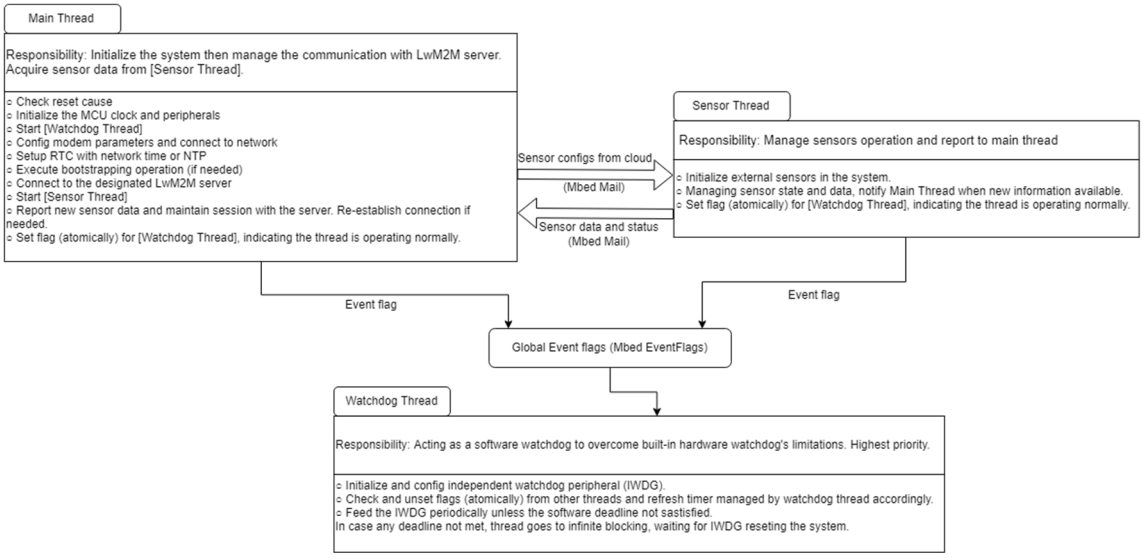

Figure 23. The software structure of the IoT application.

Generally speaking, the program consists of three major threads: main thread, sensor thread, and watchdog thread, each of which has distinct duty within the system. First,

the main thread is responsible for initialising the MCU, spawning other threads and afterwards managing the communication with the LwM2M server. The second thread, sensor thread, takes care of initialising sensors and collecting sensor data. These sensor data will be relayed to main thread via a Mbed OS mail (similar to a pipe or a queue) to be pushed to the cloud. The third thread, watchdog thread, acts as a failsafe mechanism to rescue the system from unexpected unrecoverable failures with a reset.

3.5 Software Testing

Software testing, an element within the software quality assurance process, is an important ingredient which safeguards the functionality of the program, allows work collaboration between developers and promotes good coding practices. This section provides a list of testing methods applied and applicable to this project. The names of these will-be-mentioned techniques may not follow well-known conventions.

The first testing technique used in this project is manual white box testing. These tests are to be manually performed by developers to verify whether the connectivity and LwM2M proportions of the application works as expected. First, an LwM2M Leshan server is set up on an AWS instance to carry out the role of a bootstrap and device management server. Next, Tcpdump, a network analyser tool, is installed on the server to log network traffic on ports of interest as its log can be later analysed with Wireshark. Fortunately, Wireshark provides support for decoding LwM2M and decrypting DTLS communication (provided the pre-shared key), thus making it a valuable utility to diagnose the connection problem and verify whether network data are consistent with information sent from the device. In fact, this setup combines with the server's log helped to resolve the DTLS handshaking failure while attempting certificate authentication. On the other hand, this setup also can provide an estimation on NB-IoT latency by measuring how long does it take for a ping to travel back and forth. Since NB-IoT latency can be in the degree of seconds, millisecond precision is not needed for this measurement. For instances, initial ping to google.com can take nearly 9 seconds, but subsequent ping will take much less time since the modem is already in the RRC-connected state. Later, this simple test can be improved and use to test out the reliability of the network at a given location. As Leshan server exposes a REST interface, it is possible to write an integration test which sets resources to different values then try to read it back from the device,

ensuring all the fields are functioning as expected. This check helped to detect a mistake on setting a new value to a string buffer as the application does not clear the buffer before copying in new characters. Even though these tests were addressed as manual tests at the beginning of the paragraph, most of them can later be automated when there is more time for improvements. The device will also go through security testing at a more mature state.

To increase the chance of detecting defects, the device is subjected to run multiple days continuously. The server log and device log is checked to verify that the device still operates and the server receives uplink data at expected intervals. Furthermore, on occasions where the device loses session with the server, the device manages to re-establish the connection, which adds confidence on the firmware.

As mentioned earlier, Mbed OS offers an automated testing tool called Greentea (Arm Mbed reGREssion ENvironment for TEst Automation). This utility allows writing automated unit tests, which will then be automatically flashed and executed on the target MCU. The fact that the test runs on native targets brings it as close as possible the real application, minimising the risk of passing compiler errors without detecting. However, this testing framework is still limited and can only perform assertion checks, while providing no support for convenient stubbing or mocking, things that are particularly useful in testing written code behaviour at hard-to-produce situations. As a matter of fact, Mbed OS itself uses also googletest to write off-target tests, however using this testing framework is troublesome as it is placed inside Mbed OS repository. In contrast, Greentea tests can be placed outside of Mbed OS folder. Currently, there are only a few Greentea test cases written due to time constraint, but later more tests will be added.

Despite being a norm for modern software development, continuous integration (CI) often went missing in embedded system development. Continuous integration is the practice of automating the integration process of code changes from team members in a software project. This practice often comprises automatic build, tests execution, code quality analysis, etc. In this project, a pipeline has been set up to automatically build the software, run Greentea tests, and perform static code analysis with Cppcheck. This automated process provides a reference compiling system, eliminating “it builds/passes tests on my PC” excuses.

Though it is impossible to test software exhaustively, still the testing for this software at this stage is not done properly. However, as now the way to make tests with Greentea is established, the following testing will become easier. This is an aspect of the project that could be enhanced later.

3.6 Optimising Power Consumption from Software Perspective

Considering the project accommodates NB-IoT with low-power capability, minimising the power usage becomes a point of interest in the development. The rule of thumb for preserving energy is bringing the processor and its peripherals along with external components to the best low power mode as long as possible. In this project, for the sake of simplicity while appreciating the “one step at a time” methodology, power consumption optimisation effort is conducted only for a device comprises of an STM32L4 MCU and an N211 cellular module.

Apparently, there are a few rules to follow when developing firmware for a low power device. First of all, developers should avoid making the MCU doing redundant work while avoiding polling and instead take advantage of Interrupts or DMA transfers if possible. In case polling is unavoidable or the act of avoiding causes an unjustified increase in software complexity, developers can try to limit the polling frequency to an adequate rate. Second, configuring the system clock and peripherals appropriately could reduce power consumption. In practice, lowering the system clock tends to give a slight increase in the system efficiency, hence sometimes it is worthwhile to operate the system at lower clock speed, or adjust the clock rate dynamically. Furthermore, clock source selections also have an impact on power consumption, but likely there are other things to be considered. For example, the STM32L4 multi-speed internal clock (MSI) may offer an advantage over the high-speed external (HSE) clock regarding power efficiency and bill of material. However, the internal clock suffers more drift under temperature variation compared to external oscillators’, leaving HSE the best candidate for outdoor devices. Though it is troublesome to apply a specific clock configuration in Mbed as abstraction hides away details, this goal can be achieved by modifying the source code part which initialises the clock or by reconfiguring the clock at the beginning of main(). Third, awareness of system specifics may introduce opportunities to trim down the power consumption. For example,

STM32L4 MCU offers a low power UART, which indeed could result in lower power consumption than standard UART.

According to the STM32L4 Reference Manual [31, pp. 163-169], the STM32L4 MCU features seven low power modes, and each offers different compromises over power consumption, peripheral availability, wakeup latency, and available wakeup sources. However, due to the abstraction constraint of maintaining a common API across different MCUs, Mbed OS simply divides low power modes into two levels: sleep and deep sleep. Within the STM32L4 context, the Mbed sleep mode corresponds to the SLEEP mode of the microcontroller, while the deep sleep mode corresponds to STOP2. This selection is not a coincident as STOP2 mode is the lowest power state of the MCU in which RAM contents on all banks are retained. Mbed OS provides a sleep manager for the IDLE task to decide which sleep mode the MCU should go. This sleep manager depends on a set of DeepSleepLock flags that can be raised and cleared by peripherals drivers of the OS. Whenever the sleep manager is invoked, it performs a check to see whether there is any DeepSleepLock held to determine the appropriate sleep option. For example, the MCU can go into deep sleep if the processor is not busy, and there are no high-speed peripheral active (e.g. SPI transfer ongoing).

In a sophisticated program with multiple tasks executed concurrently, employing RTOS is a good idea because the IDLE task hook is an appropriate place for bringing the system into a low power state. On the other hand, signalling and blocking utilities from RTOS such as event flags could make it easier to take advantage of the asynchrony of ISR or DMA. Another essential remark on conserving energy for RTOS applications is enabling tickless (or tick suppression) mode. This RTOS configuration disables the MCU Systick Interrupt during the deep sleep operation, thus blocking this periodical interrupt from repeatedly waking the MCU and waste energy. Figure 24 shows the difference in the wake-sleep pattern of a particular program with and without tickless configuration.

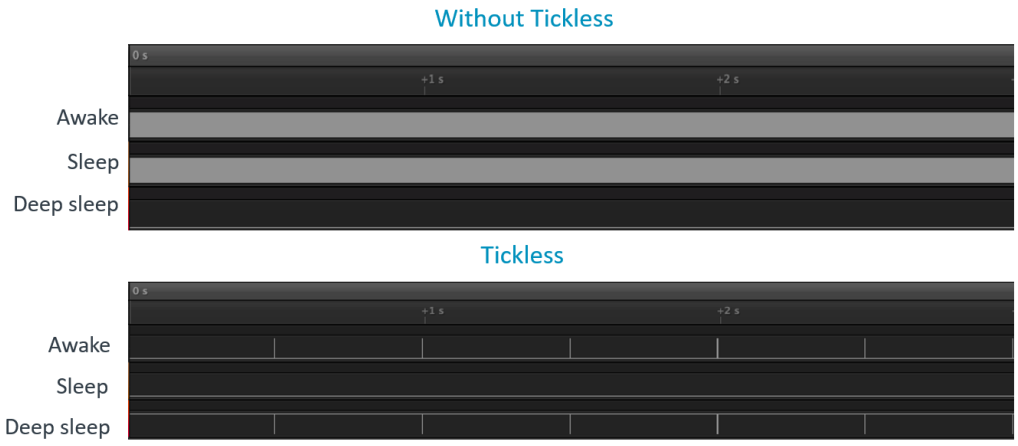

Figure 24. Comparison of awake-sleep pattern of a Mbed program with and without tickless. Copied from “Low power features in Mbed OS” [32].

Another item worth mentioning is a small drawback in the default Mbed OS tickless configuration for STM32L4. The misbehaviour appears in the form of a periodic once-per-second wakeup during long sleep of the device. Mbed OS, by default, uses a low power timer for timekeeping purpose during the deep sleep state, in this case, LPTIM1 – a low power 16-bit timer clocked from a low-speed clock source (e.g. 32kHz LSI). It turns out that this unwanted periodical wake up is caused by the overflow of the timer – which can be considered a hardware limitation. According to the Mbed OS Low Power Ticker porting guide [33], the low power timer frequency has to be at least 8kHz to ensure the resolution, thus limiting the timer overflow interval to every few seconds. Equipped with this knowledge, it is possible to increase the said timer prescaler and extend this wakeup interval to every 4 seconds, though this duration is still very restricted. This issue is addressed in a recent Mbed power consumption optimisation guide [34] along with a proposal of using the 32-bit RTC instead of LPTIM1. This solution has been confirmed working on the device used here.

The last thing in optimising power consumption is indeed evaluating the result. There are affordable measurement kits on the market for profiling consumption of low energy devices, such as NRF Power Profiler Kit or STM32 Power shield, but unfortunately, these tools are not suitable for analysing the consumption of the NB-IoT device. Even though the NB-IoT module demonstrates a low sleep current around 4 uA plus 2.6 uA from the MCU, during uplink operation the cellular module alone can spike up to a few hundred mA, going way outside the range of these tools. For now, a Keysight U1273AX digital

multimeter is used to measure the sleep current of the device and use a simple shunt resistor in conjunction with an oscilloscope to roughly capture the consumption pattern of the device. Certainly this is a far-from-perfect solution, nevertheless, it provides a fair estimation of the power consumption. The team is considering using the QOITECH Otii Arc probe, a professional energy consumption analyser offering high-resolution current measurement within the range of 0 – 5A [35]. Furthermore, this gadget is accompanied with a GUI desktop application for convenient logging and analysis, which likely will bring more insight into the device power consumption where previous profiling attempts failed to deliver.

3.7 Challenges Encountered during Project Execution

There are no projects gone through without coming across obstacles. This project was no exception, and there were indeed a few hurdles that should be mentioned. This section points out two challenges encountered during the project yet not fully solved: downlink payload length limitation (mentioned in Section 3.2.3) and the unsustainable connection session issue.

The first obstacle, the downlink payload limitation, arose during the integration of DTLS certificate authentication on the device side to secure the connection. It turned out that a payload length of 512 bytes was not sufficient for carrying out the handshaking procedure even with all the MTU and DTLS fragment size set to the most appropriate value, and according to U-blox manual this limit is fixed. As the real limit seems incorrectly documented since the received payload can be near 1500 bytes, this DTLS handshaking problem virtually does not exist. The fix for this issue was relatively simple as the only thing to do is increasing the limit of the software socket read, and it has already now been applied to the program. However, the current Mbed OS SARA-N211 driver implementation follows this stated limit and only tries to read at most 512 bytes, leaving the remaining part of the payload in the queue of the modem and mess up subsequent socket reads.

The second obstacle, the unsustainable connection session, originates from the Network Address Translation (NAT) technique used by the network operator. As commonly known, IPv4 address space is exhausted, and ISPs and network operators have been

resorted to NAT for quite some time to mitigate this IPv4 shortage conundrum. From observation, the author suspects that the technique applied in this situation is Port Address Translation (PAT). PAT is a form of dynamic NAT, translating multiple local source addresses to a single global IP address and port. As a result, it is theoretically possible to map 65535 (16-bit) source ports to a single IP. Figure 25 illustrates the usage of the PAT technique in a hypothetical network.

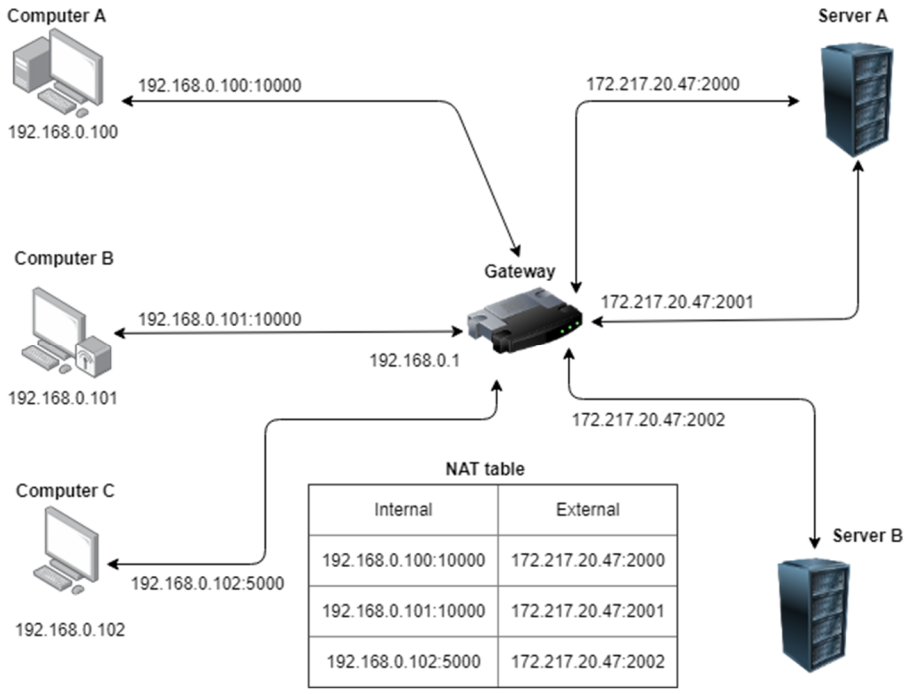

Figure 25. An Illustration of a PAT usage. Modified from techdifferences.com [36].

According to observations, every minute or more often, the operator carries out network port housekeeping and removes the UDP port associations between the cellular module and the outgoing server that stays inactive for at least a minute. This behaviour means that every time the device sends a packet to the server, the packet may look as if they originate from a different port under the server's perspective. Initially, the DTLS protocol identifies the connection based on the source IP and port [37], hence the sudden port change invalidates the security context and kills the session. An NB-IoT device can periodically send a byte to the server as a makeshift to maintain the record within the operator's NAT table. This one-byte packet does not affect regular operation since the DTLS layer discards it on the server-side as an invalid packet, yet this solution ruins the low energy consumption. There are a few resolutions perceived at the moment: migrating

to IPv6, bringing the server into the same subnet as the cellular subscriptions with VPN to avoid PAT, or using the recently proposed “Connection ID” feature in DTLS to retain the security context upon address change. It is easy to recognise that the first resolution requires changing the cellular module, and the second resolution needs close cooperation with the network operator. The third solution is probably the most favourable, yet in 2019 this feature is still an IETF draft and may not yet be supported by DTLS libraries.

4 Result and Discussion

This section states the project outcome and brings along brief discussions regarding relevant topics such as alternatives for NB-IoT, firmware upgrade, security and possible development of the project.

4.1 Project Outcome

The project turned out to be a success as the developed device is capable of using NB-IoT to register with an LwM2M server securely, and it can periodically push new weather measurements and other information to the cloud as expected. The customer was pleased to receive the device along with the supporting software, and also gives a compliment on the fast project pace after seeing the demo and testing out the device [38]. Furthermore, it has been more than half a year from the delivery, yet the project manager has not received any complaint from the customer. Considering this project a success, the upper management decided to push the project even further with more sophisticated features such as LTE-M, CAT-NB2, or 2G fallback.

The project was an excellent opportunity to explore unfamiliar topics including NB-IoT, LwM2M, Mbed OS, and to acquire new experience regarding ST MCU, microcontroller security as well as power consumption optimisation. Besides, this experiment also helped realise potential commercial cases, at the same time, identifies the limitations of the current technology and other unforeseen obstacles. Artefacts of this project are going to be used as a reference design, speeding up time to market for similar upcoming projects.

4.2 Comparison between NB-IoT with Competing Technologies

Low power wide area network (LPWAN) technologies are gaining many tractions in the industry thanks to their low power and long-range characteristics fitting many IoT applications. Currently, three LPWAN technologies standing out from the crowd: Sigfox, LoRa, and NB-IoT.

4.2.1 Comparison from Business Perspective

This subsection offers a brief overview of the business models behind these LPWAN technologies. Though deviating from the technological topic, developers may benefit from knowing some of the non-technical aspects of the technology and be better prepared for commercial projects.

Sigfox is a French company who offers a patented LPWAN solution under the same name, which operates on the unlicensed ISM band. The company owns all the technologies related to the backend, cloud server and endpoint software, but keeps the market open for device endpoints. As a result, many silicon providers such as STMicroelectronics, Atmel, and Texas Instruments have designed radio modules based on Sigfox's specification. This openness contributes significantly toward the low cost of end device radio, which Sigfox believes is the key to bring customers into their ecosystem [39]. In brief, Sigfox revenue comes from partnering with local companies to roll out Sigfox networks, which has been available at 58 countries by the time of September 2019.

LoRa (**Lo**ng **Ra**nge) is a proprietary physical layer technology owned by Semtech which operates under the unlicensed ISM spectrum, similar to Sigfox's approach. However, Semtech has a different approach compared to Sigfox's business strategy. In contrast, the company builds its business around patented hardware while recommending adopters to join the LoRaWAN alliance and design their own LoRa equipment and applications around Semtech's IC. As a result, businesses can deploy their private LoRa networks at a low cost, or even develop a customised protocol based on LoRA physical layer [40], which helps to boost the LoRa adoption, especially at places without cellular coverage. [39]

On the other hand, NB-IoT specification is a part of the 3GPP standard, which has been revised by multiple giant telecom organisations. Contrary to the mentioned technologies operating on the unlicensed spectrum, NB-IoT network operates mainly under the expensive licensed spectrum owned by mobile operators. Even though there are possibilities of operating a private LTE/NB-IoT network, this option is only available to businesses. With the involvements of multiple parties, along with the complexity of the technology and multiple certifications to be met, NB-IoT ends up as the most expensive out of these three LPWAN technologies. Notwithstanding, an NB-IoT device is more convenient to set up in areas with cellular network coverage, not to mention the high certainty in quality of service as the operating band is reserved.

4.2.2 Comparison in Terms of IoT Factors

With different physical and modulating strategies, Sigfox, LoRa, and NB-IoT exhibit different strengths and weaknesses. This section, inspired by “A comparative study of LPWAN technologies for large-scale IoT deployment” paper [41], attempts to give a rough comparison between these technologies on quality of service (QoS), battery life and latency, payload length, network range and coverage. This comparison gears more towards NB-IoT as it is a topic of this thesis.

First of all, NB-IoT offers the best QoS as it is based on LTE - a synchronous protocol that operates on the licensed spectrum. In contrast, Sigfox and LoRa are asynchronous protocols on the unlicensed spectrum, thus being more susceptible to signalling problems. On the other hand, NB-IoT is the worst candidate when it comes to battery life because it relies on synchronous communication, and the module requires a few times higher operating and sleep current compared to those of the other two. For applications where low latency is demanded, NB-IoT (with appropriate connection configurations) and class C LoRa devices (LoRa devices with always enabled receiver) are good options, though these setups are costly in terms of energy.

In terms of payload, again NB-IoT supports the largest payload size of 1600 bytes compared to the other two, though it has sometimes been seen that the real limitation depends also on the specific implementation. In the meantime, LoRa supports a maximum uplink size of 243 bytes and Sigfox 12 bytes, making them less competitive in situations

where a large amount of data/measurements is expected. From a subjective opinion, more than often, LoRa's packet size is adequate for sensor data collecting applications.

Even though these LPWANs offer much further range than most other IoT wireless technologies, it is still up to the application developer to check whether there is any support for the technology in the area of interest. Since operators run Sigfox and NB-IoT networks, when there is no immediate support nearby, LoRa becomes the only available options as it is possible to deploy private LoRa gateway. This situation often occurs in rural areas.

In summary, NB-IoT is suitable for applications with high QoS requirements and do not hold an extreme expectation on low power consumption as well as expenses. Next, LoRa is satisfactory in IoT applications which can be insensitive to latency, while prioritising higher battery life. Furthermore, it is relatively convenient to set up a private LoRa gateway where there is no immediate support for connectivity, but keep in mind that private LTE is also an option. On the other hand, Sigfox lies on the low end of the spectrum and mainly targets the most constrained and cost-sensitive applications.

4.3 Firmware Upgrade Feature Considerations

Receiving software updates seems to have become a normal part of modern life. On a regular basis, people receive smartphone application updates with new fancy features. On a regular basis, people silently receive Windows updates, which are suddenly revealed with a "Do not turn off your computer" notification at the most convenient time. Also on a regular basis, people read news where there is a zero-day vulnerability discovered, and there is a random expert who advises people to update a particular piece of software to the most recent version. There is an unofficial term known as "Patch Tuesday" referring to Microsoft's habit of releasing security patch at every second or fourth Tuesday of the month. These signs indicate that software updating has become a norm, and long gone are the days when people have to go buy a new CD containing the latest release of their favourite software. With not much of a difference, firmware upgrade is and will become more and more of an indispensable part of the IoT world, especially when the number of deployed devices is in the thousands and each of them has an Internet connection. It is easy to see that good firmware update provides a mechanism

to add new features, fix bugs, and patch vulnerabilities. However, lousy firmware update could brick the device [42], which in the best scenario requires a manual update to fix. Though this crucial feature not yet implemented for the system at hand, it has already been placed into the to-do list.

The Cloud Security Alliance organisation (CSA) provides a list of recommendations for IoT firmware upgrade processes [43]. Let us examine some relevant points (modified) to the system in this project within the list considering the context:

1) Provide a way for devices to recover upon update failure. Firmware rolling back are to be considered.

The device firmware should be able to recover from a failed update by reloading the most recent working firmware to minimise service disruption. Rolling back could be considered as a feature to recover from a fatal defect existed in the latest firmware but not in previous versions. However, this rollback action needs to be authorised to avoid downgrade attack.

2) All updatable components should be able to receive an update.

At this stage, there are only two updateable components in this IoT system: the microcontroller and the SARA-N211 modem. Besides implementing an update mechanism for the MCU, it is essential not to forget the modem is also a part of the system. SARA-N211 incorporates two different ways to deliver the update: Firmware Over The Air update (FOTA) and Firmware update Over AT (FOAT). Neither of these methods was attempted in this project.

3) Update strategy should adapt to the constraint of the system.

There are a few different update strategies perceived at the moment: full image update, package update and differential update. For the MCU context, only full image update and differential update are applicable. As NB-IoT bandwidth is limited, the differential approach may be a more pleasant way to deliver an update. For a full image update, it is possible to apply compression to reduce the size of the payload. There is one small reminder for low power system: the longer the device has to stay awake to download the update, the more energy it consumes.

4) Updates should be authenticated and its integrity protected from end to end.

Updates should be authenticated to avoid being tampered during transmission and prevent malicious code from being injected into and executed on the device. This goal can be achieved by signing the firmware as a whole, so the device can verify the authenticity before the update is carried out and every time the bootloader going to make a jump to application code. It is too worth mentioning that authentication does not protect the device from downgrade attack. Moreover, the firmware update should be transferred via a secure medium (e.g. DTLS). Also, the device may want to authenticate the update server and vice versa. It is also good to consider encrypting the firmware with a secure cipher (e.g. AES CBC-128) to ensure confidentiality, especially if it is stored in unsecured place such as external flash.

5) The system administrator should be able to schedule updates.

One benefit of scheduling update is able to avoid network congestion or perform DoS unintentionally against the update server. It is a good idea to separate update downloading from update applying. Another benefit in this scheduling ability is administrator can gradually deploy the update to a small number of devices first for testing purpose before increase the deployment scale after a reasonable time. This method can help avoid a fatal update being delivered to all devices – a business nightmare.

Updating firmware, though its concept is simple, is, in fact, a very complex feature. There are undoubtedly much more things to be considered for a full-fledged updating system.

4.4 Security Considerations Regarding Project

It is indeed very challenging and complex to thoroughly analyse and counter security threats in advance for a system; however, it is often too late to build an application then start adding security. This subsection presents a few thoughts on the microcontroller based project regarding security considerations.

According to the STM32 application note “Introduction to microcontrollers security” [44], attacks on IoT devices can be classified into three categories: software attack, hardware

non-invasive attack, and hardware invasive attack. Software attacks are efforts of exploiting programmatical weaknesses and attempts to readout or modify device data without relying on a physical element. Due to the low starting cost, relatively convenient to carry out, likewise hackers can share their malicious expertise around the Internet, software attacks are the most common type of security threats. A general practice to mitigate security risks is to read and follow standard security guidelines while refraining from inventing homegrown security scheme without sufficient expertise. This practice may include implementing a proper secure boot to block unauthorised firmware, employing trustworthy libraries, conducting adequate software quality assurance, authenticating and encrypting communication with the cloud server, implementing reliable firmware update feature to patch newly discovered exploits. For the MCU context, disabling debug port and enabling flash read-write protections are a good idea to prevent firmware cloning and leaking device secrets, though the effectiveness of such actions on STM32 is to be examined [45] [46].

General-purpose MCUs are not good candidates for withstanding hardware related attacks. However, developers should be aware of common attack strategies, and if possible, follow countermeasure instructions to mitigate the risk, which will at least cost the attacker more effort before succeeding. For example, using a cryptographic library with fake instructions could obstruct a power analysis attack, or using an internal clock source can dodge a clock glitching attempt. However, it is reasonable not to invest excessive effort on countering hardware attack unless the device was specifically designed with such criteria. Nevertheless, developers should be aware of the existing physical security features offered by their platform, which again will make attacks more difficult to succeed.

Furthermore, post-detection of malicious attempts could be considered as good-to-have countermeasures. For instance, as an IoT device must present its unique identity with the management server, it is legitimate to suspect that a particular device identity has been cloned if it appears to connect to the server from multiple places that are physically apart from each other at the same time. Nonetheless, post tampering detection efforts do not necessarily exist only in software, for example, a broken protection case might signal the device has been physically tampered.

Last but not least, it is easy to realise that good security comes not only with a lot of cost and efforts, but also comes with obstacles for debugging or investigating issues occurs after the device has been deployed, which are deemed as bad things from a business perspective. Besides, more often than not, users care less about security compared to features and costs, making rooms for poor security habits.

4.5 UDP vs TCP as Transport Layer for IoT Applications

The TCP/IP model Transport layer offers two mainstream protocols: UDP and TCP. For non-critical IoT applications, the most crucial requirement often is being able to deliver collected data to the cloud. Most of the time, TCP is the preferred solution for most Internet applications as it guarantees packet delivery, offers congestion avoidance and automatically maintains the session for the connection. On the other hand, UDP is a more lightweight protocol as it is connectionless while its packet header is smaller, making it much more suitable for low-powered devices. Aside from the benefits UDP offered, this protocol exhibits three problems that might hinder adoption: no congestion avoidance, out-of-order delivery, and no guarantee of delivery. In typical power-constrained IoT usages, small amounts of measurement data are usually sent out infrequently, thus congestion avoidance and out-of-order delivery are not issues. For the last mentioned problem, it is up to the figure of the successful delivery rate to determine whether IoT applications want to employ UDP as the transport layer.

A blog post by Karl Seguin [47] describes an experiment which attempts to investigate how reliable UDP transfers are between five AWS instances: two in New Jersey, one in Los Angeles, one in Amsterdam and the last one in Tokyo. Over the duration of seven hours, every 9 - 11 seconds, each server picks a target among the remaining ones and send 5-10 packets ranging from 16 to 1016 bytes. The result shows that the worst delivery rate is 98.55% (1.45% loss), and during the test duration, there is a short period of one to two minutes that many of the packets are lost. Interestingly, the successful delivery rate between continents is better than within the US's soil. This experiment, though only run for a short time, gives an impression on the delivery rate of UDP over the Internet.

As NB-IoT is one of the interests throughout the paper, it is worth noting that the over-the-air link-layer in NB-IoT between the UE and base station does feature acknowledgement for uplink messages. According to U-blox's document, the N211 module will attempt to retry sending the same data once if not acknowledged by the base station at the first attempt. However, it is yet to be confirmed by the writer whether the base station will need acknowledgement from UE on downlink event. If there are important packets that need to be re-sent until acknowledged, reliability ensuring mechanism from the application layer should be responsible for such duty. For example, the CoAP layer handles this role for an LwM2M application. Remarkably, using TCP (as MQTT) on NB-IoT impacts the system worse than CoAP on UDP because CoAP confirming functionality is sufficient for ensuring packet delivery, while TCP saturates the system transfer capacity sooner than UDP [48].

It is advisable for developers to run a reliability test on their chosen NB-IoT network to evaluate whether it is suitable for their application usage. The test can be as simple as an NB-IoT module tries to send and receive numbered packets at random intervals and sizes to see how the delivery rate varies over time. During such trial, packet loss pattern or other conclusions can be derived from the percentage of packets reached destination along with related timestamps. After obtaining the result, developers can in advance identify issues such as firewall filtering, network address changes or poor signal quality problem. Nevertheless, this test gives developers more insight into their system, potentially helping them to ensure the service quality as well as allowing product owners to make appropriate business decisions.

4.6 Future Developments

Excluding overengineered projects, it can be said that no engineering work is truly complete, and to create a reliably functioning system, even simple ones, takes lots of effort. This project is no exception, and in fact, there is plenty of work to be done for the device to reach a mature state. Hardware-wise, there are already demands for upgrading the cellular modem to one that supports more sophisticated features, and the reference design should take into account different sources of power input while retaining the power-efficient characteristics. Furthermore, energy harvesting is also being considered a potential enhancement for the system. Imagine an NB-IoT device capable of operating on

battery for more than a year without any recharging (which is totally feasible), and if equipped with a solar recharging capability, its operation is likely to last for quite a long time.

On the other hand, there are a lot of tasks remaining to be done on the software side. First of all, the device firmware needs to accommodate support for new modems, which will support features not available with the current modem. In case the new modules support TCP, a reference example for an MQTT application will need to be developed since that is the current de facto protocol supported by cloud services. Another highly expected feature is the support for secure firmware upgrade. Recently, Winbond introduced a new authenticated SPI flash family W74M, which opens up a proper way to store data off the MCU flash securely and authenticatable. With this new capability, it is possible to transform Mbed OS power-loss resilient LittleFS into an encrypting file system by writing encrypted data to the external flash with the key stored on the MCU, resting assure that its data is always authenticatable. An applicable cipher for this situation could be AES-GCM as the MCU will always know the address/index of the data it is reading, thus making the flash still randomly accessible under this design.

Apart from improving the device firmware, the testing aspect of the project must be improved. Aside from adding more tests, finding out a way to incorporate code coverage reporting and execution profiling could give more insight into the software, potentially reveal unforeseen issues of the system.

Out of all these enhancements, the whole bundle should be portable to newer MCU that offers more robust security features, e.g. STM32L5 - an Arm Cortex-M33 with TrustZone hardware-enforced security.

In summary, as the device is currently offering a minimum level of functionality, there is a lot of work to be done in the future. Some of these enhancements, if done in-depth, may occupy a standalone thesis.

5 Conclusion

This project successfully created a proof-of-concept low power IoT device which makes use of NB-IoT, LwM2M, and Mbed OS. The resultant device fulfils its functionality expectation of being able to deliver sensor measurements securely to the LwM2M cloud server via NB-IoT, and it has been delivered to the customer. Initial investigations on the low power capability of the current design look promising as the modem and the MCU during deep sleep consumes only approximately 7uA, while the power consumption during operating time is reasonable. Artefacts from this project are going to be used as a reference design for the company, speeding up time to market for other similar designs.

Besides exploring the new NB-IoT technology, this project also acted as an attempt to evaluate the readiness of software components, including Mbed OS and LwM2M Wakaama. Furthermore, the project helps to realise the capability and performance of the technologies used, at the same time identifying shortcomings of these components and unforeseen obstacles. Moreover, this project also establishes a template for other projects which happen to build on Mbed OS regarding development environment, Git workflow, and software testing. On the other hand, additional researches for further improvements and relevant subjects are also conducted, which may help draw out a solid road map for the project in the future.

Though this paragraph marks the end of the conclusion of this paper, it certainly does not mark the finish line of this project. This project will be continuously improved, and at its maturity, placed available under Etteplan's Device Creation service.

Referencess

[1] C. Thompson, “Here's how IoT is transforming 6 different industries,” Business Insider, 25 Oct 2016. [Online]. Available: https://www.businessinsider.com/iot-transforms-industries-2016-10. [Accessed 7 May 2019].

[2] Ericsson, “Ericsson Mobility Report June 2019,” [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2019/ericsson-mobility-report-june-2019.pdf. [Accessed 28 Jun 2019].

[3] S. B. D. A. C. B. Iago Felipe TRENTIN, “Lightweight M2M protocol: Archetyping an IoT device, and deploying an upgrade architecture,” [Online]. Available: https://ieeexplore-ieee-org.ezproxy.metropolia.fi/document/8480313. [Accessed 27 Jun 2019].

[4] M. Saarnivala, “IoT Device Management Security,” 2016. [Online]. Available: https://www.arm.com/files/event/2016_ATS_India_B4_Mikko_Saarnivala.pdf. [Accessed 7 May 2019].

[5] Open Mobile Alliance, “Lightweight Machine to Machine Technical Specification,” [Online]. Available: http://www.openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A/OMA-TS-LightweightM2M-V1_0_2-20180209-A.pdf. [Accessed 27 Jun 2019].

[6] J. Vermillard, “Bootstrapping device security with LWM2M,” [Online]. Available: https://medium.com/@vrmvrm/device-key-distribution-with-lightweight-m2m-36cdc12e5711. [Accessed 26 Jul 2019].

[7] “IPSO Smart Objects Working Group,” [Online]. Available: https://www.omaspecworks.org/about/the-oma-specworks-work-program/ipso-smart-objects-working-group. [Accessed 4 Aug 2019].

[8] M. K. H. T. Jaime Jimenez, “IPSO Smart Objects,” [Online]. Available: https://www.omaspecworks.org/wp-content/uploads/2018/03/ipso-paper.pdf. [Accessed 29 Jul 2019].

[9] Open Mobile Alliance, “OMA LightweightM2M (LwM2M) Object and Resource Registry,” [Online]. Available: http://openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html. [Accessed 10 Sep 2019].

[10] Digi, “Mesh Networking vs Cellular Technology for IoT Applications,” [Online]. Available: https://www.digi.com/videos/mesh-networking-vs-cellular-technology-for-iot-ap. [Accessed 26 Jun 2019].

[11] M. Halper, “The Top 10 Cities Implementing Connected Streetlights: Miami, Paris and Madrid on top,” [Online]. Available: https://iot-analytics.com/top-10-cities-implementing-connected-streetlights. [Accessed 26 Jun 2019].

[12] L. Amicucci, “Mesh + Cellular: Ideal Partners for Industrial IoT,” 5 Dec 2018. [Online]. Available: https://blog.nordicsemi.com/getconnected/mesh-cellular-ideal-partners-for-industrial-iot. [Accessed 27 Jun 2019].

[13] “How NB-IoT and LTE-M Fit into the IoT Ecosystem: The Future of Cellular IoT,” 31 Aug 2018. [Online]. Available: https://www.qorvo.com/design-hub/blog/how-nb-iot-and-lte-m-fit-into-iot-ecosystem-future-of-cellular-iot. [Accessed 7 Aug 2019].

[14] Rohde & Schwarz, “Narrowband Internet of Things whitepaper,” [Online]. Available: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma266/1MA266_0e_NB_IoT.pdf. [Accessed 9 Jul 2019].

[15] GSM Association, “Mobile IoT Deployments,” [Online]. Available: https://www.gsma.com/iot/deployment-map/. [Accessed 26 Jun 2019].

[16] GSM Association, “Mobile IoT commercial launches,” [Online]. Available: https://www.gsma.com/iot/mobile-iot-commercial-launches/. [Accessed 26 Jun 2019].

[17] Rewheel-Tutela, “Site density is key to LTE network performance – and critical for 5G,” [Online]. Available: http://research.rewheel.fi/downloads/Rewheel_Tutela_LTE_5G_performance_drivers_Europe_17022019_FINAL.pdf. [Accessed 26 Jun 2019].

[18] Ashish Kumar Sultania, Pouria Zand, Chris Blondia and Jeroen Famaey, “Energy Modeling and Evaluation of NB-IoT with PSM and eDRX,” [Online]. Available: https://www.researchgate.net/publication/329364141_Energy_Modeling_and_Evaluation_of_NB-IoT_with_PSM_and_eDRX. [Accessed 2 Jul 2019].

[19] U-blox, U-blox, [Online]. Available: https://www.u-blox.com/sites/default/files/SARA-N2_SysIntegrManual_%28UBX-17005143%29.pdf. [Accessed 18 Aug 2019].

[20] Borja Martinez, Ferran Adelantado, Andrea Bartoli and Xavier Vilajosana, "Exploring the Performance Boundaries of NB-IoT," 18 Feb 2019. [Online]. Available: https://arxiv.org/pdf/1810.00847.pdf. [Accessed 3 Jul 2019].

[21] JianHuaWu, "NB-IoT Technical Fundamentals," [Online]. Available: https://www.keysight.com/upload/cmc_upload/All/20170612-A4-JianHuaWu-updated.pdf. [Accessed 18 Aug 2019].

[22] Arm, "An introduction to Arm Mbed OS 5," [Online]. Available: https://os.mbed.com/docs/mbed-os/v5.12/introduction/index.html. [Accessed 26 Jun 2019].

[23] ARM Mbed, "Greentea testing applications," ARM, [Online]. Available: https://os.mbed.com/docs/mbed-os/v5.13/tools/greentea-testing-applications.html. [Accessed 25 Jul 2019].

[24] ST Microelectronics, "STM32L4 series of ultra-low-power MCUs," [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series.html. [Accessed 25 Jun 2019].

[25] ST, "STM32L4 and STM32L4+ ultra-low-power features overview," Mar 2018. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/application_note/9e/9b/ca/a3/92/5d/44/ff/DM00148033.pdf/files/DM00148033.pdf/jcr:content/translations/en.DM00148033.pdf. [Accessed 2 Sep 2019].

[26] Ublox, "U-Blox product category," [Online]. Available: https://www.u-blox.com/sites/default/files/ProductCatalog_V22_2019Feb.pdf. [Accessed 25 Jun 2019].

[27] U-blox, "U-blox Cellular Modules Open Source Software Licenses," [Online]. Available: https://www.u-blox.com/sites/default/files/products/documents/OpenSourceSWLicensesCellular_AppNote_%28UBX-13001917%29.pdf. [Accessed 25 Aug 2019].

[28] U-blox, “SARA-N2 series AT Commands Manual,” [Online]. Available: https://www.u-blox.com/sites/default/files/SARA-N2_ATCommands_%28UBX-16014887%29.pdf. [Accessed 25 Aug 2019].

[29] Eclipse Foundation, [Online]. Available: https://www.eclipse.org/org/. [Accessed 5 Aug 2019].

[30] R. Barry, “Mastering the FreeRTOS™ Real Time Kernel,” Real Time Engineers Ltd, [Online]. Available: https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. [Accessed 18 Aug 2019].

[31] “STM32L4x5 and STM32L4x6 advanced Arm®-based 32-bit MCUs Reference Manual,” ST Microelectronics, [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf. [Accessed 2019 Aug 29].

[32] B. Szatkowski, “Low power features in Mbed OS,” [Online]. Available: https://drive.google.com/file/d/1nviXI1W--9pvymZLf4FGry-lniXqifvC/view. [Accessed 12 Sep 2019].

[33] ARM Mbed, “ARM Mbed Low Power Ticker porting guide,” [Online]. Available: https://os.mbed.com/docs/mbed-os/v5.13/porting/low-power-ticker.html. [Accessed 2 Sep 2019].

[34] J. Jongboom, “Power management in Mbed OS,” [Online]. Available: https://os.mbed.com/docs/mbed-os/v5.13/tutorials/power-optimization.html. [Accessed 29 Aug 2019].

[35] “Otii Arc technical specification,” [Online]. Available: https://www.qoitech.com/products/techspec#current-measurement. [Accessed 28 Aug 2019].

[36] “Difference Between NAT and PAT,” [Online]. Available: https://techdifferences.com/difference-between-nat-and-pat.html. [Accessed 5 Sep 2019].

[37] H. T. T. F. T. G. Eric Rescorla, “The Datagram Transport Layer Security (DTLS) Connection Identifier (work in progress),” [Online]. Available:

https://tools.ietf.org/html/draft-ietf-tls-dtls-connection-id-04. [Accessed 26 Aug 2019].

[38] Etteplan, “NB-IoT based platform for wireless data transfer,” [Online]. Available: https://www.etteplan.com/references/nb-iot-based-platform-wireless-data-transfer. [Accessed 30 Aug 2019].

[39] B. Ray, “SigFox Vs. LoRa: A Comparison Between Technologies & Business Models,” 31 May 2018. [Online]. Available: https://www.link-labs.com/blog/sigfox-vs-lora. [Accessed 1 Jul 2019].

[40] Link Labs, “Symphony Link vs. LoRaWAN,” [Online]. Available: http://info.link-labs.com/hubfs/LPWAN_Technology_Explained.pdf. [Accessed 15 Sep 2019].

[41] E. B. F. C. F. M. Kais Mekki, “A comparative study of LPWAN technologies for large-scale IoT deployment,” 20 Dec 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959517302953. [Accessed 8 Sep 2019].

[42] I. Thomson, “Firmware update blunder bricks hundreds of home 'smart' locks,” 11 Aug 2017. [Online]. Available: https://www.theregister.co.uk/2017/08/11/lockstate_bricks_smart_locks_with_dumb_firmware_upgrade/. [Accessed 3 Sep 2019].

[43] S. Khemissa, “Recommendations for IoT Firmware Update Processes,” [Online]. Available: https://downloads.cloudsecurityalliance.org/assets/research/internet-of-things/recommendations-for-iot-firmware-update-processes.pdf. [Accessed 3 Sep 2019].

[44] ST, “Introduction to STM32 microcontrollers security - Application note,” Feb 2019. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/application_note/group1/9f/0b/e4/b6/75/15/4f/e2/DM00493651/files/DM00493651.pdf/jcr:content/translations/en.DM00493651.pdf. [Accessed 4 Sep 2019].

[45] S. T. Johannes Obermaier, “Shedding too much Light on a Microcontroller’s Firmware Protection,” [Online]. Available: https://www.aisec.fraunhofer.de/content/dam/aisec/ResearchExcellence/woot17-paper-obermaier.pdf. [Accessed 4 Sep 2019].

[46] “How to bypass Debug Disabling on SM32F103,” [Online]. Available: https://medium.com/@LargeCardinal/how-to-bypass-debug-disabling-and-crp-on-stm32f103-7116e7abb546. [Accessed 5 Sep 2019].

[47] K. Seguin, “How unreliable is UDP?,” 16 Oct 2014. [Online]. Available: https://www.openmymind.net/How-Unreliable-Is-UDP/. [Accessed 07 Aug 2019].

[48] L. Anna, R. Antti and S. Juha, “Impact of CoAP and MQTT on NB-IoT System Performance,” 31 Oct 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/1/7/htm. [Accessed 27 Jun 2019].

NB-IoT applicable eDRX cycle length and paging time window

Applicable eDRX cycle length: 20.48s, 40.96 s, 81.92 s (~1 minute), 163.84 s (~ 3 min), 327.68 s (~5 min), 655.36 s (~11 min), 1310.72 s (~22 min), 2621.44 s (~44 min), 5242.88 s (~87 min), 10485.76 s (~175 min)

Applicable paging time window: 2.56s, 5.12s, 7.68s, 10.24s, 12.8s, 15.36s, 17.92s, 20.48s, 23.04s, 25.6s, 28.16s, 30.71s, 33.28s, 35.84s, 38.4s, 40.96s

S1 mode
The field contains the eDRX value for S1 mode. The E-UTRAN eDRX cycle length duration value and the eDRX cycle parameter 'T_{eDRX}' as defined in 3GPP TS 36.304 [121] are derived from the eDRX value as follows:

bit 4	3	2	1	E-UTRAN eDRX cycle length duration	eDRX cycle parameter 'T_{eDRX}'
0	0	0	0	5,12 seconds (NOTE 5)	NOTE 3
0	0	0	1	10,24 seconds (NOTE 5)	2^0
0	0	1	0	20,48 seconds	2^1
0	0	1	1	40,96 seconds	2^2
0	1	0	0	61,44 seconds (NOTE 5)	6
0	1	0	1	81,92 seconds	2^3
0	1	1	0	102,4 seconds (NOTE 5)	10
0	1	1	1	122,88 seconds (NOTE 5)	12
1	0	0	0	143,36 seconds (NOTE 5)	14
1	0	0	1	163,84 seconds	2^4
1	0	1	0	327,68 seconds	2^5
1	0	1	1	655,36 seconds	2^6
1	1	0	0	1310,72 seconds	2^7
1	1	0	1	2621,44 seconds	2^8
1	1	1	0	5242,88 seconds (NOTE 4)	2^9
1	1	1	1	10485,76 seconds (NOTE 4)	2^{10}

All other values shall be interpreted as 0000 by this version of the protocol.

NOTE 3: For E-UTRAN eDRX cycle length duration of 5,12 seconds the eDRX cycle parameter 'T_{eDRX}' is not used as a different algorithm compared to the other values is applied. See 3GPP TS 36.304 [121] for details.

NOTE 4: The value is applicable only in NB-S1 mode. If received in WB-S1 mode it is interpreted as 1101 by this version of the protocol.

NOTE 5: The value is applicable only in WB-S1 mode. If received in NB-S1 mode it is interpreted as 0010 by this version of the protocol.

NB-S1 mode
The field contains the PTW value in seconds for NB-S1 mode.The PTW value is used as specified in 3GPP TS 23.682 [133a].The PTW value is derived as follows:

bit 8	7	6	5	Paging Time Window length
0	0	0	0	2,56 seconds
0	0	0	1	5,12 seconds
0	0	1	0	7,68 seconds
0	0	1	1	10,24 seconds
0	1	0	0	12,8 seconds
0	1	0	1	15,36 seconds
0	1	1	0	17,92 seconds
0	1	1	1	20,48 seconds
1	0	0	0	23,04 seconds
1	0	0	1	25,6 seconds
1	0	1	0	28,16 seconds
1	0	1	1	30,72 seconds
1	1	0	0	33,28 seconds
1	1	0	1	35,84 seconds
1	1	1	0	38,4 seconds
1	1	1	1	40,96 seconds

Reference:

Section 10.5.5.32 in 3GPP TS 24.008 V13.12.0 (2017-12)

NB-IoT Deployment Guide to Basic Feature set Requirements Version 2.0 05 April 2018
https://www.gsma.com/newsroom/wp-content/uploads/CLP.28-v2.0.pdf

NB-IoT Active timer (T3324) and TAU timer (T3412) encoding

T3412 encoding

The 3 most significant bits define the timer unit, while the 5 least significant bits define the timer value.

Bits 6-8	Timer value unit
000	10 minutes
001	1 hour
010	10 hours
011	2 seconds
100	30 seconds
101	1 minute
110	320 hours
111	timer deactivated

Example: `00100110`. Unit: 1 hour. Value: `00110` = 6. Result: 1 hour * 6 = 6 hours.

T3324 encoding

The 3 most significant bits define the timer unit, while the 5 least significant bits define the timer value.

Bits 6-8	Timer value unit
000	2 seconds
001	1 minute
010	6 minutes
111	timer deactivated

Example: `00000010`. Unit: 2 seconds. Value `00010` = 2. Result: 2 seconds * 2 = 4 seconds.

Reference: https://docs.nbiot.engineering/tutorials/low-power.html