Norwegian University of Science and Technology
&
Nordic Semiconductor

Electronic Systems Design and Innovation

Master Thesis
TFE4930

IoT Protocol Stack Current Optimizations for the nRF9160 SiP

Authors:
Simen S. Røstad

E-mail:
Simen.Rostad@nordicsemi.no

Supervisors:
Jan Tore Guggedal
Pierluigi Salvo Rossi

E-mail:
JanTore.Guggedal@nordicsemi.no
Pierluigi.Salvorossi@ntnu.no

June 15, 2020

Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU

Abstract

IoT Protocol Stack Current Optimizations for the nRF9160 SiP

by Røstad

This thesis presents current measurements of the nRF9160 configured with a common IoT use-case. With optimal power-saving features enabled, the nRF9160 is capable of consuming an average of 267mC for cellular network attachment, 213.9mC for server connection establishment, 29.52mC for PSM sleep, and 28.76mC of application payload transmission of 536-bytes every hour. This yields a theoretical battery life of 772.15 days for a reference battery capacity of 300mAh. These results are based on the nRF9160 getting PSM T3412 timer greater than the utilized publication interval of 1 hour, PSM T3324 timer of 0 seconds (disabling eDRX), C-DRX interval timer of 0.32 seconds and CAT-NB1 as LPWAN of choice. However, the overall current consumption and battery life can be greatly extended by setting a larger publication interval, extending the utilized battery capacity, and reducing the overall payload size.

Denne oppgaven presenterer aktuelle målinger av nRF9160 konfigurert for et ordinært IoT bruksområde. Med optimale strømsparende funksjoner aktivert, er nRF9160 i stand til å konsumere et gjennomsnitt på 267 mC for oppkobling til mobilnettverket, 213,9 mC for etablering av serverforbindelse, 29,52 mC i PSM-søvn og 28,76 mC for overføring av applikasjonsspesifikk data på 536 byte, en gang hver time. Dette gir en teoretisk batterilevetid på 772,15 dager for en referansebatterikapasitet på 300mAh. Disse resultatene er basert på at nRF9160 får PSM T3412 timer verdi større enn det utnyttede publikasjonsintervallet på 1 time, PSM T3324 timer verdi på 0 sekunder (eDRX deaktivert), C-DRX interval ltimer på 0,32 sekunder og CAT-NB1 som valgt LPWAN. Det det totale strømforbruket og batteriets levetid kan utvides kraftig ved å sette et større publiseringsintervall, forlenge den utnyttede batterikapasiteten og redusere den totale størrelsen på overført data.

NTNU

Acknowledgement

IoT Protocol Stack Current Optimizations for the nRF9160 SiP

by Røstad

I would especially like to thank Jan Tore Guggedal and Jon Helge Nistad for the guidance you have provided me with throughout this project. Your input has proven valuable and has hopefully steered me in a reasonable direction. I also want to thank Stian Hafskjold for early access to the online power profiler tool and clarifications regarding the LTE stack behavior of the nRF9160. Other mentionable people are Roshan Rajaratnam for helping me create a CoAP server in java and Pierluigi Salvo Rossi for being my internal supervision at NTNU.

Declaration of Authorship

I, Røstad, declare that this master thesis is my original work except where otherwise is indicated.

Simen Sigurdsen Røstad

List of Abbreviations

3GPP	Third **G**eneration **P**artnership **P**roject
API	**A**pplication **P**rogramming **I**nterface
BSD	**B**erkeley **S**oftware **D**istribution
CoAP	**Co**nstrained **A**pplication **P**rotocol
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IDE	**I**ntegrated **D**evelopment **E**nvironment
IoT	**I**nternet **o**f **T**hings
IP	**I**nternet **P**rotocol
JAR	**J**ava **AR**chive
LPWAN	**L**ow **P**ower **W**ide **A**rea **N**etwork
LTE	**L**ong **T**erm **E**volution
LWM2M	**L**ight**w**eight **M**achine to **M**achine
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
NB	**N**arrow **B**and
NCS	**N**ordic **C**onnect **S**oftware Development Kit
NTNU	Norwegian University of Science and Technology
PDCCH	**P**hysical **D**ownlink **C**ontrol **Ch**annel
PLMN	**P**ublic **L**and **M**obile **N**etwork
PSM	**P**ower **S**aving **M**ode
PTW	**P**aging **T**ime **W**indow
RAM	**R**andom **A**ccess **M**emory
RF	**R**adio **F**requency
RRC	**R**esource **R**adio **C**ontrol
RTOS	**R**eal **T**ime **O**perating **S**ystem
SDK	**S**oftware **D**evelopment **K**it
SIM	**S**ubscriber **I**dentity **M**odule
SiP	**S**ystem **i**n **P**ackage
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
UDP	**U**ser **D**atagram **P**rotocol

Contents

Chapter 1

Introduction

Throughout the information age, there has existed a tremendous push to connect everything to everything. This concept of interconnecting devices, often referred to as the Internet of Things, aims to utilize wireless technology to enable device intercommunication over short and vast distances. In the last decade, the combination of new features in cellular standards and more energy-efficient computing have spawned a new wave of cellular IoT devices focused on high mobility and power conservation. Cellular IoT devices are designed to be lightweight, compact, and maintenance-free, meaning that they cannot carry large batteries and must possess the ability to last years in between battery charges. Developers aim to make cellular IoT devices as energy efficient as possible through the exploitation of new features in cellular standards and optimizing each wireless node for current consumption and low operating expenses. Examples of such new wave IoT devices are long-range environmental monitors and GPS based asset trackers.

1.1 Problem Statement

This project carries out the design and current measurements of an nRF9160 based testbench system configured to mimic the behavior of a wireless sensor node. The purpose of this project is to identify the IoT protocol stack configuration for the nRF9160 that yields the lowest possible current consumption, and possibly uncover some tradeoffs in achieving so. In other words, **What is the lowest possible current consumption obtainable for a common IoT use-case utilizing the nRF9160?**. The process of answering this problem statement involves testing for different publication intervals, payload sizes, message-transport stack protocols, and parameters present in the LTE-M and NB-IoT cellular standards. The test results will form the basis of future implementations targeted towards making the nRF9160 more current efficient and aid developers in making informed choices when configuring the nRF9160 for power conservation.

1.2 Disposition

This project divides into eight chapters, where each chapter documents some essential elements of this research paper. Chapter two covers the theoretical motivation of the assignment by explaining in necessary detail the functionality and features of the technology utilized in this project. Chapter three documents a configurable testbench capable of mimicking different IoT use-cases. It explains every component of, as well as the synergy between firmware, software, and hardware in the system. Chapter

four documents the use-case the testbench is configured with, and why. It also covers the current measurement and simulation process, and choices made during the test phase to adequately substantiate the test results. Chapter five contains test results from two independent current measurement sources, along with the purpose of each test, utilized test parameters, and discussion of the individual tests. The final chapters, six, seven, and eight discusses and concludes the entirety of the project, and presents further work.

Chapter 2

Theory

2.1 nRF9160 SiP

The nRF9160 is an ultra-low-power System in Package with an integrated modem that is compliant with 3GPPs LTE release 13 LTE-M (CAT-M1) and release 13/14 NB-IoT (CAT-NB1, CAT-NB2) LPWAN protocol specifications. The nRF9160 is a globally certified SiP that supports operation on a wide range of LTE bands. In addition to the onboard modem, the SiP includes an ARM Cortex-M33 application processor, on-chip flash and RAM along with cryptographic security features. [25]

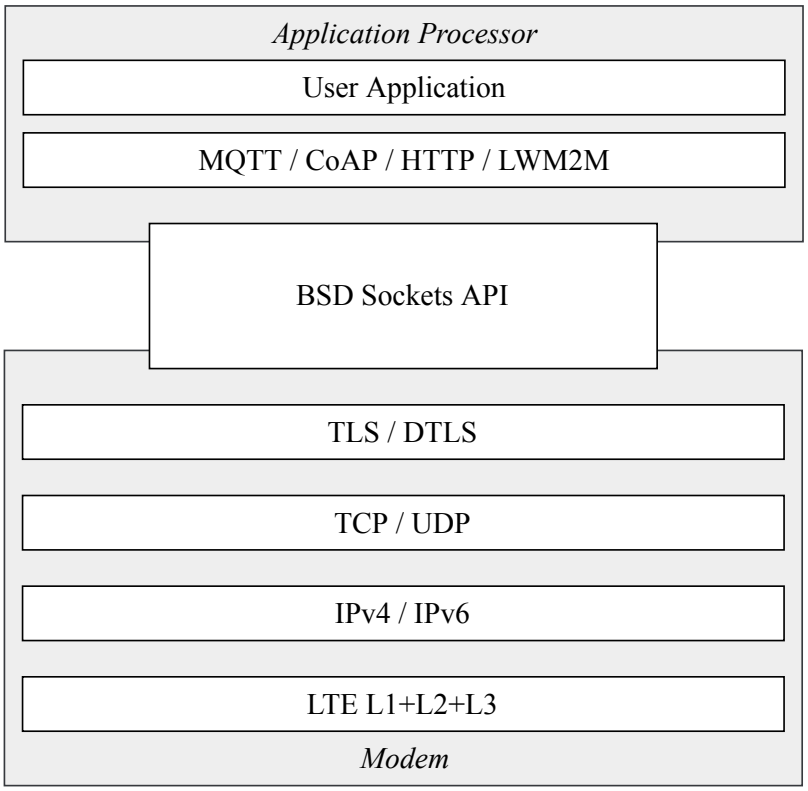

Figure 2.1: nRF9160 application- and modem firmware protocol stack. [20]

The onboard modem LTE stack layers L1-L3, and Ipv4/Ipv6 + TCP/UDP network protocols are integrated components of the modem firmware which interfaces with the application processor through a BSD secure sockets API. The BSD secure sockets API abstracts application-layer behavior and protocols such as MQTT and CoAP away from

modem coherent tasks. See figure 2.1 for an illustration of the nRF9160 network protocol stack. To enable ultra-low-power applications, the nRF9160 support PSM, eDRX, and RAI power-saving features present in the CAT-M1 and CAT-NB1 protocol standards. The nRF9160 operates between 3.0 and 5.5 volts.

2.2 nRF9160 DK

The nRF9160 Development Kit is a hardware development platform designed for the evaluation of, and firmware development for the nRF9160. Figure 2.2 depicts a physical sample of the nRF9160 DK, and in the following list, key features of the DK are listed. [24]

- nRF9160 SiP
- nRF52840 SoC
- LTE antenna
- GPS antenna
- UI elements such as Buttons, switches and LEDs.
- Segger J-Link onboard debugger
- UART interface
- USB I/O interface
- SIM card socket for nano-SIM
- External power interface
- External current measurement interface

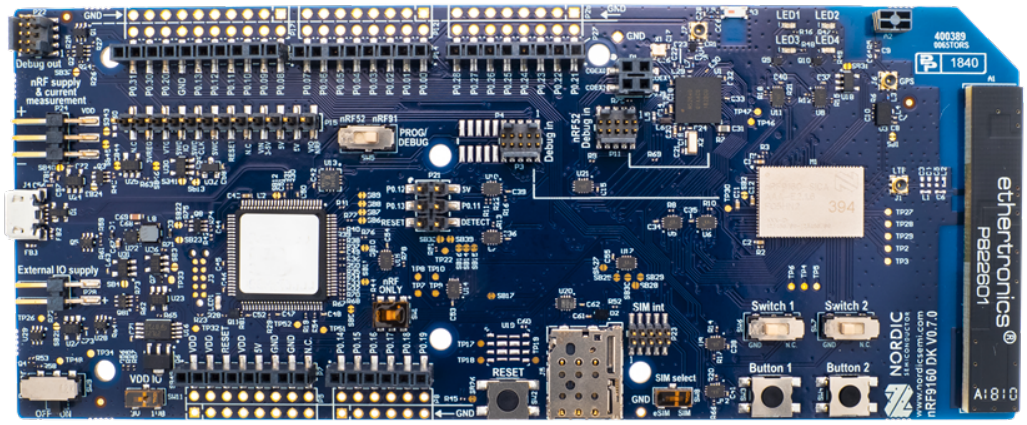

Figure 2.2: nRF9160 DK

Energy Measurement Current drawn by the nRF9160 can be monitored by connecting a power analyzer to the **VDD_nRF** bus on connector **P24** on the DK. The nRF9160 is powered either via USB or **VCC** and **GND** on external supply connector **P28**. The external Segger J-Link debugger must be disconnected from the power supply to ensure clean current measurements; this is done by setting the switch **SW1** to the leftmost position.

2.3 nRF Connect SDK

The nRF Connect SDK (sdk-nrf) is a software development kit hosted by Nordic Semiconductor. NCS depends on a set of partially open source projects that facilitates application development for the nRF9160. As of March 2020 NCS primarily depended on *nrfxlib*, *MCUboot*, and *The Zephyr Project*. [23]

nrfxlib The nrfxlib repository contains RTOS-independent libraries compatible with Nordic Semiconductors SoCs and SiPs. The *BSD library*, which is a part of *nrfxlib*, is Nordic Semiconductors implementation of the BSD sockets API, which is a set of standard function calls that can be used in a firmware application. The BSD library is the primary interface to operate the nRF9160 modem via the application processor in order to establish LTE-M, NBIOT, and GNSS connections. [26].

Zephyr Project The Zephyr Project is a scalable RTOS supporting multiple hardware architectures, optimized for resource-constrained embedded devices. NCS depends on a GitHub fork of the Zephyr Project, which is updated frequently to take advantage of new features in the RTOS. Zephyr includes standard kernel services that can be taken advantage of in application firmware development. Such kernel services include multi-threading, timers, interrupts, dynamic memory allocation, and power management. [14].

MCUBoot The MCUBoot Project is a secure bootloader for 32-bit MCUs. MCUboots intention is to provide a common infrastructure for the bootloader and system flash layout of the MCU system and enable easy software upgrade. [11]

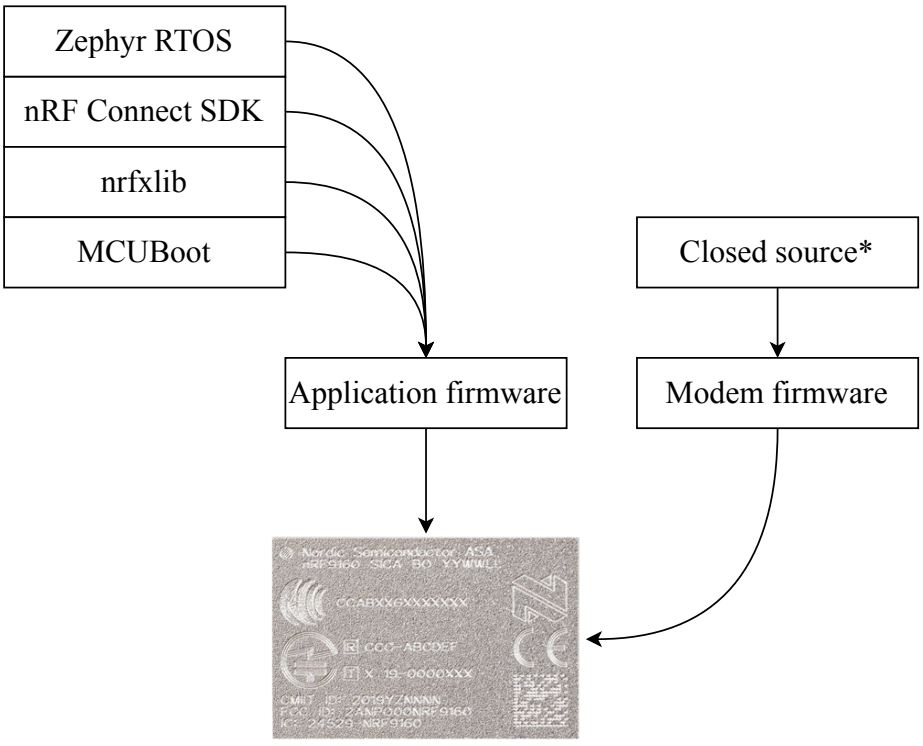

Figure 2.3: nRF9160 firmware dependencies.

2.4 The IoT Protocol Stack

The IoT Protocol Stack is visualized as an extension of the TCP/IP protocol model and involves six layers; the physical layer, data link layer, network layer, transport layer, application protocols layer, and the application service layer. The application protocols and application service layer are often combined into the application layer, and the data link and physical layers are often combined into the network access layer. Figure 2.4 illustrates the TCP/IP model along with popular protocols associated to their respective layer. The figure also illustrates each layer's contribution to the payload size of a transmitted message. [15]

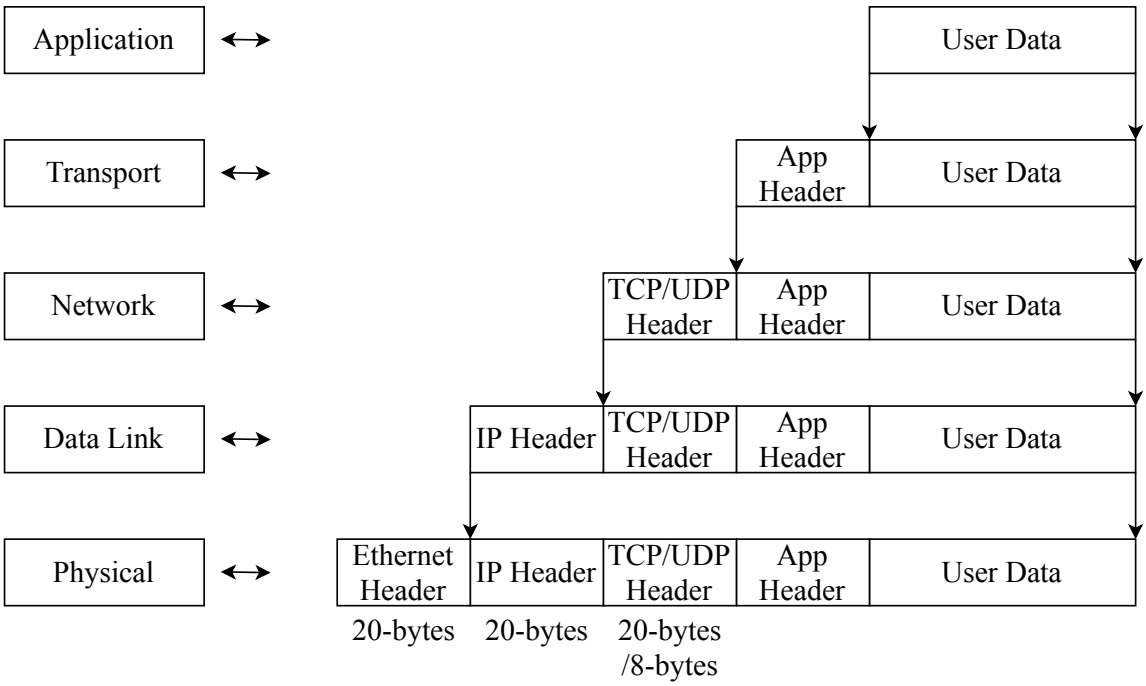

Figure 2.4: IoT Protocol Stack [29]

Application Layer Application protocols handle the communication between application layer entities and often do support readings or measurements from environmental sensors. Application protocols are used to manage control information that ultimately affects UI elements and actuators on the device. Application protocols dictate semantics and mechanisms for message exchange between communicating endpoints. There exist a range of competing application layer message protocols with MQTT and CoAP crowning as the most popular protocols in IoT.

Transport Layer The primary responsibility of the transport layer is to permit devices to communicate on a source-to-destination basis. The transport layer defines a level of services and status of the connection used when transporting data. The main protocols included in the Transport Layers are TCP and UDP. TCP provides a connected service where all packets are acknowledged. In contrast, UDP offers a fire-and-forget solution where packets are not acknowledged offering a smaller overhead in terms of

packet header size compared to TCP. Figure 2.5 illustrates the fundamental difference between TCP and UDP in how messages are transmitted between a client and server.

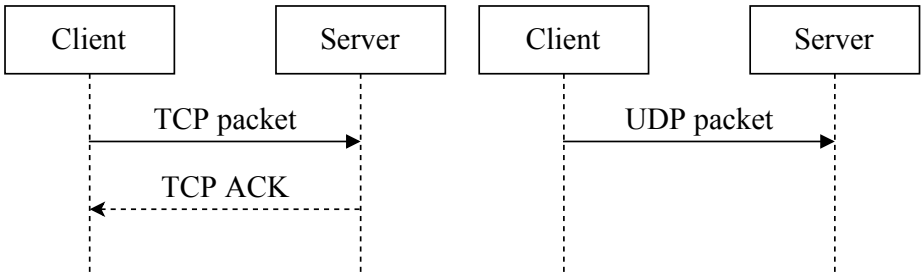

Figure 2.5: TCP vs UDP

Additionally, the TCP spesification includes the concept of retransmission. If a packet sent from either the client or server gets damaged or lost. The corresponding part will retransmitt its original TCP packet until a TCP ACK is received. In general, the TCP header conists of 20-bytes while the UDP header consist of 8-bytes.

Network Layer The network layer packages the data to be transmitted into data packets, commonly referred to as IP datagrams. The datagrams contain source and destination address information that is used to forward the datagrams across networks between hosts. The most common protocol in the internet layer is the IP protocol.

Network Access Layer The network access layer combines the data link layer and the physical layer of the OSI model. It defines specifications on how data is transmitted over the network, including how bits are electrically or optically signaled by hardware devices. The most common protocols included in the network access layer is Ethernet.

2.4.1 Transport Layer Security

Transport Layer Security provides a secure communication channel between a client and a server in transport protocols such as TCP and UDP. TLS is a cryptographic protocol which uses a handshake mechanism to negotiate various parameters to create a secure connection between a client and a server. A TLS handshake consists of a predetermined sequence of exchanged packets between the client and the server. The network sequence diagram in figure 2.6 illustrates a typical TLS handshake. TLS for UDP is often referred to as DTLS.

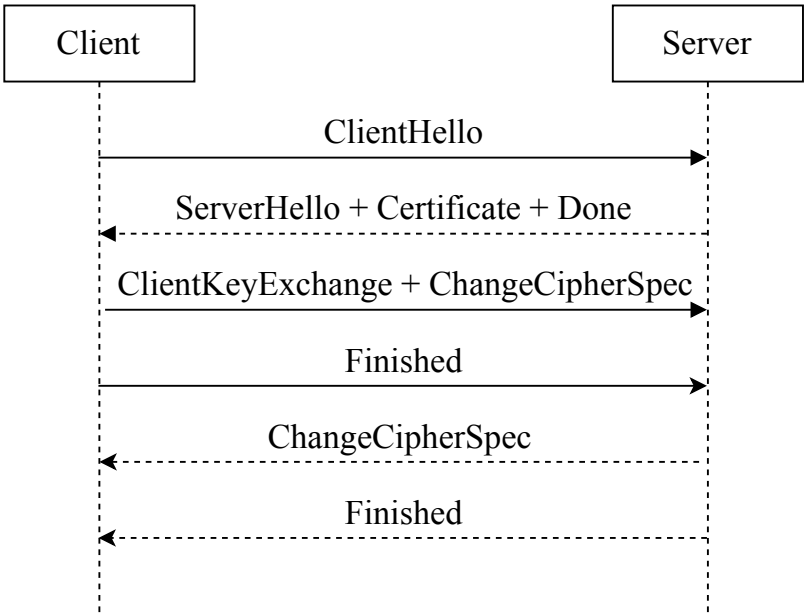

Figure 2.6: TLS Handshake

The TLS handshake consists of five overall steps. Initially, the client and server negotiate a cipher suite for the secure connection. The client transmits a *ClientHello* packet to the server containing a list of all supported cipher suites. The server then chooses a cipher suite found in that list and sends a *ServerHello* packet containing the selected cipher suite along with the public and private certificates to be used in the secure connection, and a packet telling the client that the server is done exchanging certificates. The client then authenticates the certificate and digital signature used in the connection. The last step is to perform key exchange functions that generate symmetric session keys for the client and server. After all these steps are performed, the connection is encrypted, and application data can be communicated securely.
The negotiated cipher suite determines the parameters of the TLS handshake, which effectively sets the strength of the secure connection. A cipher suite is a combination of algorithms used to negotiate security parameters during the TLS handshake. The following list contains typical algorithms of TLS/DTLS cipher suites. [6] [10]

- Key Exchange Algorithms (RSA, DH, ECDH, DHE, ECDHE, PSK)
- Authentication/Digital Signature Algorithm (RSA, ECDSA, DSA)
- Bulk Encryption Algorithms (AES, CHACHA20, Camellia, ARIA)
- Message Authentication Code Algorithms (SHA-256, POLY1305)

2.4.2 MQTT

The Message Queuing Telemetry Transport protocol is a lightweight, open, and easy-to-implement publish/subscribe messaging protocol that is designed for constrained embedded devices. MQTT is an application layer protocol that is considered a standard for communication within IoT products and is supported by most message broker/server services. MQTT enables interoperability and easy integration between supporting software solutions and IoT products. [12, 13]

Control Packet Types

The functionality present in the MQTT protocol revolves around specific MQTT Control Packets being exchanged in a specified manner between the MQTT Client and MQTT Broker. Table 2.1 lists the most fundamental Control Packet types in the MQTT specification.

Control Packet	Direction of flow	Description
CONNECT	Client ->Broker	Client request connection to Broker
CONNACK	Client <- Broker	Connection request acknowledgement
PUBLISH	Client ->Broker	Publish message containing application payload

Table 2.1: Fundamental MQTT Control Packet types.

2.4.3 CoAP

The Constrained Application Protocol is a specialized web transfer protocol for use with constrained nodes and constrained networks. CoAP provides a request/response interaction model between the application layers of the interacting client and server, where by default, messages are exchanged over UDP. [8]

Request/Response Model

CoAP requests and response semantics are carried in CoAP messages, which include either a Method Code or a Response Code. A request is either sent in a Confirmable (CON) or Non-Confirmable message (NON). Confirmable messages are confirmed with a corresponding (ACK) message, while Non-Confirmable message are not. If the endpoint does not respond to a Confirmable message, the message will be re-transmitted with exponentially increasing intervals until the acknowledgment is received.

Method Definitions

CoAP makes use of GET, PUT, POST, and DELETE methods in a similar manner to HTTP.

PUT The PUT method requests that content associated with a specified resource is to be replaced with the application payload present in the request.

2.4.4 LTE-M and NB-IoT

LTE-M and NB-IoT are modern cellular radio access technologies initially specified by 3GPP in release 13. LTE-M and NB-IoT address the fast-expanding market of low power wide area connectivity and do both undergo the LPWAN category of wireless network types. CAT-M1 and CAT-NB1 are subcategories of LTE-M and NB-IoT, which features the specifications listed in figure 2.2. [4] [5]

	CAT-M1	CAT-NB1
Bandwidth	1.4 MHz	200 KHz
Downlink	1 Mbps	∼27 kbps
Uplink	1 Mbps	∼60 kbps
Power class	23 dBm	23 dBm
Distance	∼13 Km [9]	∼15 Km [9]
Power-saving	PSM, eDRX, RAI	PSM, eDRX, RAI

Table 2.2: CAT-M1 and CAT-NB1 release 13 specifications. [19]

Discontinuous Reception

In radiocommunications, there are, in general, two types of operating modes when the User Equipment (UE) is switched on. These operating modes are often referred to as Radio Resource Control (RRC) states where the UE can exist in either connected mode (RRC connected) or idle mode (RRC idle). In RRC connected mode, the UE uploads data to the network and monitors the physical control channel (PDCCH) for information about the cellular network and potential incoming network packets. Being in RRC connected consumes a lot of power because the UEs radio either transmits data or monitors for available data, meaning the radio is always active. Discontinuous reception (DRX) allows the UE to stop monitoring the PDCCH and enter sleep mode for a certain period of time. Higher DRX interval will affect the UE's responsiveness but does, in turn, offer lower current consumption, which is especially useful for energy constraint devices, not demanding high responsiveness. Figure 2.7 illustrates RRC connected DRX (C-DRX).

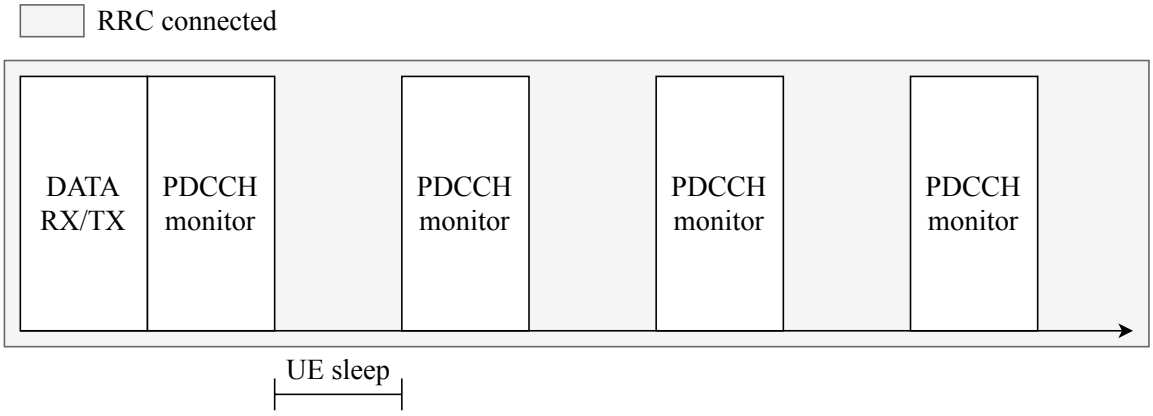

Figure 2.7: C-DRX

Upon a RRC connection release due to explicit signaling by the network or the RRC inactivity timer[1] timing out, the UE will change its RRC state to RRC idle. In RRC idle mode, the UE performs network maintenance procedures such as Public Land Mobile Network (PLMN) selection, cell selection, and reselection and receive/transmit DateTime from/to the cellular network. DRX in RRC idle mode, often referred to as I-DRX, enables the UE to sleep between such network maintenance procedures in a similar fashion as C-DRX. The first illustration in figure 2.8 illustrates the behavior of the UE in I-DRX mode.

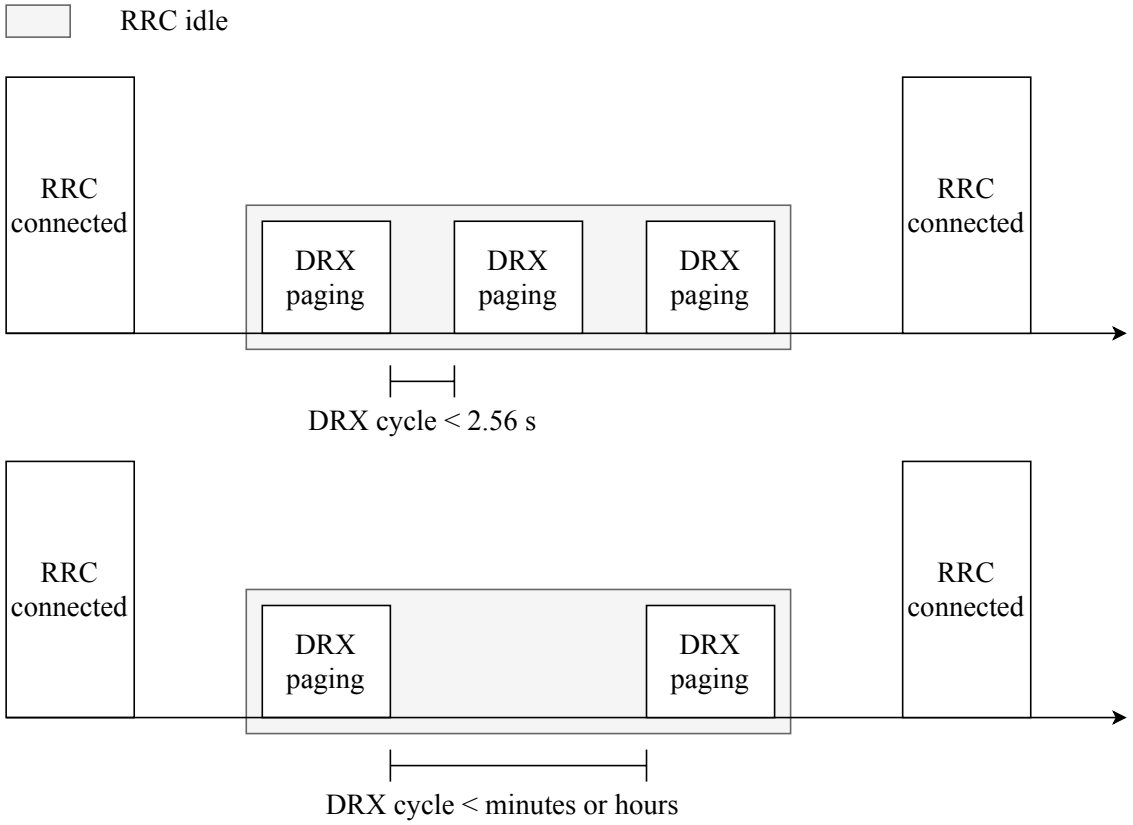

Figure 2.8: DRX vs eDRX

Extended Discontinuous Reception Extended Discontinuous Reception (eDRX) allows the UE to *extend* the sleep cycle between DRX paging to hours and minutes, offering even more significant savings in energy consumption than traditional DRX. Figure 2.8 illustrates the main difference between I-DRX and I-eDRX in RRC idle mode. The same principle applies to C-eDRX, where the C-DRX paging sleep cycles can be greatly extended.

[1]The RRC inactivity timer is a cellular network controlled timer that is based on the UE not having any downlink or uplink data within a set amount of time.

Power Saving Mode

Power Saving Mode (PSM) is a power-saving technique that further exploits the concept of sleep between communication instances in order to achieve even lower current consumption compared to eDRX. PSM introduces two new timers, Tracking Area Update (TAU) timer T3412 and Active Timer (AT) T3324. To enable PSM, the UE will include desired timer values for the two timers in either the initial connection request ATTACH[2] or in a standalone TAU request to the cellular network. When the network accepts a timer request, it will include the granted timer values in its network ATTACH-accept- or standalone TAU-accept message. Note that it is not guaranteed that the network accepts LTE parameter values requested by the UE. Figure 2.9 illustrates the correlations between PSM timers in the context of UE network communication behavior. When the cellular network has given its granted timer values, the UE can enter PSM.

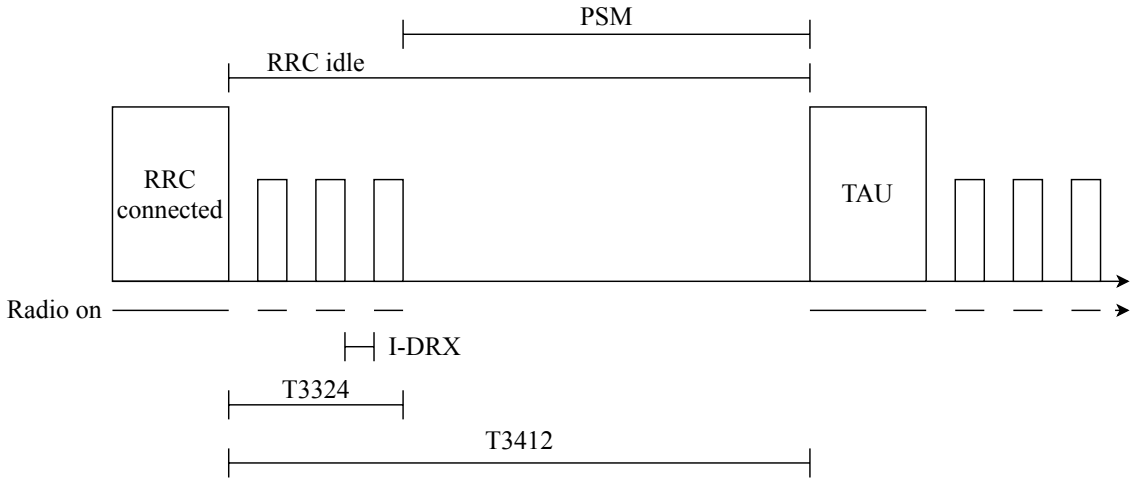

Figure 2.9: UE behaviour in PSM.

Release Assistance Indication

Release Assistance Indication (RAI) allows the UE to include an additional flag in its transmission message, signifying that it has no more data to transmit and that it does not expect any data from the cellular network. This makes the cellular network release the UE to RRC idle mode earlier than usual, enabling the UE to preserve more energy.

[2] ATTACH is the initial message transmitted by the UE signifying that it wants to connect to the cellular network.

Chapter 3

Testbench

3.1 System Synergy

The testbench system's main purpose is to measure the current consumption of the nRF9160 while it is running a configurable application firmware that supports multiple IoT use-case instances. The testbench comprises of components to measure current consumption, analyze network stack behavior, and to carry out behavior specific to a use-case. See figure 3.1 for a general overview of the testbench setup.

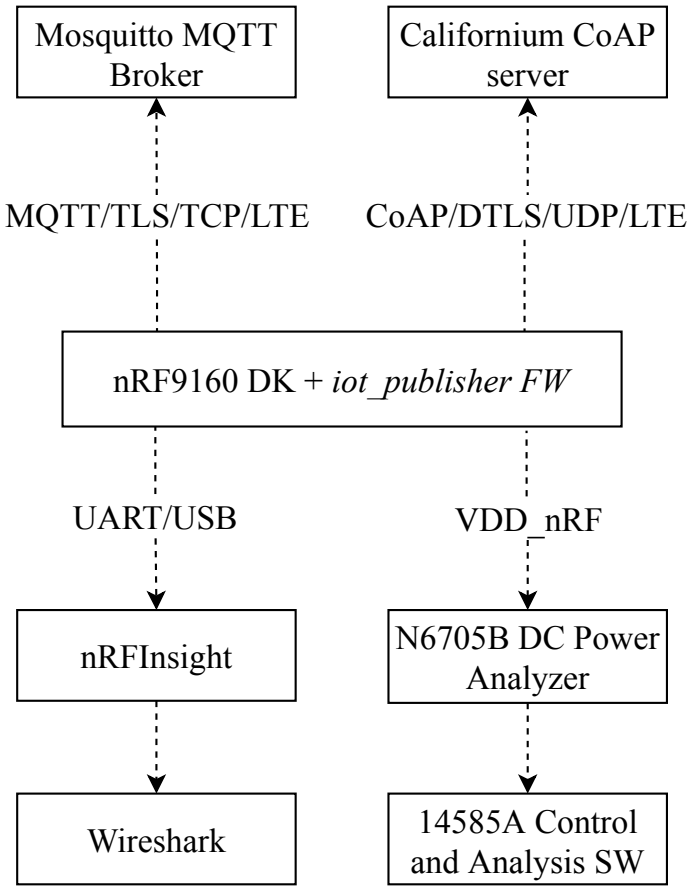

Figure 3.1: Testbench System Synergy

The application firmware runs on an nRF9160 DK exchanging data with either a Mosquitto MQTT broker or a Californium CoAP server, both running on a remote public IP server

[1]. The nRF9160 DK connects to a computer running Linux via USB where nRFinsight[1] is used to capture modem data traces that are forwarded to Wireshark to monitor network traffic between the nRF9160 and the remote MQTT/CoAP servers. To measure power consumption, a Keysight N6705B DC Power Analyzer connects to the power supply of the nRF9160 on the DK. The Power Analyzer also supplies the power of the DK to control input power parameters such as voltage and maximum current. The Power Analyzer can also simulate battery capacities[2]. The Keysight 14585A Control-and-Analysis-Software is used to manage the Power Analyzer and visualize the current consumption of the nRF9160.

3.2 The *iot_publisher* Application Firmware

The *iot_publisher* application firmware was developed according to three design objectives. These are reconfigurability, flexibility, and simplicity. The design objectives were chosen to streamline the testing process enabling fast switching between device test parameters, easy debugging upon firmware malfunction, isolation of test critical network functionality, and support for the necessary modem/network parameters. The application firmware sequentially publishes data to a remote server and supports a range of configurable options. The configurable options alter the network capabilities of the device, application payload, message format, and the frequency in which the device will publish data. The application firmware supports four network protocol stack combinations depicted in figure 3.2.

MQTT	MQTT	CoAP	CoAP
TLS (Optional)	TLS(Optional)	DTLS(Optional)	DTLS(Optional)
TCP	TCP	UDP	UDP
IP	IP	IP	IP
CAT-M1	CAT-NB1	CAT-M1	CAT-NB1

Figure 3.2: IoT Network Protocol Stack

For more information regarding the *iot_publisher* application firmware refer to the project GitHub repository [16] and the attached API reference in appendix A.2. The application firmware version GitHub commit hash for this project is **24f12578df5869609468716b80db11bbeafa58bb**.

3.2.1 Application Firmware Architecture

Figure 3.3 illustrates the correspondence between the different firmware modules utilized in the application firmware. Only the most relevant modules and interconnections are illustrated for simplicity. For extended information regarding non-project specific

[1] Nordic Semiconductor internal modem trace capture and analysis software tool.

[2] A large portion of Embedded IoT devices runs on battery, being able to simulate battery capacities enables the testbench to support a broader range of use-cases.

firmware module refer to the NCS documentation [21] and the Zephyr RTOS documentation [14].

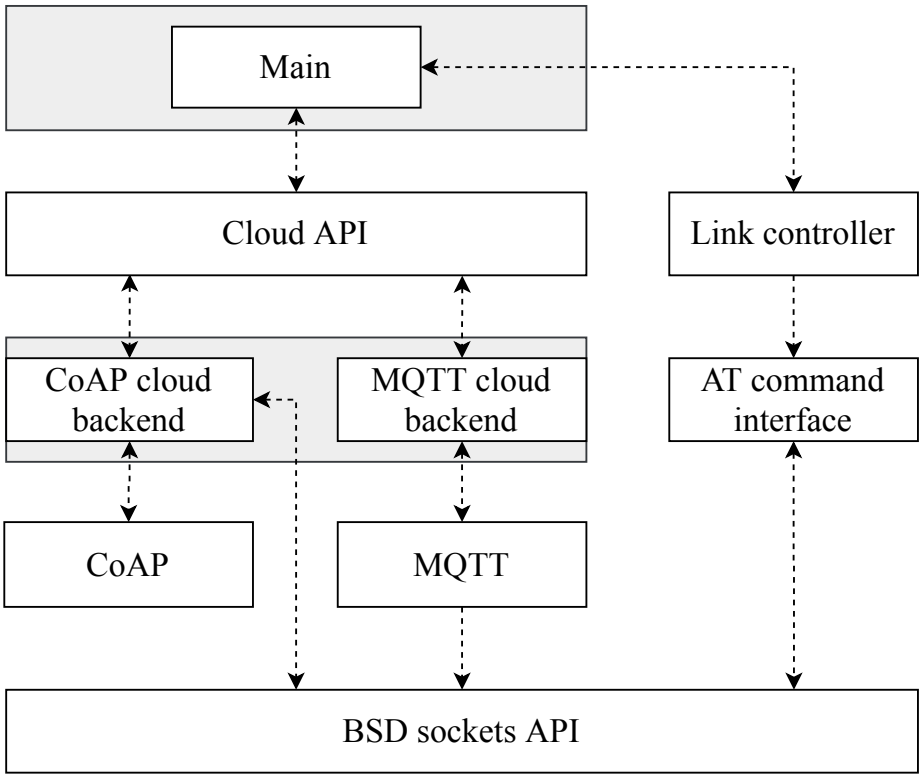

Figure 3.3: Application firmware modules, grey marks project specific libraries.

Main Governing module. Decides how frequent data is to be published to the dedicated server, payload format, -size and -content, desired message/transport layer protocol combination, and the desired LTE network parameters.

Cloud API NCS module The generic Cloud API enables cloud backends with different functionality to be interchanged by manipulating a single configurable option. This functionality maintains the integrity of the main module and does not require altering of top-level code in order to satisfy each cloud backends distinct network stack combination.

CoAP cloud backend The CoAP cloud backend is designed to support communication with a CoAP server and supports both non-secure UDP communication and secure DTLS over UDP server communications. The CoAP cloud backend is compatible with the generic cloud API.

MQTT cloud backend The MQTT cloud backend is designed to support communication with an MQTT message broker and supports both non-secure TCP communication and secure TLS over TCP server communications. The MQTT cloud backend is compatible with the generic cloud API.

CoAP and MQTT Zephyr RTOS modules The CoAP and MQTT modules formats and controls the data traffic according to the specification of the respective message protocol.

Link controller and AT command interface NCS modules The link controller module issues single or combinations of AT commands to the onboard modem to configure the desired LTE network parameters. Responses from the modem can also be subscribed to for applications depending on data received from the modem.
The link controller utilizes the AT command interface module to issue AT commands to the modem through the BSD socket API.

BSD Sockets API Zephyr RTOS module The BSD Sockets API is the primary interface for communication between the application and the nRF9160 onboard modem. [18]. See figure 2.1 in section 2.1 for a general network protocol stack implementation for the *iot_publisher* application- and modem firmware.

3.2.2 Configurations

The application firmware configurable options are a part of Zephyr RTOS's Kconfig configurable system. For more information regarding Kconfig, refer to section 2.3. Table 3.1 lists testbench relevant configurations.

Configuration	**Description**
CLOUD_MESSAGE	Message published to the server in string format.
CLOUD_PUBLICATION_INTERVAL	Interval in which messages will be published to the server.
CLOUD_BACKEND	Sets the utilized cloud backend.
MQTT_BACKEND_BROKER_HOST_NAME	MQTT broker hostname address.
MQTT_BACKEND_BROKER_PORT	MQTT broker hostname adress port.
MQTT_BACKEND_TLS_ENABLE	Enables secure connection to MQTT broker.
MQTT_BACKEND_SEC_TAG	Location of the security credentials used in the secure connection.
MQTT_BACKEND_KEEP_ALIVE	The frequency that the device sends periodic pings to server.
COAP_BACKEND_BROKER_HOST_NAME	CoAP server hostname address.
COAP_BACKEND_BROKER_PORT	CoAP server hostname adress port.
COAP_BACKEND_DLTS_ENABLE	Enables secure connection to CoAP server.
COAP_BACKEND_SEC_TAG	Location of the security credentials used in the secure connection.
COAP_BACKEND_KEEP_ALIVE	How often the device sends periodic pings to server[3].
LTE_PSM_REQ	Enable PSM.
LTE_PSM_REQ_RPTAU	Sets the PSM requested periodic TAU.
LTE_PSM_REQ_RAT	Sets the PSM requested active time.
LTE_EDRX_REQ	Enable eDRX.
LTE_EDRX_REQ_VALUE	Sets the eDRX intervals.
LTE_RAI_REQ	Sets the RAI level.

Table 3.1: *iot_publisher* configurations

3.2.3 Code Environment Setup

The application firmware code was developed with NCS on the nRF9160 DK using the Linux operating system with the tools specified in table 3.2. To install the tools required to build the firmware the following commands must be executed: [21]

```
sudo apt install --no-install-recommends git cmake ninja-build gperf \
  ccache dfu-util device-tree-compiler wget \
  python3-pip python3-setuptools python3-tk python3-wheel xz-utils file \
  make gcc gcc-multilib
sudo pip3 install west -U
```

These commands do not install the GNU Arm Embedded Toolchain and Segger J-Link Software so they must be downloaded and installed separately. Note that only version 7-2018-q2-update of the GNU Arm Embedded Toolchain is as of March, 2020 is compatible with NCS. After installation the GNU Arm Embedded Toolchain must be added to the PATH environmental variable. The recommendation is to rename the folder to "gnuarmemb", next add the following code line to the .bashrc shell script:

```
export ZEPHYR_TOOLCHAIN_VARIANT=gnuarmemb
export GNUARMEMB_TOOLCHAIN_PATH="~/gnuarmemb" # location to gnuarmemb folder
```

Create a folder with an appropriate name, access it, and clone the *iot_publisher* GitHub repository using the following command:

```
git clone https://github.com/simensrostad/fw-iot-publisher
```

Access the newly cloned repository and run the following commands to build all the NCS dependent repositories outside the *iot_publisher* folder.

```
west init -l
west update
```

After setting up NCS, the following three commands must be run from the folder containing the newly cloned repositories. The commands install additional python packages that the NCS code environment depends on.

```
pip3 install -r zephyr/scripts/requirements.txt
pip3 install -r nrf/scripts/requirements.txt
pip3 install -r mcuboot/scripts/requirements.txt
```

After the aforementioned steps are executed the file three should look like the below code listing:

```
<user specified name>
 |___ .west
 |___ mcuboot
 |___ fw-nrfconnect-nrf
 |___ nrfxlib
 |___ zephyr
 |___ fw-iot-publisher
 |___ ...

```

Building and Flashing

To build and flash the *iot_publisher* application firmware to a nRF9160 DK access the *fw-iot-publisher* folder and run the following commands with the nRF9160 DK connected to the USB interface of the PC.

```
1 west build -b nrf9160_pca20035ns
2 west flash
```

Modem Credentials

To set up a TLS/DTLS connection with a server, the onboard modem must be provided with security credentials. The security credentials enable cryptographic functionality in the onboard modem to properly negotiate the TLS/DTLS handshake and to encrypt/decrypt the secure server communications. The security credentials are flashed independently of the application firmware, directly to the modem using the *LTE link monitor* application present in the nRF Connect for Desktop application suite. nRF Connect for Desktop is downloadable through Nordic Semiconductors official website [22]. In the *LTE link monitor* application under *Certificate Manager* the desired credential combinations can be downloaded to the onboard modem and associated with a security tag. Upon a secure connection, the application firmware informs the onboard modem what credentials it wants to use by configuring the secure socket with a security tag. The modem then uses the security credentials associated with that particular security tag in the secure connection. Note that the modem can store multiple combinations of security credentials associated with *different* security tags. [28]

The nRF Connect for Desktop application suite contains a *programmer* application which can be used to flash the onboard modem with the latest firmware. The modem firmware utilized in this testbench is v1.2.0.

Tool	Type
IDE	Visiual Studio Code
Software development version Control	GitHub
Compilation tool	Cmake
Programming language	Python
Compilation tool	Device Tree Compiler
Arm processor compiler tool	GNU Arm Embedded Tool-chain
Repository manager	West v0.7.0
nRF9160 programming tool	Segger J-Link Software
nRF9160 programming tool	nRF Command-Line Tools
Development kit for nRF9160	nRF9160 DK v0.9.0
C compiler	GCC
nRF9160 programming tool	Ninja
Operating system	LINUX v18.04
Modem/link analyzer	LTE link monitor v1.1.1
Modem network tracer	nRFinsight v.1.1.5
Network packet capturer	Wireshark

Table 3.2: Tools and equipment utilized during firmware development

3.3 Test Servers

In order to properly test the *iot_publisher* application firmware, test servers for the configurable network stacks was created with support for secure communication channels to properly handle and decrypt/encrypt transferred data for different application firmware use-cases. The test servers are set up to have as many commonalities as possible in order to properly compare the different configurations of the *iot_publisher* application firmware.

Figure 3.4: Eclipse Californium CoAP and Mosquitto MQTT server frameworks, logos.

By default, the test servers listens on all IPv4 and IPv6 addresses on the local machine, port 1993 for the CoAP server and 1992 for the MQTT server. To test and monitor the secure connections the test servers uses Pre-shared-key authentication. PSK does not support the strongest ciphersuites available for TLS and DTLS, but allows for network-transport-layer-data-traffic decoding in wireshark[4]. Both servers uses the following PSK credentials, and ciphersuite in their TLS and DLTS connection:

- Pre-shared Key: 73656372657450534b ("secretPSK" in hex representation)
- PSK identity: Client_identity
- Ciphersuite: **TLS_PSK_WITH_AES_128_CBC_SHA256**

The testbench system also supports a more secure cipher suite based on ECDHE, **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256**. This cipher suite is very common in modern network solutions and would typically be supported by a finished IoT products demanding a highly secure connection. This level of security is often required for MQTT and CoAP communications with cloud providers' standards solutions.

[4]Wireshark can be configured to decrypt secure communication between client and server by associating the pre-shared key with the cryptographic protocol.

3.3.1 MQTT Eclipse Mosquitto Broker

The configurations and executables for the test servers can be cloned using the following command. The GitHub repository is located at [17]. GitHub commit hash **aab877cb0854bf96d582092d547f302e39a2589d**

```
1 git clone https://github.com/simensrostad/iot-test-servers
```

The MQTT broker test server uses Eclipse Mosquitto v.1.6.8. To download Mosquitto run the following command:

```
1 sudo apt-get install mosquitto
```

In order to run the MQTT TLS server run the following command:

```
1 cd iot-test-servers/mqtt_tls_test_server
2 mosquitto -c mosquitto.conf -v
```

The MQTT broker can be configured by manipulating the *mosquitto.conf* file with valid configurations found in the Mosquitto MQTT broker documentation. [3]

3.3.2 CoAP Eclipse Californium Server

A CoAP/DTLS server was created in java utilizing the Eclipse Californium CoAP server framework [2] v.2.0.0. The server supports a single customizable CoAP storage *resource* that retains the application payload of incoming CoAP PUT requests[5], by default the resource is named *iot_publisher*. The CoAP server was developed with the tools specified in figure 3.3.

Tool	Type
Build automation tool	Maven
Programming language	Java DK v11.0.6
Code library	Eclipse Californium CoAP/DTLS framework v.2.0.0

Table 3.3: Tools and equipment utilized during development of the CoAP test server.

To run the CoAP test server, access the folder containing the server JAR executables and run one of the following commands depending on if the application firmware is configured to use DTLS + UDP or plain UDP without security.

```
1 cd iot-test-servers/coap_dtls_test_server
2 java -jar coap-dtls-server.jar PSK   # Secure option
3 java -jar non-secure-coap-server.jar # Non-secure option
```

The executable JAR file uses the server configuration present in the *Californium.properties* file located in the same folder. These configurations can be altered to change the behavior of the CoAP test server.

[5] If the server receives multiple PUT requests addressing the *iot_publisher* resource, the server will replace the older retained application message with the new one.

Chapter 4

Use-Case and Test Conditions

4.1 Wireless Sensor Network Node

A common use-case of portable embedded devices enabled by wireless technology is a WSN node. A WSN node is typically a low powered device that continuously samples sensor data and transmits the data sequentially to a higher power node over a wireless network. These types of low-powered devices are often utilized as environmental monitors where the device reports changes in some environmental factors such as pressure, temperature, and position. This use-case configuration mimics the behavior of a typical low powered WSN node connected to the LTE network transmitting an application payload of a predetermined size sequentially. The WSN node use-case is ideal when testing current consumption over different IoT Protocol Stack configurations. This is due to the non-complex behavior of the WSN node, making testing, debugging, and reconfiguring easier. In the undermentioned lists, the constant and variable application firmware configurations for this use-case are listed. The constant configurations do not change throughout *physical* testing, which aids in isolating the variable configurations that are of interest to be evaluated.

Constant Test Configurations

- Application Payload: 536 byte JSON object string[1]
- QoS: Non-confirmable (CoAP), At most once (MQTT)
- CoAP storage resource, MQTT topic: iot_publisher
- MQTT Broker keepalive: negligible[2]
- Mobile Network Provider (MNO): Telia

Variable Test Configurations

- Message/transport protocol
- Publication interval
- PSM T3412 and T3324 timers
- eDRX interval timer
- LPWAN (Cellular network mode)
- RAI

[1] The JSON object string format is utilized during physical testing with the DC Power Analyzer testbench setup, while testing with the Online Power Profiler utilizes variable payload sizes with undefined content.

[2] A MQTT broker keepalive time exceeding the device publication interval is always requested, ensuring that the link is kept alive solely by publication messages.

Application Payload and Network Sequence Diagrams The application payload comprises of multiple JSON string objects containing data retrieved from the onboard modem of the nRF9160. This payload includes LTE network data, GPS data, and general device data, such as SiP supply voltage. Also, data from an external accelerometer is included. All data value entries have corresponding timestamps representing the time the data was sampled in UNIX time. See appendix A.1 for the entire application payload JSON string object. The WSN node use-case supports two message protocols that manifest different network stack behavior, MQTT and CoAP. The sequence diagrams illustrated in figure 4.1 and 4.2 describes the behaviour of the *iot_publisher* application firmware configured with the WSN node use-case for both the MQTT and CoAP stack configurations. The network sequence diagrams represent the message, transport, and internet layer communication between the client (nRF9160) and the server.

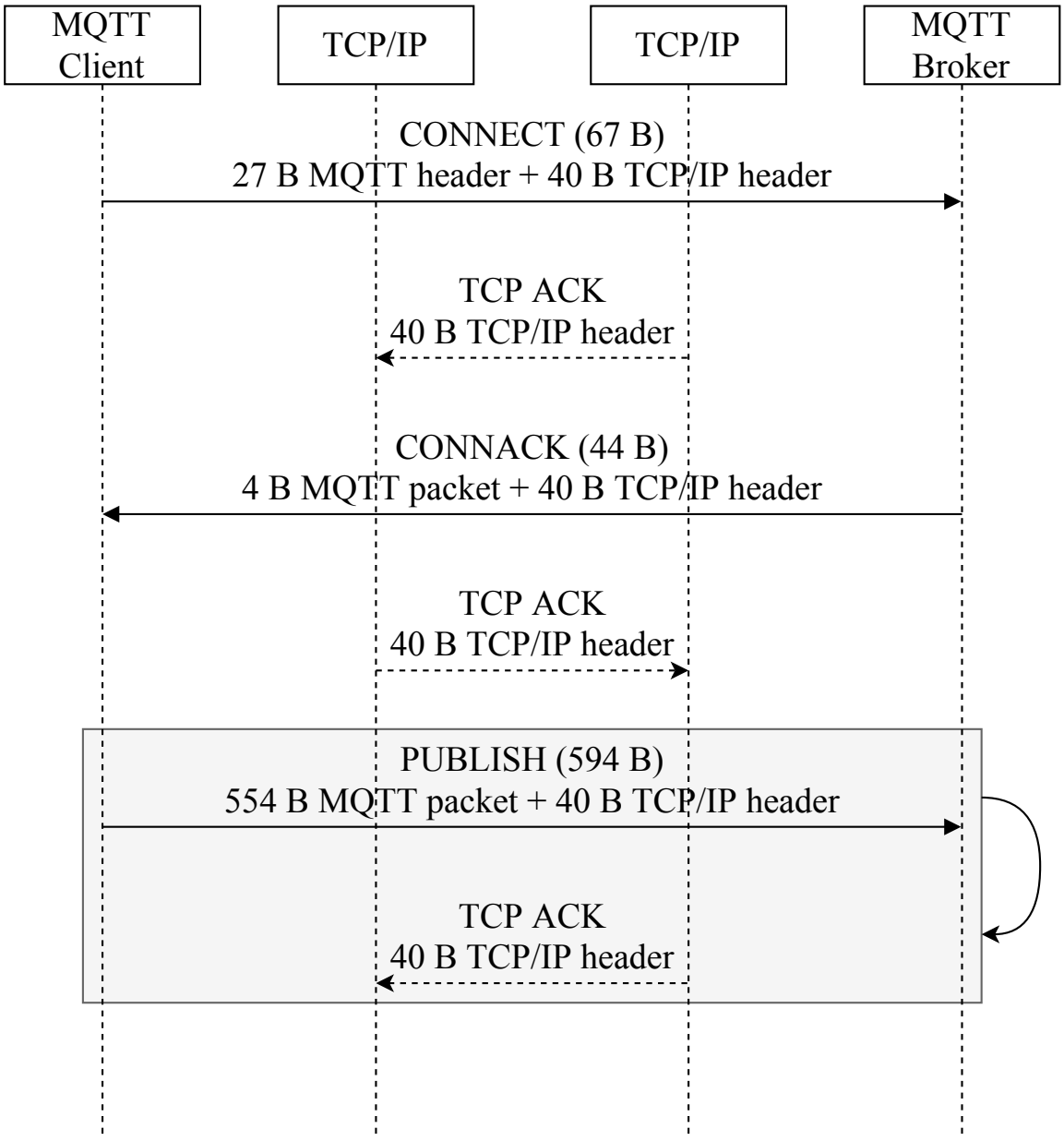

Figure 4.1: MQTT/TCP network stack implementation.

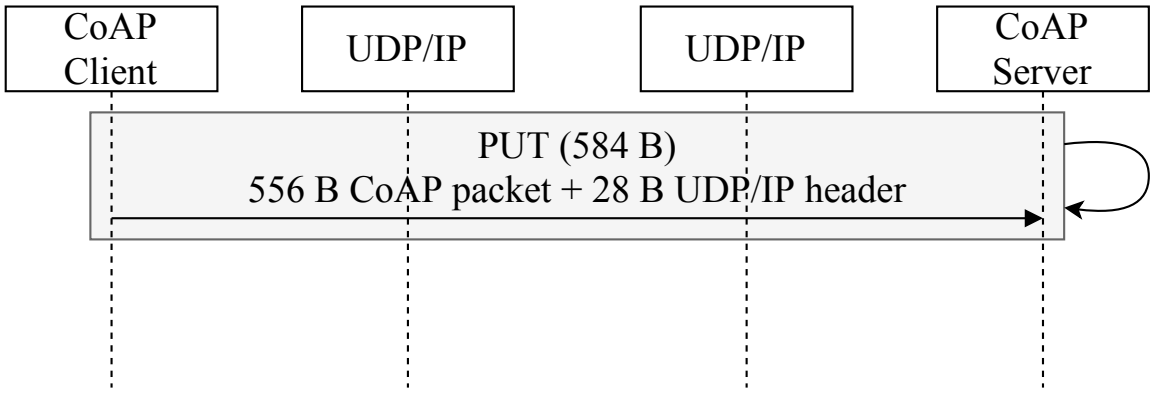

Figure 4.2: CoAP/UDP network stack implementation.

Use-Case Parameters The device's configurable publication interval is respected by disabling frequent ping requests to the server. The ping requests purpose is to keep the MQTT connection alive by upholding the MQTT server's keepalive time. By default, the MQTT client requests a connection keepalive greater than its publication interval, meaning that the MQTT connection is kept alive solely by publication messages. CoAP does not possess the concept of keepalive timeout. However, like MQTT, the underlying transport layer "connection" depends on the NAT session timeout. [7]

The Quality of service of publication messages is set to the lowest possible. *At most once* for MQTT and *non-affirmative* for CoAP. This level of QoS means that none of the messages containing the application payload published to the server gets an acknowledgment at message-protocol level. This suits the low powered WSN node use-case where limiting network traffic is more important than acknowledging every publication of sensor readings. Even though the use-case parameters do not support higher levels of QoS, it still exists a fundamental difference between the CoAP and MQTT stack configurations. MQTT uses TCP, which relies on transport level acknowledgment while CoAP, which relies on UDP, does not. This means that even though there exist no acknowledgement at message-protocol level for MQTT, all messages are still acknowledged at transport level.

4.2 Current Measurement and Simulations

The testing was conducted with two primary tools of measuring the total average current consumption of the nRF9160 configured with the WSN node use-case specified in section 4.1.

1. Online Power Profiler (OPP), theoretical setup [27]
2. DC Power Analyzer (DCPA) setup described in section 3.1, physical setup

Tradeoffs between OPP and DCPA The OPP is theoretical with certain constraints. It is limited to a transmission output power of 23, 10 or 0 dBm, UDP packets only, nRF9160 supply voltage level of 3.7V, no support for C-DRX estimations and no support for eDRX Paging Time Window (PTW). On the other hand, the OPP allows for fast result generation from input parameters, which makes it suitable to test over a vast amount of input parameters in a stable and controlled environment. In contrast to the OPP, the DCPA setup has no restrictions but generally consumes more time per test, especially in tests depending on intervals and timers spanning several hours. The application firmware also has to be flashed with new configurations between every test, which is not feasible when performing extended testing of hundreds of stack configurations. In addition to monitoring current consumption in the DCPA setup, the IoT protocol stack of the UE needs to be continuously monitored with nRFinsight/Wireshark to ensure that no unexpected behavior occurs during testing, polluting the results. Such unexpected behavior can be a modem/cellular network related (for instance, not getting the requested LTE parameters from the cellular network and SIM card not shutting down properly in PSM) or firmware related (crashing/halting). The OPP produces ideal results that are most likely not obtainable during real-world physical testing. However, the intention of using the OPP is not to provide accurate real-world-reflective current consumption results but mapping the impact of different UE configurations / LTE parameters have on current consumption for the given use-case. In other words, the relationship between different IoT protocol stack configurations with respect to current consumption. For stack configurations not supported by the OPP such as TCP traffic, the DCPA setup is utilized.

Method of Testing The testing was conducted in two stages. Initially, the OPP was used to map the current consumption of the nRF9160 over supported LTE- and device configurations. Then, the configurations from the OPP results yielding the most promising results were applied to the *iot_publisher* application firmware during physical testing with DCPA setup. The test results are presented in section 5.0.1 in the following format. Each test is associated with a number identifying the test, a purpose of the test, test configurations, utilized current measuring tool, results in the form of graphical depictions, and evaluation of the test. The graphical representations are based on measured values located in appendix A.6 and A.5. The. The evaluation contains battery lifetime estimations. This is to provide a *human graspable* metric relative to current consumption averages. This makes it easier to highlight good and bad results based on each configuration's contribution to the current consumption of the UE.

Battery Lifetime Estimations Battery lifetime estimations are calculated based on ideal conditions (100 % battery discharge efficiency) using equation 4.1 and 4.2 using a *reference* battery capacity of 300mAh. The following abbreviations are referred to throughout testing and discussion.

- UE: Device Under Test, nRF9160 DK
- BC: Battery capacity in mAs
- TAC: Total average current consumption in mA
- PSM: PSM event in mC
- PSMD: Duration of PSM event in seconds
- ATT: LTE attach to cellular network in mC
- CON: DNS + TLS handshake (optional) + MQTT connect (optional) in mC
- PUB: Publication of application data in mC.
- PUBD: Duration of publication event in seconds
- HS: 3600, seconds in a hour.
- HD: 24, hours in a day.

Measurements based on total average current (OPP):

$$\frac{BC}{TAC * HS * HD} = \underline{\underline{Days}} \tag{4.1}$$

Measurements based on charge (DCPA):

$$((BC - ATT - CON) * \frac{PSMD + PUBD}{PSM + PUB}) \div (HS * HD) = \underline{\underline{Days}} \tag{4.2}$$

Payload size The payload size referred to in the test results under *Test configurations* is the total size of the payload accumulated down to transport layer level. This means that the size of the payload reflects a combination of the application payload, MQTT/CoAP header, and TCP/UDP header.

Chapter 5

Results

5.0.1 OPP - Test #1; Baseline Simulations

Purpose of the test The purpose of this test is to identify the current consumption of the UE over varying payload sizes with PSM and eDRX disabled. This is to provide worst-case baseline measurements that can be compared against in tests where power-saving features are enabled.

Test Configurations

- Message protocol: NA
- Publication interval: 1 hour
- Payload size: Varying
- PSM disabled
- eDRX disabled
- C-DRX: 0.32 seconds CAT-M1, not supported in OPP for CAT-NB1
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer disabled
- Voltage: 3.7V

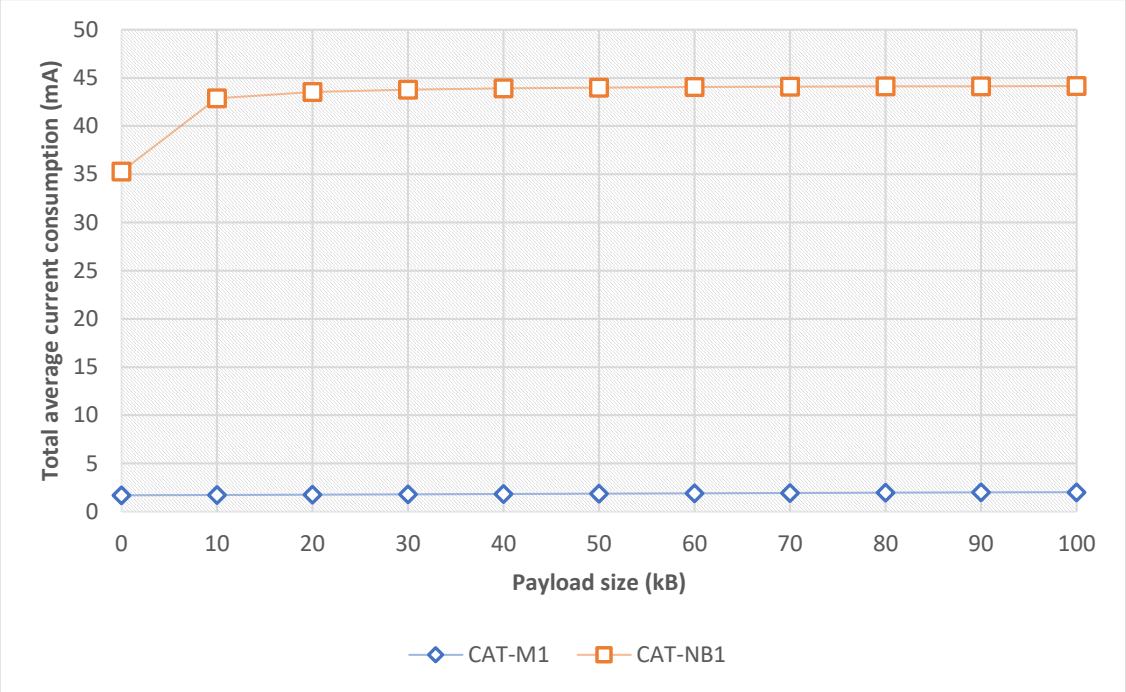

Figure 5.1: OPP - Test #1; Baseline Simulations

According to 5.1, a significant difference in current consumption at mA level is not noticeable until the payload size is incremented in terms of kilobytes. The total average current consumption for 0kB and 10kB transmissions for CAT-M1 and CAT-NB1 is 1.7mA to 1.73mA for CAT-M1 and 35.3mA to 42.87mA for CAT-NB1. Battery lifetime estimations with the reference battery charge for the aforementioned current consumption averages amounts to 7.353 days (1.7mA), 7.225 days (1.73mA), 0.3541 days (35.3mA) and 0.2916 days (42.87mA). This means that if the UE continuously stays in RRC connected, the difference between transmitting 100-byte payload sizes in contrast 10kB payload is only $\sim$ 3 hours for CAT-M1 and 1.5 hours for CAT-NB1. The test results suggest a very high radio start-up current consumption, which favors larger batched data transmissions at less frequent transmission intervals compared to smaller packets published at more frequent intervals. Being in RRC connected mode consumes 1.7mA on average for CAT-M1. That is for small payload sizes. In contrast, publishing 100kB payloads consumes 2.02mA on average in RRC connected mode. This fact suggests that the majority of the UE's current consumption occurs during network RRC connected network procedures such as monitoring the PDCCH and not when actually publishing application-specific data.

5.0.2 OPP - Test #2; PSM enabled - Variable Publication Intervals

Purpose of the test The purpose of this test is to look at the current consumption of the UE when PSM is enabled over different data publication intervals.

Test Configurations

- Message protocol: NA
- Publication interval: Varying
- Payload size: 100 bytes
- PSM T3412: Varying
- PSM T3324: 0 seconds
- eDRX disabled
- C-DRX: 0.32 seconds CAT-M1, not supported in OPP for CAT-NB1
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: 0 seconds
- Voltage: 3.7V

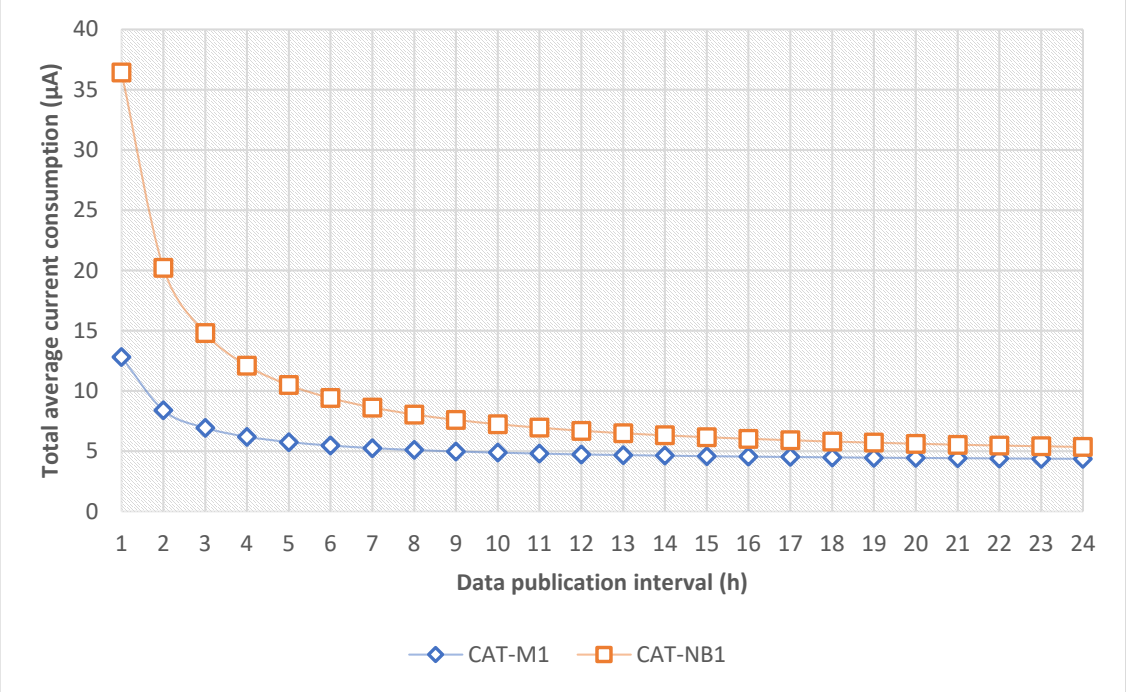

Figure 5.2: OPP - Test #2; PSM enabled - Variable Publication Intervals

Figure 5.2 illustrates a clear trend in the UE's current consumption, starting at 12.81μA for CAT-M1 and 36.43μA for CAT-NB1 when using a publication interval of 1 hour. The graphical trend in the figure displays a convergence of the UE's current consumption around the PSM floor current of 4μA [25] which makes sense becuase the UE spends most of its time in PSM. At 24 hours between transmissions, the UE consumes, on average 4.37μA for CAT-M1 and 5.35μA for CAT-NB1 yielding a theoretical battery lifetime of 2860.4 days (7.831 years) for CAT-M1 and 2336.4 days (6.397 years) for CAT-NB1 when calculating for the reference battery capacity. The difference in battery life when comparing the current consumption of the UE with and without PSM enabled is $\sim$ 2.6 years for publication intervals of 1 hour when using CAT-M1 and $\sim$ 1 year for CAT-NB1 proving that PSM offers substantial increases in current consumption.

5.0.3 OPP - Test #3; PSM enabled - Variable Payload Sizes

Purpose of the test The purpose of this test is to look at the current consumption of the UE with PSM enabled over increasing payload sizes.

Test Configurations

- Message protocol: NA
- Publication interval: 1 hour
- Payload size: Varying
- PSM T3412: 1 hour
- PSM T3324: 0 seconds
- eDRX disabled
- C-DRX: 0.32 seconds CAT-M1, not supported in OPP for CAT-NB1
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: 0 seconds
- Voltage: 3.7V

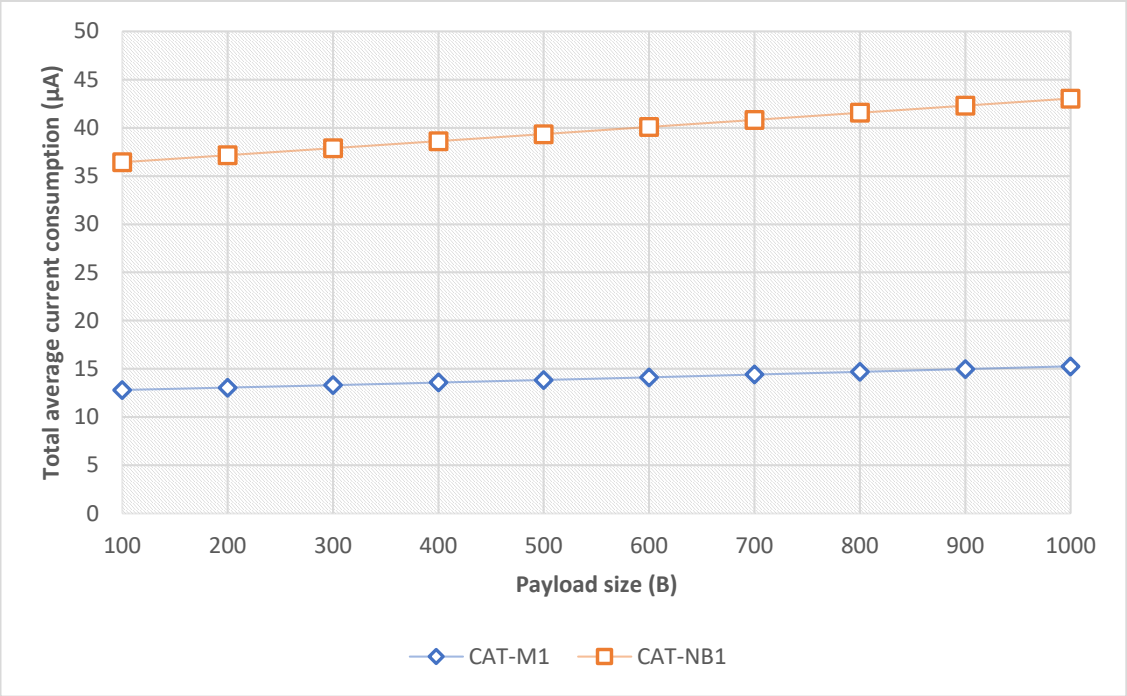

Figure 5.3: OPP - Test #3; PSM enabled - Variable Payload Sizes

With PSM enabled a substantial increase in battery life can be achieved by reducing the payload in terms of bytes. The average difference in battery life between increments of 100 bytes is about 400 hours, meaning that if the application payload is reduced with only 100 bytes, it amounts to about 17 days of extended battery life. The graph in figure 5.3 displays a clear trend in the UE's current consumption, starting at 12.81μA (975.8 days) and 36.43μA (343.1 days) for CAT-M1 and CAT-NB1 respectively when using a payload size of 100 bytes. When utilizing a 1kB payload size, the average current consumption amounts to 15.27μA (818.7 days) and 43.04μA (297.34 days) for CAT-M1 and CAT-NB1. By reducing the payload with 900 bytes, $\sim$ 520 days of prolonged battery life is achieved. This highly encourages compression and alternative serialization formats to reduce the overall payload size.

5.0.4 OPP - Test #4; eDRX enabled - Variable I-eDRX Intervals

Purpose of the test The purpose of this test is to look at the current consumption of the UE for different I-eDRX intervals when PSM is disabled.

Test Configurations

- Message protocol: NA
- Publication interval: 1 hour
- Payload size: 100 bytes
- PSM disabled
- eDRX interval: Varying
- C-DRX: 0.32 seconds CAT-M1, not supported in OPP for CAT-NB1
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: 0 seconds
- Voltage: 3.7V

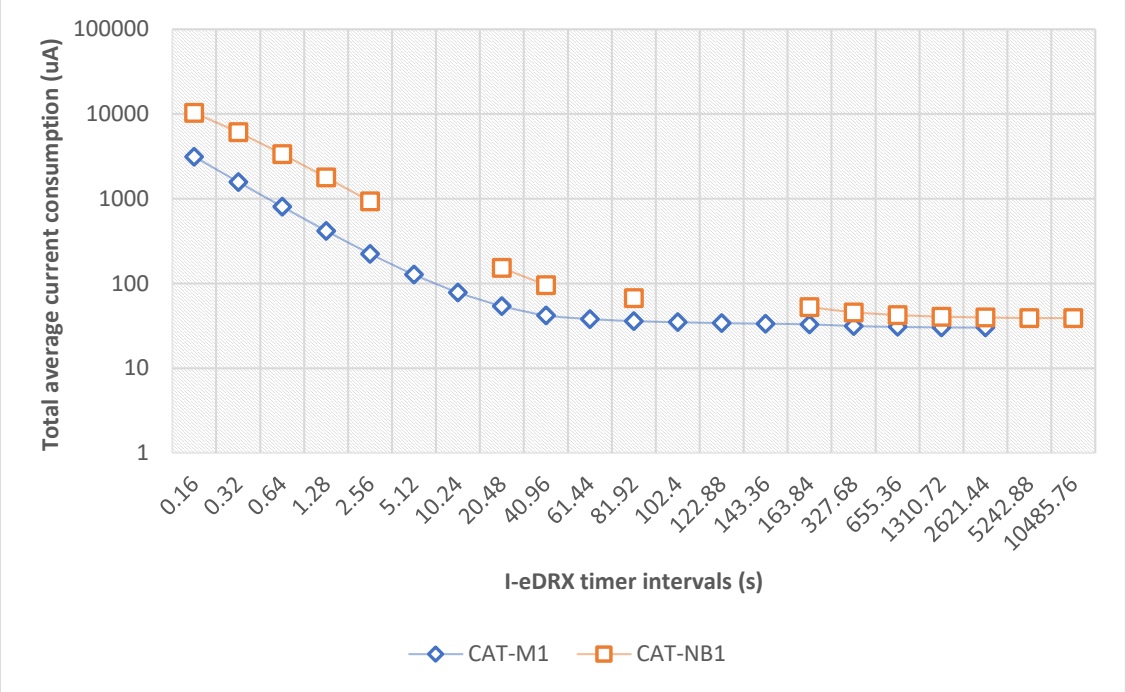

Figure 5.4: OPP - Test #4; eDRX enabled - Variable I-eDRX Intervals. Note that the y-axis scales logarithmically

The current consumption of the UE for different eDRX intervals starts at 3.14mA (CAT-M1) and 10.34mA (CAT-NB1) for eDRX intervals of 0.16 seconds and approaches the idle eDRX floor current of $\sim 25\mu$A. This number is heavily influenced by the SIM clock stop current because the SIM does not shut down during I-DRX sleep (frequently accessed). When comparing the current consumption of PSM and eDRX utilized independently, PSM has the lowest consumption of 12.81μA compared to 30.17μA for eDRX with the longest allowed value for CAT-M1 at 2621.44 seconds and 10485.6 for CAT-NB1. Long I-eDRX intervals affect the responsiveness of the device. In the case of an I-eDRX interval of 2621.44 seconds, one single I-DRX paging event occurs in-between data transmissions when transmitting with 1-hour intervals. The UE consumes less current taking advantage of PSM but will be less responsive than if standalone eDRX is used instead. The onboard modem of the nRF9160 supports different eDRX intervals

depending on the LPWAN. This is visible in the depicted graph where specific eDRX interval values are not present in the measurements. When comparing the battery life duration of standalone PSM versus standalone I-eDRX in CAT-M1, the PSM configuration yields a battery lifetime of 975.8 days while the I-eDRX configuration yields a battery lifetime duration of 414.3 days making the PSM configuration over twice as energy-efficient. For CAT-NB1, this is also the case. If the I-eDRX interval of the UE is increased to a maximum of 10485.76 seconds over a publication interval of 24 hours, the UE still consumes a higher sleep current in eDRX. This is also highly colored by the SIM clock stop current and the fact that every 10485.76 seconds, a DRX paging event occurs consuming additional current. Figure 5.5 graphically represents the same test but for I-eDRX values spanning from 40.96 to 10485.76.

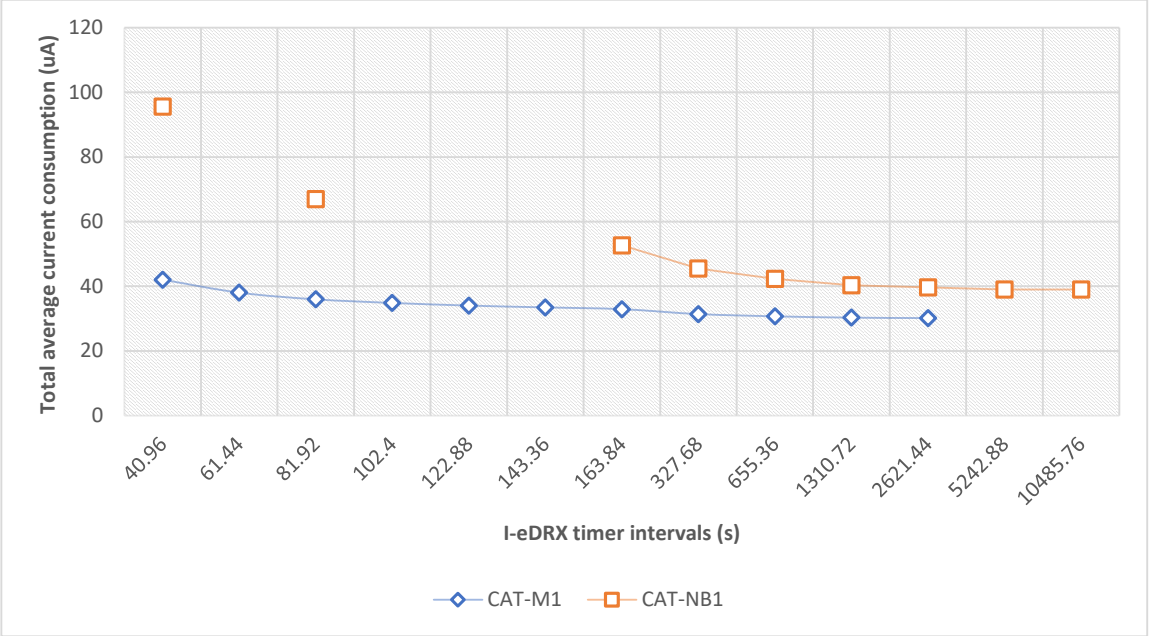

Figure 5.5: OPP - Test #4; eDRX enabled - Variable I-eDRX Intervals. Zoomed in from I-eDRX interval 40.96. Linear y-axis.

5.0.5 OPP - Test #5; PSM and eDRX enabled - Variable I-eDRX Intervals

Purpose of the test The purpose of this test is to look at the current consumption of the UE for different I-eDRX intervals when PSM is enabled.

Test Configurations

- Message protocol: NA
- Publication interval: 1 hour
- Payload size: 100 bytes
- PSM T3412: 1 hour
- PSM T3324: 10 and 30 seconds
- eDRX interval: Varying
- C-DRX: 0.32 seconds CAT-M1, not supported in OPP for CAT-NB1
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: 0 seconds
- Voltage: 3.7V

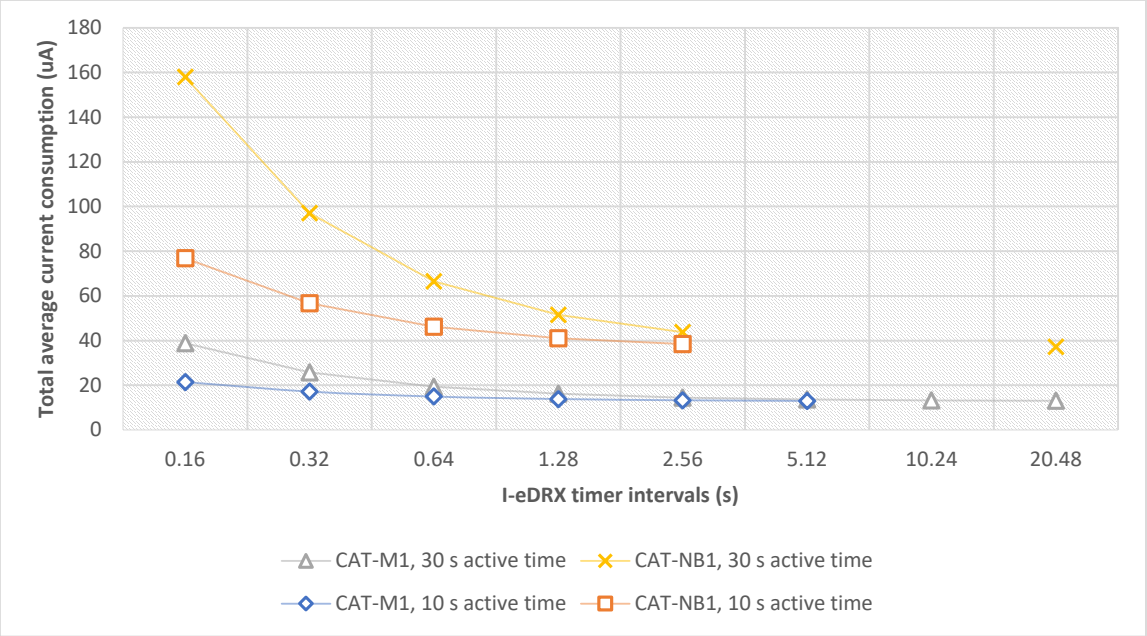

Figure 5.6: OPP - Test #5; PSM and eDRX enabled - Variable I-eDRX Intervals

Figure 5.6 illustrates the current consumption of the UE when both eDRX and PSM power-saving features are enabled using a PSM T3324 timer value of 10 and 30 seconds for both LTE modes. The graph states that CAT-NB1 benefits the most of increasing the duration of the I-eDRX timer interval for both PSM active timer durations because of its higher current consumption in general. When the I-eDRX timer interval is set to 20.48 seconds, the UE manages to perform one single DRX paging interval during the PSM active timer period before it enters PSM mode. For CAT-M1 the current consumption of the UE for an I-eDRX interval timer of 20.48 amounts to 13.06μA or $\sim$ 957.1 days which is an 18.7-day decrease compared to not utilizing I-eDRX at all essentially setting the PSM T3324 timer to 0 seconds making the UE go straight to PSM after RRC connected mode.

5.0.6 OPP - Test #6; PSM enabled - Variable C-DRX Intervals

Purpose of the test The purpose of this test is to look at the current consumption of the UE for different C-DRX intervals when PSM is enabled.

Test Configurations

- Message protocol: NA
- Publication interval: 1 hour
- Payload size: 564 bytes
- PSM T3412: 1 hour
- PSM T3324: 0 seconds
- eDRX disabled
- C-DRX: Variable
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: 60 seconds
- Voltage: 3.7V
- LPWAN: CAT-M1

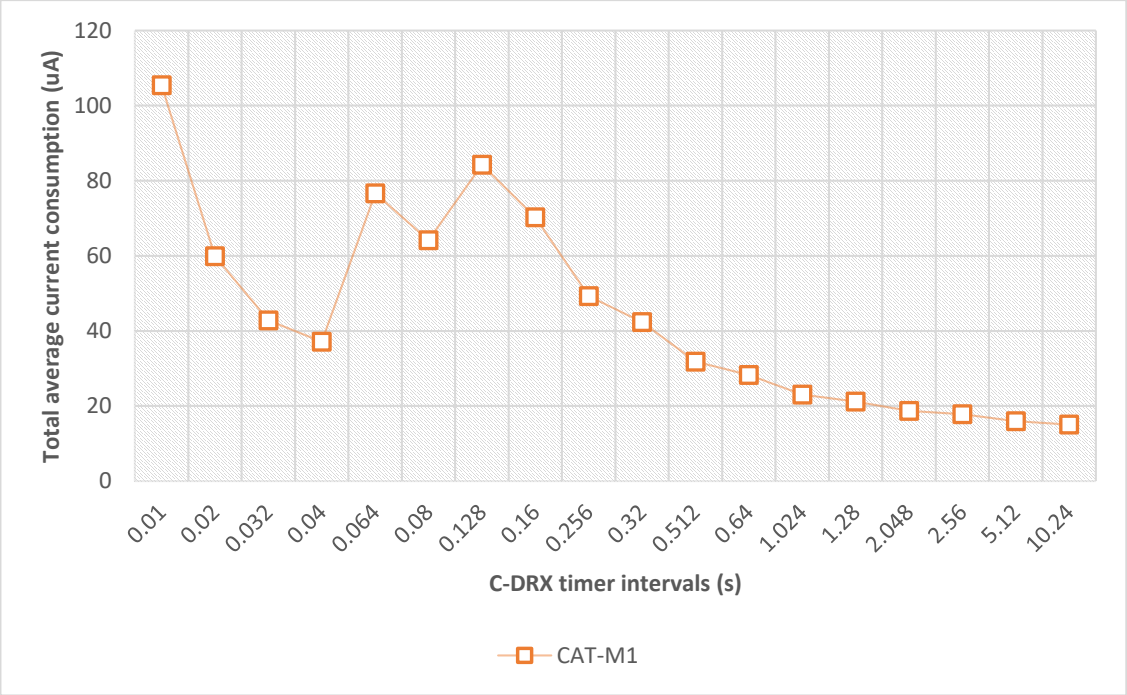

Figure 5.7: OPP - Test #6; PSM enabled - Variable C-DRX Intervals

See figure 5.7. With an inactivity time of 60 seconds, the UE consumes at average 105.05μA with a C-DRX interval of 0.01 seconds, yielding a battery lifetime estimation of 118.54 days. If the highest allowed value for C-DRX is utilized (10.24 seconds), the current consumption averages at 15.05μA, yielding an estimated battery life of 830.56 days, which is about eight times as effective. It should be noted that this is for quite a large inactivity time that greatly exceeds the highest allowed value for C-DRX. The lower inactivity time, the less C-DRX intervals contribute to the UE's current conservation unless PSM is disabled. When PSM is enabled, the UE enters PSM after RRC connected if PSM AT (T3324) is set to 0 seconds, and in PSM, the UE consumes less current than in between C-DRX and I-DRX paging events. The current measurements in this test gave very inconsistent values between 0.04 and 0.512 seconds of C-DRX. The expected trend is a graph that decreases exponentially. Perhaps these inconsistent values are due to corrupt sampled values in the OPP.

5.0.7 OPP - Test #7; LTE event charge - Variable Payload Sizes

Purpose of the test The purpose of this test is to look at the total charge per LTE publication event over various payload sizes.

Test Configurations

- Message protocol: NA
- Publication interval: NA
- Payload size: Various
- PSM: NA
- eDRX: NA
- C-DRX: 0.32 seconds CAT-M1, not supported in OPP for CAT-NB1
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: 0 seconds
- Voltage: 3.7V

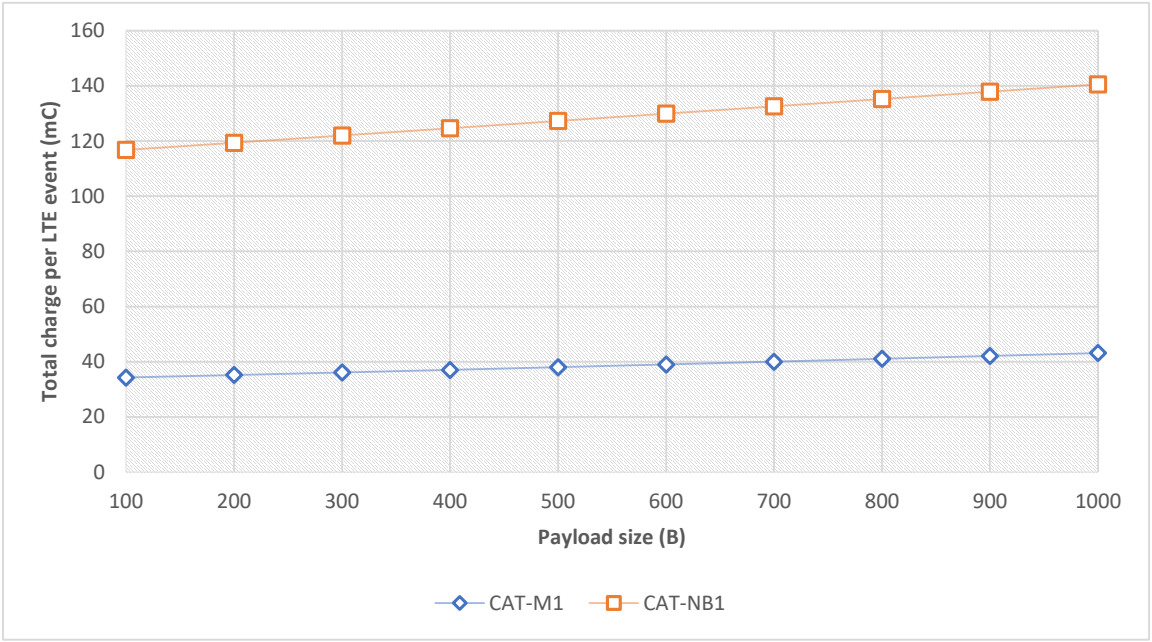

Figure 5.8: OPP - Test #7; LTE event charge - Variable Payload Sizes

The graph in figure 5.8 depicts a linear trend between payload size and charge per LTE event for both LPWAN technologies. Utilizing this linear trend to calculate for charge per payload byte results in LTE event costs of 7.5μC per payload byte + 33.17μC base charge for CAT-M1 and 26.5μC per payload byte + 114.01μC base charge for CAT-NB1. The base charge is the total charge cost of transmitting a LTE event without any payload.

5.0.8 DCPA - Test #1; Various Configurations

Purpose of the test The purpose of this test is to look at the real-world current consumption of the UE when publishing the reference application payload in a sequential manner over various stack configurations.

Test Configurations

- Message protocol: MQTT / CoAP
- Publication interval: 1 hour
- Payload size: 584/594 bytes
- PSM T3412 timer: 70 minutes
- PSM T3324 timer: 0 seconds
- eDRX disabled
- C-DRX: 0.32 seconds
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: NA
- Voltage: 3.7V

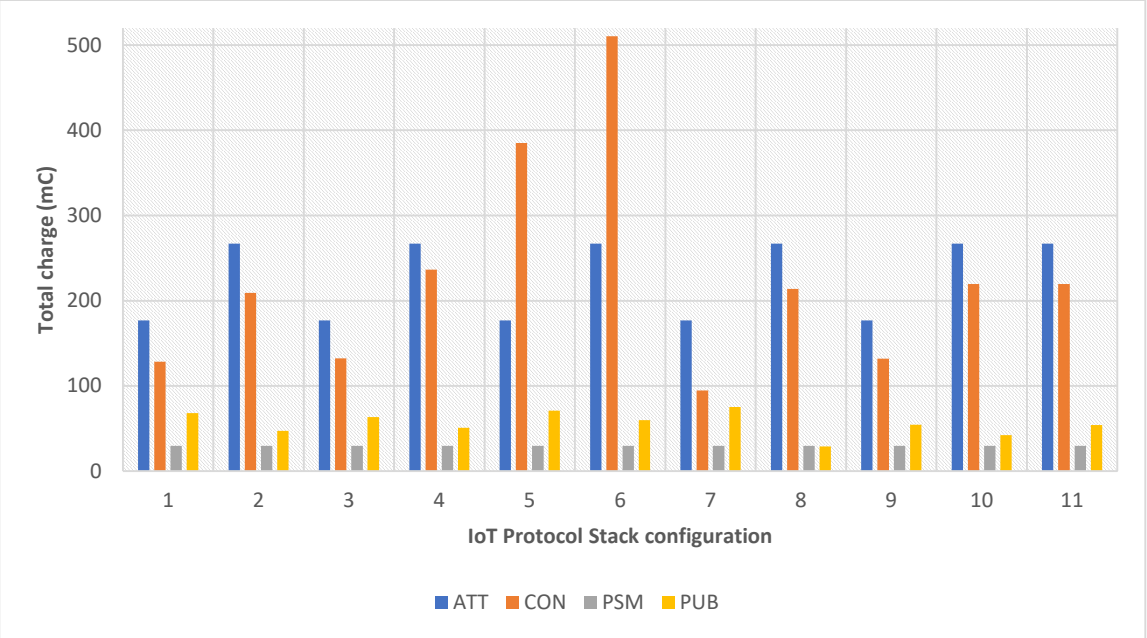

Figure 5.9: DCPA - Tests; Various Configurations

1. MQTT + TCP + CAT-M1
2. MQTT + TCP + CAT-NB1
3. MQTT + TCP/TLS(PSK) + CAT-M1
4. MQTT + TCP/TLS(PSK) + CAT-NB1
5. MQTT + TCP/TLS(EDCHA) + CAT-M1
6. MQTT + TCP/TLS(EDCHA) + CAT-NB1
7. CoAP + UDP + CAT-M1
8. CoAP + UDP + CAT-NB1
9. CoAP + UDP/DTLS(PSK) + CAT-M1
10. CoAP + UDP/DTLS(PSK) + CAT-NB1
11. CoAP + UDP/DTLS(PSK) + CAT-NB1, RAI

The graph in figure 5.9 illustrates current measurements across four types radio segments over eleven individual tests. The type of segment and its purpose are listed in section 4.2. The ATT segment is equal for each test and only measured once for each LPWAN. In the case of the ATT segment, CAT-NB1 consumes more current than CAT-M1 when attaching to the cellular network, by about 90mC. Note that this is the case of the modem having "cached" the cellular tower it connects to. This is probably the more common case for most IoT WSN nodes and can be compared to being in close proximity to a single cell tower over longer durations. This type of retention can drastically reduce the time the modem needs to connect to a cellular tower.

For CON segments, the measurements display the same trend where CAT-M1 is more energy-efficient than CAT-NB1. In general, the MQTT/TCP based configurations have higher current consumption than the CoAP/UDP based configurations. This is due to the fact that MQTT establishes a connection at the message-protocol level during this segment. This means that a CONNECT will be transmitted to the MQTT broker with a corresponding TCP ACK received, and a CONNACK will be receiving with a corresponding TCP ACK transmitted. In CoAP/UDP, only a single DNS request is requested during a CON segment. Comparing the non-secure tests to the PSK and EDCHA based cipher suite options, the results are also as expected for CON segments. The non-secure configurations do not have a TLS/DTLS handshake and thus consumes less current. The PSK based handshake consumes a little more than the non-secure option and less than the EDCHA based handshake. The EDCHA based handshake uses far more than the PSK cipher suite. This is mainly because of the exchange of a much larger certificate for the connection and the computing needed to encrypt/decrypt data.

In the DCPA measurements, the average PSM floor current and modem off[1] current was measured to $\sim 8.2\mu$A average current. This is 4.2μA more than specified in the specification for the nRF9160. This number is also constant through all the DCPA tests and a number identical for both LPWAN technologies. PSM for one hour amounts to $\sim$ 29.52mC.

For PUB segments, the total charge was, in general, higher for CAT-M1 than for CAT-NB1. This is directly contradictory to the OPP simulations, where the CAT-M1 results always yielded the lowest current consumptions. The PUB segments for non-secure, PSK, and EDCHA were about the same having a difference of the only mC per LTE event, translating to +-30 days in battery lifetime estimations, proving that a very secure connection does not have to be very expensive in terms of current consumption over time. The TLS/DTLS handshake will ideally only contribute a one time cost to the consumption. From that point, every publication packet has a limited amount of overhead associated with the particular encryption. The fact that every transmission has a very expensive baseline start-up cost and that each byte has less impact that the start-up cost supports that fact. The measurements were also taken in ideal conditions when the RRC inactivity time was close to zero. This can be noticed by the impactless contribution RAI has to current consumption, as seen in test number 10 and 11.

[1] Modem off current is the current drawn by the nRF9160 when the UE has disconnected from the cellular network and shut down.

5.0.9 DCPA - Test #2; RAI

Purpose of the test The purpose of this test is to look at the real-world current consumption of the UE when publishing the reference application payload in a sequential manner over various stack configurations with an extended RRC inactivity timer.

Test Configurations

- Message protocol: CoAP
- Publication interval: 1 hour
- Payload size: 584 bytes
- PSM T3412 timer: 70 minutes
- PSM T3324 timer: 0 seconds
- eDRX disabled
- C-DRX: 0.32
- TX output power: 23dBm
- Clock stop current: 30μA
- Inactivity timer: $\sim$ 12 seconds
- Voltage: 3.7V
- LPWAN: CAT-NB1

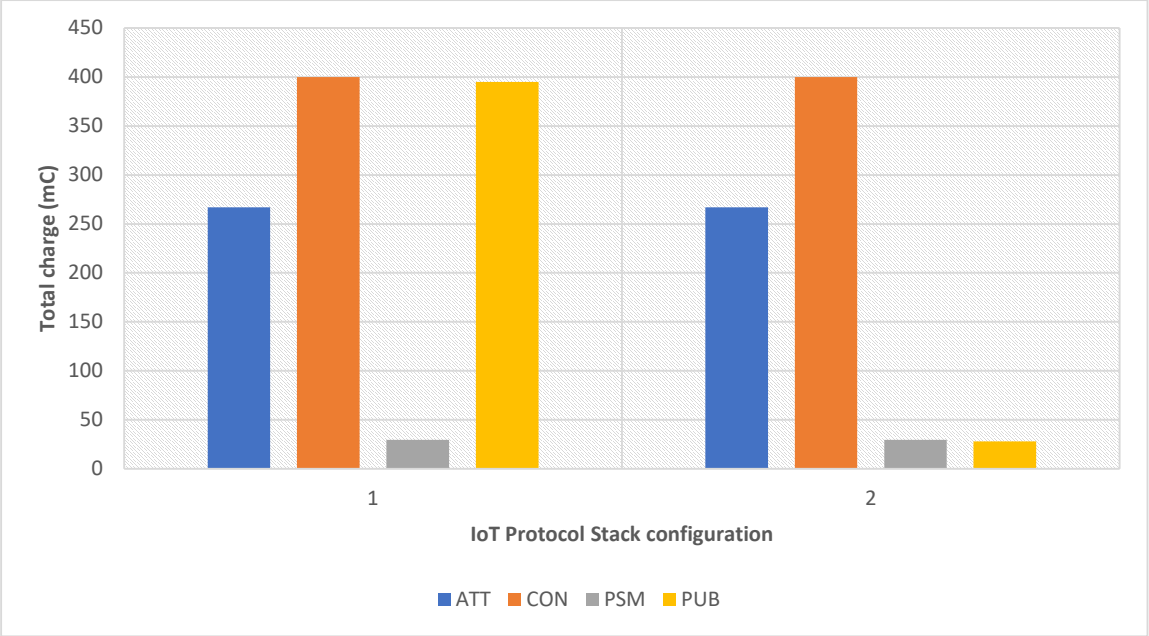

Figure 5.10: DCPA - Test #2; RAI with $\sim$ 12 seconds RRC inactivity timer

1. CoAP + UDP/DTLS + CAT-NB1
2. CoAP + UDP/DTLS + CAT-NB1, RAI

The graph in figure 5.10 displays the effects of RAI in the situation where an extended inactivity time has been given. RAI effectively shortens the time in RRC connected mode with about 10 seconds from 12 to 1.47 seconds, proving the effects of RAI. Note that RAI is only compliant with CAT-NB1 as per nrf9160 specification, and the benefits gained by enabling RAI is mostly limited to the given RRC inactivity timer. In the case of 12 seconds of inactivity time, the difference in charge per LTE event amounts to 395.03mC (106.26 days) without RAI and 28.15mC (780 days) with RAI enabled, which is about 13 times more current effective per LTE event.

Chapter 6

Discussion

6.1 Test Results

Compression and Serialization The results suggest that using compression algorithms and alternative serialization formats to reduce the size of the application payload probably is not beneficial as seen per device if the obtained size reduction is not in the kB domain *when power-saving features are **not** enabled*. This is because of the little impact small payload sizes have on current consumption when continually being connected to the network. On the other hand, considering data transfer costs charged by the MNO and potential cloud services, limiting the size of the payload can have a viable accumulative impact. In a more extensive system comprising hundreds, even thousands of devices, optimizing packet sizes and formats will have a sizeable accumulative impact, and therefore reducing the transferred packet sizes must be considered. On the other hand, conserving data can have a tremendous impact when power-saving features are enabled, as shown in OPP - Test #3 where 100 bytes amounts to over 400 hours additional battery life. In a more extensive system, this would amount to substantial accumulative savings across only limited of the numbers of system nodes.

DCPA vs OPP Because the OPP does not support receiving packages, the comparison between the OPP and DCPA is based on DCPA test #1 subtest 7 through 10 (CoAP/UDP configuration). By taking the utilized payload of 564 bytes in DCPA and multiplying it with the charge per event (ratio found in 5.0.7 + the base charge), the total charge per PUB event amount to 37.4mC for CAT-M1 and 129mC for CAT-NB1 based on the OPP numbers. Comparing this to the DCPA test measurements of 75.2mC for CAT-M1 and 28.76mC for CAT-NB1, it is clear that the OPP and DCPA in this particular test case directly contradict each other. In the DCPA test, it is CAT-NB1 that possesses the lowest current consumption, not CAT-M1, that is stated throughout the OPP testing.

It is suspected that the CAT-NB1 measurements for DCPA test # 1 subtest 8 did not yield the correct readings, which could be due to a number of reasons. Considering the DCPA #1 tests for CoAP/DTLS(PSK), a lot more sensible numbers are given, where the DTLS(PSK) configuration utilizes less current consumption for PUB segments than wits MQTT/TLS(PSK) counterpart. In general, the measurements from the OPP outperforms the test results from the physical testing. This is expected as the OPP measurements are sampled under highly ideal conditions unobtainable when measuring in live networks. The reason why CAT-NB1 performs better than CAT-M1 for PUB segments could perhaps be because UE got C-DRX during DCPA testing and that the OPP does not support C-DRX for CAT-NB. Another observation is that CAT-NB1 performs better

in the DCPA tests for segments involving the most amount of data in the transfer. Perhaps the bandwidth of the UE when using CAT-NB1 is sufficient enough to transfer small packet sizes at the same rate as CAT-M1, enabling the UE to go to sleep at a similar time for both cellular technologies. When transferring larger packet sizes, the UE has to stay connected for an extended amount of time when using CAT-NB1. On the contrary, CAT-M1's superior bandwidth enables the UE to transfer the large data chunk at a much faster rate and go to sleep at an earlier point in time, conserving energy.

MQTT vs CoAP There is a noticeable difference in current consumption between the MQTT and CoAP based configurations. On almost all accounts, the CoAP solution outperforms the MQTT solutions. This is due to the fact that MQTT has more overhead in its communication compared to CoaP. But, this does not mean that CoAP over UDP is the most reasonable choice for every application. In fact, for larger publication intervals, it might be more important to utilize a message protocol depending on acknowledgment, such as MQTT over TCP for the client-server communication. This is due to the fact that UDP packets are not guaranteed to arrive at the destination port because of network packet loss. On the other hand, it is possible to implement acknowledgment at the message-protocol level for CoAP, which resolves this issue, but this is highly dependent on the client and server implementation. Another reason why UDP based communications might be a better choice than TCP is the retransmission functionality present in the TCP specification. In particularly lossy networks, TCP retransmissions of packets can have a noticeable effect on the current consumption of constrained devices were in the worst case, and the UE will use a lot of current retransmitting TCP packets which effectively forces the UE to stay in RRC connected mode.

Another issue that is contributing to the overall current consumption in either message-protocol is the fact that the resource name and the topic name has to be present in every CoAP request and MQTT PUBLISH message. This is to properly address the information carried in the message. Over time, transmitting as little as 13 bytes (iot_publisher) would amount to hundreds of hours of wasted battery life depending on the size of the battery, refer to OPP test #3.

Chapter 7

Conclusion

This thesis presents current measurements of the nRF9160 configured with a common IoT use-case. With optimal power-saving features enabled, the nRF9160 is capable of consuming an average of 267mC for cellular network attachment, 213.9mC for server connection establishment, 29.52mC for PSM sleep, and 28.76mC of application payload transmission of 536-bytes every hour. This yields a theoretical battery life of 772.15 days for a reference battery capacity of 300mAh. These results are based on the nRF9160 getting PSM T3412 timer greater than the utilized publication interval of 1 hour, PSM T3324 timer of 0 seconds (disabling eDRX), C-DRX interval timer of 0.32 seconds and CAT-NB1 as LPWAN of choice. However, the overall current consumption and battery life can be greatly extended by setting a larger publication interval, extending the utilized battery capacity, and reducing the overall payload size.

This level of current consumption enables the nRF9160 to be deployed in remote areas over long durations of time. By utilizing a reasonably small battery capacity of 300mAh, the nRF9160 can, under ideal conditions, last years in between each battery charge while still maintaining a small form factor and publication frequency. This ideal current-consumption-to-publication-ratio widens the field of application for the nRF91 enabling it to be fitted on smaller and more demanding objects and subjects.

Chapter 8

Further Work

8.1 Use-Case Expansion

This project only documents the current consumption of a WSN node use-case where the UE publishes data sequentially to the remote server. There are countless IoT applications where the current consumption of the device needs to be mapped in order to propose new current efficient solutions. For instance, use-cases where the server requests information of the device, meaning that data publication is triggered by the server and not the client and use-cases that introduce GPS behavior.

8.2 Coverage of multiple TLS/DTLS Cipher Suites

There exists a vast amount of different TLS/DTLS cipher suites that can be utilized to ensure a secure connection to TCP/UDP-based server implementations. A lot of these cipher suites are based on different technologies and algorithms that each contribute differently to the application's security and the UE's current consumption. The overhead associated with each cipher suite should be mapped to find the sweet spot between security and UE current consumption.

8.3 Extended Coverage of LTE Parameters

This thesis did not cover LTE power-saving parameters such as eDRX Paging Time Window (PTW) and potentially other LTE parameters designed to reduce the current consumption of the UE. The comparison between CAT-M1 and CAT-NB1 also needs further testing to determine which of the two cellular technologies that truly the most energy preserving based on the fact that the DCPA and OPP test results gave contradicting answers.

8.4 Power Saving Library

A possible implementation based on the test results of this project is a power-saving library (PSL) for the nRF9160. The purpose of such a library would be to abstract away LTE parameter optimizations from the developer by providing functionality that automatically reconfigures the LTE modem in real-time. This would counteract the fact that power-saving features in CAT-M1 and CAT-NB1 networks may or may not be supported by the different network providers. The PSL would provide functionality for the nRF9160 that always makes sure the device uses the lowest amount of power

possible, given the circumstances. In other words, if given LTE parameters are not sufficient enough, the PSL would engage fallback functionality that actively requests less optimal LTE parameters and even shuts down the modem in between transmissions to preserve current, avoiding prolonged durations in RRC connected mode. A suggested API reference for a PSL is located in appendix A.4. Figure 8.1 illustrates a potential position of the PSL in the application firmware stack.

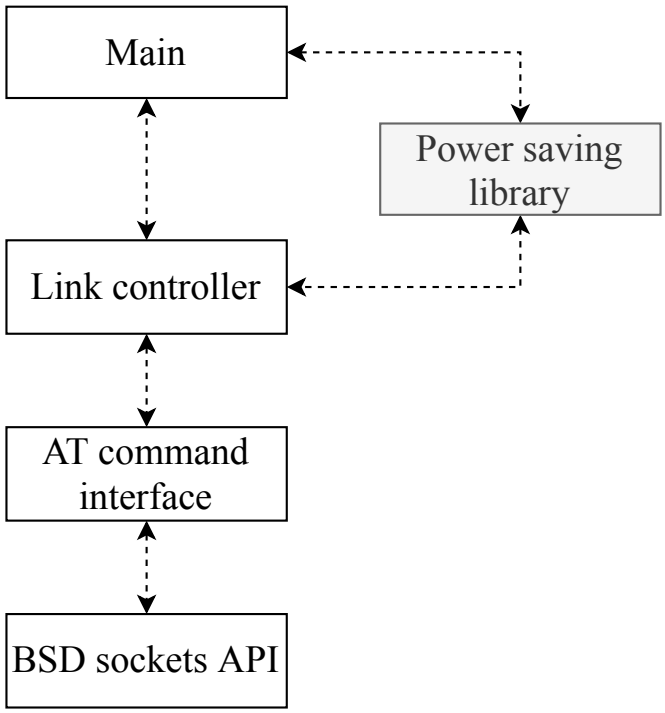

Figure 8.1: Application firmware PSL stack placement.

List of Figures

Bibliography

[1] Contabo. *Contabo*. Available at https://contabo.com/.

[2] Eclipse Foundation. *Californium GitHub Repository*. Available at https://github.com/eclipse/californium.

[3] Eclipse Foundation. *Eclipse Mosquitto*. Available at https://mosquitto.org/.

[4] GSMA. "LTE-M Deployement Guide to Basic Feature Set Requirements". In: (2019).

[5] GSMA. "NB-IoT Deployement Guide to Basic Feature Set Requirements". In: (2018).

[6] HiveMQ. *TLS/SSL - MQTT Security Fundamentals*. Available at https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/.

[7] IEEE. *NAT Behavioral Requirements for TCP*. Available at https://tools.ietf.org/html/rfc5382.

[8] IETF. *The Constrained Application Protocol*. Available at https://tools.ietf.org/html/rfc7252.

[9] Petter Myhre. *Field-testing how distance affects the behavior of LTE-M and NB-IoT*. Available at https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/field-testing-how-distance-affects-the-behavior-of-lte-m-and-nb-iot.

[10] Patrick Nohe. *Cipher Suites: Ciphers, Algorithms and Negotiating Security Settings*. Available at https://www.thesslstore.com/blog/cipher-suites-algorithms-security-settings/.

[11] none. *MCUboot*. Available at http://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/mcuboot/index.html.

[12] none. *MQTT documentation*. Available at http://mqtt.org/.

[13] OASIS. *MQTT Version 3.1.1*. Available at http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.

[14] The Zephyr Project. *The Zephyr Project Documentation*. Available at http://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/zephyr/index.html.

[15] Ammar Rayes and Samer Salam. "Internet of Things: From Hype to Reality". In: Revision 2 (1993).

[16] Simen S. Røstad. *IoT Publisher Application Firmware Repository*. Available at https://github.com/simensrostad/iot_publisher.

[17] Simen Sigurdsen Røstad. *IoT Test Servers*. Available at https://github.com/simensrostad/iot-test-servers.

[18] Zephyr RTOS. *BSD Sockets*. Available at https://docs.zephyrproject.org/latest/reference/networking/sockets.html.

[19] Rohde & Schwartz. “Power saving methods for LTE-M and NB-IoT devices”. In: (2019).

[20] Nordic Semiconductor. *BSD library*. Available at https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrfxlib/bsdlib/README.html.

[21] Nordic Semiconductor. *nrf*. Available at https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/introduction.html.

[22] Nordic Semiconductor. *nRF Connect for Desktop*. Available at https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-desktop.

[23] Nordic Semiconductor. *nRF Connect SDK Tutorial*. Available at https://devzone.nordicsemi.com/nordic/cellular-iot-guides/b/getting-started-cellular/posts/nrf-connect-sdk-tutorial.

[24] Nordic Semiconductor. *nRF9160 DK*. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nrf91_dk%2FUG%2Fnrf91_DK%2Fintro.html&cp=2_0_4.

[25] Nordic Semiconductor. *nRF9160 Product Specification v1.0*. https://infocenter.nordicsemi.com/pdf/nRF9160_PS_v1.0.pdf (2019/05/29).

[26] Nordic Semiconductor. *nrfxlib*. Available at http://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrfxlib/README.html.

[27] Nordic Semiconductor. *Online Power Profiler*. Available at https://devzone.nordicsemi.com/nordic/power/.

[28] Nordic Semiconductor. *Security Tags*. Available at https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrfxlib/bsdlib/doc/security_tags.html.

[29] Micrium Embedded Software. *IoT Protocol Stack Options*. Available at https://www.micrium.com/iot/internet-protocols/.

Appendix A

A.1 Example Application Payload

Code/format.JSON

```
{
  "dev": {
    "v": {
      "band": 3,
      "nw": "NB-IoT GPS",
      "iccid": "89450421180216254864",
      "modV": "mfw_nrf9160_1.1.0-40.rc",
      "brdV": "nrf9160_pca10090",
      "appV": "1.0.2"
    },
    "ts": 1581412370663
  },
  "roam": {
    "v": {
      "rsrp": 64,
      "area": 3305,
      "mccmnc": 24202,
      "cell": 34237203,
      "ip": "10.81.160.193"
    },
    "ts": 1581438337056
  },
  "bat": {
    "v": 4125,
    "ts": 1581438332150
  },
  "acc": {
    "v": [6.442969, 7.256921, -1.019891],
    "ts": 1574251825404
  },
  "gps": {
    "v": {
      "lng": 10.437156350160675,
      "lat": 63.421399614436105,
      "acc": 5.192617416381836,
      "alt": 152.19400024414062,
      "spd": 0.0854216143488884,
      "hdg": 0
    },
    "ts": 1581438332000
  }
}
```

A.2 MQTT Backend API Reference

```
/*
 * Copyright (c) 2020 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-BSD-5-Clause-Nordic
 */

/**@file
 *@brief MQTT Backend library header.
 */

#ifndef MQTT_BACKEND_H__
#define MQTT_BACKEND_H__

#include <stdio.h>
#include <net/mqtt.h>

/**
 * @defgroup mqtt_backend MQTT Backend library
 * @{
 * @brief Library to connect a device to a MQTT Backend message broker.
 */

#ifdef __cplusplus
extern "C" {
#endif

/** @brief MQTT Backend topics, used in messages to specify which
 *         topic that will be published to.
 */
enum mqtt_backend_topic_type {
  MQTT_BACKEND_TOPIC_MSG = 0x1
};

/** @brief MQTT Backend notification event types, used to signal the application. */
enum mqtt_backend_evt_type {
  /** Connected to MQTT broker. **/
  MQTT_BACKEND_EVT_CONNECTED = 0x1,
  /** MQTT broker ready. */
  MQTT_BACKEND_EVT_READY,
  /** Disconnected from MQTT broker. */
  MQTT_BACKEND_EVT_DISCONNECTED,
  /** Data received from MQTT broker. */
  MQTT_BACKEND_EVT_DATA_RECEIVED,
  /** FOTA update done, request to reboot. */
  MQTT_BACKEND_EVT_FOTA_DONE
};

/** @brief Struct with data received from MQTT broker. */
struct mqtt_backend_evt {
  /** Type of event. */
  enum mqtt_backend_evt_type type;
  /** Pointer to data received from the MQTT broker. */
  char *ptr;
  /** Length of data. */
  size_t len;
};

/** @brief MQTT Backend topic data. */
struct mqtt_backend_topic_data {
```

```
60   /** Type of topic that will be published to. */
61   enum mqtt_backend_topic_type type;
62   /** Pointer to string of application specific topic. */
63   char *str;
64   /** Length of application specific topic. */
65   size_t len;
66 };
67
68 /** @brief MQTT Backend transmission data. */
69 struct mqtt_backend_tx_data {
70   /** Topic that the message will be sent to. */
71   struct mqtt_backend_topic_data topic;
72   /** Pointer to message to be sent to the MQTT broker. */
73   char *str;
74   /** Length of message. */
75   size_t len;
76   /** Quality of Service of the message. */
77   enum mqtt_qos qos;
78 };
79
80 /** @brief MQTT Backend library asynchronous event handler.
81  *
82  *  @param[in] evt The event and any associated parameters.
83  */
84 typedef void (*mqtt_backend_evt_handler_t)(const struct mqtt_backend_evt *evt);
85
86 /** @brief Structure for MQTT Backend broker connection parameters. */
87 struct mqtt_backend_config {
88   /** Socket for MQTT broker connection */
89   int socket;
90   /** Client id for MQTT broker connection. */
91   char *client_id;
92   /** Length of client_id string. */
93   size_t client_id_len;
94 };
95
96 /** @brief Initialize the module.
97  *
98  *  @warning This API must be called exactly once, and it must return
99  *           successfully.
100 *
101 *  @param[in] config Pointer to struct containing connection parameters.
102 *  @param[in] event_handler Pointer to event handler to receive MQTT Backend
103 *             module events.
104 *
105 *  @return 0 If successful.
106 *           Otherwise, a (negative) error code is returned.
107 */
108 int mqtt_backend_init(const struct mqtt_backend_config *const config,
109           mqtt_backend_evt_handler_t event_handler);
110
111 /** @brief Connect to the MQTT broker.
112  *
113  *  @details This function exposes the MQTT socket to main so that it can be
114  *           polled on.
115  *
116  *  @param[out] config Pointer to struct containing the connection parameters,
117  *                     the MQTT connection socket number will be copied to the
118  *                     socket entry of the struct.
119  *
```

```
120  *  @return 0 If successful.
121  *            Otherwise, a (negative) error code is returned.
122  */
123 int mqtt_backend_connect(struct mqtt_backend_config *const config);
124
125 /** @brief Disconnect from the MQTT broker.
126  *
127  *  @return 0 If successful.
128  *            Otherwise, a (negative) error code is returned.
129  */
130 int mqtt_backend_disconnect(void);
131
132 /** @brief Send data to MQTT Backend broker.
133  *
134  *  @param[in] tx_data Pointer to a struct containing data to be transmitted to
135  *                     the MQTT broker.
136  *
137  *  @return 0 If successful.
138  *            Otherwise, a (negative) error code is returned.
139  */
140 int mqtt_backend_send(const struct mqtt_backend_tx_data *const tx_data);
141
142 /** @brief Get data from MQTT broker.
143  *
144  *  @return 0 If successful.
145  *            Otherwise, a (negative) error code is returned.
146  */
147 int mqtt_backend_input(void);
148
149 /** @brief Ping the MQTT broker. Must be called periodically
150  *         to keep connection to broker alive.
151  *
152  *  @return 0 If successful.
153  *            Otherwise, a (negative) error code is returned.
154  */
155 int mqtt_backend_ping(void);
156
157 #ifdef __cplusplus
158 }
159 #endif
160
161 /**
162  *@}
163  */
164
165 #endif /* AWS_IOT_H__ */
```

Code/mqtt_backend.h

A.3 CoAP Backend API Reference

```
/*
 * Copyright (c) 2020 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-BSD-5-Clause-Nordic
 */

/**@file
 *@brief CoAP library header.
 */

#ifndef COAP_BACKEND_H__
#define COAP_BACKEND_H__

#include <stdio.h>

/**
 * @defgroup CoAP library
 * @{
 * @brief Library to connect the device to a UDP server.
 */

#ifdef __cplusplus
extern "C" {
#endif

/** @brief CoAP notification event types, used to signal the application. */
enum coap_backend_evt_type {
  /** Connected to the CoAP server. */
  COAP_BACKEND_EVT_CONNECTED = 0x1,
  /** CoAP server ready.  */
  COAP_BACKEND_EVT_READY,
  /** Disconnected from the CoAP server. */
  COAP_BACKEND_EVT_DISCONNECTED,
  /** Data received from the CoAP server. */
  COAP_BACKEND_EVT_DATA_RECEIVED,
  /** CoAP Backend library error. */
  COAP_BACKEND_EVT_ERROR,
  /** CoAP Backend Fota done, request to reboot. */
  COAP_BACKEND_EVT_FOTA_DONE
};

/** @brief Struct with data received from UDP server. */
struct coap_backend_event {
  /** Type of event. */
  enum coap_backend_evt_type type;
  /** Pointer to data received from the UDP server. */
  char *ptr;
  /** Length of data. */
  size_t len;
};

/** @brief UDP backend transmission data. */
struct coap_backend_tx_data {
  /** Pointer to message to be sent to UDP server. */
  char *str;
  /** Length of message. */
  size_t len;
};

```

```
60 /** @brief CoAP library asynchronous event handler.
61  *
62  *  @param[in] evt The event and any associated parameters.
63  */
64 typedef void (*coap_backend_evt_handler_t)(const struct coap_backend_event *evt);
65
66 /** @brief Structure for UDP server connection parameters. */
67 struct coap_backend_config {
68   /** Socket for UDP server connection */
69   int socket;
70 };
71
72 /** @brief Initialize the module.
73  *
74  *  @warning This API must be called exactly once, and it must return
75  *           successfully.
76  *
77  *  @param[in] config Pointer to struct containing connection parameters.
78  *  @param[in] event_handler Pointer to event handler to receive CoAP module
79  *                           events.
80  *
81  *  @return 0 If successful.
82  *           Otherwise, a (negative) error code is returned.
83  */
84 int coap_backend_init(const struct coap_backend_config *const config,
85           coap_backend_evt_handler_t event_handler);
86
87 /** @brief Connect to the UDP server.
88  *
89  *  @details This function exposes the UDP socket to main so that it can be
90  *           polled on.
91  *
92  *  @param[out] config Pointer to struct containing connection parameters,
93  *                     the UDP connection socket number will be copied to the
94  *                     socket entry of the struct.
95  *
96  *  @return 0 If successful.
97  *           Otherwise, a (negative) error code is returned.
98  */
99 int coap_backend_connect(struct coap_backend_config *const config);
100
101 /** @brief Disconnect from the UDP server.
102  *
103  *  @return 0 If successful.
104  *           Otherwise, a (negative) error code is returned.
105  */
106 int coap_backend_disconnect(void);
107
108 /** @brief Send data to the UDP server.
109  *
110  *  @param[in] tx_data Pointer to struct containing data to be transmitted to
111  *                     the UDP server.
112  *
113  *  @return 0 If successful.
114  *           Otherwise, a (negative) error code is returned.
115  */
116 int coap_backend_send(const struct coap_backend_tx_data *const tx_data);
117
118 /** @brief Get data from the UDP server.
119  *
```

```
120  *  @return 0 If successful.
121  *            Otherwise, a (negative) error code is returned.
122  */
123 int coap_backend_input(void);
124
125 /** @brief Ping the UDP server. Must be called periodically
126  *         to keep socket open.
127  *
128  *  @return 0 If successful.
129  *            Otherwise, a (negative) error code is returned.
130  */
131 int coap_backend_ping(void);
132
133
134 #ifdef __cplusplus
135 }
136 #endif
137
138 /**
139  *@}
140  */
141
142 #endif /* COAP_BACKEND_H_ */
```

Code/coap_backend.h

A.4 PSL API Reference

```
/*
 * Copyright (c) 2020 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-BSD-5-Clause-Nordic
 */

#ifndef PSL_H__
#define PSL_H__

#include <zephyr/types.h>

/**
 * @defgroup psl Power Saving Library
 * @{
 * @brief Library that configures the LTE modem in real time based on
 *        application firmware configurations and requested/given LTE
 *        parameters.
 */

#ifdef __cplusplus
extern "C" {
#endif

enum psl_evt_type {
        /* Current consumption < Latency. */
  PSL_MODE_HIGHEST,
        /* Current consumption = Latency. */
        PSL_MODE_MEDIUM,
        /* Current consumption > Latency. */
        PSL_MODE_LOWEST,
        /* Parameters requested by the PSL. */
        PSL_LTE_PARAMETERS_REQUESTED,
        /* Optimal parameters given by the network. */
        PSL_LTE_OPTIMAL_PARAMETERS_GIVEN,
        /* Current battery percentage*/
        PSL_LTE_BATTERY_LIFETIME,
};

enum psl_rai_type {
        /* Control plane one response. */
        PSL_RAI_ONE_RESPONSE,
        /* Control plane no response. */
        PSL_RAI_NO_RESPONSE
};

struct psl_cfg_param {
        /* PSM TAU timer. */
        float psm_tau;
        /* PSM active timer. */
        float psm_at;
        /* eDRX interval timer. */
        float edrx_int;
        /* eDRX Paging Time Window. */
        float edrx_ptw;
        /* Release Assistance Information. */
        struct psl_rai_type rai;
};

struct psl_cfg {
```

```
60          /* UE publication interval. */
61          int pub_int;
62          /* Size of sequential payloads. */
63          int payload_size;
64          /* Parameters given by the network. */
65          struct psl_cfg_param param;
66  };
67
68  struct battery_level {
69          /* Battery percentage. */
70          int battery_lvl;
71          /* Battery lifetime estimation days. */
72          int battery_lifetime_days;
73  };
74
75  struct psl_evt {
76          /* Power Saving Library event type. */
77    enum psl_evt_type type;
78          /* Additional information included in the PSL event. */
79    union {
80
81      int bat_lvl;
82                  /* Parameters given by the network. */
83                  struct psl_cfg_param param;
84    };
85  };
86
87  /* Asynchronous event handler notifying the calling module of PSL events. */
88  typedef void(*psl_evt_handler_t)(const struct psl_cfg *const evt);
89
90  /** @brief Initialize the power saving library.
91   *
92   *  @return 0 If the operation was successful.
93   *            Otherwise, a (negative) error code is returned.
94   */
95  int psl_init(psl_evt_handler_t handler);
96
97  /** @brief Update the power saving library with the latest LTE parameters and
98   *         application firmware configurations.
99   *
100  *  @param[in] cfg Pointer to a configuration struct.
101  *
102  *  @return 0 If the operation was successful.
103  *            Otherwise, a (negative) error code is returned.
104  */
105 int psl_param_update(struct psl_cfg *cfg);
106
107 #ifdef __cplusplus
108 }
109 #endif
110
111 #endif /* PSL_H__ */
```

Code/psl.h

A.5 DCPA Current Measurements

No inactivity time										
		ATT		CON		PSM		PUB		Battery lifetime
MQTT + TCP	LTE-M	176.8	mC	128.4	mC	29.52	mC	68.2	mC	460.73
	NB-IoT	267	mC	209.12	mC	29.52	mC	47.4	mC	585.15
MQTT + TCP/TLS(PSK)	LTE-M	176.8	mC	132.3	mC	29.52	mC	63.3	mC	485.03
	NB-IoT	267	mC	236.6	mC	29.52	mC	50.8	mC	560.32
MQTT + TCP/TLS/EDCHA*	LTE-M	176.8	mC	385	mC	29.52	mC	71.1	mC	447.34
	NB-IoT	267	mC	510.5	mC	29.52	mC	59.9	mC	503.27
CoAP + UDP	LTE-M	176.8	mC	94.8	mC	29.52	mC	75.2	mC	429.95
	NB-IoT	267	mC	213.9	mC	29.52	mC	28.76	mC	772.15
CoAP + UDP/DTLS(PSK)	LTE-M	176.8	mC	132.13	mC	29.52	mC	54.5	mC	535.83
	NB-IoT	267	mC	219.7	mC	29.52	mC	42.3	mC	626.62
CoAP + UDP/DTLS(PSK), RAI	NB-IoT	267	mC	219.7	mC	29.52	mC	54	mC	538.95
Extended Inactivity time ∼12 sec										
CoAP + UDP	NB-IoT	267	mC	400	mC	29.52	mC	395.03	mC	106.26
CoAP + UDP, RAI	NB-IoT	267	mC	400	mC	29.52	mC	28.15	mC	780.14

Table A.1: DCPA setup test results

A.6 OPP Current Simulations

TAC: Total Average Current

OPP TEST #1

Payload (KB)	TAC CAT-M1		TAC CAT-NB1	
0	1.7	mA	35.3	mA
10	1.73	mA	42.87	mA
20	1.77	mA	43.54	mA
30	1.8	mA	43.79	mA
40	1.83	mA	43.92	mA
50	1.86	mA	44	mA
60	1.9	mA	44.05	mA
70	1.93	mA	44.09	mA
80	1.96	mA	44.12	mA
90	1.99	mA	44.14	mA
100	2.02	mA	44.16	mA

OPP TEST #2

PSM TAU inter	TAC CAT-M1		TAC CAT-NB1	
1	12.81	uA	36.43	uA
2	8.4	uA	20.22	uA
3	6.94	uA	14.81	uA
4	6.2	uA	12.11	uA
5	5.76	uA	10.49	uA
6	5.47	uA	9.41	uA
7	5.26	uA	8.63	uA
8	5.1	uA	8.05	uA
9	4.98	uA	7.6	uA
10	4.88	uA	7.24	uA
11	4.8	uA	6.95	uA
12	4.73	uA	6.7	uA
13	4.68	uA	6.49	uA
14	4.63	uA	6.32	uA
15	4.59	uA	6.16	uA
16	4.55	uA	6.03	uA
17	4.52	uA	5.91	uA
18	4.49	uA	5.8	uA
19	4.46	uA	5.71	uA
20	4.44	uA	5.62	uA
21	4.42	uA	5.54	uA
22	4.4	uA	5.47	uA
23	4.38	uA	5.41	uA
24	4.37	uA	5.35	uA

OPP TEST #3

payload size	TAC CAT-M1		TAC CAT-NB1	
100	12.81	uA	36.43	uA
200	13.06	uA	37.16	uA
300	13.31	uA	37.89	uA
400	13.58	uA	38.62	uA
500	13.85	uA	39.35	uA
600	14.12	uA	40.09	uA
700	14.41	uA	40.82	uA
800	14.69	uA	41.56	uA
900	14.98	uA	42.3	uA
1000	15.27	uA	43.04	uA

OPP TEST #4

eDRX interval	TAC CAT-M1		TAC CAT-NB1	
0.16	3140	uA	10340	uA
0.32	1580	uA	6090	uA
0.64	806.97	uA	3350	uA
1.28	418.5	uA	1780	uA
2.56	224.26	uA	930.93	uA
5.12	127.15	uA		uA
10.24	78.51	uA		uA
20.48	54.2	uA	152.62	uA
40.96	42.05	uA	95.58	uA
61.44	38.04	uA		uA
81.92	35.97	uA	67	uA
102.4	34.86	uA		uA
122.88	34.04	uA		uA
143.36	33.48	uA		uA
163.84	32.93	uA	52.69	uA
327.68	31.41	uA	45.53	uA
655.36	30.72	uA	42.27	uA
1310.72	30.31	uA	40.32	uA
2621.44	30.17	uA	39.66	uA
5242.88		uA	39.01	uA
10485.76		uA	39.01	uA

OPP TEST #5

eDRX interval	TAC CAT-M1		TAC CAT-NB1	
0.16	21.43	uA	76.82	uA
0.32	17.15	uA	56.67	uA
0.64	14.93	uA	46.25	uA
1.28	13.82	uA	41.04	uA
2.56	13.27	uA	38.43	uA
5.12	12.97	uA		uA

OPP TEST #5

eDRX interval	TAC CAT-M1		TAC CAT-NB1	
0.16	38.81	uA	157.97	uA
0.32	25.82	uA	97.06	uA
0.64	19.33	uA	66.53	uA
1.28	16.15	uA	51.57	uA
2.56	14.48	uA	43.75	uA
5.12	13.64	uA		uA
10.24	13.2	uA		uA
20.48	13.06	uA	37.2	uA

OPP TEST #6

C-DRX interval	TAC CAT-M1	
0.01	105.45	uA
0.02	59.9	uA
0.032	42.82	uA
0.04	37.13	uA
0.064	76.63	uA
0.08	64.17	uA
0.128	84.28	uA
0.16	70.29	uA
0.256	49.26	uA
0.32	42.35	uA
0.512	31.81	uA
0.64	28.22	uA
1.024	23	uA
1.28	21.21	uA
2.048	18.68	uA
2.56	17.78	uA
5.12	15.98	uA
10.24	15.05	uA

OPP TEST #7

payload size	CAT-M1 (C-DRX 0.32)		CAT-NB1
100	34.28		116.77
200	35.18		119.38
300	36.1		122.01
400	37.05		124.64
500	38.03		127.28
600	39.02		129.93
700	40.03		132.58
800	41.06		135.23
900	42.1		137.89
1000	43.15		140.55