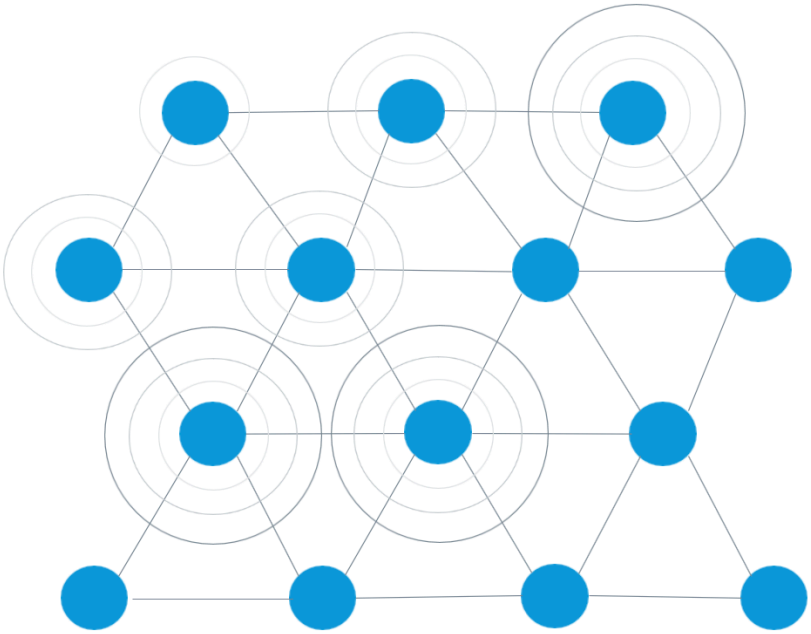
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Stine Åkredalen

Automatic dynamic tuning of Bluetooth mesh networks for optimal performance

Master's thesis in Cybernetics and Robotics
Supervisor: Geir Mathisen
Co-supervisor: Omkar Kulkarni
July 2022



NTNU
Norwegian University of Science and Technology

NORDIC SEMICONDUCTOR

Stine Åkredalen

Automatic dynamic tuning of Bluetooth mesh networks for optimal performance

Master's thesis in Cybernetics and Robotics
Supervisor: Geir Mathisen
Co-supervisor: Omkar Kulkarni
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MASTER THESIS DESCRIPTION

Candidate:	**Stine Åkredalen**
Course:	**TTK4900 Engineering Cybernetics**
Thesis title (Norwegian)	**Automatisert dynamisk regulering av Bluetooth mesh nettverksparametere for optimal ytelse**
Thesis title (English):	**Automatic dynamic tuning of Bluetooth mesh networks for optimal performance**

Thesis description: The settings of vital protocol parameters are expected to have influence on the performance of a Bluetooth mesh network. Further, it is assumed that an optimal tuning of the protocol parameters will depend on the given mesh technology. Thus, optimal tuning of the protocol parameters also must take into account the specific mesh topology.

Build on the specialization project "Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks", this master thesis shall investigate a method for dynamic tuning of Bluetooth mesh networks for optimal performance. As Bluetooth nodes often are battery powered, energy consumption should be part of the term performance, together with connectivity properties.

The tasks will be:

1. Conduct a literary study of protocol parameters' influence on the performance of Bluetooth mesh networks and procedures for adjustments of the parameters.

2. Propose an algorithm for automatic dynamic tuning of Bluetooth mesh networks for optimal performance. Also, suggest a system for verifying the proposed algorithm.

3. As far as time permits, implement the suggested algorithm from previous point.

Start date: 1st of March 2022
Due date: 27th of July 2022

Thesis performed at: Department of Engineering Cybernetics
Supervisor: Professor Geir Mathisen, Dept. of Eng. Cybernetics
Co-supervisor: Omkar Kulkarni, Senior R&D Engineer at Nordic Semiconductor

Abstract

This report introduces the basic concepts of the Bluetooth mesh network technology and discusses its measured performance in terms of latency and reliability through published papers and, through independent research, proposes a novel algorithm to dynamically and automatically regulate some of the protocol-specific parameters available for tuning in Bluetooth mesh in real-time. Latency and reliability tests have been conducted on a large-scale Bluetooth mesh network located at Nordic Semiconductor office space in Trondheim, Norway. The goal was to test the Bluetooth mesh network performance and compare the results when using the proposed algorithm with previously found optimal static parameter values for the same network. The algorithm regulates the protocol-specific parameters, Network Transmit Count and Relay Re-transmit Count, as well as the non-protocol-specific parameter device radio transmit power. The results reveal that the energy consumption of the devices used in the test-bed could be reduced by 93% using the proposed algorithm compared with using the default transmit power settings for the Nordic Semiconductor nRF52840 chip. The network reliability, reaching only 99,73%, was slightly lower compared with the results using the optimally tuned static parameters suggested in the report Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks [2] which gave perfect reliability at 100%. Network latency results using the algorithm came out to be almost twice the recorded values of its static competitor. This indicates that the algorithm, although performing well considering power usage and reliability, gives a higher first-transmit message loss count. This effect is most likely due to the reduced device radio transmit power. The algorithm achieves a trade-off between network traffic, power usage, reliability, and latency. Hence, tuning of the algorithm parameters (constants) could be improved to balance the performance better or be adjusted to fit specific application priorities. In other applications where some latency is accepted but reliability and battery life are more important, such as in data gathering- or sensor networks, the algorithm may be a helpful support tool.

Sammendrag

Denne rapporten introduserer de grunnleggende konseptene for Bluetooth mesh nettverksteknologien og diskuterer dens målte ytelse når det gjelder latens og pålitelighet gjennom publiserte artikler og, gjennom uavhengig forskning, foreslår en ny algoritme for dynamisk og automatisk å regulere noen av de protokollspesifikke parametrene som er tilgjengelige for innstilling for Bluetooth mesh i sanntid. latens- og pålitelighetstesting har blitt utført på et fullskala Bluetooth mesh-nettverk lokalisert ved Nordic Semiconductors kontorlokaler i Trondheim, Norge. Målet var å teste Bluetooth mesh nettverksytelse og sammenligne resultatene ved bruk av den foreslåtte algoritmen med tidligere funnet optimale statiske parameterverdier for samme nettverk. Algoritmen regulerer de protokollspesifikke parameterne, Network Transmit Count og Relay Re-transmit Count, så vel som den ikke-protokollspesifikke parameterenhetens radiosendeeffekt. Resultatene viser at energiforbruket til enhetene som brukes i testoppsettet kan reduseres med 93% ved å bruke den foreslåtte algoritmen sammenlignet med å bruke standard innstillingene for radiosendeeffekt for nRF52840-brikken til Nordic Semiconductor. Nettverkspåliteligheten, som nådde 99,73%, var litt lavere sammenlignet med resultatene ved å bruke de optimalt innstilte statiske parameterne foreslått i rapporten Optimalisert innstilling av Bluetooth-mesh-parametere for trådløse lysstyringsnettverk [2] som ga perfekt pålitelighet med 100% meldingssuksess. latensresultatene ved bruk av algoritmen viste seg å være nær det dobbelte av de registrerte verdiene til dens statiske konkurrent. Dette indikerer at algoritmen, selv om den fungerer bra med tanke på strømforbruk og pålitelighet, gir et høyere antall meldingstap ved første sendingsforsøk. Denne effekten skyldes mest sannsynlig enhetens reduserte radiosendeeffekt. Algoritmen oppnår en trade-off mellom nettverkstrafikk, strømforbruk, pålitelighet og latens. Derfor vil tuning av algoritmeparametrene (konstanter) kunne forbedres for å balansere ytelsen bedre eller justeres for å tilpasse spesifikke applikasjonsprioriteter. I andre applikasjoner der noe ventetid er akseptert, men pålitelighet og batterilevetid er viktigere, slik som for eksempel i datainnsamlings- eller sensornettverk, kan algoritmen være et nyttig støtteverktøy.

Preface

This thesis is based on the further work as suggested in the specialization project *Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks* for TTK4550. Nordic Semiconductor has provided all hardware and software tools used for testing and development. The performance test, Ethernet communication tools and Bluetooth mesh test-bed utilized in this report have been created by the Bluetooth mesh team at Nordic Semiconductor of which I have been a part of as a student intern over the past two years. The rest of this thesis is my own independent work.

Acknowledgements

I would like to take this opportunity to thank professor Geir Mathisen for his valuable feedback and continuous counseling during the work process of writing this report. Thank you to my colleagues at Nordic Semiconductor: Hasan Qaq for his help with creating the colorful Python plotting scripts, insightful discussions and for all the carefree coffee breaks, and Anders Storrø who made the large-scale network testing possible by installing the test-bed and for always providing technical support in times of need. Lastly, I would like to thank my supervisor at Nordic Omkar Kulkarni for sharing his knowledge through the many engaging discussions we have had during this semester and for those to come.

As an honorable mention, I want to acknowledge my colleague and then mesh team member Erik Vincent Robstad who, as my previous lab partner in TTK4147, was courageous enough to put his own reputation on the line when introducing me to Nordic Semiconductor.

Table of Contents

List of Figures

List of Tables

1 Introduction

Bluetooth (BT) was initially released in year 2000, intended as a simple cable replacement technology [24]. Since its release, this wireless technology has taken several leaps to enhance its performance to keep up with the needs of popular smart, wireless and low-power devices. In 2010, Bluetooth Low Energy (BLE) was made available, making the BT technology applicable with wireless smart-, wearable- and home devices which typically are, due to their small size, very power restricted. The latest addition, released in 2017, is the Bluetooth mesh (BTM) standard, enabling many-to-many communication over BLE radio. The mesh technology opens up a new world of home- and commercial smart wireless applications for the Bluetooth standard. For example, BTM networking has made it possible to control building services such as lighting, HVAC (Heating, Ventilation and Air Conditioning) systems and predictive maintenance, and to wirelessly connect with every control point and automate their behavior. This smart building technology has made living and work environments more comfortable, efficient, safe and all at a lower cost [26].

A BTM network can be configured in various ways to fit the application use-case based on the expected throughput, network size and physical structure, ambient noise and power considerations as evaluated by the network administrator. However, maintaining a well-fit parameter configuration is a tedious task considering that the environment and topology of the mesh often are dynamically changing, for example, wirelessly tracked boxes being moved around a busy warehouse. Varying traffic and external interference must also be expected and may change the need for configuration for optimal performance. Automatic maintenance of mesh networks has therefore become a "hot" topic among mesh researchers and developers.

Depending on the System on a Chip (SoC) manufacturer, the parameter settings of a BTM device may have different presets, or default values, with no guarantee that these configurations are going to be optimal for the application that the developer is working on. Therefore, this report will investigate how some of these parameters, along with the device radio transmit power, can be automatically tuned in real-time to optimize network reliability, while also reducing the overall power consumption, by implementing a distributed algorithm.

This report will introduce the BTM basic terms, concepts and specification, present an overview of published material on the subject of mesh optimization, both on the tuning of parameters and automated mesh extension mechanisms, and then explore the possibilities of designing an automatic dynamic de-centralized algorithm for BTM to optimize its performance. The proposed algorithm will be tested, in isolation and in combination with a performance test. The performance test will be run on a full-scale BTM network consisting of 100 nodes in an active office environment. The algorithm's effect will be evaluated by comparing the results from the same performance test with a set of default values. The algorithm design is presented in such a way that the code could be recreated for further work.

2 Bluetooth mesh specification and basic concepts

Bluetooth mesh (BTM), which may also be referred to as "mesh" in this report, is a networking technology built on top of the existing Bluetooth Low Energy BLE radio, which was designed for very low power operations [22]. BTM extends the capabilities of BLE by providing concurrent many-to-many network communication. This enables the Bluetooth technology to meet the enormous expectations of modern *smart* applications and the IoT (Internet of Things) where "everything talks to everything". BTM also has the advantage of being a complete, full-stack solution. This makes development with mesh a smooth affair as everything from the very low-level physical layer to the high-level model/application layer is defined.

The BTM networking solution is thoroughly described in its specification document [29] as published by the Bluetooth Special Interest Group (SIG). The SIG is an organization consisting of member companies who participate in the development of the mesh standard through research and committee work, allowing BTM to continuously evolve [25]. This chapter aims to give an introductory description of the BTM networking protocol, some key concepts and terms needed to be able to reflect on the performance measures and challenges that will be discussed later in this report.

2.1 Stack architecture

The BTM architectural stack is divided into 8 layers according to their respective responsibilities, see Figure 1. The bottom layer is the BLE stack which consists of multiple layers of its own. The mesh stack is entirely dependent on the availability of the BLE stack [24]. The *bearer layer* defines how mesh packets, or Protocol Data Unit (PDU), will be handled by the mesh communication system [24]. Two types of bearers are used in BTM: the Advertising bearer (ADV bearer) which uses the BLE radio to scan for nearby devices and to advertise (transmit) mesh packets onto the network, and the GATT bearer which allows devices without native mesh ADV bearer to join the network and communicate with the other *nodes* (mesh device members of a network), using a protocol known as the *Proxy Protocol*. A network node may or may not have their *proxy feature* enabled, but those who do can convert and relay messages between the two types of bearers.

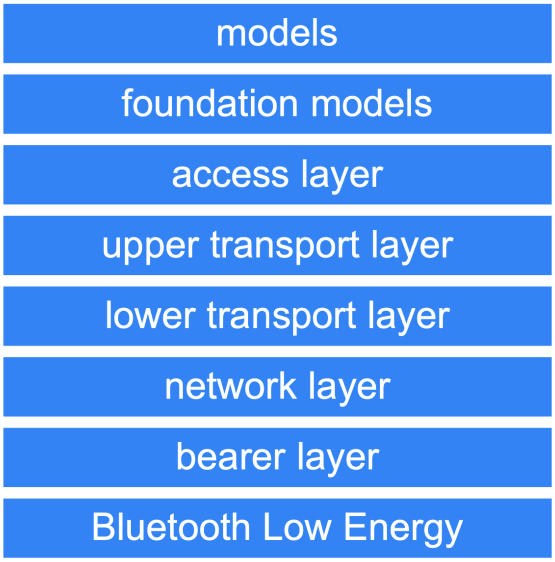


Figure 1: The Bluetooth mesh stack architecture. The very bottom layer is the BLE stack, upon which the mesh system is dependent.

Source: Bluetooth.com

The *network layer* keeps track of message addressing and prepares the PDU to be transported by

the bearer layer or formatted to be interpreted by the transport layer. The (upper- and lower) *transport layers* handles message segmentation and reassembly of packets (PDUs) when needed and security operations such as encryption, decryption- and authentication processes. The *access layer* is responsible to format the packet data in such a way that is useful to the above *model layer* which defines the models. A *model* defines a set of states, message types and other associated behavior for the mesh element [29]. The model layer can sometimes be referred to as the *application layer* as the *model handlers* define the application-specific logic. The *foundation model layer* defines the mandatory mesh models which handle basic configurations and network management.

2.2 Node roles, interactions and features

Member nodes of a BTM network may inhabit various roles, each with different properties and tasks. Figure 2 shows how the different types of nodes contribute to the network communication flow. The **proxy node** enables devices that do not support the mesh ADV bearer[1] to join the network by re-transmitting the signals from the GATT bearer to the ADV bearer. In Figure 2 one can see how the smartphone with a standard BT interface is connecting to the mesh network through the proxy node, labeled "P".

A **relay node** has one of the most essential tasks in the mesh network - to relay, or re-transmit received messages to all other nodes in its range. The relay node uses the ADV bearer and its radio is always on. The relay node reads the message's unique sequence number and may discard the message is recognized by the cache[2], meaning it has previously been relayed from this node. Each time a message is relayed, its Time-To-Live (TTL) value is deducted by 1. When the message TTL reach 1 in value, it can no longer be relayed. This is to limit the message propagation, saving network traffic.

A **friend node** is, like the relay node, always active as it stores messages addressed to neighboring **low power nodes** which are only active during set duty cycles to limit their power consumption.

2.3 Messaging and packet structure

BTM is a messaging-based protocol. Unlike other networking protocols, BTM uses a *managed flooding* networking strategy, meaning that messages are not routed[3] along any specific path [27]. Instead, when a message is being transmitted from its origin node, all nodes within radio range will receive the message. The nodes which have the *relay* feature enabled will then re-transmit the message to all other nodes in range, and so on. This way, the transmitted message will have multiple paths from source to destination, adding an element of reliability as there is no single point of failure [28].

BTM use a publish/subscribe communication scheme. A mesh *client* may multi-cast messages to a group address, where only the nodes subscribing to this group address will accept the message. The message will then be processed further up through the higher stack levels until it reaches the model/application layer. Here the message payload can be interpreted and used for application purposes. Group addresses in the range 0xFF00 through 0xFFFF are reserved for fixed group addresses, such as 0xFFFF which target all nodes (broadcast) [29]. The client may also unicast to a specific node's address. During the provisioning of a mesh device, the *provisioner* is responsible for allocating an unique unicast address to each member of the network.

The BTM packet format, see Table 1, leaves 12-16 bytes for the application payload depending on the network MIC. 1 byte is usually reserved for the opcode specifying the type of model message [29]. The remaining 11-15 bytes are available for other parameters, such as a value measured by a sensor or an on/off lighting command. For payloads that exceed the payload limit of 12-16 bytes,

[1] A bearer is a service that enables data transmission between different network interfaces.

[2] Cache is a special type of memory which holds copies of frequently used data, making it more accessible to the CPU when needed

[3] Network routing is the process of selecting a path across one or more networks.

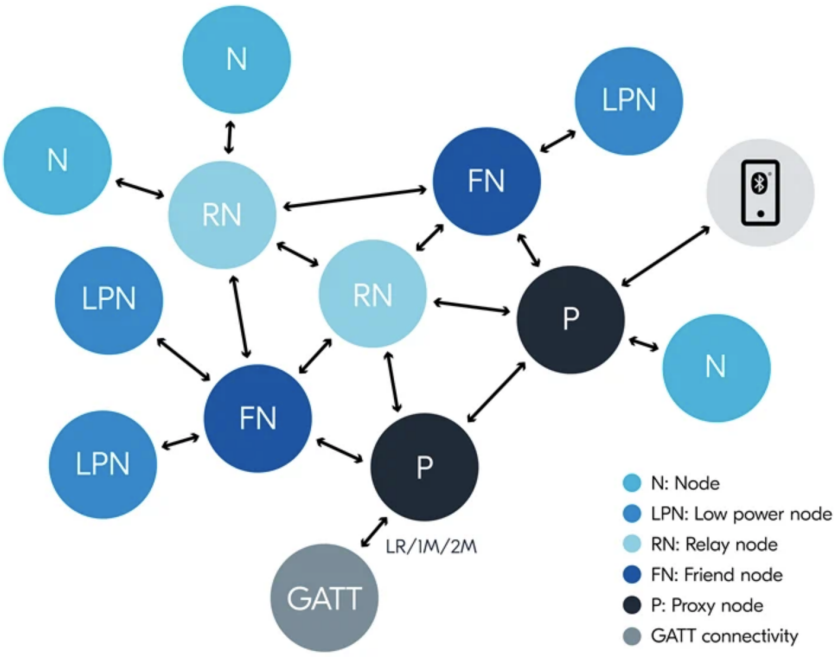


Figure 2: The Bluetooth mesh nodes can inhabit various roles. Relay and Friend nodes are responsible for message delivery to low power and other nodes. Smartphones or other devices using a standard (non-mesh) Bluetooth interface must connect through a proxy node to be able to join the network.

Source: Nordic Semiconductor

there is a process of Segmentation And Re-assembly (SAR) which requires more processing time. The maximum number of segments for a transmitted or received message is 32.

Bluetooth Mesh Packet Format								
Bytes	1		1		3	2	2	12 or 16	4 or 8
Parameters	IVI	NID	CTR	TTL	Seq number	Src addr	Dst addr	Payload	NWK MIC

Table 1: The Bluetooth mesh packet format leaves 12 or 16 bytes of payload, of which one byte is usually reserved for the opcode, depending on the network and application MIC size.

The BTM packet header contains parameters for security and encryption (IVI, Message Integrity Check (MIC), and Network Identification), message behavior and identification (CTL, TTL, and sequence number) and addressing (source and destination addresses).

2.3.1 Network protocol parameters

When a message is being transmitted to the network, the origin node may choose to repeat the number of times the message is transmitted on the network layer. This parameter is called the **network transmit state** which controls the Network Transmit Count (NTC) of additional transmits and the interval between each transmit, the Network Transmit Interval (NTI). The NTI has a minimum of 10ms + 0-10ms random delay, which gives an average of 15 ms per hop[4]. In BTM, a *hop* is referring to how many times a message has been relayed by a relay node. Each transmit sends a copy of the same message. This means that the same sequence number is being used each time. When a relay node receives any amount of copies of this message, only the first one to arrive

[4]A hop is the traversing from one node to another

will be relayed and the others discarded due to the cache remembering the sequence number of the first message.

The relay node has a similar parameter, namely the **relay re-transmit state**, defined by the Relay Re-transmit Count (RRC) and Relay Re-transmit Interval (RRI). The RRI has the same minimum and average time as the NTI. The model layer also has the option to set a re-transmit count for additional message copies being sent. This parameter is called Publish Re-transmit Count (PRC), sometimes also called *model retries*. The PRC is a very powerful transmit parameter as a message will be sent with a unique sequence number for each transmit, making each of them applicable for both the NTC and the RRC configured for the nodes in the network. For example, with a PRC of 1 and NTC of 1, the same message will be transmitted from the origin node a total of 4 times.

2.4 Quality of Service

When talking about performance, or Quality of Service (QoS)[5], of BTM in this report, the main focus will be latency- and reliability performance and device power consumption. **Latency** is usually measured by the Round-Trip Latency (RTL) of a message, meaning the time it takes for a message to be transmitted from the source to its target destination and back again to the source. Single-Trip Latency (STL) may also be used as a latency measure. STL is the time it takes for a message to travel from its source to its target destination. **Reliability** is a measure of consistency of a procedure or method. When testing the performance of BTM in this report, the reliability is defined as the percentage of times a message can be delivered to a target node under given conditions.

Minimizing device power consumption extends battery life and, as a result, makes for better products. *Transmit Power (TXP)* and the total amount of time that the radio is active (scanning cycles and *windows*) are the main factors affecting the power consumption in a BLE device [14]. TXP is the amount of power input into the radio signal and is proportional to the signal's effective range. The higher the TXP, the farther the signal can travel, and the more obstructions it can effectively penetrate. This makes the TXP parameter relevant, not only when considering device power consumption, but for reliability and latency as well.

[5]Quality of service (QoS) is the description or measurement of the overall performance of a service, such as a telephony or a computer network

3 Previous work

What does optimizing a mesh network mean? A quick Google search reveals an overwhelming interest in the optimization of IoT- (Internet of Things) and smart applications, including their popular choice of communication: mesh networks. The Bluetooth mesh (BTM) technology is no exception. There are multiple reports, methods, and algorithms readily available on the subject, varying from optimal power control to automatic network joining and routing mechanisms. A network may also benefit from optimized messaging protocol settings, device operation settings like radio transmit power and scanning cycles, and optimized topological configuration such as relay node selection and density. This section will attempt to present an overview of the previous works on these subjects and, based on this material, justify the thesis' focus of research.

3.1 Optimized tuning of Bluetooth mesh parameters for lighting control networks control

The report *Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks* [2] was written by the report author as a pre-study for this very thesis. The report focus lies on the specific case of BT mesh lighting control networks and how these can be tuned for optimal performance in terms of the network parameters such as network transmit count (NTC), publish re-transmit count (PRC) and relay re-transmit count (RRC). Lighting control networks typically have strict requirements on reliability and latency performance, making this the focus of the Quality of service (QoS). The goal was to find the most optimal parameter configuration for lighting control applications using BTM.

Figure 3: Map for lighting scenario 3: One-to-Many - Multiple rooms with limited relay configuration. Nodes within the scenario area (marked in orange) are targets ("field nodes") for this scenario. The relay nodes used for the limited relay configuration are colored pink. For the all-relay configurations, all nodes included in the scenario are configured as relays. The tester node is marked in blue.

Source: *S. Åkredalen [2]*

Performance tests were run with various network parameters available for tuning on a large-scale network as shown in Figure 3. The goal was to provide insight into how the performance of the mesh network changes when varying the network parameters. The report concludes that the choice of network parameters is not arbitrary, it states that: "Optimal network parameter configuration depends on the use-case and priorities, network topology, traffic (throughput), external disturbances and network size." (Åkredalen, 2022, p. 1).

Parameters → / Configuration ↓	**NTC**	**RRC**	**Relays**
1st	0	0	All relays
2nd	2	0	All relays
3rd	2	2	All relays
4th	0	0	Limited
5th	2	0	Limited
6th	2	2	Limited

Table 2: Test parameter configurations for the "Multiple room" (78-node network) lighting scenario.

Source: *S. Åkredalen [2]*

Single-Trip time (avg.) → / Configuration ↓	**Mean [ms]**	**Median [ms]**	**Max [ms]**	**Msg. success rate**
1st	18.71	16.65	53.43	100%
2nd	18.34	16.24	53.53	100%
3rd	19.77	17.13	54.06	100 %
4th	17.81	15.77	52.67	99.93%
5th	20.01	17.62	58.87	100 %
6th	20.36	17.24	61.55	99.97 %
6th with PRC = 2	22.58	19.67	62.56	100%

Table 3: Average test results for all nodes included in the Multiple room lighting scenario. The configurations are described in Table 2.

Source: *S. Åkredalen [2]*

Table 3 reveal the final results from the performance test run on the large-scale network. With a dense relay deployment ("All relays" in Table 3) a reliability of 100% was achieved for all configurations, although increasing the RRC gave an increase in latency. This indicates that packets are getting lost due to the increased traffic related to the RRC parameter and that some messages are received at their destination on a later transmit attempt. With the sparse relay deployment ("Limited" in Table 3) a slight decrease in reliability as well as increased latency was recorded with the higher RRC count configuration. The report concludes with a suggested baseline configuration as shown in Table 4.

Parameter	**Value**	**Comment**
NTC	2	Could be replaced with a high relay node density
RRC	0	Increase if traffic or disturbance is high
PRC	0	Should be used in combination with RRC
TTL	-	Depend on network size

Table 4: Suggested parameters for BT mesh lighting control networks before tuning

Source: *S. Åkredalen [2]*

However, since the operating conditions and environment of a network might change during its lifetime, the report suggests that the network performance could potentially benefit from having the parameters continuously updated during run-time.

3.2 SiLabs: Bluetooth mesh network performance report

Silicon Labs (SiLabs), an American technology company that designs and manufactures semiconductors, silicon devices, and software [15], SiLabs has also published a report on the mat-

ter: "Bluetooth mesh network performance" where they have tested for latency, throughput, and scalability. The report concludes that unsegmented messages are of great importance to reduce the overall latency, and that relay selection becomes critical as network size increases, considering both latency and reliability. The overall reliability with the different network sizes, varying from a small 24 node network to a full scale 240 node network, all came out to be above 99%, see Figure 4. The parameters used during testing is given in Table 5. Variation in network transmit counts (NTC, RRC, PRC) was not tested for.

Parameter	**Value**
TTL	7
NTC : NTI	3 : 10 ms
RRC : RRI	3 : 10 ms
PRC	0

Table 5: Static network parameter configuration as used during testing in SiLabs report *Bluetooth mesh network performance* [12]

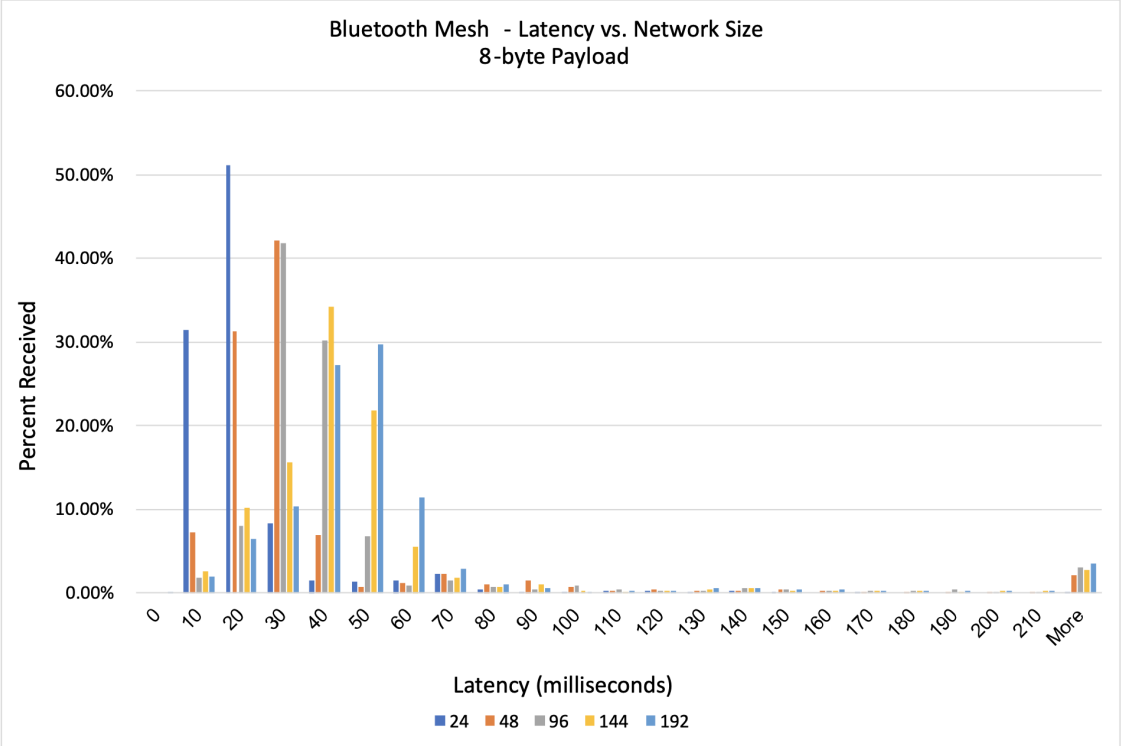


Figure 4: SiLabs latency and reliability results for an 8-byte payload for multiple network sizes, each presented with its own color.

Source: SiLabs [12]

SiLabs have also published their own parameter tuning guide for optimizing BT mesh networks [13], in which their only comment on how to set the network repetition parameters says to only use network/relay re-transmits in areas with very few relays. They do not mention any recommended values.

3.3 Ericsson: Large scale Bluetooth mesh Testing

Swedish company Ericsson, a founding member of the Bluetooth SIG and leading provider of Information and Communication Technology (ICT) worldwide [8], conducted a series of performance

tests on a BTM network in 2017 [9]. The paper provides insight into BTM reliability performance with different relay node configurations. Their test site consisted of 879 devices, including everything from window sensors, occupancy sensors, HVAC[6] sensors, actuators, and lighting, distributed over approximately 2,000 sqm. Note: the tests have not been run on an actual physical network of mesh nodes. Instead, Ericsson have created a full stack implementation of the Bluetooth Mesh Profile running in a system-level simulator.

The first test was conducted using only the mandatory (baseline) network configurations message cache and TTL (Time to Live). The results can be seen in Figure 5. With the sparse relay node deployment, evenly distributed and with a density of 1.4%, the packet delivery success within their set 300ms limit came out to be 99.1%. The report defines 300 ms as the human limit of perception and is therefore used as a requirement for the network as it includes lighting [9]. The traffic was simulated to be about 150bps. With a more dense relay deployment of 5.6%, the reliability dropped to 97.5% and got even worse with higher traffic use-cases where it dropped to as low as 69.2%.

	Baseline			Enhanced		
	Low traffic	Medium traffic	High traffic	Low traffic	Medium traffic	High traffic
Sparse deployment	99.1%	95.4%	84.3%	>99.9%	>99.9%	>99.9%
Dense deployment	97.5%	88.7%	69.2%	>99.9%	>99.9%	>99.1%

Figure 5: Ericsson BT mesh reliability performance results table

Source: Ericsson.com

Next, an enhanced network configuration was tested. This setup included first hop message repetitions, also known as NTC, and advertising randomization. The results show enhanced reliability compared with the previous baseline configuration. The sparse relay node deployment resulted in a 99.9% reliability for all traffic cases. The dense relay node deployment did, as with the baseline configuration, lower the reliability but only with the highest traffic case. Thus, the enhanced configuration proved to be better suited to handle the cases with increased traffic.

[6]HVAC: heating, ventilation, air-conditioning

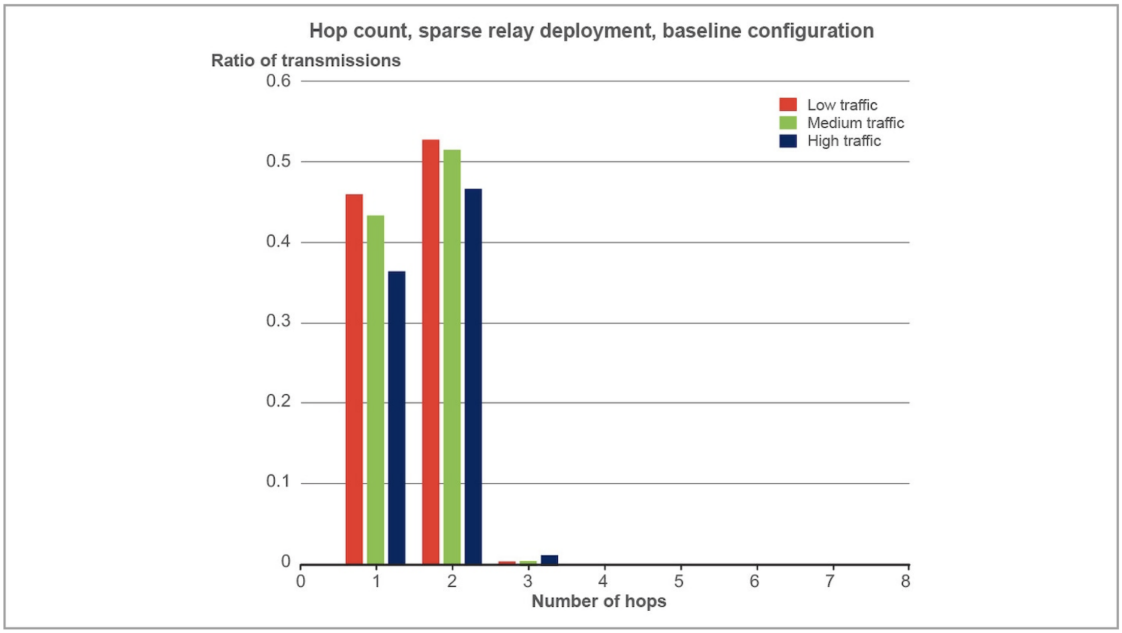


Figure 6: Ericsson BT mesh reliability performance results with the sparse relay node deployment.

Source: Ericsson.com

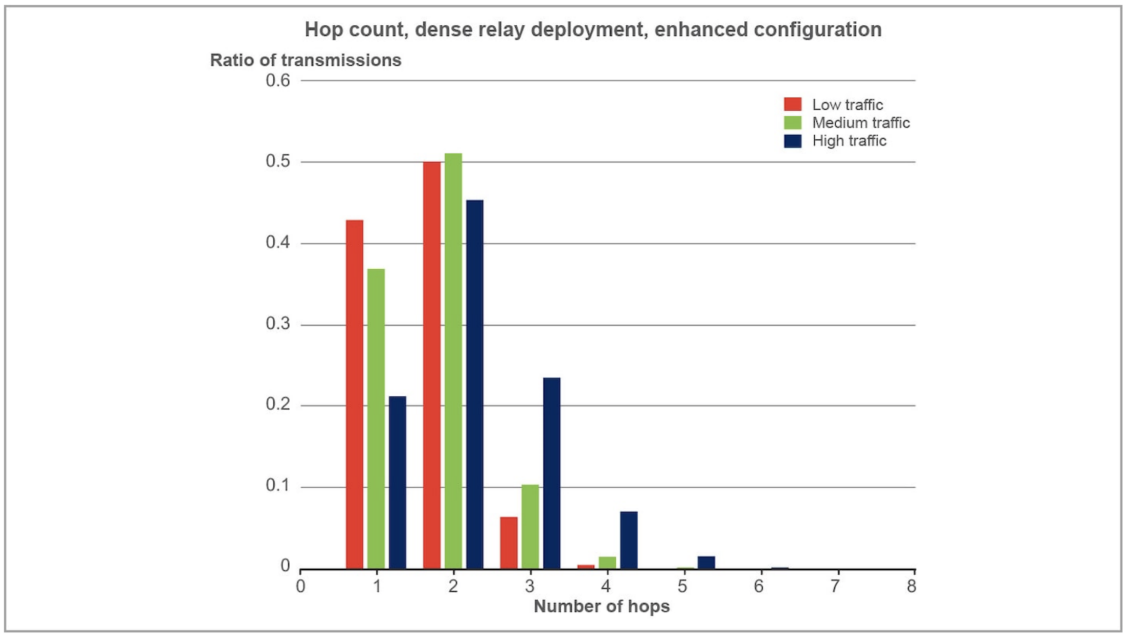


Figure 7: Ericsson BT mesh reliability performance results with the dense relay node deployment.

Source: Ericsson.com

The paper also compares the number of message hops and the density of relay nodes. The results can be seen in Figure 6 - 7. Although the number of re-transmissions, for the most part, stayed the same for the dense and sparse relay deployment. A higher hop count was recorded for the dense deployment. Latency was not presented as part of the results for this report.

The report concludes that the best performance among the studied cases is obtained when deploying a relay density corresponding to roughly 1.5%, which corresponds to six relays for every 1,000sqm with this test setup.

3.4 Automatic Bluetooth mesh network joining algorithms

Various experimental approaches are currently being developed to provide BLE (Bluetooth Low Energy) mesh networks the ability to provision[7] new nodes to a lower cost (in terms of power consumption) as well as to eliminate the need for a smartphone to perform the procedure. Let's take a look at a few examples.

3.4.1 FruityMesh

FruityMesh, a BLE master-slave connection-oriented protocol able to build and manage all connections without any necessary user interaction [7]. The algorithm uses a clustering technique where the location of each node, and the size of the cluster it is part of, are used as a criterion when connecting to other nodes in the network. The algorithm also includes an automatic discovery and handshaking[8] procedures, as well as a "self-healing" functionality for when connections break or new nodes are added. Figure 8 presents a simplified flowchart of how the FruityMesh network is created. A connecting node can take on the role of a Master (or Parent) or as a Slave (or Child) depending on cluster size. Compatible devices can simply be flashed with the FruityMesh source code and will then connect in an instant, ready for mesh communication. According to the documentation [7], power consumption is minimal with the correct settings.

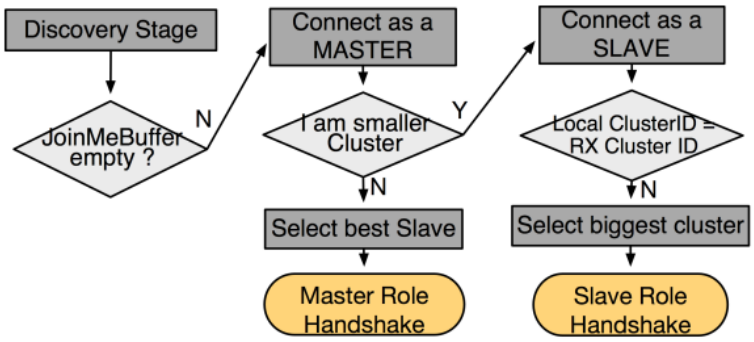


Figure 8: Simplified flowchart of the FruityMesh network build up

Source: Nieto-Taladriz, Murillo, Pollin [19] - University of Leuven, Belgium

In the report *Towards Efficient BLE Mesh: Design of an Autonomous Network Joining Algorithm* by Nieto-Taladriz, Murillo and Pollin (2019) [19], FruityMesh is used as a baseline for their work on refining the Master/Parent selection procedure by scoring the various node candidates. The results reflect an improved latency, reliability, and scaling of the mesh network.

3.4.2 NeoMesh

NeoMesh is an autonomous and self-containing wireless mesh-network technology developed by the Danish tech-company NeoCortec [18]. This ad hoc[9] mesh network technology replaces the need of the traditional central Network Manager as all network nodes link to each other automatically, forming a dynamic network that works even if nodes change position or are replaced. NeoMesh is not made for BT mesh specifically but has re-purposed the old "legacy"[10] mesh networks –

[7] Provisioning is the process of adding a new, un-provisioned device to a Bluetooth mesh network, such as a light bulb. - Bluetooth.com

[8] Handshaking is the automated process for negotiation of setting up a communication channel between entities. - Techopedia.com

[9] An ad hoc network is one that is spontaneously formed when devices connect and communicate with each other.

[10] Legacy technologies are systems, technologies, software, or hardware that is outdated or obsolete. - BeInformed.com

like Zigbee, Thread, WiFi, and Bluetooth. According to NeoCortec, NeoMesh is providing a second-generation mesh technology by adapting the basic mesh principles, making these previously obsolete wireless mesh technologies ready to use in the world of wireless, low-power networks for IoT applications, see Figure 9.

Figure 9: Automated agriculture is a possible use-case for NeoMesh, allowing for a low cost, low energy, scalable and dynamic solution for farmers.

Source: NeoCortec

3.5 Minimizing BLE device power consumption

Another way to improve BT mesh network performance is to optimize the battery lifetime of the devices by minimizing their power consumption. The subject of low-power is crucial in developing functional and sustainable IoT applications [10]. Both FruityMesh and NeoMesh address the importance of power consumption by integrating low power by design in their algorithms [7] [18].

With the BLE protocol, the two main factors affecting power consumption in the device are the amount of power transmitted and the time that the radio is active (during scanning and advertising intervals) [14]. In the report *BLE Parameter Optimization for IoT Applications* published on HAL open science [16], an algorithm based on the configuration of the BLE protocol parameters *scanning interval*, *scanning window* and *advertising interval* is presented to minimize power consumption. The report results indicate a significant gain in the lifetime of the devices compared to the SIG profile recommended configuration, showing that BLE, and therefore also BT mesh, is suitable for a wide range of IoT applications when tuned correctly.

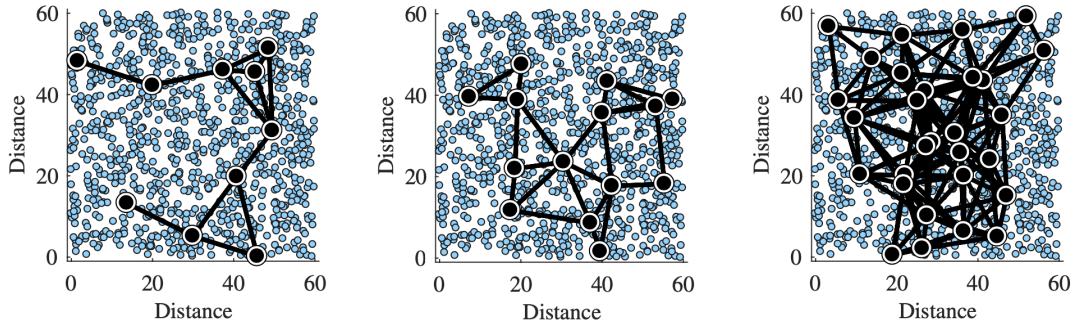
Apart from tuning the scanning and advertising intervals to save power, the Optimized tuning report [2] suggests another parameter that could potentially be tuned for power-saving purposes: transmit power (TXP). The report states: "Minimizing the TXP value will not only spare the network of traffic, as fewer relay nodes will receive and then relay messages but also node power consumption." [2]. Minimizing the TXP of the device will directly reduce the power drainage during scanning procedures. Silabs state in their Bluetooth documentation that by reducing the TXP from 8 dBm to 0 dBm the power consumption can be reduced by more than 120% using 100 ms advertising intervals, and 105% with 1 s advertising intervals [14].

Another study [1] also suggests that the latency and reliability performance in BT mesh can be greatly enhanced by optimizing advertising/scanning parameters, implementing power control techniques, and customized relaying. Default parameter configurations were otherwise used during the performance testing. However, the test-bed only consisted of 20 nodes over an area of 600 sqm.

Comparing this to similar performance tests conducted by Silabs [12] (240-node network over 2200 sqm) and Åkredalen [2] (100-node network over 1400 sqm), this 20-node network is fairly small and may therefore not face the same challenges related to scaling (mainly reduced latency and reliability performance) as was reported with the larger networks.

3.6 Relay node selection for Bluetooth mesh networks

Closely related to minimizing power consumption is network relay selection. SiLabs comment in their BT mesh tuning report [13] states that relay nodes must be chosen cautiously due to their traffic-inducing nature (because of the flooding mesh) and power consumption cost as they are constantly scanning for incoming advertising packets and then re-transmitting them. SiLabs recommend making use of the nodes in the network which are connected to the building's electrical system, such as is often the case with light bulbs and configure them to be relays as they are not power-constrained.



(a) *Greedy Connect* algorithm. (b) *K2 Pruning* algorithm. (c) *Dominator!* algorithm.

Figure 10: Illustration of the relay distribution with the three different algorithms: Greedy Connect, K2 Pruning, and Dominator!. Each relay is represented as a black dot, with links between them

Source: Hansen et al. with the Department of Electronic Systems, Aalborg Universuty

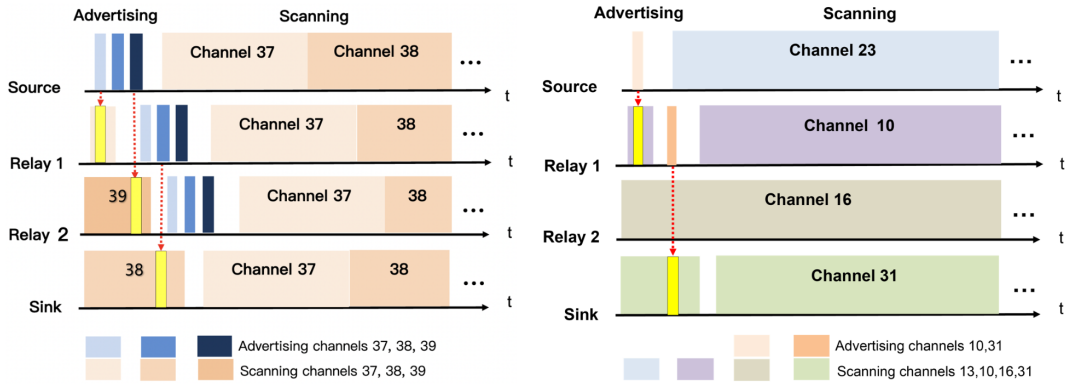
A report by Hansen et al. published by the Department of Electronic Systems at Aalborg University considered three different relay selection algorithms focusing on distributing the relay nodes to improve the *packet drop rate*, or reliability, of the BT mesh network [11]. The algorithms evaluated in their network simulation was *Greedy Connect*, *K2 Pruning* and *Dominator!*. The algorithms are run in a distributed (decentralized) manner. The simulated results show that all algorithms provide better overall network reliability compared with the case where all nodes are acting as relays. See Figure 10 for an illustration of the relay distribution with the three different algorithms. The *Dominator!* comes out on top by completing the relay selection efficiently and is also prone to topology changes making it a good choice for dynamic networks such as with mesh.

Another paper [6] published by Warsaw University of Technology has attempted to optimize (minimize) the network relay count by formulating a Minimal Relay Tree (MRT), both as a heuristic algorithm as well as an exact by using integer linear programming[11]. The sum of all performed simulations suggests that efficient relay management can reduce the network energy consumption by up to 12 times! The report also points out the trade-off between energy usage and efficient message delivery.

[11] An integer linear programming is a [optimization] problem in which the decision variables are further constrained to take integer values. Both the objective function and the constraints must be linear - ScienceDirect.com.

3.7 BLE scanning procedures and Routing algorithms for Bluetooth mesh

BTM uses a *managed flooding* technique to relay messages in the network instead of routing as explained in Section 2. This flooding technique has been proven to scale poorly due to the broadcasting storm that it induces, resulting in reduced reliability [12] [2]. The *ACE algorithm*, as presented in a report published on MDPI in 2021 [31], is a routing mechanism that addresses the mesh flooding issue while exploring the use of a wider frequency range within the BLE frequency domain. The authors have designed an algorithm that uses *heartbeat packets*, which is already an integrated part of mesh, for route creation and maintenance. Different BLE channels are automatically, and adeptly, scheduled for each network node locally. See Figure 11 for an illustration of how the algorithm varies from the standard BT mesh. The report concludes that the ACE algorithm proves to be a highly efficient packet collision avoidance mechanism, reducing the end-to-end latency by 16% and improving the overall network reliability by 30% even with heavy traffic. Since managed flooding is mainly a scalability issue for mesh, it is promising that ACE seems to work more efficiently with increased network size and node density.



(a) Bluetooth Mesh (b) ACE

Figure 11: The figure illustrates the different scanning schemes for Bluetooth mesh compared to the ACE algorithm. Bluetooth mesh only utilizes three channels (37, 38, and 39) for both scanning and advertising. With ACE however, scanning and advertising get separate channel subsets.

Source: Yang, Li, Lv, Gao, Qiao, Liu and Dong [31]

Institute of Electrical and Electronics Engineers (IEEE) published an article in their Internet of Things Journal from March 2020 that concluded that BTM is: "vulnerable to external interference, even when limiting the frequency channels to only 3 out of the 40 available Bluetooth Low Energy (BLE) frequency channels" [21], which could easily lead to congestion problems in the three advertising (ADV) channels. This suggests that extended utilization of the currently unused BLE frequency channels could be a possible tool for performance optimization of BT mesh.

3.8 Next steps

This chapter has presented only a few of the existing algorithms, suggestions, and methods developed to optimize BT mesh networks, considering power consumption, interference, automatic joining, and parameter tuning. However, of the many automated optimization approaches presented, none of them have considered the option of tuning the NTC, PCR, and RRC parameters. The report on optimized lighting control networks [2] does however suggest this as a possible research field. The NTC, PCR, and RRC parameters are protocol specific for BTM and will from now on be referred to as the *transmit count* parameters. Each additional transmit count adds an element of redundancy to the message transmission but also adds to the total traffic in the network. Increased traffic could potentially lead to packet collision and network constipation ("bottlenecks"), which

again leads to poor network reliability and increased latency.

Multiple SoCs (System on a Chip) supporting BTM are available on the market. Depending on the manufacturer, the default values set for the protocol-specific (and non-protocol-specific) parameters may vary. The lack of any standardization or common guidelines on this matter is obvious when comparing the papers Silabs [12], Ericsson [9] and Åkredalen [2]. Silabs use a NTC = RRC = 3 configuration, while Ericsson only use an unspecified but non-zero NTC value. Åkredalen conclude with an optimal parameter setting of NTC = 2 and RRC = 0 [4]. Deciding on which values to choose for these parameters to achieve optimal performance for a specific application is not always obvious and usually requires a certain degree of technical insight into how the mesh protocol operates. Additionally, since the traffic and environment typically are non-constants in a mesh network, a *dynamic tuning* could be necessary for optimal performance. Maintaining a properly tuned set of parameters for such a network would be a tedious task for any Network Manager to handle. Also, a global tuning for any of these parameters may not be optimal as the network nodes may be subjected to different physical obstacles, environment disturbance, relay node density, and so on. The challenge of a non-constant, non-homogeneous environment and the trade-off between redundancy and the negative effects of increased traffic in the network is an important incentive to regulate these parameters, both dynamically and automatically, to improve the overall performance of BTM.

A few of the studies [7] [18] mention power control and one [16] mention the optimization of scanning cycles explicitly. However, TXP itself is rarely included as part of the research. TXP is an interesting non-protocol parameter to consider for optimization as it does not only greatly impact the device power consumption but can also enhance network reliability. A higher TXP means that more nodes can be reached, but, as BTM communication rely on relaying of messages and managed flooding, an increase in TXP means more relays will be triggered and more traffic is generated. Thus, TXP has a similar trade-off as with the message transmit counts.

The next chapter will explore the possibilities of developing an algorithm, running locally on every node in the network, that will tune these parameters, protocol specific: NTC, RRC and non-protocol specific: TXP, in real-time. The effect of the algorithm will be documented by running performance tests on a large-scale BT mesh network in an active office environment, with and without the use of the algorithm. The goal for the algorithm will be to achieve high reliability with acceptable latency, while potentially reducing the overall network power consumption.

4 The algorithm: Key idea and specification

The algorithm to be developed will regulate the number of message *transmit counts*: Network Transmit Count (NTC) and Relay Re-transmit Count (RRC), and the device radio Transmit Power (TXP) to optimize reliability and minimize the power consumption of the Bluetooth mesh (BTM) network. Although increasing the message transmit counts have the effect of adding redundancy, it also induces traffic which could potentially cause more packet collisions and may result in poor reliability. The TXP has a similar trade-off; if set too high, the transmitted message will have a longer reach, potentially triggering many relay nodes at the same time, which will cause more traffic when relayed. But if the TXP is set too low, the message might not even reach its nearest relay, get corrupted, lost, or otherwise compromised due to interference or external disturbance. Hence, it is important that the algorithm carefully regulate these parameters (NTC, RRC, and TXP) to obtain an optimal network performance. Regulating these parameters in real-time with help of an algorithm is a novel way of improving the trade-off between performance, traffic, and power consumption of mesh devices. This mechanism goes beyond the typical implementation of a standardized BTM device.

This Chapter describes the distributed algorithm *specification* through a set of listed criteria points and high-level logic diagrams. The specification will be used as a roadmap and quality control when implementing the algorithm later on.

4.1 Specification

This specification explains the acceptance criteria that define successful achievement for the algorithm. The key idea is that each relay node in the mesh network will take on the role of a "master" and maintain a table of its associated neighbor nodes by periodically requesting status updates. Upon receiving a status request, the neighbor node will respond with a status message containing its Received Signal Strength Indicator (RSSI), transmit Time-To-Live (TTL), NTC and TXP to be used as input to the algorithm regulator running on the Relay node. Figure 12 illustrates the main system regulation loop.

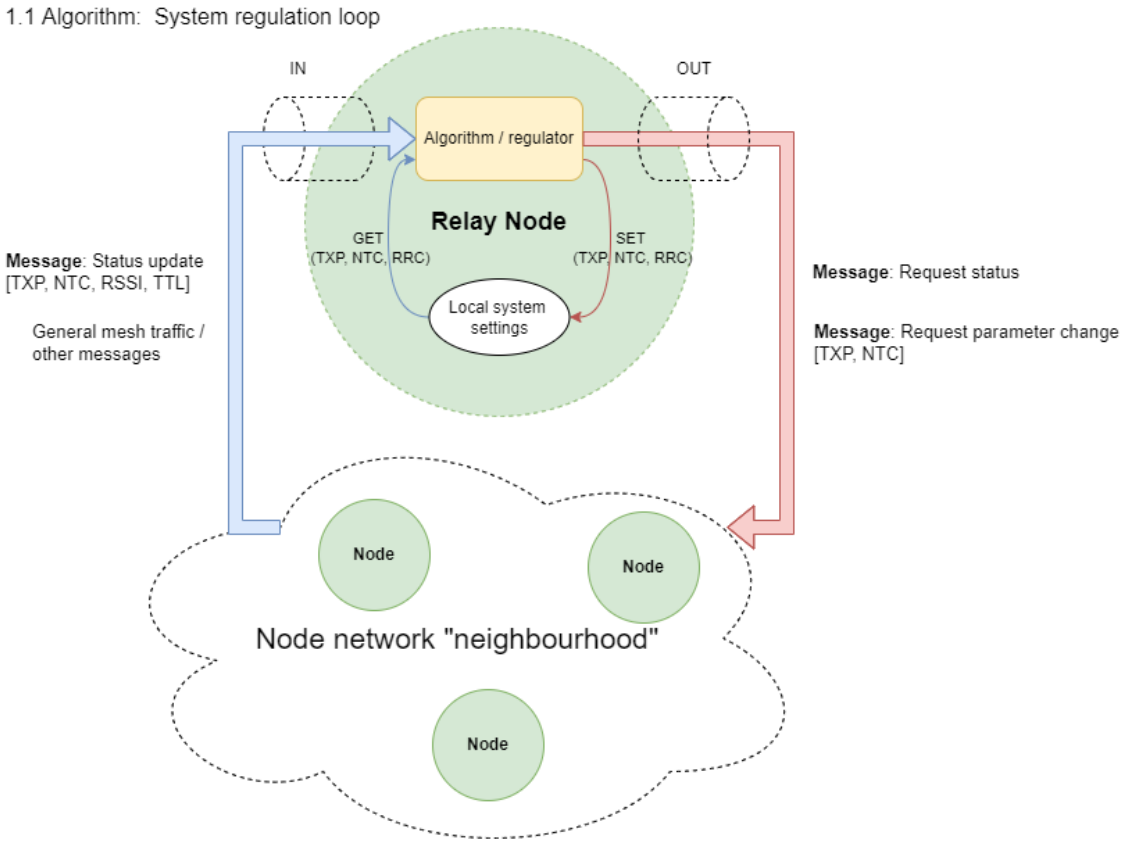


Figure 12: The Relay will act as a "master" that will regulate its own and its neighbour node's parameters by periodically requesting status updates, as well as request parameter changes when needed.

Given below is a list of criteria for the algorithm:

1. Each Relay node (not the capital letter for denoting the specific relay *role*, not the relay state) will periodically request a status message from nearby nodes.
2. The Relay will have to keep track of which nodes should be considered a "neighbor". For a node to be considered a neighbor, the average hop count for its status messages must be less than 0.5. This is to make sure that each Relay only manages the nodes within its immediate proximity.
3. Upon receiving a status update from a neighboring node, the Relay will first check if the origin node's RSSI, which is the strength of the signal at which the neighbour node received the last status request message [23], is out of range: [-a, -b] dBm. If this is the case, a message will be sent back to the origin node requesting it to change its TXP value. The Relay node must determine how to regulate the TXP to meet the RSSI range criteria. The flow chart in Figure 13 shows the high-level logic of the Relay upon receiving a status message for a neighbour node.
4. The Relay will check the node's Status Received Rate (SRR). The Relay will keep track of how many status messages are received from each neighbour node and how many status requests it has sent out in total. If the SRR falls below X%, then the Relay will attempt to mend the SRR by enhancing the *message redundancy* by increasing the following parameters: Relay NTC and RRC and origin node's NTC, will by referred to as *transmit counts* from now on. The high-level logic of this mending procedure is shown in Figure 14.
5. The algorithm will keep track of the total number of messages (α), both general network messages and algorithm-related messages combined, originating from its neighboring nodes.

A high α value would indicate that the message flow through the Relay is getting out of control and needs to be regulated. If the total (α) exceeds a certain limit per time interval, then the NTC will have to be decreased for some nodes. A priority may be given to the nodes with the lowest RSSI, which relay-path would be most prone to disturbance, to keep their current NTC. See Figure 15 for a simplified logic for this event. The total relay traffic (α) to be measured per time unit is hence a function of all the neighboring nodes' NTC count, see Equation 1.

$$\alpha \; = \; \upsilon \; (\; NTC \; \forall \; nb\_nodes \;) \tag{1}$$

Hence, the transmit count parameter (TC), of which the Relay NTC, Relay RRC, and neighbour NTC values will follow, will be a function of the two factors SRR and α, see Equation 2 below.

$$TC \; = \; \nu \; (\; SRR, \; \alpha \;) \tag{2}$$

6. The Relay node will have to manage its associated neighbour nodes whilst being some other Relay's neighbour as well. This is to maintain a strong path connection throughout the entire mesh network.

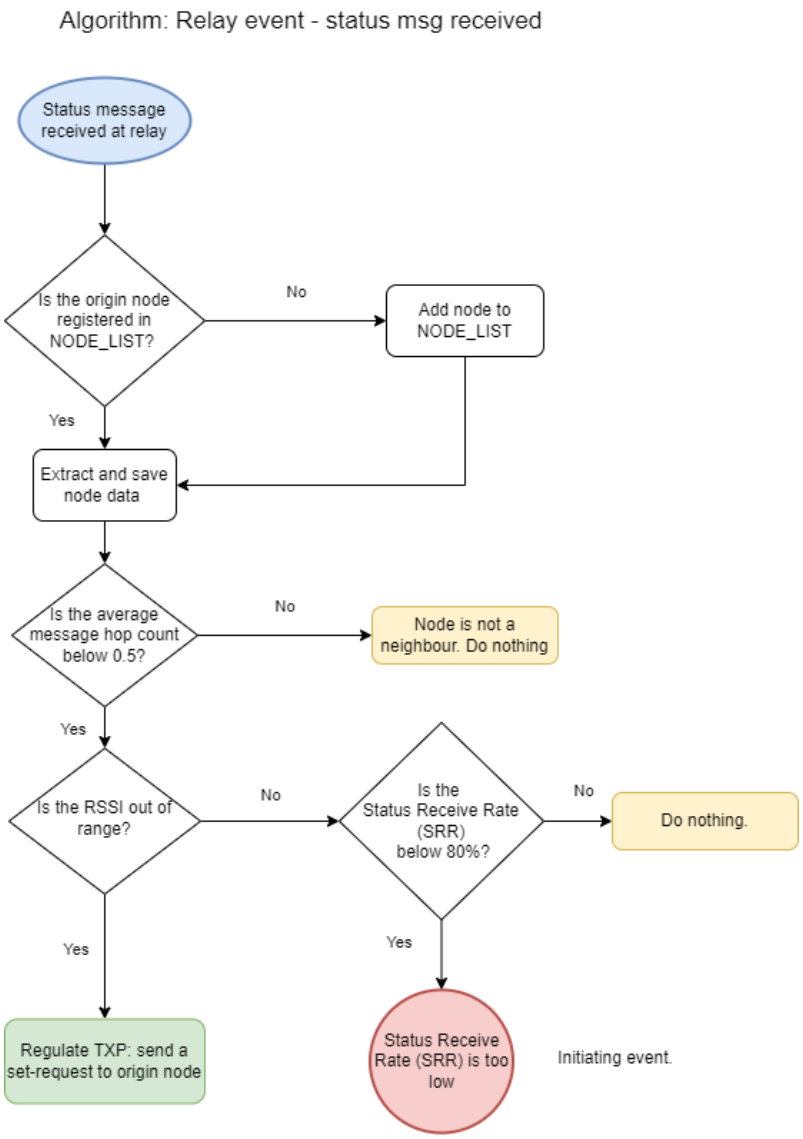


Figure 13: Simplified flowchart logic for the relay node after receiving a status message from an arbitrary node. The RSSI is regulated to stay within a given limit to save device power consumption. If the Status Receive Rate (SRR) is too low, a mending process is initiated, see Figure 14.

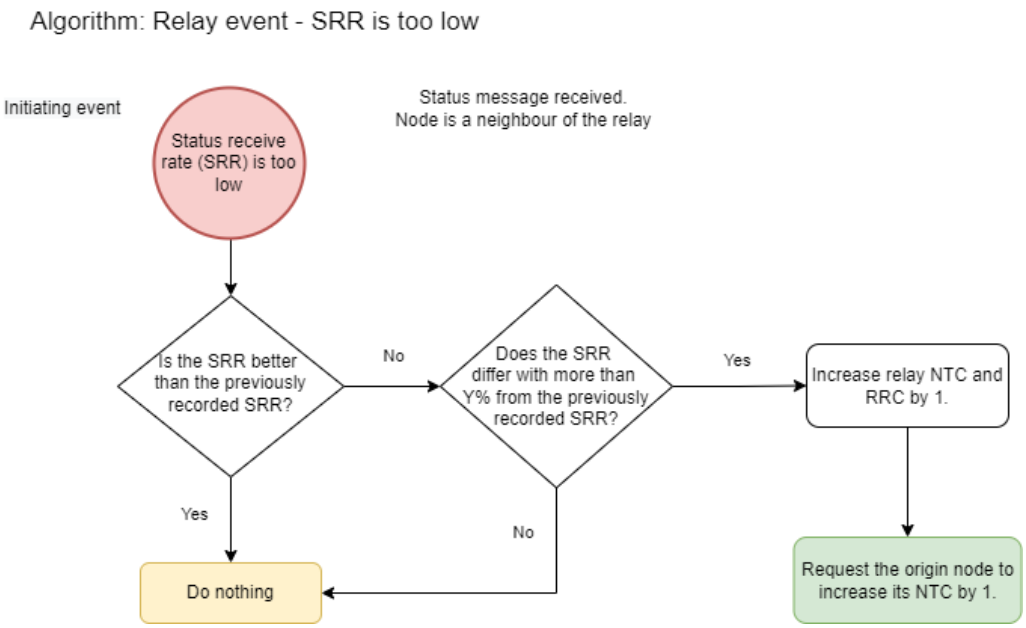


Figure 14: Simplified flowchart logic for the relay node after receiving a status message from a neighbouring node with poor Status Receive Rate (SRR). The Relay may change its own local parameters or send a set-message to the neighbour node requesting it to change theirs.

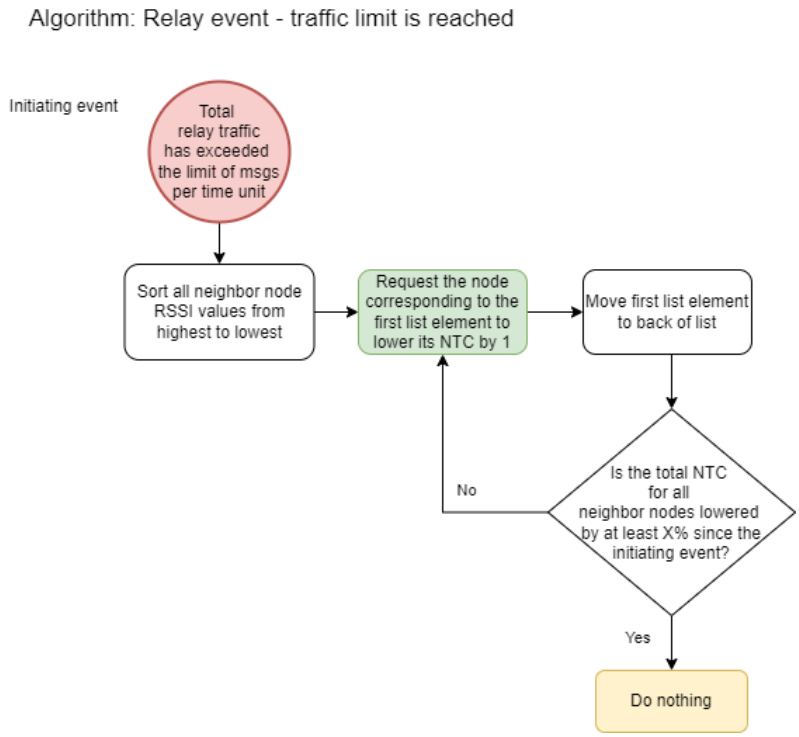


Figure 15: Simplified flowchart logic for the relay node after traffic flow has exceeded its limit

Note that a node's SRR it is only affected by the local NTC and the NTC at the Relay. This is because the status request messages are originating from the Relay itself. However, a relay node's most important job is to relay messages of which may originate from a node far across the network. For the Relay to be able to relay these messages along an equally strong path as when communicating directly within its neighbourhood, it has to regulate its RRC accordingly (RRC = NTC), see the example in Figure 16.

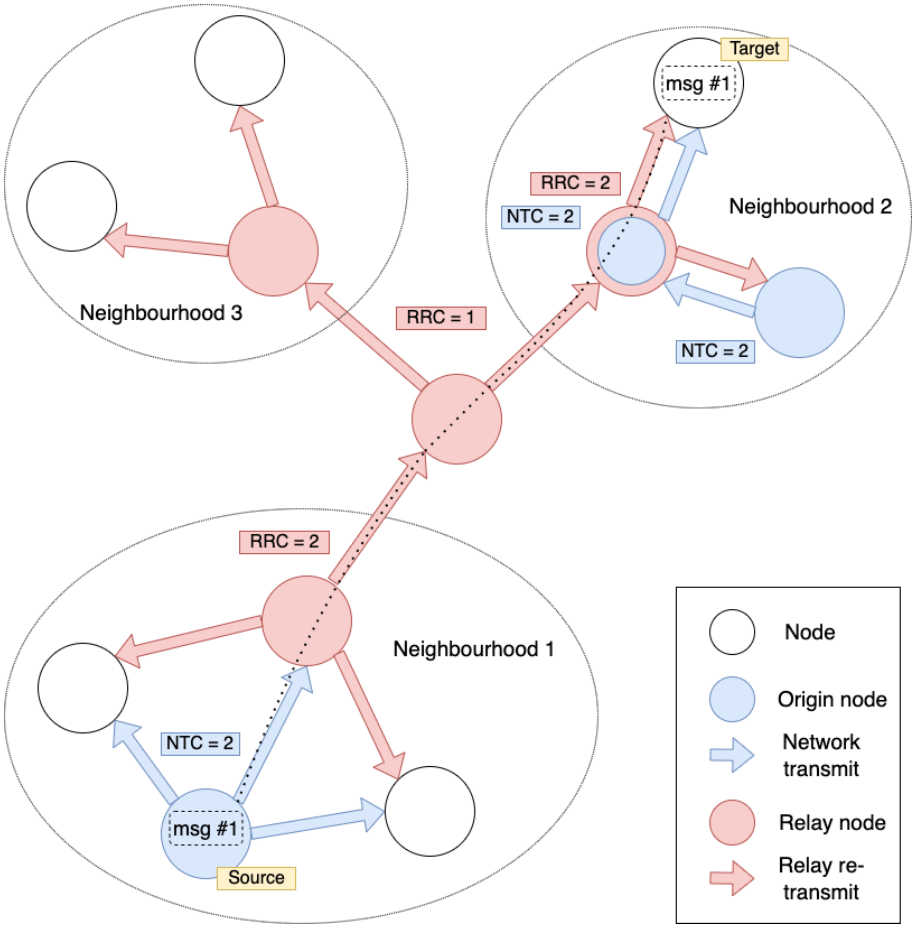


Figure 16: A relay node will have to maintain an equally strong connection with its neighbourhood for relayed messages as with self-transmitted messages. The dotted line shows how a message's delivery success is dependent on its target neighbourhood relay node's RRC, not the NTC.

All details in the listed points and figures in this subsection are required for the algorithm to achieve its intended goal of an optimized BT mesh network considering both power consumption and reliability. Any additional logic or design solutions are acceptable as long as they do not compromise these requirements.

4.1.1 KPI and assumptions

An assumption for the algorithm is that the relay node coverage in the network is, somewhat, sufficient. Some of the reports in the previous works [6] [11] did explore the effect of various relay selection algorithms, thus, this issue will not be included with this algorithm.

The ultimate goal for the algorithm is to be able to minimize the overall power consumption of the BT mesh network by optimizing the devices' radio TXP and to enhance reliability (successful message delivery) through regulation of transmit counts (NTC and RRC) in real-time. Latency is considered a KPI (Key Performance Indicator) for some network types, such as with lighting control networks. However, although latency will be evaluated, this will not be the main focus of

the algorithm. To summarize, the KPIs for the algorithm are the following:

- Overall network power usage
- Overall network reliability

The next chapter will introduce the physical test-bed of nodes that will be used during development and for performance testing of the algorithm, as well as all software and hardware tools used in the development.

5 Test-bed and tools

The Bluetooth mesh (BTM) network used for performance testing in this report is a large-scale 100-node network, or "test-bed", located in Nordic Semiconductor's 4th-floor office in Trondheim. The test-bed was installed during the spring of 2021 and is the very same network that was used in the report *Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks* [2]. Figure 17 shows the floor map and the approximate placement of each node. The nodes are mounted on the ceiling plates in corridors, offices, etc., see Figure 20 and 19.

Figure 17: Large scale network office floor map - Nordic Semiconductor Trondheim. The pink dots indicate the approximate placement of each node (PCA20036 kit) in the network. The entire floor is approximately 1350 square meters.

Source: Koteng / Nordic Semiconductor

5.1 Hardware tools

The test-bed consists of 100 custom-made PoE (Power over Ethernet) hardware development kits (DK) labeled PCA20036, a printed circuit assembly (PCA) featuring the Nordic Semiconductor nRF52840 microchip [4], the W5500 WIZnet integrated circuit (IC) TCP/IP embedded Ethernet controller [30], a high-power light-emitting diode (HP LED), an antenna and an Ethernet port for power supply and virtual serial communication[12]. See Figure 18 for a simplified block diagram for the PCA20036 board. The kits can receive device firmware updates (DFU) and other commands directly using the same Ethernet connection. Ethernet back-channel connectivity allows for time-sync analysis through real-time logging during testing.

[12] A virtual serial port over Ethernet provides all of the functionality of a physical COM interface, able to transmit serial data over a network (Internet or LAN). Source: serial-over-ethernet.com

The nodes are all interconnected through three stacked[13] PoE switches, one 24-port and two 48-port, connected to a dedicated computer.

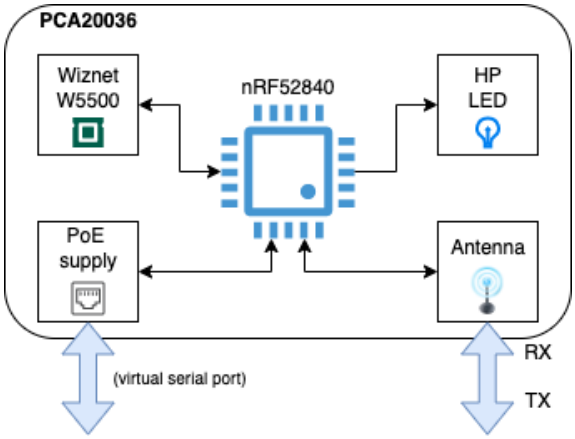


Figure 18: PCA20036 block diagram. The main components of the PCA used during testing are the nRF52840 microchip, the WIZnet W5500 IC, a high-power LED, an antenna and, an Ethernet port for power supply and serial connectivity.

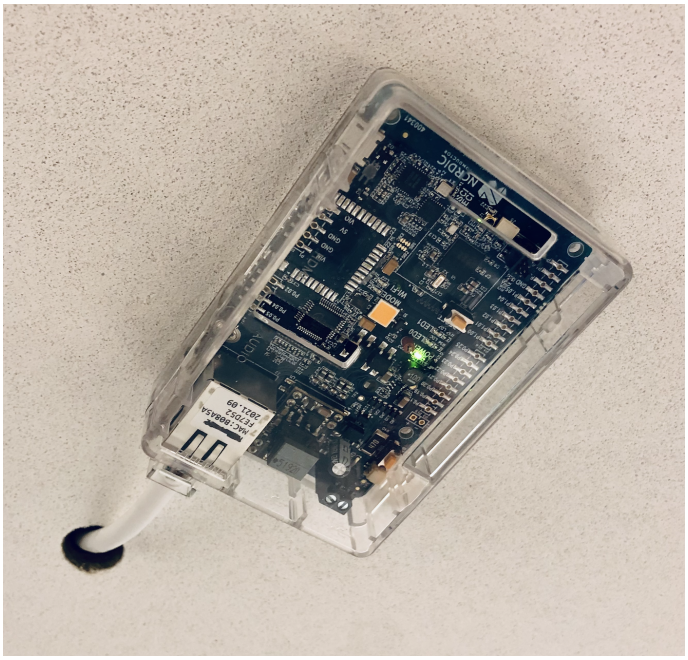


Figure 19: A PCA20036 kit mounted on a ceiling plate in the Trondheim office of Nordic Semiconductor. Each kit is enclosed in a Raspberry Pi casing for protection from dust.

5.1.1 Utilizing and access of the test-bed

The dedicated test computer can be remotely controlled from anywhere using a remote desktop client[14], internal VPN access and a WiFi connection, see Figure 22. See Figure 21 for the PoE switch in one of the server rooms that is connected to the dedicated test-bed computer.

[13] In networking, the term "stack" (or stackable) refers to a group of physical switches that have been cabled and grouped in one single logical switch. Source: Cisco

[14] A remote desktop client let you use and control a remote PC from another device

Figure 20: PCA20036 kit placement in office hallway in Trondheim. 100 kits are evenly distributed throughout the office space.

Source: Private photo

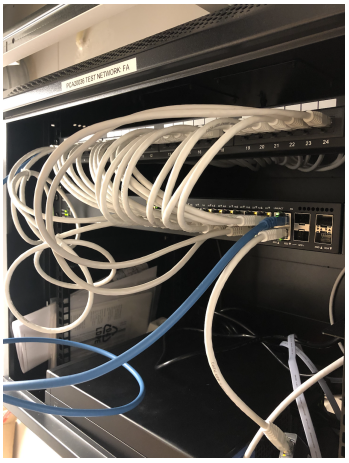


Figure 21: A POE switch in on of the server rooms

Source: Private photo

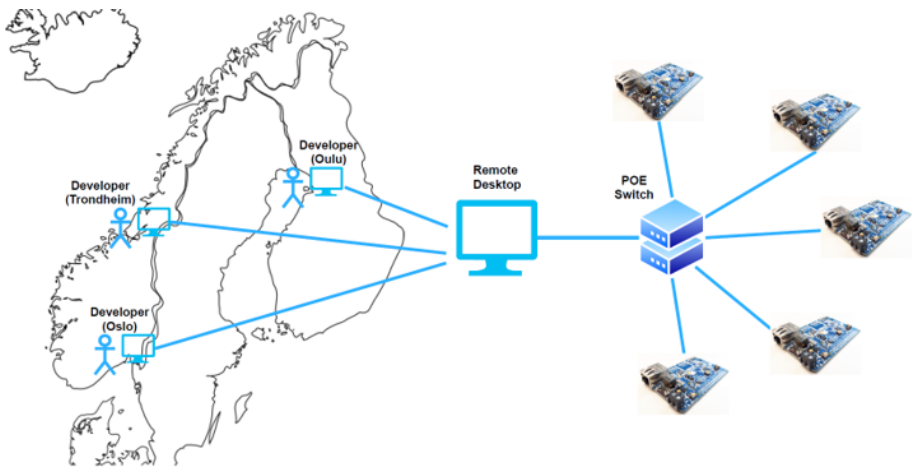


Figure 22: The test-bed can be accessed from anywhere. The dedicated computer (labeled "Remote Desktop" in the figure) can be controlled using a remote desktop client on a developer's private computer.

The performance test scripts or other command scripts, such as to configure network parameters, etc., can be launched from the connected computer's terminal.

5.2 Software tools

The algorithm will be developed using the Nordic nRF Connect Software Development Kit (SDK), or nCS, a developing environment for Nordics nRF52 and nRF53 series SoCs (System on a Chip) [3], including the nRF52840 which is used for the PCA20036 board. The nCS source code includes everything from drivers, libraries, examples, and radio protocols. The repository can be found publicly on GitHub and contains Nordic's additions to open source projects Zephyr RTOS (real-time operating system) and MCUboot (a secure bootloader[15] for 32-bit microcontrollers [17]). Together with the tool-chain listed in the SDK documentation [5], the SDK provides developers with a complete solution for them to develop applications using Nordic's wireless low-power nRF52, nRF53, and nRF91 series devices.

The latency and reliability performance test and custom Ethernet commands to be used during testing in this report are the same as was used in the author's previous report [2]. The test code was developed as part of a Nordic student summer project in 2021. The code is not publicly available but some high-level logic and basic concepts will be presented in Section 9.

[15] A bootloader is a special operating system software that loads into the working memory of a computer after start-up. Source: ionos.com

6 Data emulator: Development

As a first step in the development of the optimization algorithm, a Python script, see Appendix A, is created to simulate the messaging flow of a small Bluetooth mesh (BTM) network and also emulate the regulation logic of the specified algorithm as given in the specification (Section 4.1). The script has three classes: one for *Relay nodes*, one for *Edge nodes* - nodes with their relaying feature disabled only to act as a neighbour of the Relay, both with their associated class functions which handle incoming and outgoing messages and other necessary logic, and one for the emulator itself. For simplicity, the function definitions are hidden and only the function bodies are shown with their inputs and output. An exception is made for the `Emulator` class `_run`-function which shows how the main loop of the emulator works.

The node objects created in the program are initialized from the `nodes` dictionary, where they have a pre-defined relative position. An Edge node's distance from the Relay node, together with its current Transmit Power (TXP) value, is used to calculate the assumed path-loss indicating the node's RSSI value. The main program, see bottom of the script, creates one Relay and two Edge objects for this simplified demo. The Relay is periodically requesting status updates from the Edge nodes every 10 seconds. Integrated with the code is a *message loss factor* used to determine the chance of message success. The TXP, Network Transmit Count (NTC) and Relay Re-transmit Count (RRC) values will alter this factor, simulating the disturbances in an actual networking scenario. The main loop will also check the current traffic flow through the relay node every 30 seconds and adjust the overall message transmit count for the neighboring edge nodes if needed. The Relay will update its `relay_table`, containing the parameters of all nodes providing it with status messages, for every status update received.

6.1 Emulator execution demo

The program execution, see the terminal output below, demonstrates how the Relay node (address: R01) is tracking and mending each of the Edge nodes' (addresses: E01 and E02) Status Received Rate (SRR) and network power consumption by regulating its own parameters (TXP, NTC, and RRC) or sending out SET message requests to change TXP or NTC values for the Edge nodes. The Relay node must maintain a table with all its neighbour nodes' parameters. The terminal output shows a snapshot from time `t`, where `t` is some time that has passed and the nodes SRR is getting too low, till `t` + 120 where the SRR with both Edge nodes have been mended. The set-point for the regulation was set to 0.8. Note that the SRR value does not reflect the actual reliability goal of the network in terms of successful arrival of messages. Instead, the SRR is a minimum local reliability measure between one network node and its closest relay. In reality, a message will have multiple pathways and is not dependent on this single connection.

```
TERMINAL OUTPUT for: python opt_alg_emulator.py

---------------------------- Time:  t + 0 ----------------------------------------

R00  ------- GET (ntc: 1 rrc: 1 txp: 2) -------> ALL
<E01> GET message received
E01  ------ STATUS ------>  R00
<R00> STATUS received:  {'node_id': 1, 'addr': 'E01', 'rssi': -74,
'hop': 1, 'txp': 2, 'ntc': 0, 'traffic': 3}
<R00> E01  has too *low* RSSI (-74)
R00  ----- SET TXP (ntc: 0 rrc: 0 txp: 4) ----->  E01
<E02> GET message received
E02  ------ STATUS ------>  R00
<R00> STATUS received:  {'node_id': 2, 'addr': 'E02', 'rssi': -42,
'hop': 1, 'txp': 0, 'ntc': 1, 'traffic': 6}
<R00> E02  has too *low* SRR (0.692308)
<R00> Minimal SRR decrease detected. Wait...
```

```
---------------------------- Time: t + 10 ----------------------------------------

R00  ------- GET (ntc: 1 rrc: 1 txp: 4) -------> ALL
<E01> GET message received
E01  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 1, 'addr': 'E01', 'rssi': -74,
'hop': 1, 'txp': 2, 'ntc': 0, 'traffic': 3}
<R00> E01  has too *low* RSSI (-74)
R00  ----- SET TXP (ntc: 0 rrc: 0 txp: 4) -----> E01
<E01> SET request for TXP received. TXP is now 4
<E02> GET message received
E02  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 2, 'addr': 'E02', 'rssi': -42,
'hop': 1, 'txp': 0, 'ntc': 1, 'traffic': 6}
<R00> E02  has too *low* SRR (0.714286)
<R00> Minimal SRR decrease detected. Wait...
```

At time t + 0 (above), the Relay broadcasts a GET message requesting Edge node status updates. Responses are received from both Egde nodes, E01 and E02. The E01s' RSSI is evaluated to be too low. The Relay then answers with a SET-request to increase the TXP to better the chances of successful message delivery. The SET message is unicasted with the same parameters as the target Edge node as they currently stand in the relay table. The E02s' SRR is evaluated to be too be below the lower limit of 0.8. However, the relay chooses to wait before sending a new SET-request because the node has previously been regulated and the change in SRR since then is marginal.

At t + 10 the E01's RSSI is still too low and the status message indicates that the previous SET message did not get through. This is also seen in the GET broadcast at t + 10 where the relay's TXP is set to 4 which is always set to the highest value for all neighbour nodes in the relay table. A new SET-request is sent from the relay to E01 and this time the Edge node successfully receives the message and adjusts its parameters. E02 has too low SRR but relay waits as the decrease from the last regulation is still small.

```
---------------------------- Time: t + 50 ----------------------------------------

R00  ------- GET (ntc: 1 rrc: 1 txp: 4) -------> ALL
<E01> GET message received
E01  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 1, 'addr': 'E01', 'rssi': -64,
'hop': 1, 'txp': 4, 'ntc': 0, 'traffic': 1}
<R00> E01  has too *low* SRR (0.666667)
<R00> Adjusting Relay RRC to 1
<R00> Adjusting Relay NTC to 1
R00  ----- SET NTC (ntc: 1 rrc: 1 txp: 4) -----> E01
<E01> SET request for NTC received. NTC is now 1
<E02> GET message received
E02  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 2, 'addr': 'E02', 'rssi': -42,
'hop': 1, 'txp': 0, 'ntc': 1, 'traffic': 2}
<R00> E02  has too *low* SRR (0.733333)
<R00> E02 SRR regulation is too slow. Continue regulation...
<R00> Adjusting Relay RRC to 2
<R00> Adjusting Relay NTC to 2
R00  ----- SET NTC (ntc: 2 rrc: 2 txp: 0) -----> E02
<E02> SET request for NTC received. NTC is now 2
```

At t + 50 we can see an example of how the E02's SRR is still too low after 5 status messages. The relay now concludes that even though the SRR has improved since the last regulation, the process

is too slow. The relay adjusts its parameters and sends a new SET-request to E02. E01 now has an acceptable RSSI (with a TXP of 4 dBm) but the SRR is too low and the relay regulates this by increasing the message transmit count.

```
---------------------------- Time: t + 80 ----------------------------------------

R00  ------- GET (ntc: 2 rrc: 2 txp: 4) -------> ALL
<E01> GET message received
E01  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 1, 'addr': 'E01', 'rssi': -64,
'hop': 1, 'txp': 4, 'ntc': 1, 'traffic': 4}
<R00> E01  has too *low* SRR (0.722222)
<R00> E01  is improving. No further changes are necessary
<E02> GET message received
E02  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 2, 'addr': 'E02', 'rssi': -42,
'hop': 1, 'txp': 0, 'ntc': 2, 'traffic': 12}
<R00> E02  has too *low* SRR (0.777778)
<R00> E02  is improving. No further changes are necessary
<R00> Traffic is too high. Initiating regulation process...
<E02> SET request for NTC received. NTC is now 1
<R00> regulation process is complete.
```

At t + 80 the relay checks the traffic message flow (amount of messages received at the relay from neighbour nodes per time unit). It has exceeded its limit. The relay responds by decreasing the NTC value for the Edge node with the highest SRR in the relay table as this is most likely the most robust path in terms of message loss and chances are that a decrease in NTC for this node will not affect the SRR significantly.

```
---------------------------- Time: t + 120 ---------------------------------------

R00  ------- GET (ntc: 2 rrc: 2 txp: 4) -------> ALL
<E01> GET message received
E01  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 1, 'addr': 'E01', 'rssi': -64,
'hop': 1, 'txp': 4, 'ntc': 4, 'traffic': 15}
<R00> E01  is OK. (SRR: 0.8125)
<E02> GET message received
E02  ------ STATUS -----> R00
<R00> STATUS received:  {'node_id': 2, 'addr': 'E02', 'rssi': -42,
'hop': 1, 'txp': 0, 'ntc': 1, 'traffic': 2}
<R00> E02  is OK. (SRR: 0.8125)
```

By t + 120 both nodes are at an acceptable SRR level and will not receive any further parameter change requests before the SRR drops below 0.8 or the RSSI is out of range.

Figure 23 - 24 shows the final results for the data emulator. Both Edge nodes are converging towards their initial goal of 0.8 SRR while their NTC values are being regulated on demand. The RSSI is being kept within its acceptable limit, in this emulation: [-70, -20] dBm, by regulating the TXP, as seen in Figure 24.

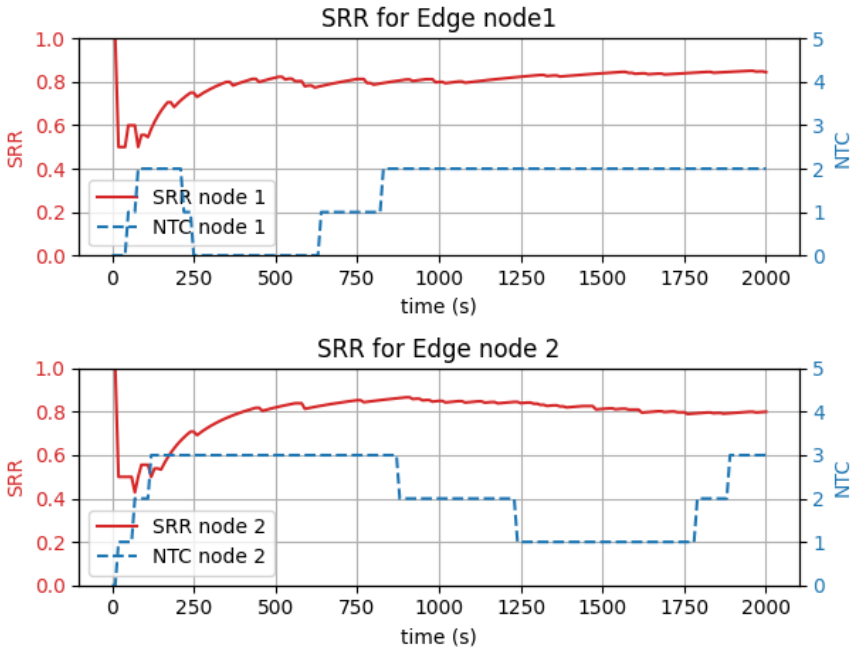


Figure 23: The graphs show the SRR and NTC values for both Edge nodes over a period of 2000 seconds.

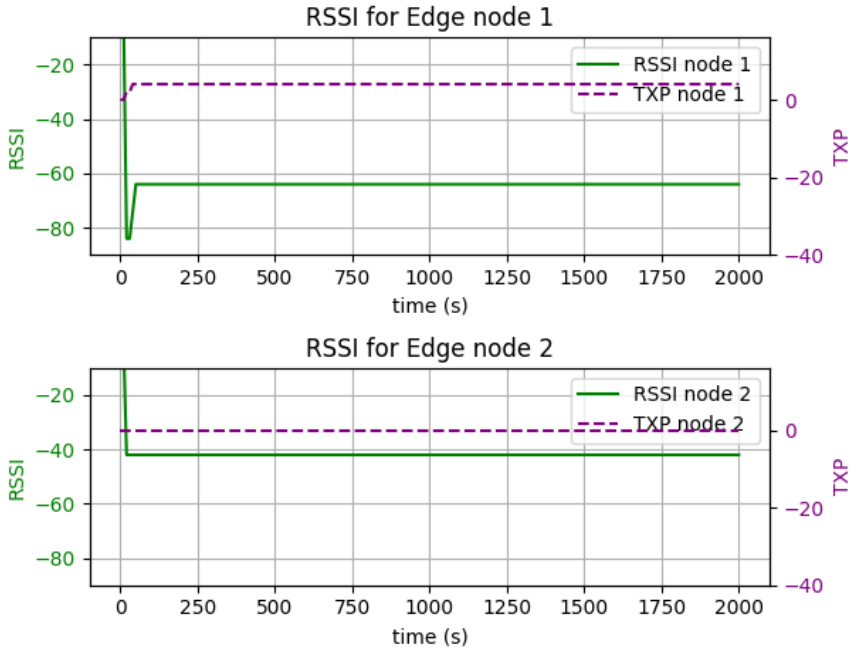


Figure 24: The graphs show the RSSI and TXP values for both Edge nodes over a period of 2000 seconds.

The TXP and message transmit counts (NTC and RRC) for the Relay node is shown in Figure 25. The relay switches NTC (RRC = NTC) and TXP according to which Edge node is the target of its transmitted message. The average TXP over the period of 2000 seconds is 3.66 and the average NTC is 2.32. Without the algorithm, the Relay would have static values for these parameters which would most likely force it to use higher values to successfully connect with all its neighbour nodes.

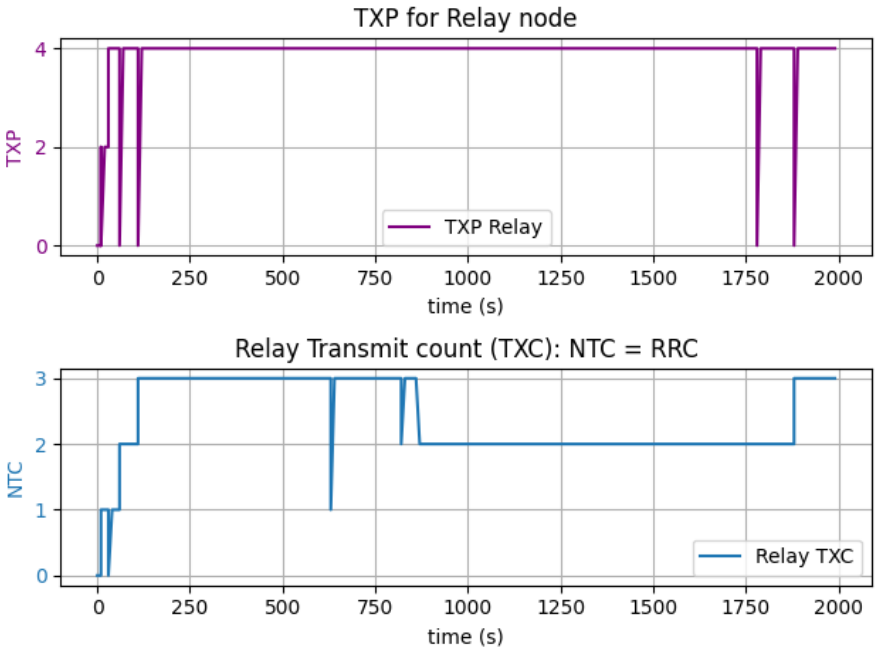


Figure 25: The graphs show the transmit count (NTC and RCC) and TXP values for the master Relay node over a period of 2000 seconds.

The emulator demo has shown that the algorithm logic works, in theory, as the specification intended. The algorithm succeeds in regulating the SRR, thus improving reliability for the neighbour nodes while also minimizing power consumption by transmitting messages with the TXP corresponding to the highest demand in the *neighbourhood*. The node TXP is based on its current RSSI level. However, the data emulator cannot perfectly simulate actual mesh traffic and the problems that might arise in a real large-scale network. Some of these problems are for example blocking of signals due to office employees moving around (also called "shadowing"), nearby equipment operating on the same Bluetooth Low Energy (BLE) frequency creating disturbance, or the actual traffic impact a node's increase in RRC and NTC will have on the network. Therefore, the same logic will have to be embedded on actual mesh nodes using the full-sized network as was presented in Section 5. In the next chapter, some details on the embedded implementation work will be discussed, along with some relevant challenges that came with it.

7 Embedded implementation and function testing: Development

The embedded algorithm was developed using the software tools as presented in Section 5.2, coded in C language. A 22-node sub-portion of the large-scale network was used during most of the embedded development, see Figure 26. This sub-network will be referred to as *zone 1*. In the following chapters, the network nodes with their relay feature disabled will be referred to as *Edge nodes* and the rest as *Relay nodes*.

Figure 26: The isolated embedded function testing was run on a 22-node sub-portion of the large-scale 100-node network, called Zone 1.

7.1 Design challenges

While working on the embedded algorithm, some new challenges surfaced that were not apparent while testing with the emulator. Some were new because of the limited and non-realistic scope of the data emulator, while others were due to mesh stack limitations. In this subsection, these challenges will be discussed in detail and their solutions are justified.

7.1.1 Direct transmission for accurate RSSI reading

The 2nd specification criteria point in Section 4.1 states that the Relay will have to keep track of which nodes are in its neighbourhood and that the STATUS messages received from a node must not exceed 0.5 hops on average to be considered a neighbour. However, since the reading of the Received Signal Strength Indicator (RSSI) is an essential part of the algorithm logic, the messages need to be direct transmissions. If a STATUS message at some point made a hop, meaning that it was relayed once before it arrived at its destination, then the RSSI reading would give the radio strength between the relaying node and the destination node, not between the origin node and destination node. Because of this, it was more convenient to set all GET and STATUS messages' transmit Time-To-Live (TTL) value to 0. This will not only ensure direct transmission of the messages and correct RSSI values but also limit the traffic impact the algorithm-generated messages has on the network.

7.1.2 Cluster regulation limitation

The 4th criteria point in the specification states that the Relay will attempt to mend a node's Status Received Rate (SRR) by increasing the message transmit count (relay NTC, relay RRC and target node NTC). However, since these values may be different between the neighbour nodes, the relay will have to limit its transmit parameter regulation to follow the *highest demand* in the relay table, meaning the maximum value of NTC for all neighbouring nodes in the table and so on. The periodic GET-message could potentially be unicasted to each neighbour node with the transmit parameters matching the target node's parameters from the relay table, but as mesh use a *managed flooding* (see Section 2) this would cause unnecessary amounts of traffic. Another reason for the relay to use the highest-demand transmit parameters at all times is that when relaying a message the relay's network layer is handling the request, not the model/application layer. The network layer does not have access to the relay table containing the specific target node's parameters. Even if a work-around was possible, which would involve "pipelining" information from the network layer to the application layer, checking the message destination, verifying neighbourhood membership and then setting local transmit parameters before resuming the relaying work on the network layer. This would require additional processing time which would affect the overall network latency. Because of this limitation in the BT mesh stack, the embedded algorithm will only regulate on a "cluster-level", meaning that each relay node will maintain an optimized transmit parameter setting (NTC, RRC and TXP) to best meet the demand of all the nodes in its neighbourhood.

7.1.3 Multiple relay coordination

The emulator did not test having multiple nodes acting as relays, actually relaying of messages, or how the relays may interfere with each other. A relay receiving a SET-message from another relay will simply have to ignore that message. If the relay was to update any of its transmit parameters according to the request from a neighbouring relay, it would not be able to maintain an optimal parameter set for its own neighbourhood of Edge nodes which could potentially end in loss of reliability. Let's assume that the STATUS-messages exchanged between two relay nodes were sufficient enough for the relay to choose its transmit parameters, including other relays in its neighbourhood. Recall that a relay's transmit parameters (NTC, RRC = NTC and TXP) will be set according to the highest demand of all neighbouring nodes. This leads to another problem: if relay A has relay B in its neighbourhood and B is "struggling" with just *one* of its neighbours and must increase its TXP to a high value, then this TXP value will most likely be used by relay A as well since B's TXP might be the highest value amongst all A's neighbours. This could again propagate to other relays in A's neighbourhood and cause the entire network to be unnecessary "loud", resulting in poor reliability due to noise and increased power usage at the same time.

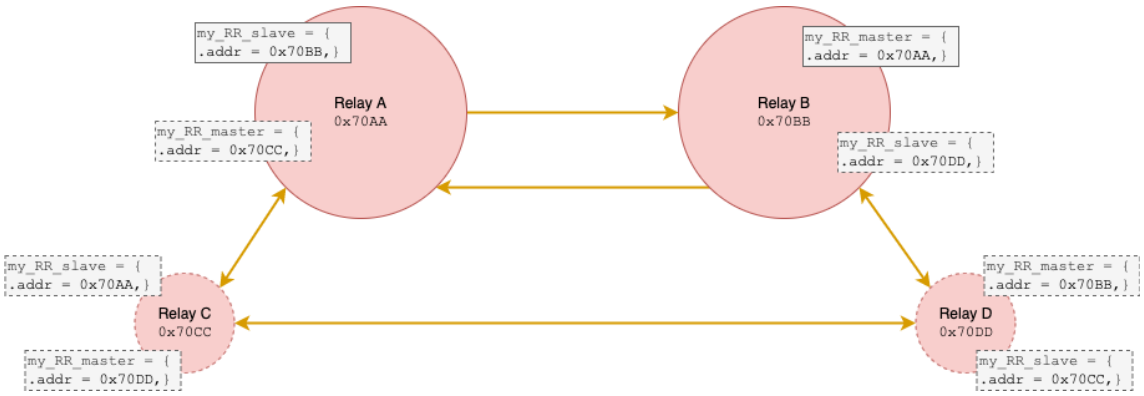


Figure 27: Every Relay will have a master and a slave tag. Their addresses cannot be the same. Each of these tags will hold an address and a RSSI and TXP value. Only the addresses are shown in the figure.

The 6th criteria point in the specification states that there must be some relationship between the

relay nodes to tie the network together. A logical criterion for all relays is to have at least one relay in its neighbourhood that has a RSSI within excepted range. This is to ensure a strong relay path throughout the mesh network. However, with the current logic, this proved to be difficult without experiencing the "loud neighbourhood"-issue. The suggested solution is that all relays hold a Relay-to-Relay (RR) slave tag: `struct my_RR_slave`, and a Relay-to-Relay master tag: `struct my_RR_master`, see Figure 27. The `my_RR_master` and `my_RR_slave` tags will hold an address, RSSI and TXP value corresponding to its RR-master and RR-slave accordingly. A relay's RR-slave and RR-master tag cannot have the same node address, and each relay may only hold one of each. A relay cannot be anyone's neighbour. Instead, the relays will have a special type of relationship: a RR-master will be able to provide a minimum TXP value for its RR-slave so that the two will have a sufficient radio signal strength between them. A detailed description of this Relay-to-Relay relationship and messaging scheme will be given in Section 8. With this added logic, all relays have a guarantee of having an acceptable signal strength between at least one other relay in the mesh network, while also being able to maintain a regulated connection with the Edge nodes in its neighbourhood.

Additionally, a *neighbour tag* was added to the application logic to make sure that only one relay node is controlling each Edge node. Each node holds its own 16-bit integer which is initialized to `0x0000` and will be overwritten by the master relay (neighbourhood master, not RR-master) node's address.

7.1.4 Dynamic change of master

Testing with the embedded algorithm on zone 1 (Figure 26), revealed that the nodes in the network have the need to dynamically change their master Relay. It was, naively enough, assumed that the nodes would be requested to join the neighbourhood of the Relay that reached out the first and that this would be the Relay that laid the closest in physical range.

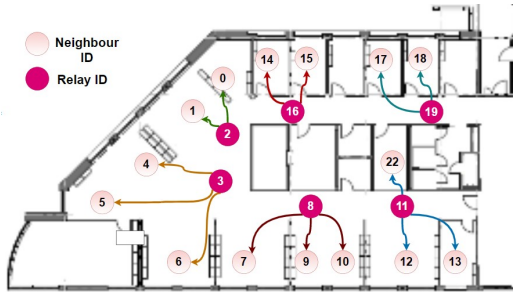


Figure 28: Zone 1: Network clusters with dynamic master change. The coloured arrows represent the established Relay-Edge neighbour relationships.

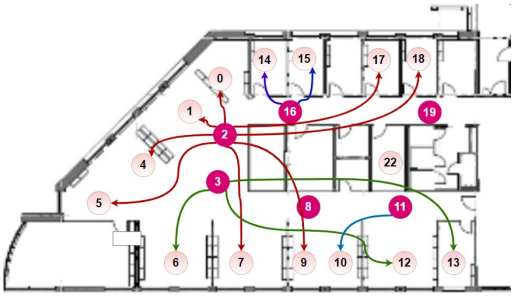


Figure 29: Zone 1: Network clusters without dynamic master change. The coloured arrows represent the established Relay-Edge neighbour relationships.

Figure 29 displays how the neighbourhood clusters were formed after the algorithm had been running in isolation, meaning no other tests or applications are running on the nodes, over a period of a few minutes. This "malformed cluster" configuration resulted in a noisy neighbourhood where a relay claimed a far-away node as its neighbour and would start "shouting" with an increased TXP to make sure that this particular neighbour node could receive its messages with an acceptable RSSI. When this happens, the relay will disturb the communication of its other neighbours. As the network nodes all are initialized with a TXP of 0 dBm, a relay can in practice, although with poor reliability and signal strength, reach nodes that are as far as 15-20 meters away judging from the results in Figure 29.

To avoid this issue, a new logic is implemented: if a node receives a GET-request (which contains the source node TXP), the Edge node can check if this new Relay can provide it with a lower RSSI while also transmitting with a lower or equal TXP than its current master Relay. If this is the case, the Edge node may change its master. A timeout feature was also added so that if an Edge node does not receive a GET message from its master within 20 seconds it will find a better

match. This was necessary because a Relay with a low TXP value might, on rare occasions, reach an Edge node with its GET message, making it a neighbour while no other Relays can "compete" with this low level of TXP. However, the ongoing GET-STATUS message exchange might suffer from poor reliability or might not exist at all.

7.1.5 Mesh traffic scanning

An important input to the algorithm is the total mesh traffic originating from a Relay's neighbourhood nodes being picked up by the Relay every N second. If the traffic gets too high, the Relay will start decreasing the NTC values for its neighbouring nodes. The incoming message could either be addressed directly to the relay itself or as a message to be relayed on the network. The "scanning" was implemented as a "hook"-function that triggers the application layer from the network layer *after* each time an incoming message has been processed (making sure that the process itself is not waiting for this function and adding to the processing time). The function, defined in the application layer, updates a counter that is incremented each time the message's source address matches one of the neighbouring nodes in the relay table. However, the network layer performs a cache-check before encryption of the message, revealing transmit context info such as source address. This is the address that the application layer needs to check in order to update its counter. The cache check dismisses any incoming message with a previously seen sequence number. Thus, the traffic that a transmitting node with a non-zero NTC is generating is not fully accounted for but it is still affecting the network. Luckily, the solution was quite simple. The Relay already knows the NTC of each of its neighbouring nodes, so any number of incoming messages from either of these will have to be multiplied with their respective relay table NTC value +1 (since NTC is *additional* message transmits).

7.1.6 "Sliding window" regulation

In the emulator, the SRR was calculated with every STATUS message received. The value was simply a ratio between the number of outgoing GET messages sent from the Relay and the number of incoming STATUS messages received for each Edge node. This gave the smooth regulation appearance as seen in Section 6.1. However, this way of presenting the SRR is not optimal as any number of lost messages would become irrelevant for the regulation over time ($t \to \infty$). Therefore, the calculation of the SRR was changed to being the average of received / not received STATUS messages within a current "window". Figure 30 shows an example of how the SRR is calculated with the new technique, here with a window of 4.

4								
5	TX	1	1	1	1	1	1	1
6								
7	RX	1	0	1	1	0	0	0
8								
9	SRR	1			0.75			
10						0.5		
11							0.5	
12								0.25

Figure 30: The figure shows the "sliding window" regulation technique with a window of 4 units. The TX row holds the outgoing GET messages, while the RX row holds the current received status for each message accordingly. The SRR is calculated based on the 4 last entries.

7.2 Embedded execution demo

The terminal snapshot in Figure 31 displays some of the various real-time logs that are generated during the run-time of the algorithm, similar to those with the data emulator (Section 6.1). These Ethernet logs are the input to a Python script running on the computer. The script will continuously scan for log messages, identifying and collecting the data, processing and finally plotting the data to make the results easier to read. This is important because the logs will become too many to make use of when the algorithm is running on the full-scale 100-node network.

Figure 33-34 presents the results for two Edge nodes sharing the same Relay master after an isolated run of the algorithm, meaning that no other application or test messages are being sent in the network. The algorithm was run over the course of approximately 30 minutes (or 2000 seconds). The set-point for the algorithm was 0.8 Status Revived Rate (SRR). The data was collected from the log output, like those as shown in the terminal window, from Relay node 16, see Figure 26. Figure 32 shows the result for the master Relay node 16. Overall, the demo reveals that the embedded implemented algorithm works as intended and achieves the goal of stabilizing the SRR for the Edge nodes around their set-point. In this demo, the SRR was sampled every 10th STATUS message received and with a *regulation window* of only 10 messages. Thus, the graphs only show changes in 10% bulks, giving them their uneven appearance like a saw-tooth as seen in the figures.

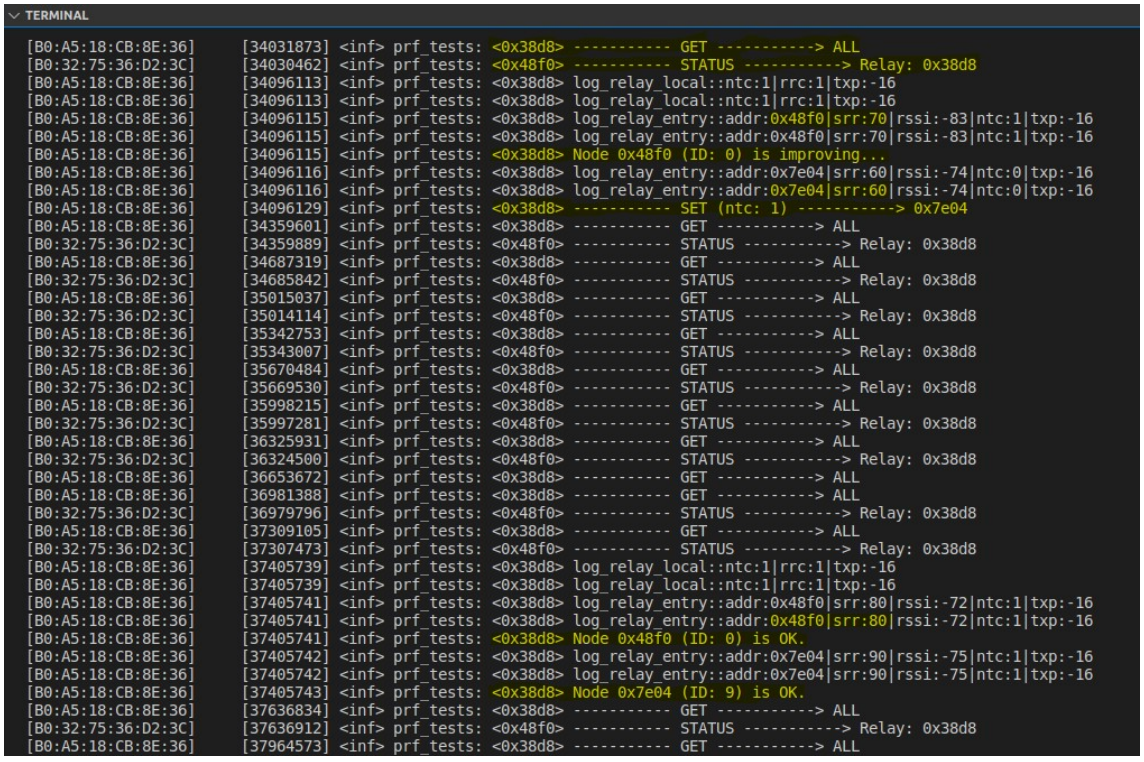


Figure 31: Embedded algorithm execution demo showing some of the Ethernet logs generated during run-time. The relay (address: `0x38d8`) is here seen sending out GET and SET messages, receiving STATUS messages, as well as keeping track of the neighbouring node's Status Receive Rate (SRR).

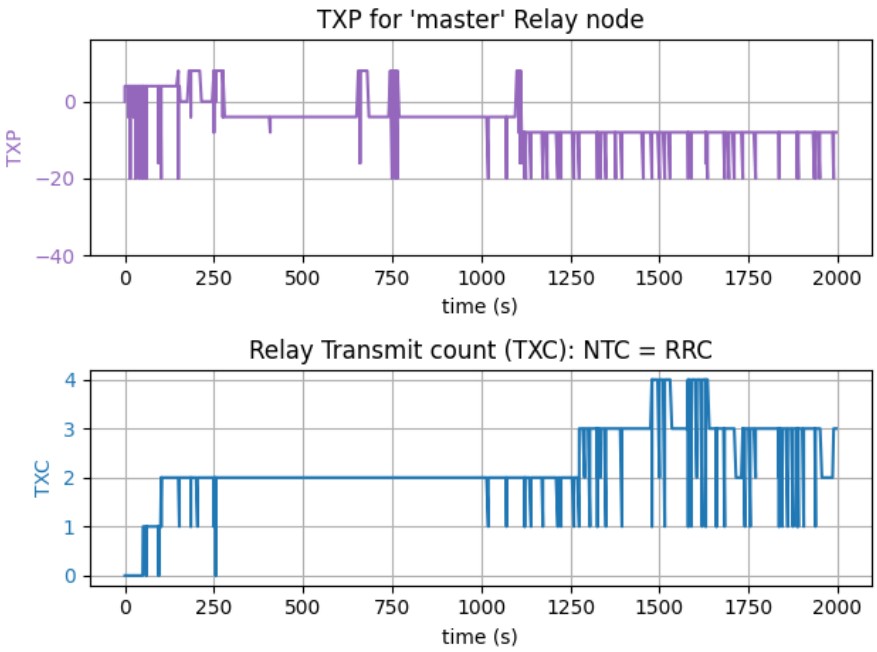


Figure 32: Results for network node 16 acting as Relay after an isolated run of the algorithm on a 22-node sub-portion of the large scale network. Average values for the Relay over the time period: TXP: -10 dBm, TXC: 2

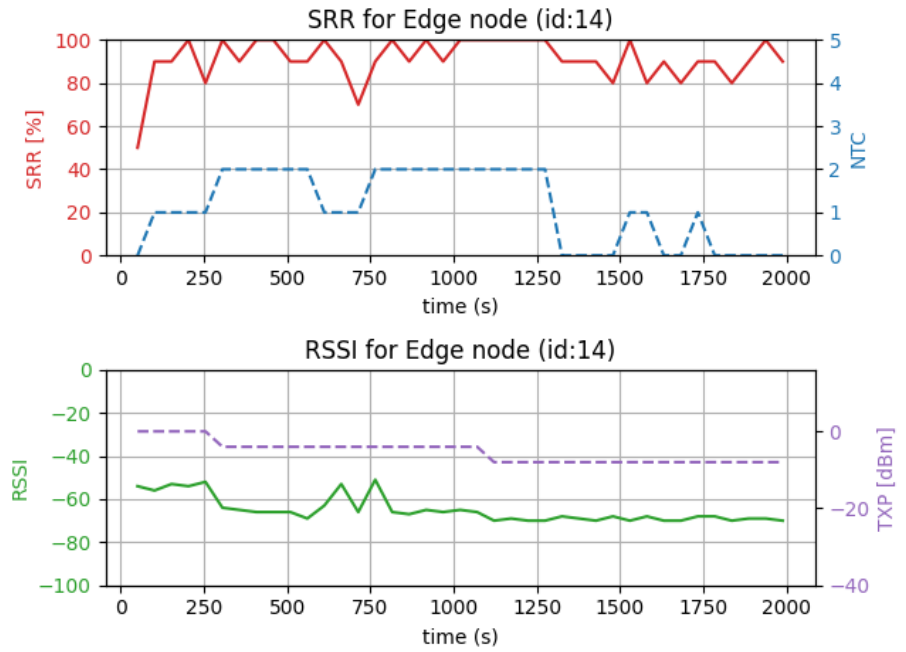


Figure 33: Results for network node 14, sharing master Relay 16 with Edge node 15, after an isolated run of the algorithm on a 22-node sub-portion of the large-scale network.

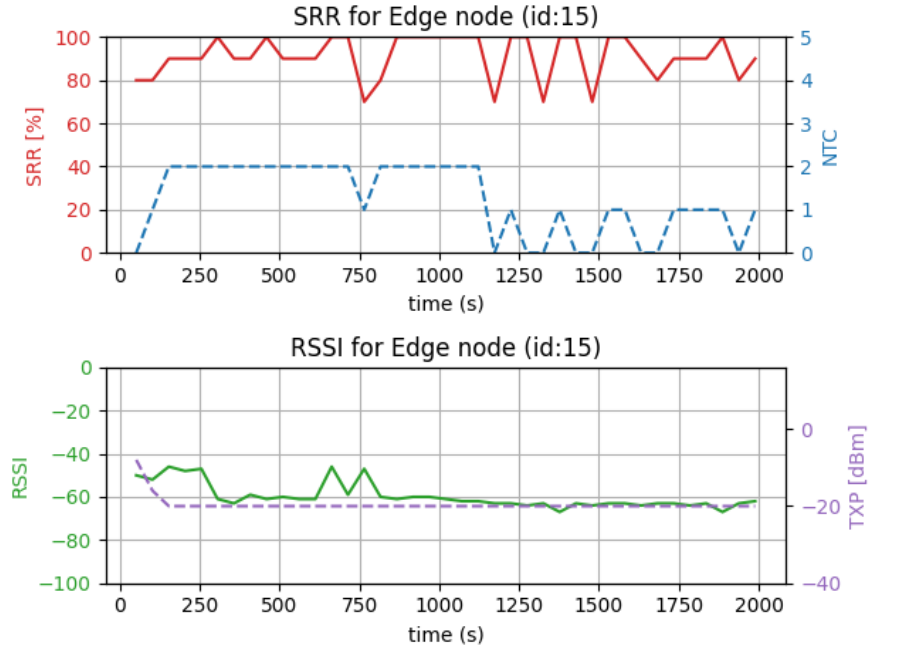


Figure 34: Results for network node 15, sharing master Relay 16 with Edge node 14, after an isolated run of the algorithm on a 22-node sub-portion of the large-scale network.

8 The algorithm: Design

This section describes the *design*, meaning how the specification criteria in Section 4.1 is implemented in detail. At the end of this chapter, a review of the specification criteria is conducted to verify that all points have been included in the design.

The design contains two parts: a BT mesh model, and an application. The model describes the message-based server-client communication, and the application describe the algorithm logic processing the input from the received messages and generating an output forming the regulation loop as illustrated in Figure 12. The final design which is presented in this section is the result of an iterative process adjusting the logic to improve the speed and performance of the parameter regulation using the data emulator (Section 6) and finding solutions with the limitations of embedded programming when implementing the final code (Section 7).

8.1 Model layer: Client and Server models

The algorithm will utilize a custom-made BT mesh model for all communication. The model will have both server and client functionality. The client will operate with the requesting messages GET and SET, while the server will respond with a STATUS message. Table 6 lists the different message types and their parameters. The message types are identified by their unique operation code (OP code). All messages are within the size of a single BLE (Bluetooth Low Energy) packet, allowing an 11-byte payload [29], and are therefore all sent unsegmented. All messages are sent unacknowledged, meaning that the client will not wait for the server to respond.

Model message types	**OP code**	**Parameters**	**Type**

STATUS	0xDA	`int8_t rssi`
		`unit8_t ntc`
		`int8_t tx_txp`
		`bool is_nb`
		`bool is_my_rr_slave`
			Unacknowledged
			Unsegmented

SET	0xDB	`uint8_t ntc` `int8_t txp`	Unacknowledged Unsegmented
GET	0xDC	`int8_t tx_txp`	Unacknowledged Unsegmented

Table 6: The algorithm will utilise a custom model with the three message types: STATUS, SET and GET. The table lists the contents of each message type.

8.1.1 Client design

The client part of the model will act upon receiving STATUS messages, as described in Table 6, from server nodes:

1. Upon receiving a STATUS (`0xDA`) message:
    1.1 Extract the message contents: `rssi`, `ntc`, `txp`, `is_nb` and `is_my_rr_slave`, and from header: `addr` and `rx_rssi`
    1.2 Call client STATUS model handler, see "Client model handler: STATUS handler" in Section 8.2.6, passing all extracted contents.

The client can be called upon for sending messages. The client has the following functions for sending messages:

```
/** Broadcasts a mesh optimization algorithm GET message. */
int bt_mesh_opt_alg_send_get(struct bt_mesh_opt_alg_mod *mod,
```

```
            struct bt_mesh_msg_ctx *ctx,
            struct bt_mesh_opt_alg_get_msg *msg);

/** Sends a mesh optimization algorithm SET message
 * to request parameter change at the target server. */
int bt_mesh_opt_alg_send_set(struct bt_mesh_opt_alg_mod *mod,
            struct bt_mesh_msg_ctx *ctx,
            const struct bt_mesh_opt_alg_set_msg *msg);
```

Each function takes the model, message sending context (destination address, TTL etc.) and the message to be sent as input. The client will prepare and then forward the message to the lower layers of the stack (Section 2.1).

8.1.2 Server design

The server part of the model is responsible for responding to the client's GET and SET requests. The messages are described in Table 6. In detail, the server shall do the following:

1. Upon receiving a GET (`0xDC`) message:
    1.1 Extract `addr` and `rx_rssi` from header.
    1.2 Extract message contents: `txp`.
    1.3 Call server GET model handler, see "Server model handlers: GET handler" in Section 8.2.4, passing all extracted contents.
2. Upon receiving a SET (`0xDB`) message:
    2.1 Extract `addr` from header
    2.2 Extract message contents: `txp` and `ntc`.
    2.3 Call server SET model handler, see "Server model handlers: SET handler" in Section 8.2.5, passing all extracted contents.

The server will respond with a STATUS message when receiving a GET message. The server has the following function for sending the message:

```
/** Sends a mesh optimization algorithm STATUS message
* back to the requesting client; rsp to GET.*/
int bt_mesh_opt_alg_send_status(struct bt_mesh_opt_alg_mod *mod,
            struct bt_mesh_msg_ctx *ctx,
            struct bt_mesh_opt_alg_status_msg *msg);
```

The function takes the model, message sending context and the message to be sent as input. The server will then forward the message to the lower layers of the stack.

8.2 Application layer: Node roles and handlers

The application layer lies on top of the model layer and is handling the algorithm-specific logic, see Figure 35. This logic is divided into different *handlers* which are called either periodically from a work queue or in case of an event, like when receiving a message. Some of these handlers are used only by some nodes according to their *roles* and behaviour. The upcoming subsections will describe the various modules, handlers, node-specific behaviour and necessary structures that make up the design of the algorithm.

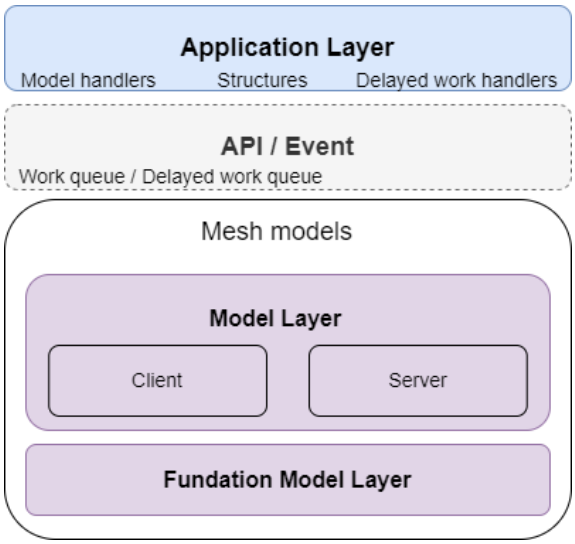


Figure 35: The application layer lies on top of the model layer (2.1). Model handlers, delayed work handlers and application-specific structures are all defined here.

8.2.1 Node roles and structures

The algorithm defines two types of *roles* for the nodes: Relay nodes and Edge nodes (note the capital letter denoting the Relay *role*, not the relay *state* as previously described in Section 2.3.1). Nodes in the network which have enabled the relay feature are considered Relay nodes, and the rest are Edge nodes. A Relay node will act as both a client and a server, and an Edge node will only act as a server.

All Edge nodes must hold a variable of the type `nb_tag`, or *neighbour tag*, containing a 16-bit address initialized to `0x0000` (non-valid BTM address), a 8-bit RSSI and a TXP value containing its current Relay master's info, see Table 7. Relay nodes will have two variables, `my_rr_master` and a `my_rr_slave`, similar to the *neighbour tag*, defining its current Relay-to-Relay relationship which will be further described in Section 8.2.2. The structure `relay_status` holds information about whether or not the node is currently acting as a relay, and its relay re-transmit state (RRC and RRI).

Node identity structures	**Parameters**	**Host**

`nb_tag`	`uint16_t addr`
	`int8_t txp`
	`int8_t rssi`
		Edge

`my_rr_master`	`uint16_t addr`
	`int8_t txp`
	`int8_t rssi`
		Relay

`my_rr_slave`	`uint16_t addr`
	`int8_t txp`
	`int8_t rssi`
		Relay

`relay_status`	`bool relay_feat_enabled`
	`uint8_t retrans_interval`
	`uin8_t retrans_count`
		All

Table 7: The algorithm will utilise three types of identity tags for the nodes. Some of the tags are role-specific (Edge or Relay only).

A Relay will maintain a table of its associated Edge nodes, updating their info with each STATUS message received. The `relay_table`, will hold 100 (or the number of nodes in the network) entries of the data structure as described in Table 8:

uint16_t address	struct parameters			
	int8_t rssi	uint8_t ntc	int8_t txp	bool is_nb
	int8_t srr_q[SRR_Q_SIZE]	float srr	float prev_srr	

Table 8: Required data for each Edge node entry in `relay_table` maintained by the Relay node.

Note: The `srr_q[]` must function as a First In First Out (FIFO) container, and should be able to hold at least 10 (8-bit) integers, all initialized to 1. The container will hold the received status for the nth last messages. Specific usage of this will be described in Sections 8.2.3 and 8.2.6.

The Relay must have the following defined **variables**: `uint32_t total_mesh_traffic` - a count of all scanned mesh traffic originating from neighbor nodes, and `uint8_t msg_tx_count` - relay NTC and RRC will follow this value. All variables should be initialized to 0.

The Relay must have the following defined global **constants**: `SRR_SET_POINT` (0-100) - the SRR regulation set-point, `RSSI_MIN`, `RSSI_MAX`, `NTC_MAX`, `NTC_MIN`, `RRC_MAX`, `RRC_MIN`, `TXP_VALID_VALUES[]` - as specified by nRF Connect Software development kit (nCS)[3], `RELAY_GET_PER` - how often the Relay GET message is published, `RELAY_MAX_TRAFFIC` - the threshold of which the total traffic from neighbouring nodes must be reduced, `RELAY_TRAFFIC_CHECK_PER` - how often the Relay will check the current traffic state and `NTC_DEDUCT` - the percentage of the total neighbourhood NTC must be reduced after the traffic threshold is exceeded.

The Relay will continuously scan for incoming mesh traffic. All traffic originating from its neighbours will be counted and saved to `total_mesh_traffic`.

8.2.2 Messaging: Regulation and forming of node relationships

A Relay will periodically broadcast a GET message and use the contents of the STATUS messages that it receives in response as input for the regulation of NTC, RRC and TXP for its neighbouring nodes (or "slaves") and itself. The Relay will periodically check the SRR for each neighbour and evaluate the RSSI with each STATUS message received. The Relay may send a SET message to request a specific node in its neighbourhood to change its parameters. The mesh packets originating from neighbouring nodes per time will be tracked by the Relay. If the threshold for acceptable traffic is exceeded, then a reduction of the total neighbourhood NTC is initiated. A message exchange example between a master Relay node and a neighbouring Edge node is shown in Figure 37. More details and specifics of the regulation is given in the handler descriptions in Section 8.2.3 - 8.2.6.

Edge nodes will choose their master Relay to regulate them based on the GET messages they receive. The Edge node will evaluate the received-RSSI extracted from the GET message header and message payload TXP (the TXP value used by the transmitting Relay) and choose the Relay with the most efficient RSSI to TXP relationship (a high RSSI to a low TXP is preferred). This master evaluation is a continuous process and Edge nodes may change their master at any time, see Figure 36. This way, *neighbourhoods* of Edge nodes will form, each with their chosen master Relay, or *neighbourhood master*, to regulate its neighbours. A timer is started at the Edge node when a master is accepted and then cancelled and restarted for every GET-message received from the master Relay. This is to make sure that the Edge node always has a responsive master.

When a Relay receives a GET message from another Relay, it will not accept the requesting Relay as its neighbourhood master (`is_nb = False`). Relays cannot be part of another Relay's neighbourhood. However, it may choose the requesting Relay as its Relay-to-Relay(RR)-master, or change from its current RR-master to this requesting Relay if it provides a better RSSI to TXP ratio. If so, the relay will update the parameters of its `my_RR_master` tag. The relay will respond to its RR-master with a STATUS message tagged with a `is_my_RR_slave` flag (binary value, true or false), see Figure 38. The receiving relay will check the `is_my_RR_slave` flag. If true, it will save the STATUS message's address, TXP and RSSI value to its `my_RR_slave` tag. If the RSSI is out of range for a STATUS message sent from a node matching the relay's RR-slave address, then it will adjust its `my_RR_slave` tag TXP value, moving either up or down the `TXP_VALID_VALUES[]`. For

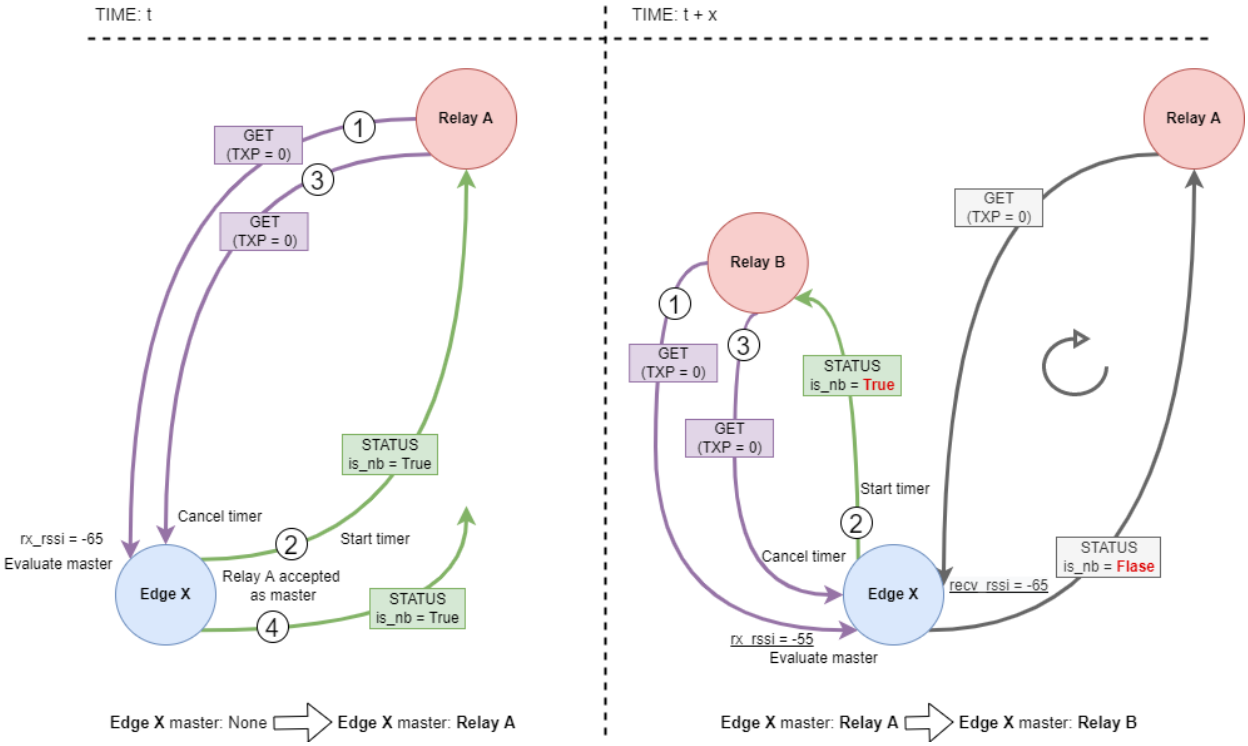


Figure 36: An Edge node may change its master Relay if the requesting Relay provides a better match. In the figure above, the Edge node X first accepts Relay A as its master (lhs) and then switches to Relay B as this relay offers a higher RSSI with the same TXP as it is closer in physical range (rhs).

example, it the received TXP value was 0 dBm, and the RSSI is too high, then set `my_RR_slave.txp` to -4 dBm.

A Relay should at all times be configured to use its *optimally regulated transmit parameters* (TXP, NTC, RRC = NTC) so that when relaying messages in the network, these are the transmit parameters that are used. These optimal parameters must be set to match the highest NTC value of all its Edge node neighbours (`relay_table[n].params.ntc`), and the TXP that is the highest of all its Edge node neighbours (`relay_table[n].params.txp`), RR-slave (`my_rr_slave.txp`) and RR-master (`my_rr_master.txp`). However, an exception is made when sending a SET message. Before sending the SET message, the Relay will set its transmit parameters according to the target node's values as they are listed in the `relay_table[target_index].params`. After the message is sent, the Relay should immediately change back to its optimally regulated transmit parameters.

8.2.3 Delayed work items

Delayed work is a thread-based work queue defined in Zephyr Real-Time Operating System (RTOS) [20]. When a *delayable work item* is scheduled, its callback-function (`_cb`) will be executed at the host after a defined *timeout*, unless the delayable is cancelled before the timeout is reached. A few delayed works are used in the embedded implementation of the algorithm, see Table 9.

Delayed work	Execution	Timeout	Host
`opt_alg_send_get_work`	Periodic	`RELAY_GET_PER`	Client
`opt_alg_check_traffic_work`	Periodic	3 * `RELAY_GET_PER`	Client
`opt_alg_check_srr_work`	Periodic	10 * `RELAY_GET_PER`	Client
`opt_alg_master_timeout_work`	On-demand	4 * `RELAY_GET_PER`	Client and Server

Table 9: Delayed work items in the embedded algorithm implementation.

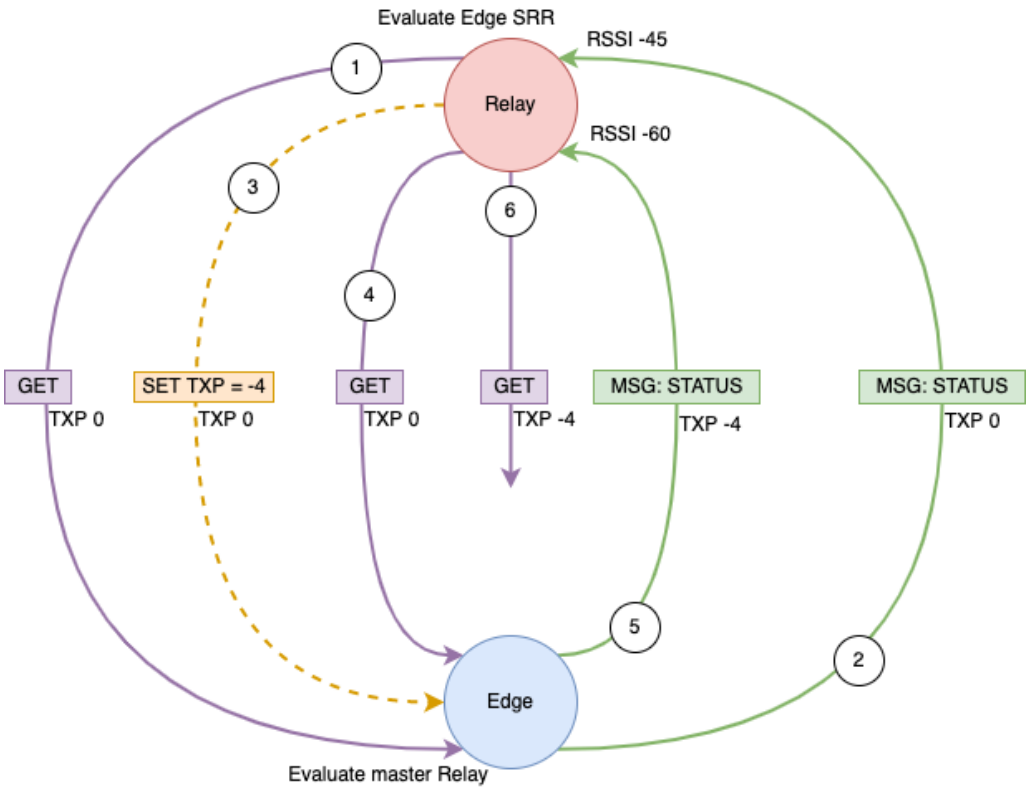


Figure 37: Message exchange example between a Relay node and a neighbouring Edge node. The Relay node periodically broadcasts a GET-message to which it expects a STATUS-message in return. Based on the STATUS-message contents, the Relay keeps track of the Edge nodes Status Received Rate (SRR) and may request the Edge node to change its local parameters such as NTC or TXP by unicasting a SET-message. The Relay node may also change its own local parameters.

The `opt_alg_send_get_work` callback, `opt_alg_send_get_work_cb`, is responsible for broadcasting the periodic GET message for the Relay. The callback have the following execution:

1. Initialize a GET-message.
2. Push 0 to each node's `srr_q` (Table 8) in the `relay_table`.
3. Get local system's TXP and save value to GET-message.
4. Set transmit parameters (NTC, RRC = NTC and TXP) according to the highest value of each parameter of the neighbouring nodes listed in the `relay_table`.
5. Broadcast (using address `0xFFFF`) GET-message with TTL set to 0.
6. Reschedule `opt_alg_send_get_work`.

The `opt_alg_check_traffic_work` callback, `opt_alg_check_traffic_work_cb`, is responsible for checking the current neighbourhood traffic and to react by reducing the total neighbourhood NTC count if the traffic exceeds a pre-defined limit. The callback has the following execution:

1. Initialize a SET-message.
2. If the total amount of mesh messages originating from the client Relay's neighbours (`total_mesh_traffic`) has exceeded the maximum amount of allowed traffic (`RELAY_MAX_TRAFFIC`), then:

   2.1 Starting with the node with the highest SRR, send a SET-message (8.1.1) with a deducted NTC value to the node. The TTL may be set to higher than 0.

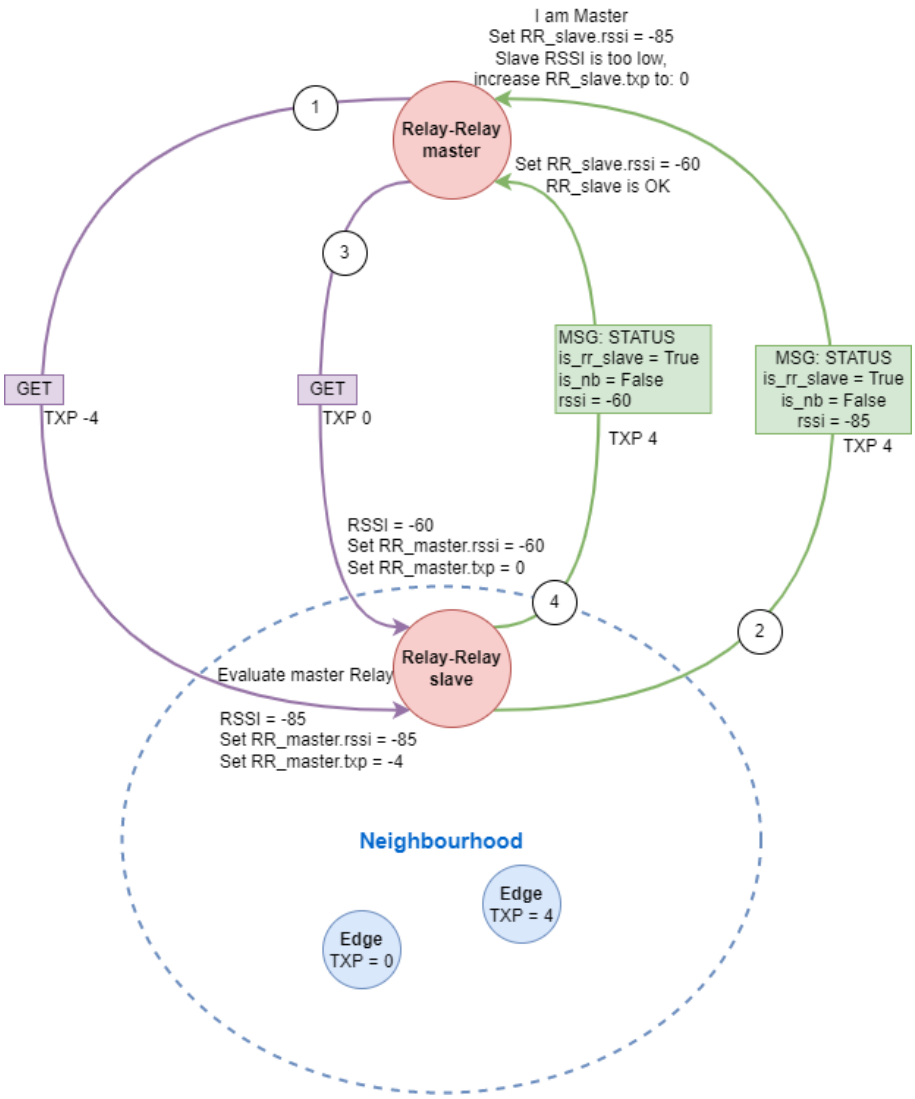


Figure 38: A relay will update its TXP to be the highest of all TXP values in: its relay table (neighbouring Edge nodes), its Relay-Relay slave and its Relay-Relay master. The figure shows how the RR-slave finds its highest TXP value with one of its neighbouring Edge nodes. The RR-master's minimum required value is then not used.

2.2 The same node can not be deducted more than once.

2.3 Continue the procedure until the sum of the neighbourhoods NTC is reduced with at least `NTC_DEDUCT`%.

3. Set the mesh traffic count to 0.

4. Reschedule `opt_alg_check_traffic_work` with its set timeout as given in Table 9.

The `opt_alg_check_srr_work` callback, `opt_alg_check_srr_work_cb`, is responsible for checking all neighbouring Edge nodes' SRR and increasing the nodes' transmit counts if necessary. The callback has the following execution:

1. Initialize a SET-message.

2. Set `prev_srr` euqal to `srr`.

3. Calculate the average value of the `srr_q` for every entry, for which the `is_nb` is True, in the `relay_table` (Table 8), and save to `srr`.

4. For all the neighbour nodes in the `relay_table`:

   4.1 If the node `srr` is 0, break the neighbour relationship by setting `is_nb` to False.

   4.2 If the node `srr` is greater or equal to the `SRR_SET_POINT` (for example 0.8), do nothing.

   4.3 If the node `srr` is less than the regulation set-point and has not improved since the last SRR-check (`prev_srr`), then send a SET message (8.1.1) to the node requesting it to increase its NTC value if not already at maximum capacity.

5. Reschedule `opt_alg_check_srr_work` with its set timeout as given in Table 9.

The `opt_alg_master_timeout_work` callback, `opt_alg_master_timeout_work_cb`, is responsible for resetting the nodes current master if communication has been lost. The callback has the following execution:

1. If calling node (host) is an Edge node:

   1.1 Set the `nb` master address to `0x0000` (blank address).

   1.2 Set the `nb` master rssi to NULL (no value / invalid).

   1.3 Set the `nb` master TXP to NULL.

2. Else if calling node (host) is a Relay:

   2.1 Set the `my_rr_master` address to `0x0000` (blank address).

   2.2 Set the `my_rr_master` RSSI to NULL.

   2.3 Set the `my_rr_master` TXP to NULL.

8.2.4 Server model handlers: GET handler

The node will respond with a STATUS-message upon receiving the periodic GET-message. When receiving the NB-message ("neighbour message") the node will check if the request came from its "master" relay's address or if it has no master before updating its *neighbour tag*.

Upon receiving a GET message:

1. If `addr` equals the local (own) address, do nothing and return.

2. Initialize a STATUS message.

3. If `relay_status.relay_feat_enabled` Table 7 is False:

   3.1 If `addr` matches `nb_tag.addr`, then update `nb_tag` parameters with corresponding extracted contents (`addr`, `rx_rssi` and `txp`), and then cancel and reschedule `opt_alg_master_timeout_work`.

   3.2 Else if `nb_tag.addr` is blank (`0x0000`), then update `nb_tag` parameters with corresponding extracted contents. Then, reset network transmit state (NTC and NTI) to default values and reschedule `opt_alg_master_timeout_work`.

   3.3 Else, if the requesting Relay can provide a better RSSI (`rx_rssi`) to TXP ratio (with some criteria of minimal improvement), then update `nb_tag` parameters with corresponding message contents, and cancel, and reschedule `opt_alg_master_timeout_work`.

4. Else if `relay_status.relay_feat_enabled` is True:

   4.1 If `addr` matches `my_rr_master.addr`, then update `my_rr_master` parameters with corresponding extracted contents, and cancel and reschedule `opt_alg_master_timeout_work`.

   4.2 Else if `my_rr_master.addr` is blank and message `addr` does not equal `my_rr_slave.addr`, then update `my_rr_master` parameters with corresponding extracted contents. Then, reset network transmit state to default values and reschedule `opt_alg_master_timeout_work`.

4.3 Else, if the requesting Relay can provide a better RSSI (`rx_rssi`, received from the header / message context) to TXP ratio, then update `my_rr_master` parameters with corresponding extracted contents, and cancel and reschedule `opt_alg_master_timeout_work`.

5. Fetch current system values: *TXP*, *NTC* and save to STATUS message.
6. If `addr` matches `nb_tag.addr`, set STATUS message `is_nb` to True. Else, set STATUS message `is_nb` to False.
7. If `addr` matches `my_rr_master.addr`, set STATUS message `is_my_rr_slave` to True. Else, set STATUS message `is_my_rr_slave` to False.
8. Send STATUS message (8.1.2) with destination address set to `addr`

8.2.5 Server model handlers: SET handler

Upon receiving a SET-message, the node will update its parameters according to the request, but only if sent by its master Relay.

Upon receiving a SET message:

1. If `addr` equals `nb_tag.addr`:
   1.1 If `txp` is not None: Update system TXP value accordingly and return.
   1.2 If `ntc` is not None: Update system NTC value accordingly and return.
   1.3 Else, do nothing. Only the node's master may request parameter change.

8.2.6 Client model handler: STATUS handler

Upon receiving a STATUS message, the client will:

1. Initialize a SET-message
2. Allocate or fetch existing node index (`n_idx`) for the origin node in the `relay_table`.
3. If message `is_nb` equals True but node was not a neighbour with last STATUS received, then reinitialize (clear) entry corresponding to index in `relay_table`.
4. Update `relay_table` entry (`relay_table[n_idx]`) with the STATUS message contents: `ntc`, `txp` and `is_nb`, and from header content: set `rssi` equal to `rx_rssi`.
5. Set first element in `srr_q` to 1.
6. If `is_my_rr_slave` equals True:
   6.1 If message `addr` equals `my_rr_master.addr`, skip all subsequent sub-points and continue.
   6.2 If `my_rr_slave.txp` is greater then message `txp`, skip next sub-point and continue.
   6.3 Update the `my_rr_slave` with the message contents `addr`, `txp` and `rssi`.
   6.4 If the `rx_rssi` received from the header (message context) is out of range [`RSSI_MIN`, `RSSI_MAX`], then update `my_rr_slave.txp` to the next lower or higher TXP value according to the `TXP_VALID_VALUES`. If the node cannot be adjusted any further, then do nothing.
7. Else if `is_my_rr_slave` equals False:
   7.1 If `my_rr_slave.addr` equals message `addr`, then reset `my_rr_slave` structure values to NULL.
8. If `is_nb` equals True, continue to next point, else do nothing and return.
9. If the `rx_rssi` is out of range [`RSSI_MIN`, `RSSI_MAX`], then send a SET message to the origin node with the next lower or higherTXP value according to the `TXP_VALID_VALUES`. If the node cannot be adjusted any further, then do nothing.

8.3 Specification review

Table 10 gives a brief summary of the design solutions answering to the specification in Section 4.1. The leftmost column is referring to the numbered specification criteria list from the same section.

Specification listing	Design reference	Note
1.	8.2.3 `opt_alg_send_get_work`	A GET message is broadcasted periodically
2.	8.2.3 `opt_alg_send_get_work` 8.2.6 Status handler	TTL is set to 0 for all GET messages sent to ensure direct transmission with no hops. A neighbour tag is used to keep track of neighbouring nodes.
3.	8.2.6 Status handler	A SET message is sent if the RSSI is out of range.
4.	8.2.3 `opt_alg_send_get_work` 8.2.6 Status handler 8.2.3 `opt_alg_check_srr_work`	The `srr_q[]` is updated with 0's and 1's with each sent GET and received STATUS message respectively. The SRR is checked periodically and a SET message is sent to increase the node NTC if necessary.
5.	8.2.3 `opt_alg_check_traffic_work`	Traffic is periodically checked and total neighbourhood NTC is decreased if the traffic limit is exceeded. Total traffic from the neighbourhood can be tracked by implementing a "hook"-function from the network layer.
6.	8.2.2 Messaging 8.2.1 Node roles and structures	A special form of Relay-to-Relay (RR) relationship is established instead of letting Relays be neighbours. The RR connection is ensuring a sufficient radio signal strength between the Relays.

Table 10: Design specification review.

9 Performance testing: Collected data and results

In this chapter, the logic and procedure of the test that will be used to determine the network performance are presented, along with the results for different test runs with varying parameters. The performance is measured with and without using the algorithm (static values). The tests are run on the large-scale network as presented in Section 5. The optimized parameter set found in the report Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks [2], see Table 4 in Section 3, will be used as one of the static competitors to the algorithm.

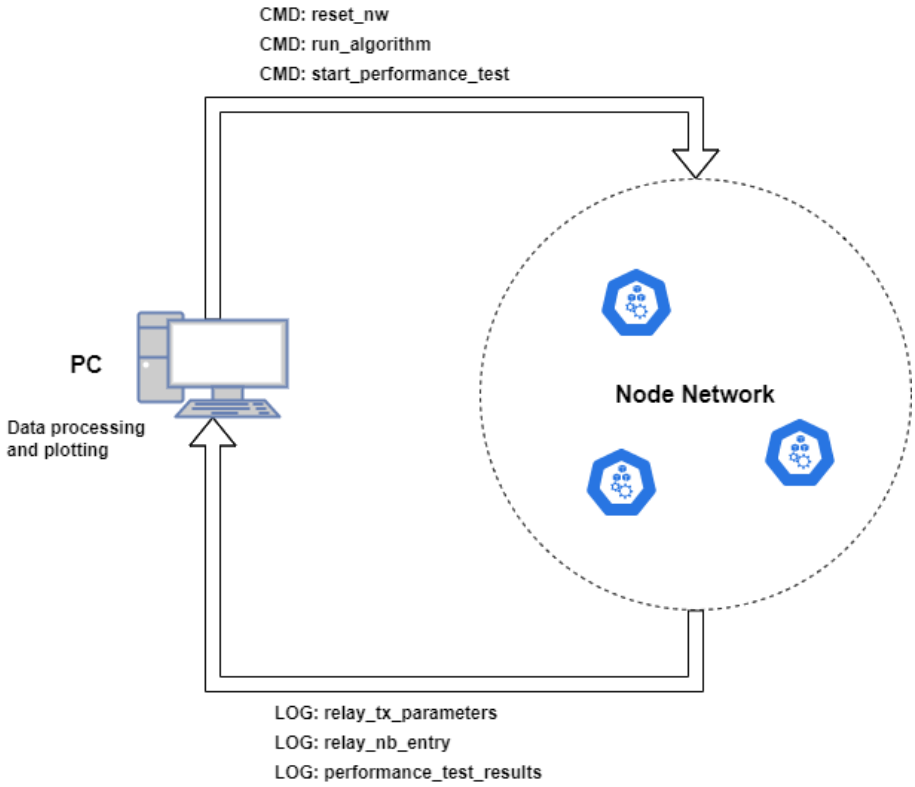


Figure 39: Communication between the remote PC and the network during testing

The test code itself is owned by Nordic Semiconductor and is not publicly available. Figure 41 and 42 give a high level description of the test flow and the messaging logic. The custom mesh model and application which make up the algorithm, as described in Section 4, are added as an extension to the already existing performance test.

Figure 40 shows how the network nodes were configured during the performance testing. Only the nodes in office corridors are configured as relays (marked in pink on the map). The performance test designates a single node in the network as a *tester node* (marked in blue on the map). This tester node will send 100 messages to each of the remaining network nodes, one at a time, measuring the Round-Trip Latency (RTL) of each acknowledged message it receives. The entire network is time-synchronized, giving the nodes the ability to log the time of arrival for every message they receive. This way, the Single-Trip Latency (STL) is also recorded. The results, containing STL times, transmit power (TXP) and Network Transmit Count (NTC) values are logged over Ethernet in real-time. Once received at the test computer, the data is analyzed in a Python script. Here, the average-, median-, minimum-, and maximum latency values, average TXP and NTC values are calculated, along with the total amount of received messages. The results are then processed in a plotting script. See Figure 39 for a view of the information flow.

Figure 40: Node configurations as used during performance testing. The tester node (blue) will send 100 messages to each of the remaining nodes in the network, Relays (dark pink) as well as Edge nodes (light pink).

If nothing else is stated, the tests are run using the initial conditions as given in Table 11. 100 messages will be sent to each target with each test run, adding up to a total of 9900 messages. The messages are transmitted in bursts of 10 at a time. The algorithm was configured as listed in Table 12. As the table shows, an alternative version of the algorithm was tested in addition to the default configuration which follows the specification and design from Chapters 4.1 and 8. This alternative version has a different NTC/RRC relationship than what is specified in the algorithm specification. The results for all test-runs are given in Table 13 and Figure 43. When running the performance test with the algorithm, the algorithm is set to run for two minutes before the test starts and then continues to run throughout the duration of the test. This is to ensure that the transmit parameters (NTC, RRC and TXP) will be tuned from start to finish during testing.

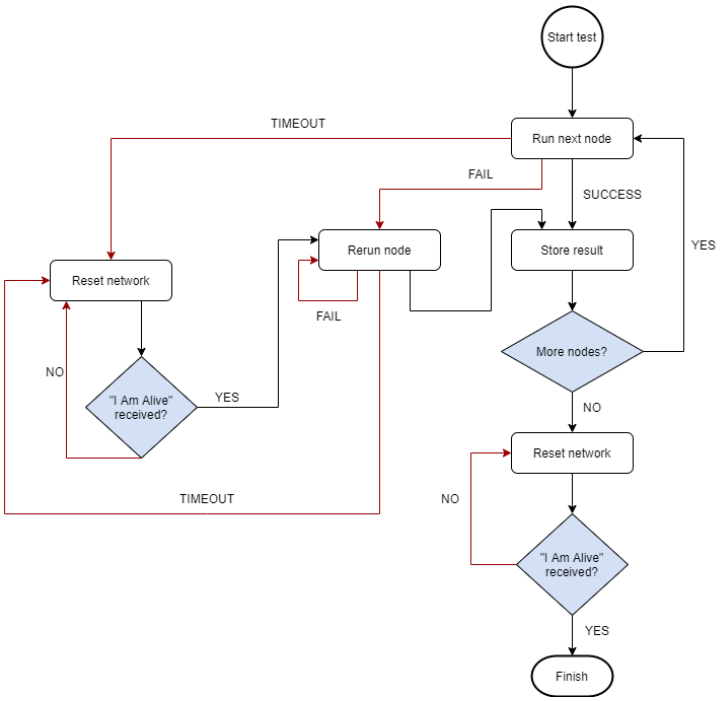


Figure 41: The diagram shows the general flow of the Python scripts executing the performance test

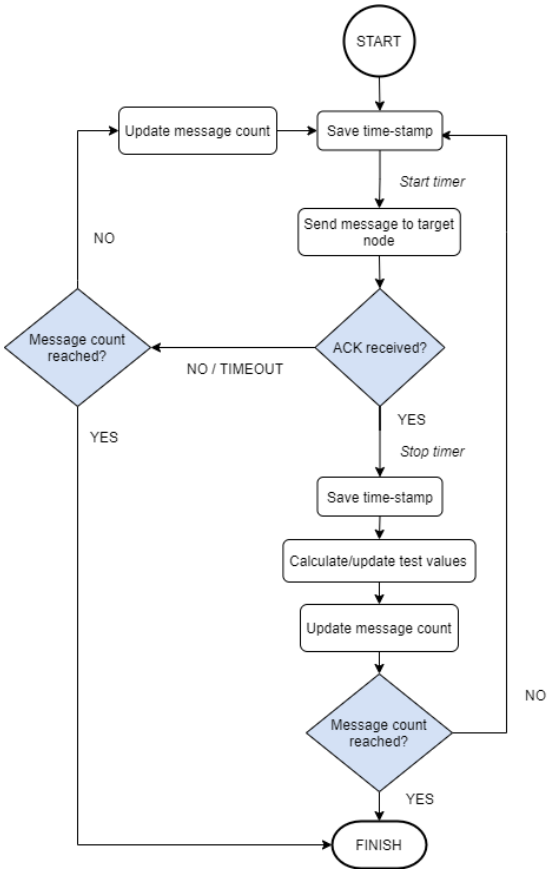


Figure 42: The diagram shows the acknowledged logic flow of the tests for the tester-node (on the embedded side). The tester-node will iterate through all targets. The target can either be a unicast or group address.

Constraint	**Value / Type**
Message type	Acked. Unsegmented
Message amount per node	100
Publish re-transmit count (PRC)	0
Publish re-transmit interval (PRI)	400 ms
Message interval	400 ms
Addressing	Unicast
Time-to-Live (TTL)	7
Network transmit interval (NTI)	20 ms
Relay re-transmit interval (RRI)	20 ms
TXP	0 dBm

Table 11: Initial test conditions.

Algorithm parameter configurations	**Value**		**Note**
	Default	*Alt. v.*	
RSSI range	[-85, -65]	”	
TXP range	[-40, 8]	”	
MAX_TXC	4	”	Max value for NTC and RRC
MIN_TXC	0	”	Min value for NTC and RRC
SRR_SET_POINT	0.8	0.9	
RELAY_GET_PER	10	”	
SRR_Q_SIZE	10	”	
RELAY_MAX_TRAFFIC	160	200	
NTC_DEDUCT_PER	10%	”	
RRC	NTC	NTC / 2	RRC is dependent on NTC

Table 12: Algorithm configurations used during testing. Only the default configuration follows the algorithm specification.

Test results for 100-node network: Reliability & TXP					
Avg. tester NTC	*Avg. network RRC*	*Algorithm*	*Msg. Success*	*Avg. TXP*	Ref. in Figure 43
0	0	No	98,97 %	0 dBm	”NTC0”
2	0	No	100 %	0 dBm	”NTC2”
1	1	Yes - *Default*	99,73 %	-11.76 dBm	”ALG_cfg1”
1	0	Yes - *Alt. v.*	99,87 %	-12.01 dBm	”ALG_cfg2”

Table 13: Average reliability and transmit power results for the 100 nodes with static and dynamic (with the algorithm) values. The different configurations for the algorithm, *Default* and *Alt. v.*, is referring to Table 12.

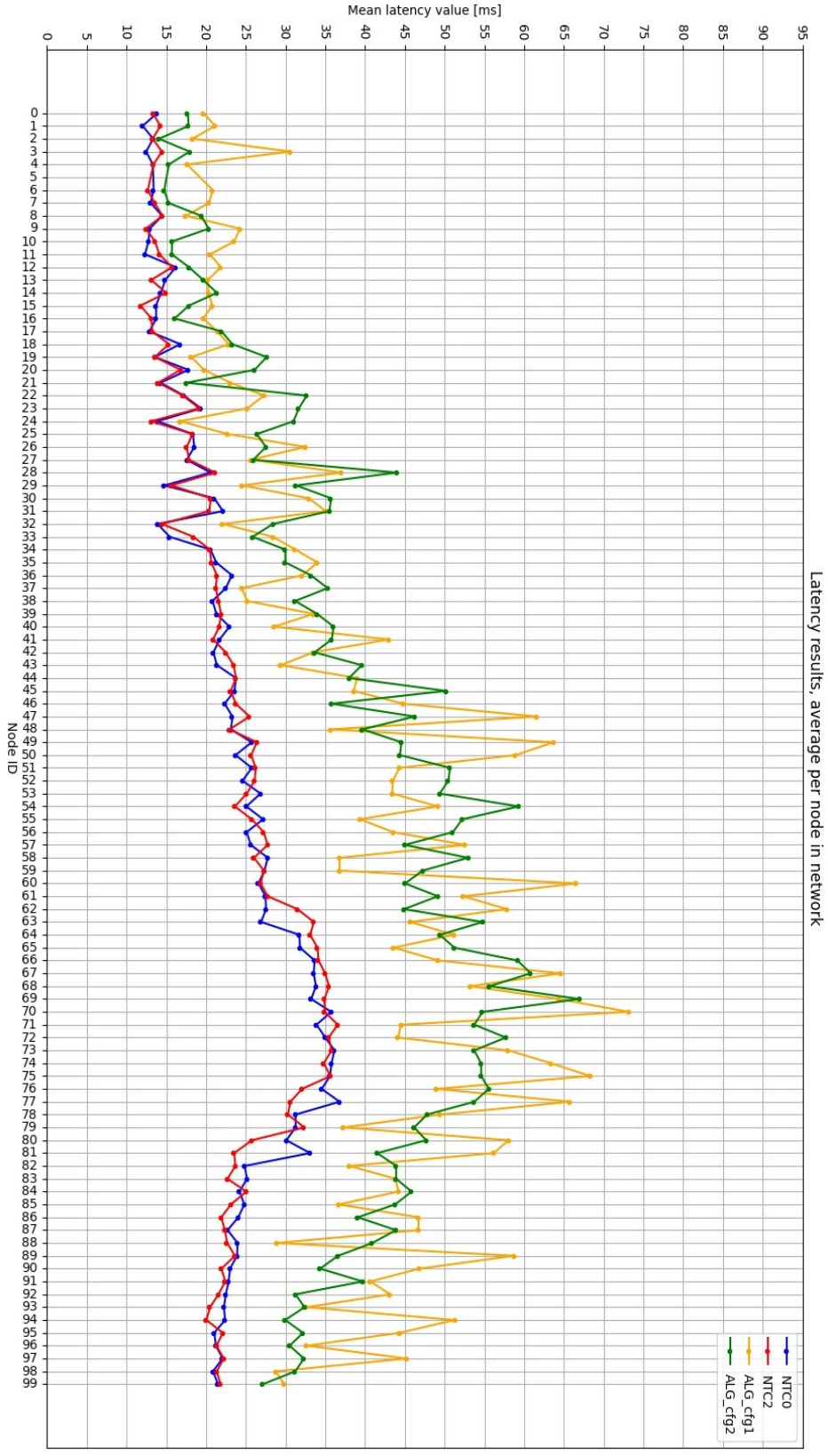


Figure 43: Latency results comparing static and dynamic (with the algorithm) parameter values. The plot shows the mean latency value for each node in the network.

10 Discussion

The developed algorithm has succeeded in monitoring and regulating the state of the network in real-time as documented in the embedded execution demo in Section 7.2. All criteria points as listed in the specification were fulfilled in the final implementation despite the limitations of embedded coding. An assumption for the algorithm was that the network needs to have a somewhat sufficient relay node coverage. The algorithm design diminishes the importance of this assumption by introducing the Relay-to-Relay (RR) relationship and by giving the Edge nodes the ability to dynamically change their master as discussed in Section 7. If the radio connection between two connecting relay nodes is weak or the distance between them is large and therefore unreliable, the TXP regulation will attempt to mend the connection. The algorithm design was made considering a dynamic network and thus the need for dynamic node relationships. Hence, the algorithm is able to handle situations where network nodes are moved or if signals are blocked due to environmental changes or disturbances. However, more testing to document the efficiency and limitations of this aspect is necessary and was not possible due to time limitations for this report.

10.1 Network performance

While running tests on the large-scale network it became apparent that the integrated time-synchronization was not perfectly accurate. From debugging and experience gained from working with the test-bed, it can be concluded that the time-synchronization has an uncertainty of approximately +/- 5-7 ms. This was discovered as some latency values from nodes laying in close proximity to the tester node came back negative, meaning that the received time-stamp at some of these nodes was recorded to be slightly less than the time-stamp on transmit at the tester. This is possible as the actual Single-Trip-Latency (STL) between them was small relative to the uncertainty in the network. Nevertheless, the results provide valuable information about reliability and power usage for the nodes, and the latency results are still meaningful when comparing the different configurations.

The reliability and device transmit power (TXP) results in Table 13 show that the algorithm gives a network reliability of 99.73 % (using the *default* configuration), while also operating at a very low TXP level at only -11.76 dBm. Comparing with the results using the optimally tuned static parameters as suggested in the report *Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks* [2], which gave perfect reliability at 100%, one can say that the algorithm performs at an acceptable level of reliability. This suggests that the Key Performace Indicators (KPIs) in Section 4.1 has been met. According to SiLabs documentation on how to optimize BLE devices [14], their SoCs are set to 8 dBm by default. The nRF52 chip by Nordic Semiconductor used during testing in this report has the default TXP set to 0 dBm according to its specification [4]. On the 100-node network that was subjected to testing in this report, one can reduce the device power consumption by as much as 93% compared with the manufacturer's default setting when using the optimal value as chosen by the algorithm. With battery-operated devices, which are usually very power restricted, this could increase the life span considerably. The reduction in power level by virtue of this algorithm can also help bring down the overall noise level in the Bluetooth advertising channels, making BLE devices and networks able to coexist with less interference.

Although the algorithm proved effective in terms of reliability and power consumption, the latency result as presented in Figure 43 is less impressive. The mean latency for the average node in the network is roughly twice the size of its static competitors, although slightly better with the *alternative configuration* which will be discussed later. This could be explained by the fact that by reducing the TXP as greatly as it was, it decreases the chance of a message successfully arriving at its destination on the first transmission attempt. The weaker the signal the more prone it is to disturbance and interference. Hence, a late arrival of a message, on a later transmit attempt either from its source (NTC) or from a relay (RRC), will cause the latency to increase as seen in the results. The parameters NTC and RRC came out to be, on average for all 100 network nodes, respectively, 1 and 1. The negative effect of an increase in RRC, as documented in [2], was believed to be mitigated due to the decrease in TXP as fewer relay nodes will reach each other, thus

limiting the traffic that this redundant messaging usually cause. However, perfect reliability was not achieved and the latency results indicate that packets are getting lost. The algorithm reveals a trade-off between network traffic, power usage, reliability, and latency. Tuning of the algorithm parameters (constants) could be improved to balance the performance better in this regard.

The acceptable Receive Signal Strength Indicator (RSSI) range used during testing could potentially be re-evaluated and further optimized. A less conservative TXP regulation could reduce the packet-loss issue that was discussed before but also increase the power consumption. It will simply depend on the application use-case and its priorities in terms of quality of service (QoS). Additionally, deciding on what is an acceptable RSSI is not trivial to begin with as it depends on the sensitivity of the radio antenna of the device and other factors as well. The RSSI range as listed in Table 12 was chosen based on the previous testing with the network and knowledge of the nodes that were used.

10.2 The alternative algorithm

Åkredalen [2] draws the conclusion that the first transmission from the source node to the first relay is the most vulnerable and therefore the most critical path for the message. This makes the NTC the most important parameter to consider for this particular path. Since the NTC has a much lower impact on the overall network traffic as it is only effective until the message reaches the first relay after being transmitted from the source node, this parameter can be increased more freely without seeing any significant packet-loss due to increased network traffic. Åkredalen tests with multiple parameter configurations and all came out with a reliability above 99.9%. However, the default TXP parameter setting of 0 dBm was used in all configurations. This makes it more plausible that the messages sent from a source node will reach one or more relays on the first transmit than with a more restrictive TXP. This implies that the algorithm, which in comparison greatly limits the radio range of each network node, is even more dependent on the NTC parameter. The algorithm specification in Section 4.1 clarifies that the RRC value of the relay must follow the NTC to maintain an equally strong path to its neighbouring nodes when relaying as when regulating their SRR. However, this might not be necessary. Since a message has multiple pathways once initially relayed, the relay-to-destination path is not equally as critical as the source-to-relay path. This implies that the RRC, although it may be dependent on NTC in the regulation, might not have to be strictly the same value. To test this theory, an alternative configuration was tested for the algorithm, see "*Alt. v.*" in Table 12. The alternative configuration was chosen to be less conservative with the NTC regulation, allowing a higher traffic threshold and an increased set point for the regulator, while letting the RRC be only half of the local relay's NTC. A slight enhancement in reliability and latency was recorded. Given that the tests were run at different times, with possibly some variation in disturbance and other factors due to the changing environment in the office, these small variations might have been random. To further test the theory of an alternative NTC to RRC relationship, more testing is needed and was not conducted for this report.

11 Conclusion

The framework of the algorithm has proven effective. The results reveal that the network nodes are able to achieve an acceptable reliability at 99.97% while operating with a much lower transmit power than the preset provided by the manufacturer. However, the algorithm achieves a trade-off between network traffic, power usage, reliability, and latency. Tuning of the algorithm parameters could be improved to balance the performance better. This would require extensive testing and was not possible with this report due to time limitations.

As is, the algorithm performs well in terms of reliability and has proven to effectively decrease the device power consumption. Due to the poor latency performance, the algorithm may not be the best choice for lighting control applications or other use-cases that have strict timing demands. In other applications where some latency is accepted but reliability and device life span are more important, such as data gathering or sensor networks, the algorithm may be a good support when optimizing the network parameters.

12 Further work

During the work process of this report, some ideas surfaced on how to further explore the potential of Bluetooth mesh (BTM). Due to timing restrictions and the limited scope of this report, some ideas simply had to be left out. Hopefully, some of these ideas can inspire future work with fellow mesh enthusiasts.

12.1 Further testing and development of the algorithm

The performance test used in this report was not fully able to utilize the individually tuned Network Transmit Count (NTC) value for all network nodes as found by the algorithm. Only *one* tester-node was used; therefore, only this node's network transmit state was able to influence the outcome of the test results. It would be interesting to extend the test coverage to include sensor and data gathering networks where multiple nodes transmit messages onto the network before being picked up by a central gateway. The algorithm has the potential to fit variations of priority-sets depending on the application use case. With further testing and experimentation with tuning of algorithm parameters such as maximum allowed traffic, Status Receive Rate (SRR) set-point and acceptable Radio Signal Strength Indicator (RSSI) range, one could accomplish pre-defining various algorithm presets. To advance even further, the algorithm could be extended to take a weighted priority list as input, defined by the application developer, and then tune its parameters to fit these criteria.

12.2 Bluetooth mesh specification alteration

As discussed in Section 7, the transmit parameter regulation was limited to operate on a "cluster level" due to mesh stack limitations. When a relay node is relaying a message, it is handled by the lower network layer which does not have access to the target data as collected by the application layer. Nor does it have the option to change its parameters on this layer of the stack. However, this could be an interesting subject for further research. By altering the BTM stack, the network layer could be modified so that it's able to adjust its transmit parameters to fit the message target destination. By doing so, there is potential for further optimizing reliability, latency, and power consumption in the network. This *adaptive relaying* could potentially be the next enhancement to the BTM specification.

12.3 Merging with existing optimization techniques

The algorithm shows great promise when it comes to power optimization judging by the results. By combining the algorithm's optimization logic with already existing techniques, for example, the one on scanning cycles as discussed in [16], the overall power usage for the network could potentially become even more effective. Researching the algorithm's capacity to work together with existing optimization techniques, and its ability to perform according to its initial goal while doing so may determine how well-suited a candidate it is for the next addition to the BTM specification.

Bibliography

[1] A. Aijaz et al. 'Demystifying the Performance of Bluetooth Mesh: Experimental Evaluation and Optimization'. In: (2021).

[2] S. Åkredalen. 'Optimized tuning of Bluetooth mesh parameters for wireless lighting control networks'. In: (2022).

[3] Nordic Semiconductor ASA. *Nordic library and source code*. URL: https://github.com/nrfconnect/sdk-nrf (visited on 25th Oct. 2021).

[4] Nordic Semiconductor ASA. *nRF52840 chip*. URL: https://www.nordicsemi.com/Products/nRF52840 (visited on 18th Sept. 2021).

[5] Nordic Semiconductor ASA. *Software development kit (SDK) for Bluetooth mesh*. URL: https://www.nordicsemi.com/Products/Development-software/nRF5-SDK-for-Mesh (visited on 25th Oct. 2021).

[6] Beben, Bak and Sosnowski. 'Efficient relay node management method for BLE MESH networks'. In: (2019).

[7] BlueRange. *FruityMesh documentation*. URL: https://github.com/mwaylabs/fruitymesh/wiki (visited on 25th Mar. 2022).

[8] Ericsson. *Ericsson in Sweden*. URL: https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/sweden (visited on 4th Jan. 2022).

[9] Ericsson. *White paper: Bluetooth mesh networking*. 2020. URL: https://www.ericsson.com/en/reports-and-papers/white-papers/bluetooth-mesh-networking.

[10] GE. *Why Low-Energy Solutions Are Needed To Power The IoT Revolution*. URL: https://www.ge.com/news/reports/low-energy-solutions-needed-power-iot-revolution (visited on 25th Mar. 2022).

[11] Hansen et al. 'On Relay Selection Approaches in Bluettoth Mesh Networks'. In: (2018).

[12] Silicon Labs. *Bluetooth mesh network performance*. URL: https://www.silabs.com/documents/public/application-notes/an1137-bluetooth-mesh-network-performance.pdf (visited on 18th Nov. 2021).

[13] Silicon Labs. *Bluetooth Mesh Parameter Tuning for Network Optimization*. URL: https://www.silabs.com/documents/public/application-notes/an1316-bluetooth-mesh-network-optimization.pdf (visited on 23rd Mar. 2022).

[14] Silicon Labs. *Optimizing Current Consumption in Bluetooth Low Energy Devices*. URL: https://docs.silabs.com/bluetooth/2.13/general/system-and-performance/optimizing-current-consumption-in-bluetooth-low-energy-devices (visited on 25th Mar. 2022).

[15] Silicon Labs. *SiLabs - About us*. URL: https://www.silabs.com/about-us (visited on 4th Jan. 2022).

[16] A. Liendo et al. 'BLE Parameter Optimization for IoT Applications'. In: (2018).

[17] MCUBoot. *MCUBoot*. URL: https://docs.mcuboot.com/ (visited on 20th July 2022).

[18] NeoCortec. *Technology - NeoMesh*. URL: https://neocortec.com/technology/ (visited on 18th Mar. 2022).

[19] C. Nieto-Taladriz, Y. Murillo and S. Pollin. 'Towards Efficient BLE Mesh: Design of an Autonomous Network Joining Algorithm'. In: (2019).

[20] Zephyr Project. *Zephyr Project Documentation*. URL: https://docs.zephyrproject.org/latest/ (visited on 11th July 2022).

[21] R. Rondón et al. 'Understanding the Performance of Bluetooth Mesh: Reliability, Delay, and Scalability Analysis'. In: (2020).

[22] Nordic Semiconductor. *What is Bluetooth mesh?* URL: https://www.nordicsemi.com/Products/Bluetooth-mesh/What-is-Bluetooth-mesh (visited on 30th Apr. 2022).

[23] LSA Technology Services. *Received Signal Strength Indicator*. URL: https://teamdynamix.umich.edu/TDClient/47/LSAPortal/KB/ArticleDet?ID=1644 (visited on 5th May 2022).

[24] Bluetooth SIG. 'Bluetooth Mesh Networking - An Introduction for Developers v1.0.1'. In: (2020).

[25] Bluetooth SIG. *Bluetooth Special Interest Group - About Us*. URL: https://www.bluetooth.com/about-us/vision/ (visited on 18th Nov. 2021).

[26] Bluetooth SIG. *Build a Smarter Building with Blue*. URL: https://www.bluetooth.com/bluetooth-resources/smartbuilding-infographic/ (visited on 4th July 2022).

[27] Bluetooth SIG. *Managed flooding*. URL: https://www.bluetooth.com/blog/an-intro-to-bluetooth-mesh-part2/ (visited on 24th Jan. 2022).

[28] Bluetooth SIG. *Mesh networking*. URL: https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/ (visited on 18th Sept. 2021).

[29] Bluetooth SIG. *Mesh Profile 1.0.1*. URL: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/ (visited on 18th Sept. 2021).

[30] WIZnet. *W5500 IC Ethernet controller*. URL: https://www.wiznet.io/product-item/w5500/ (visited on 1st May 2022).

[31] M. Yang et al. 'ACE: A Routhing Algorithm Based on Autonomous Channel Scheduleing for Bluetooth Mesh Network'. In: (2021).

Acronyms

ADV bearer Advertising bearer. 2, 3

BLE Bluetooth Low Energy. 1, 2, 5, 15, 32, 55

BT Bluetooth. 1, 3, 34

BTM Bluetooth mesh. 1–6, 10, 15–17, 28, 42, 58

FIFO First In First Out. 43

nCS nRF Connect Software development kit. 43

NTC Network Transmit Count. 4–6, 9, 10, 16, 17, 22, 28, 30–32, 34, 36, 43–47, 49–51, 55, 56, 58

NTI Network Transmit Interval. 4, 5

PDU Protocol Data Unit. 2, 3

PRC Publish Re-transmit Count. 5, 6, 9

QoS Quality of Service. 5, 6, 56

RRC Relay Re-transmit Count. 5, 6, 8, 9, 16, 17, 28, 31, 32, 34, 42, 43, 51, 55, 56

RRI Relay Re-transmit Interval. 5, 42

RSSI Received Signal Strength Indicator. 17, 28, 30, 32, 33, 35, 42–44, 47, 49, 56, 58

RTL Round-Trip Latency. 5, 50

RTOS Real-Time Operating System. 44

SAR Segmentation And Re-assembly. 4

SoC System on a Chip. 1, 55

SRR Status Received Rate. 18, 22, 28, 30, 32, 34, 36, 37, 43, 45, 46, 49, 56, 58

STL Single-Trip Latency. 5, 50, 55

TTL Time-To-Live. 3, 4, 10, 17, 33, 41, 45, 49

TXP Transmit Power. 5, 13, 16, 17, 22, 28, 30–32, 34–36, 42–44, 47, 48, 50, 51, 55, 56

Appendix

A Emulator script

```
# file name: opt_alg_emulator.py

from cmath import sqrt
from pydoc import classname
import random
import time
import math

nodes = {
    0: {'addr': 'R00', 'Position': [0, 0]},
    1: {'addr': 'E01', 'Position': [8, 9]},
    2: {'addr': 'E02', 'Position': [5, 4]},
}

RSSI_MIN = -70
RSSI_MAX = -20
NTC_MAX = RRC_MAX = 4
NTC_MIN = RRC_MIN = 0
TXP_MAX = 16
TXP_MIN = -40

# ------------------------ CONFIGURATIONS ---------------------- #
# The RELAY_MAX_TRAFFIC determines how many mesh (algorithm or otherwise) messages
# it can receive per time unit before the transmit count (NTC and RCC) must be lowered.
RELAY_MAX_TRAFFIC = 40

# Send a GET message every X second
RELAY_GET_PER = 10

# Check the relay traffic every X second
RELAY_TRAFFIC_CHECK_PER = 30

# Determines how much of a change the SRR must have since the previous adjustment before another
SRR_REGULATION_BAND = 0.05

# MIN_MEND_RATE is the minimum SRR increase from previous regulation after MAX_MEND_TRIES
MIN_MEND_RATE = 0.3
MAX_MEND_TRIES = 5

# Determines how much the total neighborhood NTC should be reduced per traffic mending event
NTC_DEDUCT = 0.1
# -------------------------------------------------------------- #

class Edge():
    def __init__(self, node_id):

    def _get_rssi(self, pos1: int, pos2: int) -> float:

    def _tx_status_msg(self, addr: str) -> bool:

    def _rx_get_msg(self, relay) -> dict:
```

```
    def _rx_txp_msg(self, value: int) -> None:

    def _rx_ntc_msg(self, value: int) -> None:


class Relay():
    def __init__(self, node_id: int) -> None:

        self.relay_table = {
        1: {'addr': 'E01', 'rssi': 0, 'txp': 0, 'ntc': 0,
        'tx_txp': 0, 'msg_count': 0, 'srr': 1, 'prev_srr': 1, 'avg_hop': 1},
        2: {'addr': 'E02', 'rssi': 0, 'txp': 0, 'ntc': 0,
        'tx_txp': 0, 'msg_count': 0, 'srr': 1, 'prev_srr': 1,'avg_hop': 1.2},
        }

    (...)

    def _tx_get_msg(self) -> bool:

    def _tx_set_ntc_msg(self, addr: str) -> bool:

    def _tx_set_txp_msg(self, addr: str) -> bool:

    def _set_tx_param(self, addr: str):

    def _add_to_msg_tx_cnt(self, node_id: int) -> list:

    def _get_highest_srr_node_id(self, _dict: dict, _highest_srr: int) -> int:

    def _check_traffic(self) -> None:

    def _rx_status(self, msg: dict) -> list:
        # Follows the logic of the client model handler

class Emulator:

    def __init__(self) -> None:

    def _run(self, relay_list: list, edge_list: list) -> None:
        _time = 0
        _total_ntc = 0
        _prev_total_ntc = 0
        _try_cnt = 0

        while True:
            for relay_obj in relay_list:
                print('_________________ Time: ', _time,'_________________\n')
                if relay_obj._tx_get_msg():
                    for edge_obj in edge_list:
                        status_msg = edge_obj._rx_get_msg(relay_obj)
                        if edge_obj._tx_status_msg(relay_obj.addr):
                        # Relay _rx_status() decides if regulation is necessary:
                            reg_param_list = relay_obj._rx_status(status_msg)
                            if reg_param_list is not None:
                                if  reg_param_list[0]== 'txp':
                                    edge_obj._rx_txp_msg(reg_param_list[1])
                                elif  reg_param_list[0]== 'ntc':
                                    edge_obj._rx_ntc_msg(reg_param_list[1])
```

```
                                else:
                                    print('ERROR: no such parameter')

                    time.sleep(RELAY_GET_PER)
                    _time+=RELAY_GET_PER


                    # Check traffic every RELAY_TRAFFIC_CHECK_PER second:
                    if _time \% (RELAY_TRAFFIC_CHECK_PER/RELAY_GET_PER) == 0:
                        relay_obj._check_traffic()

#---------------------------------- main -------------------------------------

e1 = Edge(1)
e2 = Edge(2)
edge_nodes = [e1, e2]

r1 = Relay(0)
relay_nodes = [r1]

emu = Emulator()
emu._run(relay_nodes, edge_nodes)
```

B Embedded code

The actual embedded code making up the developed algorithm contains several files and libraries. Parts of this code is owned by Nordic Semiconductor and is not currently public. Therefore, only a simplified structure view of the embedded main code is given here. Only the most important function bodies and definitions are included. Model definitions such as message contents etc. are not given here. See Section 8 for more logic details.

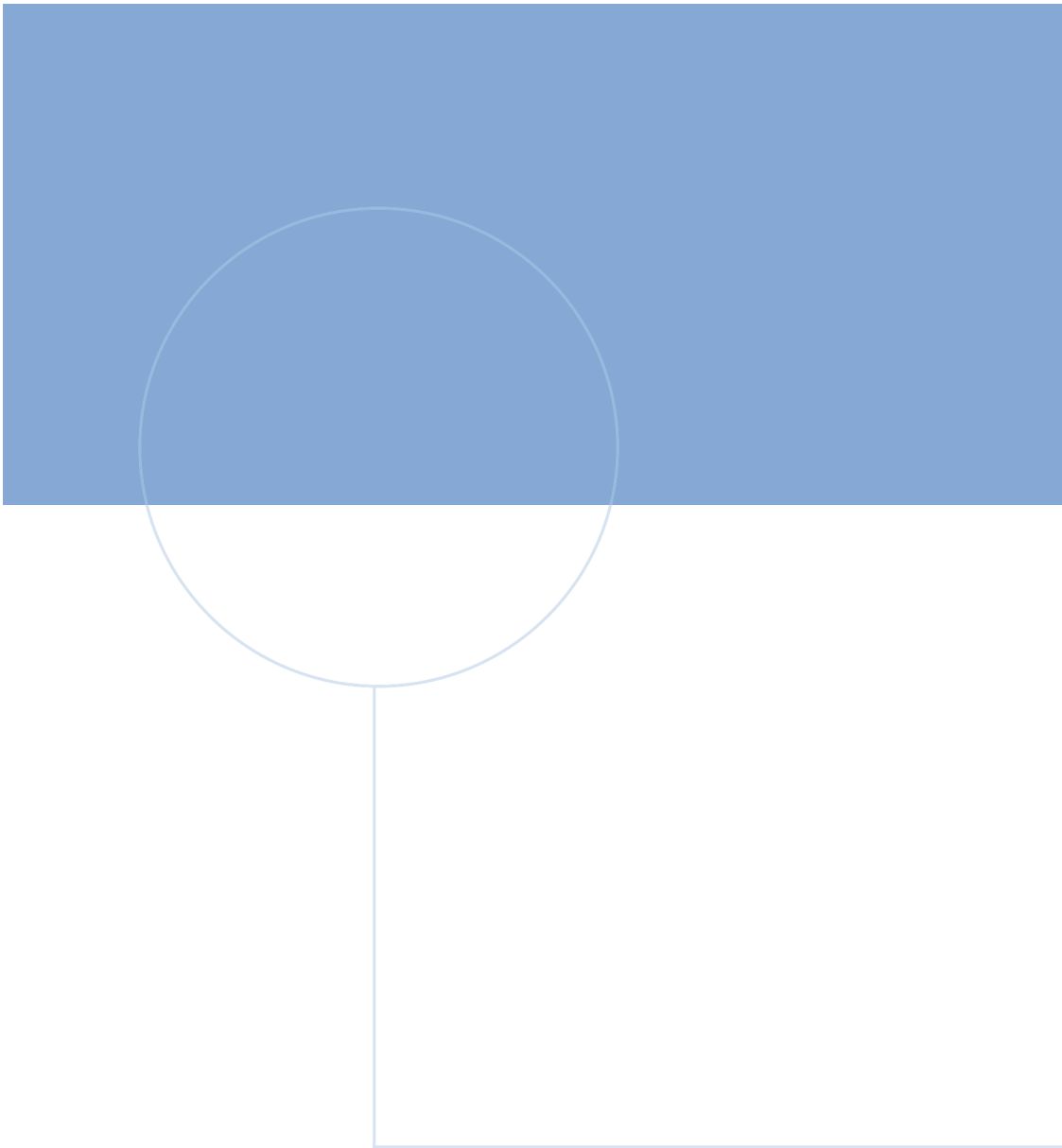
```
// file name: ./mesh_opt_alg/src/main.c

/////////////// CONFIGURATIONS /////////////////

#define RSSI_MIN -85
#define RSSI_MAX -65

/** Message Transmit Count: NTC and RRC*/
#define MSG_TXC_MAX 4
#define MSG_TXC_MIN 0
#define SRR_SET_POINT 0.8

/** Send a periodic GET message every X second*/
#define RELAY_GET_PER 10
#define SRR_CHECK_PER ((SRR_Q_SIZE * RELAY_GET_PER)+1)

#define MASTER_TIMEOUT (5 * RELAY_GET_PER)

/** Max mesh packets originating from a relays neighbours
 *  before reducing the overall neighbourhood NTC*/
#define RELAY_MAX_TRAFFIC 160

#define RELAY_TRAFFIC_CHECK_PER (3 * RELAY_GET_PER)
#define NTC_DEDUCT_PER_EVENT 0.1
#define MAX_NTC_REDUCTION_ITR 10


/////////////// Structures /////////////////

#define RELAY_TABLE_SIZE 100
#define SRR_Q_SIZE 10

/** Additional parameters for a node in the relay table. */
static struct node_param_t {
        /** Last recorded RSSI value for node */
        int8_t rssi;
        /** Last recorded NTC value for node */
        uint8_t ntc;
        /**  */
        int8_t txp;
        /** Status Received Rate calculated from the average of srr_q */
        float srr;
        /** Previous Status Received Rate */
        float prev_srr;
        /** A container (FIFO) holding the latest received STATUS message stats */
        int8_t srr_q[SRR_Q_SIZE];
        /** The average of hop_queue container */
        bool is_nb;
};
```

```
/** Mandatory parameters for each node entry in the relay table. */
struct relay_table_entry {
        /** Address of STATUS message origin node */
        uint16_t addr;
        /** Holds additional origin node parameters */
        struct node_param_t node_param;
};

static struct relay_table_entry relay_table[RELAY_TABLE_SIZE];

static const int8_t txp_table[6] = { -40, -20, -16, -8, -4, 0, 4, 8 };

static uint32_t relay_total_mesh_traffic = 0;

static struct relay_para relay_state = {
        .retrans_count = 0,
        .retrans_int = 20,
        .feat = BT_MESH_FEATURE_DISABLED,
};

struct bt_mesh_msg_ctx tx_alg_ctx = {
        .addr = 0x0000,
        .send_ttl = TTL_ALG_DEFAULT,
        .app_idx = 0,
};

/** Relationship data for the node */
struct tag_t {
        uint16_t master_key_addr;
        int8_t master_txp;
        int8_t master_rssi;
};

struct tag_t nb_tag, rr_master_tag, rr_slave_tag;


/////////////// Model transmit functions /////////////

int bt_mesh_opt_alg_send_get(struct bt_mesh_opt_alg_mod *mod,
                        struct bt_mesh_msg_ctx *ctx,
                        struct bt_mesh_opt_alg_get_msg *msg);

int bt_mesh_opt_alg_send_status(struct bt_mesh_opt_alg_mod *mod,
                        struct bt_mesh_msg_ctx *ctx,
                        struct bt_mesh_opt_alg_status_msg *msg);

int bt_mesh_opt_alg_send_set(struct bt_mesh_opt_alg_mod *mod,
                        struct bt_mesh_msg_ctx *ctx,
                        const struct bt_mesh_opt_alg_set_msg *msg);


////////////// Tools and help functions ///////////

static void get_txp(uint8_t handle_type, uint16_t handle, int8_t *txp_lvl){};

static void set_txp(uint8_t handle_type, uint16_t handle, int8_t txp_lvl){};
```

```
/** Set the local node's Network Transmit State */
static void set_net_trans(uint8_t _count, uint8_t _intr){};

/** Set the local node's Relay Re-transmit State */
static void set_relay(struct relay_para state){};

/** This function is being triggered every time the local node receives a
* BT mesh message on the network layer of the stack.
* The function keeps track of the total amount of traffic originating form neighbouring nodes
*/
void hook_network_pkt_rx(struct net_buf_simple *buf, struct bt_mesh_net_rx *rx){};

/** Returns table index on success, else -1 */
static uint8_t get_relay_table_index(uint16_t node_addr){};

/** Returns table index on success, else -1 */
static uint8_t get_txp_table_index(int8_t txp_val){};

static bool rssi_in_range(int8_t val){};

static bool ntc_is_in_range(uint8_t val){};

static void update_tx_param(uint16_t addr){};

static void opt_alg_update_relay_table(struct bt_mesh_msg_ctx *ctx,
                        const struct bt_mesh_opt_alg_status_msg *status,
                        uint8_t n_idx){};

/** Sends a SET_message requesting target (neighbour "nb") (n_idx) to change its TXP.
 * Returns:
 * 0 if SET-message was sent successfully,
 * 1 if no further change is possible,
 * 2 if the TXP argument (n_txp) is invalid,
 * or negative if message transmit failed */
static int request_nb_txp_change(struct bt_mesh_opt_alg_mod *mod,
                    struct bt_mesh_msg_ctx *rcvd_ctx,
                    uint8_t n_idx){};

static uint8_t is_not_in_list(uint8_t val, int8_t list[], uint8_t len){};

/** Returns the relay_table index of the node with the highest srr,
 * excluding all indexes in exclude_list,
 * or -1 if no more nodes can be reduced
  */
static uint8_t get_highest_srr_node_idx(int8_t exclude_idx_list[]){};

/** Sends a SET_message requesting target (neighbour "nb") (n_idx) to increase its NTC.
 * Returns:
 * 0 if SET-message was sent successfully,
 * 1 if no further change is possible,
 * or negative if message transmit failed */
static int request_nb_ntc_increase(struct bt_mesh_opt_alg_mod *mod,
                    struct bt_mesh_msg_ctx *rcvd_ctx,
                    uint8_t n_idx){};

static uint8_t get_total_nbh_ntc(){};
```

```
////////////////// Model handlers /////////////////

static int opt_alg_mod_status_handler(struct bt_mesh_opt_alg_mod *mod,
                          struct bt_mesh_msg_ctx *ctx,
                          const struct bt_mesh_opt_alg_status_msg *status);

static void opt_alg_mod_get_handler(struct bt_mesh_opt_alg_mod *mod,
                                struct bt_mesh_msg_ctx *ctx,
                                struct bt_mesh_opt_alg_get_msg *get);

static void opt_alg_mod_set_handler(struct bt_mesh_opt_alg_mod *mod,
                              struct bt_mesh_msg_ctx *ctx,
                              struct bt_mesh_opt_alg_set_msg *set);


///////////////// Delayed work ////////////////////

static struct k_work_delayable opt_alg_send_get_work;
static struct k_work_delayable opt_alg_traffic_work;
static struct k_work_delayable opt_alg_master_timeout_work;
static struct k_work_delayable opt_alg_check_srr_work;

/** All delayables have their own handlers for when/if they time out */


///////////////// Ethernet command handlers ////////////////

static void mesh_opt_alg_run(struct pca20036_rx_ctx *ctx,
                             struct net_buf_simple *buf)
{
        if(relay_state.feat == BT_MESH_RELAY_ENABLED){
                k_work_reschedule(&opt_alg_send_get_work, K_NO_WAIT);
                k_work_reschedule(&opt_alg_traffic_work, K_SECONDS(RELAY_TRAFFIC_CHECK_PER));
                k_work_reschedule(&opt_alg_check_srr_work, K_SECONDS(SRR_CHECK_PER));
        }
}

static void mesh_opt_alg_stop(struct pca20036_rx_ctx *ctx,
                              struct net_buf_simple *buf)
{
        k_work_cancel_delayable(&opt_alg_send_get_work);
        k_work_cancel_delayable(&opt_alg_traffic_work);
        k_work_cancel_delayable(&opt_alg_check_srr_work);
}


////////////////////////////////////////////////////////

void main(void)
{
        int err;

        pca20036_dbg_pins_init();
        k_work_init_delayable(&unacked_ctx.work, unacked_tx_work_cb);
        k_work_init_delayable(&opt_alg_send_get_work, opt_alg_send_get_work_cb);
        k_work_init_delayable(&opt_alg_traffic_work, opt_alg_traffic_work_cb);
        k_work_init_delayable(&opt_alg_master_timeout_work, opt_alg_master_timeout_work_cb);
        k_work_init_delayable(&opt_alg_check_srr_work, opt_alg_check_srr_work_cb);
```

```
        LOG_INF("- DFU Version: %d -", CONFIG_TFTP_DFU_APP_VERSION);

        pca20036_cmd_init(NULL);

        LOG_INF("- Initiated -");

        /* DHCP may not be leased yet - check with get_dhcp_leased_flag() */

        printk("Initializing mesh...\n");

        /* Initialize BT Mesh, provision and configure local device */
        err = bt_enable(NULL);
        if (err) {
                BT_ERR("Bluetooth init failed (err %d)", err);
        }

        bt_ready(0);
}
```



NTNU
Norwegian University of
Science and Technology

NORDIC®
SEMICONDUCTOR