Buffer management policies for BLE-based multi-hop communication

Soman, A.

Award date:
2014

Technische Universiteit
Eindhoven
University of Technology

Department of Electrical Engineering
Electronic Systems

Buffer Management Policies for BLE-based Multi-hop Communication

Master Thesis

Ashutosh Soman
Student ID: 0869737

Supervisors:
Dr. Sander Stuijk
Dr. Majid Nabi Najafabadi
Dr. Yan Zhang

1.0 version

Eindhoven,
27 October 2015

Abstract

Bluetooth Low Energy (BLE) has gained a lot of popularity since its introduction in the Bluetooth Core 4.0 standards, mainly due to its compatibility with modern smartphones. BLE provides low energy and low cost solution for low data rate wireless communication. To reduce complexity, BLE supports only peer-to-peer, master-slave communication. This proves to be a roadblock in its application in many moderately large wireless sensor network (WSNs) applications where line of sight communication is not sufficient. We tackle this problem by introducing multi-hop communication over BLE. This work proposes a novel routing mechanism and data transfer functionality over BLE radio and link layer, which is tested with an actual implementation on custom hardware from IMEC. The data transfer in BLE over multiple hops is essentially a series of connection events where each connection event can transfer large amount of data to its neighboring node resulting in each node storing large amount of data. But if complete buffer space is provided to a single node in a connection event, the other nodes willing to send data through the same relay node will not get buffer space till all the data from previous node is forwarded and the relay node has empty buffer space. This may cause starvation of the nodes. This leads to a need of buffer management strategy in BLE multi-hop networks. We introduce two buffer allocation strategies in this work namely '*1-by-k* buffer allocation mechanism'and 'buffer allocation based on average number of connections'. The strategies moderate the factor (k and average number of connections) deciding the fraction of the buffer space which is allocated to a requesting node. These proposed strategies are tested with a high level simulation in MATLAB for their performance in non-mobile networks. The simulation results show that 'buffer allocation based on average number of connections'method performs better in the tested scenarios.

Acknowledgement

I am sincerely grateful to my supervisors Dr. Sander Stuijk and Dr. Majid Nabi Najafabadi at the university for their exemplary guidance, critical insight and encouragenent throughout this thesis work. Special thanks to Dr. Yan Zhang at IMEC for her constant motivation and valuable insights without which this work would not be possible. I would also like to thank IMEC and its employees for providing me friendly and motivating environment for my research. Thanks to EIT ICT labs and Dr. Bas Luttik for introducing me to this project.

I am forever grateful to my parents and family for their love and support, without which, this endeavor would not be possible. Finally, my hearty thanks to my friends Sivakumar Thangamuthu and Govind Narayan for their encouragement, moral support and good times.

Contents

Chapter 1

Introduction

Wireless Sensor Networks (WSNs) have been in focus of lots of research recently. The reason is their usefulness in a vast number of applications such as indoor automation, environmental monitoring, industrial automation to name a few. A typical WSN consists of a large number of processing units called nodes which communicate with each other via their low power wireless radio transceivers. These nodes act as sensors which sense the data from the environment, actuators which change the state of the environment in some way or relays which are primarily used for data transfer from source (sensor) to the sink. As these networks are designed to run for months or years without human intervention in remote areas, the individual nodes are typically constrained in terms of energy, memory, processing power and transmission power. These constraints have motivated researchers to invent innovative and novel ways to reduce energy consumption in WSNs. Since the wireless transmission is one of the most energy consuming activities in WSNs, many protocols and standards like IEEE 802.15.4 [1], Bluetooth Low Energy [2], ANT [10] have been introduced and studied extensively, which enable energy efficient and low to moderate data rate wireless communication amongst WSN nodes.

Bluetooth Low energy (BLE) has gained a lot of popularity among these technologies since its introduction, majorly due to its compatibility with modern smartphones. This has led to its employment in many applications where direct radio line of sight communication is possible such as body sensing, proximity based information retrival etc. BLE standard was originally developed at Nokia under the name Wibree in 2006 [14]. The Bluetooth SIG (Special Interest Group) incorporated it in Bluetooth 4.0 standards in 2010. A subset of the Bluetooth 4.0 core specification, BLE, provides a lightweight solution to the problem of low power wireless communication.

1.1 BLE in Wireless Sensor Networks

Since its introduction in Bluetooth 4.0 standards, BLE has gained huge popularity in wireless sensor networks research and applications. It provides energy efficient wireless communication at low development cost due to reduced complexity. This is because BLE is optim-

ized for one-to-one communication and supports only a simple star topology. This along with relaxed Adaptive Frequency Hoppong (AFH), reduces silicon footprint of the BLE protocol stack resulting in less complex implementation as compared to classic Bluetooth and IEEE 802.11 (Wi-Fi). BLE provides high robustness to interference by using Adaptive Frequency Hopping (AFH) [6] technique. With 128-bit AES encryption [21], it provides highly secure communication between the nodes. With many of the new generation of smartphones supporting BLE, it provides a convenient way for data monitoring and data logging to users. BLE provides a low latency, energy efficient and robust solution, necessary for applications in sensor networks making it an ideal choice [16]. With potentially very small silicon footprint and its compatibility for smartphones, the technology is embraced by many of the network designers for low cost, low power applications.

To reduce complexity, BLE adapts star topology (piconet). Scatternet topology is not permitted. Therefore BLE network becomes a purely peer-to-peer, master-slave network where there is one master communicating with many slaves. This makes BLE useful for applications such as remote controlling of actuators, or indoor sensing networks with short range communications.

1.2 Contribution

As discussed in section 1.1, BLE is ideal for use in short range communication in WSN with its low latency, low power consumption and low cost. But still it remains a network which is essentially only useful for peer-to-peer communication as it only supports star topology (piconet) with no support for scatternet, which is essentially many piconets communicating withing one another with the help of common nodes in individual piconets, as compared to classic Bluetooth. This restriction makes BLE unfit for establishing multi-hop networks, in effect restricting coverage of the network. As other properties of BLE are quite fitting for sensor networks, it is quite interesting to study the possibility of increasing the coverage of BLE network. In this work, we aim at establishing multi-hop links between the nodes using BLE thus increasing the network coverage and connectivity.

In BLE, data transfer is done by making connection events to the peer. So, the actual data can be transferred only after establishing a connection between two nodes. This process of establishing connections is more expensive in terms of latency as well as power than the actual data transfer. Thus to forward data and to act as a relay, a node needs to store a large amount of data in the buffer before forwarding it towards the destination node, so as to reduce the number of expensive connection establishments and transfer as much data as possible in one connection event. So, if we assign the whole data buffer to a single node, another node wanting to forward data through same relay node has to wait till the buffer is empty. This results in the starvation of the nodes and delay in overall end to end data transfer between source and destination nodes. This thesis describes solutions to this problem by partitioning the buffer and reserving different partitions for different sending nodes. The purpose of this thesis is to find a buffer management strategy which provides fairness in the allocation of buffer space while trying to minimize average end-to-end delay

in the transfer of the data.

The typical use for such multi-hop BLE networks can be small networks for indoor industrial automation and sensor networks. It may include specific applications such as safety sensor networks and data logging networks or it can be used for backbone networks providing relays for the applications mentioned above. These sensor or data logging networks work with large amounts of data, generated sporadically at the sensors. This kind of data, specially logging data, needs to be stored to the last byte. However, it is delay tolerant to some extent as the data is usually stored in files at the sink for later analysis. BLE based multi-hop networks seems fit for these applications.

1.3 Report Overview

Chapter 2 explains the problem of multi-hop link establishment and necessity of a good buffer management strategy in detail. Chapter 3 reviews related researches in the field of multi-hop link demonstration using BLE and different buffer management strategies in multi-hop networks. Chapter 4 focuses on the basics of BLE protocol in detail. In Chapter 5, the routing algorithm used to establish multi-hop links in this thesis project is discussed. The buffer management strategies and algorithms used to solve the problem are explained in Chapter 6. Chapter 7 presents the experiments conducted for verification and comparison of different buffer management strategies intorduced in Chapter 6. It discusses the obtained results from these experiments. Chapter 8 concludes this thesis report.

Chapter 2

Problem Statement

BLE is designed for low data rate master-slave communication. The star or piconet topology allows a slave node to be connected to a single master. By keeping the topology simple, the Bluetooth SIG aims to make BLE less complex than classic Bluetooth. This design choice in BLE network topology makes it a purely peer-to-peer network. Though the popularity of BLE is on the rise, mainly because of its compatibility with today's smartphones, many designers refrain from using it for communication in certain applications where multi-hop communication is involved because BLE inherently does not support mesh topologies and multi-hop communication.

Even though there are other protocols such as 802.15.4 which are designed for multi-hop communication and mesh networking topologies, it is interesting to investigate if multi-hop link communication can be demonstrated using BLE. It is also interesting to understand what problems we face in achieving such feat.

One of the major road blocks in BLE multi-hop communication can be storage of data at intermediate nodes. Data transfer in BLE requires two nodes to pair with each other in a connection event. A slave node at a time can be part of only a single connection. Thus, for an intermediate node to forward the data, it needs to store the data from the whole connection event rather than just forwarding it as soon as it is received. This leads to the need of a buffer management strategy.

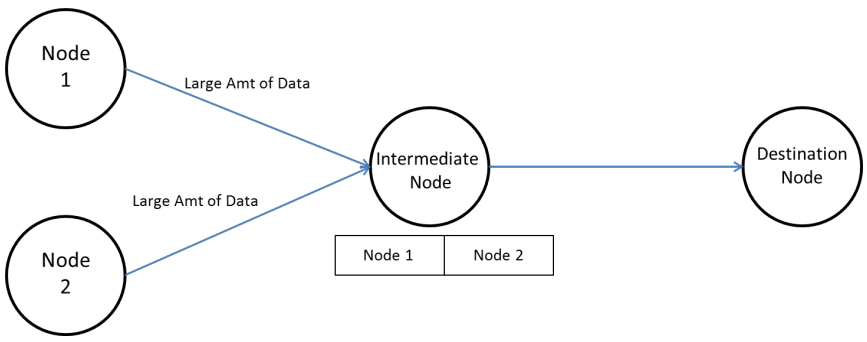

Figure 2.1: Scenario: Two multi-hop communications throught a common node with BLE

A very naive strategy would just accept the data from a node until the buffer is full and then inform the other node to stop sending the data. But this can get complicated if another node wishes to send data through the same intermediate node. In this scenario, a second node asking for multi-hop connection through the same intermediate node has to wait for a quite long time as it can only try to send the data once the intermediate node has forwarded its current data and has some space to accommodate new data. The example in Figure 2.1 explains the scenario quite well. Here node 1 first starts the multi-hop communication and establishes connection with intermediate node. Now, if whole buffer is allocated to node 1, then node 2 cannot send the data till the intermediate node has forwarded the data to the destination. This scenario depicts the need of proper buffer management in case of BLE multi-hop networks. This leads us to a problem of discovering a buffer management strategy which gives a fair allocation of buffer to each participating node and which tries to minimize the average end-to-end delay of data sent from all sources to the sink. The allocation should be done in such a way that it should not lead to a deadlock or a bottleneck situation due to limited available buffer space.

Chapter 3

Related Work

This chapter discusses research in the fields of buffer management to manage limited buffer space at individual nodes in WSNs and multi-hop communication using BLE. We first review the work done related to multi-hop routing over BLE and then move on to different buffer management strategies that are introduced to have communication between the nodes in WSNs.

3.1 Multi-hop Communication over BLE

As BLE is primarily a peer-to-peer communication technology supporting only star topology, there has not been much work to have multi-hop communication or mesh topology using BLE. Main reason is the existance of other powerful protocols such as IEEE 802.15.4 LR-WPAN [8] which inherently support mesh topologies. Nevertheless, as mentioned earlier, BLE possesses some characteristics which are ideal for WSNs inspiring people to study the possiblity of multi-hop communication using the same. Haatanen et al. in [9] have developed a multi-hop link over BLE for a specific scenario. Though not successfully implemented due to limitations of hardware platform used at that time, they propose to use the whole BLE stack for the means of transportation and to implement routing mechanism on a higher layer or outside the stack. They use BLE advertisements and ATT profile for transfer of data among the nodes. This results in overhead due to upper layers reducing the amount of data that can be transferred. Konstantin Mikhaylov and Jouni Tervonen, in [12] refine the methods used in [9]. They implement multi-hop routing as a Generic Attribute (GATT) service called MHTS. Their routing mechanism is based on reading and writing of GATT attributes, resulting in latency expensive establishment of connection for even setting up the route to a remote node increasing the latency for route discovery and data transfer.

This thesis on the other hand, implements multi-hop routing over the link layer of BLE stack providing efficient solution to the problem. The mechanism uses link layer advertising packets to find the routes and to establish the connection. The data is exchanged after the connection is established using link layer data packets to reduce the overhead of BLE

host layers such as GATT and GAP. Thus more amount of actual data can be transferred in a single packet. As the route discovery is done in advertising mode, no connection establishment is required for finding the routes to the destination, reducing the latency. By avoiding implementation of data transfer as a service over GATT layer, i.e. by avoiding the implementation of route discovery as an application over BLE stack, we can have any application over the network layer (over the route discovery) increasing the flexibility of the implementation. Thus, by implementing routing over the link layer, we add flexibility and reduce latency as compared to the approaches used above.

3.2 Buffer Management Strategies

The multi-hop communication mentioned in [9] and [12] also assume that sufficient buffer space is available for the communication and no congestion or bottleneck takes place during the communication. But in practice, the buffer space is limited and thus, fair allocation and congestion or deadlock management needs to be taken into account.

There has been extensive research on buffer management in different types of wireless communications. Many of the papers like [3][4][18], though, talk about push-out policies. A push-out policy essentially means a policy to throw out some of the existing or newly arriving packets in the buffer, to accommodate new ones. This kind of policy is suitable for situations where it does not matter if a few packets are lost. This thesis describes the scenario where one cannot afford a loss of packets though some delay is tolerable as the possible applications for this work are industrial data logging networks where data needs to be transferred without any packet loss.

Shigang Chen and Na Yang in [5] solve the problem of congestion using lighweight buffer management in WSNs by introducing *1/k-buffer* solution. In this solution, every sending node is allowed to send data equal to only *1/k* of free buffer space. This available buffer space is broadcast time to time. This is done as a safety measure to avoid congestion. To manage the buffer space competely and fairly, the algorithm decreases the value of k by 1 if no congestion occurs for a long time. On the other hand, if the congestion is observed in the network, k is increased by 1. Thus over the time, value of k is optimized for optimal usage of buffer.

While not in the field of WSN in particular, the buffer management policy by Ramesh Nagarajan [13] provides a different point of view to look at the problem. In this paper, he suggests a mechanism with two thresholds in which nodes send a message to vary the transmission rate based on the available buffer space. If there is less space available to store incoming data i.e. if the buffer is full upto the higher threshold, the node broadcasts a message to reduce the transmission rate of the senders. On the other hand, if the buffer is filled less than the lower threshold, the node broadcasts the message so that sender can increase its transmission data rate. Thus in this work, the author suggests to manage buffer space by varying the datarate at the sending nodes. In our case, however, we aim to reduce the number of transmissions to reduce the latency and power consumed. Another limiting aspect in using this method for BLE multi-hop communication is that BLE supports fixed

datarate of 1 Mbps which can not be varied based on the buffer space available at the peer node.

CODA [22] avoids bottlenecks due to limited buffer by combining various methods such as open-loop hop-by-hop backpressure, and closed-loop multi-source regulation to control data rate of the peers. The open-loop hop-by-hop backpressure mechanism is a quick response method which propagates 'pressure'from destination to source by limiting the incoming traffic at immideate neighbour whereas closed-loop multi-source regulation depends on communication of control message to all the sources in closed loop to regulate the data rate and avoid the network saturation. These methods are applicable for a networks which support variable data rate and where no expensive connection is needed for transfer of data from one node to other.

An important point here is that, all the above papers look at a connection as end-to-end connection or no connection at all. In BLE, however, we only have a connection to the peer and multi-hop communication can be seen as a series of connection events. Thus the nodes do not have information about the destination. This makes it expensive to propagate the buffer information down or up the stream. With high cost of connection event and fixed data rate in BLE, we aim towards reducing the number of connections made to the neighbours thus reducing end-to-end latency. Therefore, handshake signals seems a fitting option in this case.

Chapter 4

Bluetooth Low Energy Protocol

BLE wireless technology is a low energy, short range radio technology. It was introduced as a part of the Bluetooth 4.0 core specification and is optimized specifically for small battery operated devices that require almost no power [11]. It is designed to run for years on a single coin cell. BLE supports only simple star topology (piconet) to significantly reduce implementation complexity. With a small silicon footprint, it is ideal for use in indoor automation systems.

For multi-hop communication using BLE, the routing algorithm is implemented over BLE physical and link layer. Thus, it is important to study and understand the capabilities and limitations of the protocol. Hence this chapter explains the BLE protocol standard in detail. We first explain BLE physical layer or radio in brief. The link layer needs to be manipulated properly for using it in multi-hop communication. Therefore, also the link layer is described in more detail.

4.1 Radio

The BLE radio is constrained by the requirement that it should be able to be implemented using the same RF chain as Bluetooth. Though it is based on classic Bluetooth, BLE is not compatible with it. BLE features a new protocol stack with completely new physical and link layer specifications. It uses the 2.4 GHz ISM band with GFSK modulation. AFH is used to make it robust against the interference inherent in the ISM band due to other wireless technologies. BLE also incorporates 24-bit CRC to improve robustness further. It divides the ISM band in 40 channels, each 2MHz wide. Out of these, 3 channels are dubbed as advertising channels and are used for transfer of advertising packets. The rest of the 37 channels are used for data transfer in a connection event. The advertising channels (37, 38 and 39) are placed in a way in the spectrum so as to avoid interference from frequencies used by IEEE 802.11 (WiFi) as these channels fall in between channels 1, 6 and 11 of 2.4 GHz WiFi. To reduce complexity further, frequency hopping is done only on data channels. Rather than having a time-based frequency hopping, frequency is hopped for each transaction. BLE provides an over the air data rate of 1 Mbps.

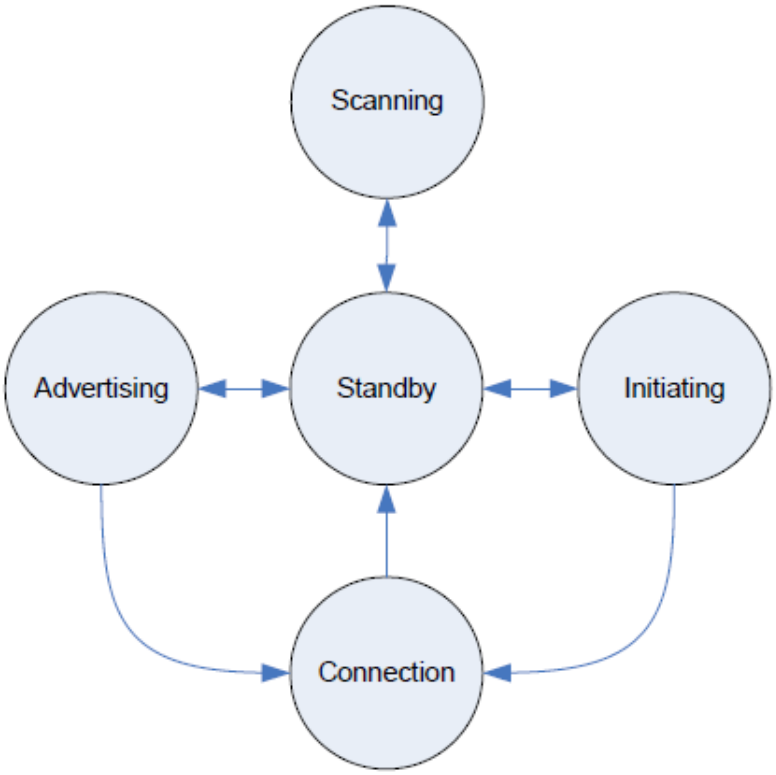

Figure 4.1: BLE Link Layer State Machine [19]

4.2 Link Layer

Similar to the physical layer, BLE protocol specifications present a new link layer. The BLE link layer specification defines 5 states namely Standby state, Scanning state, Advertising state, Initiating state and Connection state in which BLE link layer can work. Figure 4.1 shows the state machine for BLE link layer. BLE allows only one state to be active at a time. When not transmitting or receiving any packet, the link layer should be in the standby state. In the scanning state, the link layer will be listening to advertising packets. The scanning state can be entered from the standby state. The link layer is said to be in the advertising state if it is transmitting advertising packets. This state can only be entered from the standby state. The link layer in the initiating state must listen to advertising packets from a specific device and respond to it by a connection request. It must be entered from the standby state. The connection state is entered when a device establishes connection with another device. It can be entered from the advertising state, acting as a slave or from the initiating state, acting as a master.

Unlike classic Bluetooth, BLE supports a single packet structure for all kinds of payloads. Every packet starts with a preamble used for frequency synchronization and AGC (Automatic Gain Control) training. It is 1 byte in length. The preamble is followed by 4 bytes of access address. Access address acts as a unique id for a particular connection. For

advertising event, access address must always be 0x8E89BED6. For data channel access address is randomly generated but has few restrictions. Actual link layer Protocol Data Unit (PDU) follows the access address, which in turn is followed by 24-bit CRC. Link layer PDU can have any size between 2 to 39 bytes, resulting in a complete BLE packet length between 10 to 47 bytes. Figure 4.2 shows the BLE packet structure.

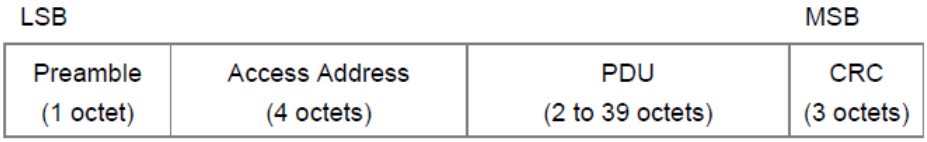

Figure 4.2: BLE packet structure [2]

The link layer PDU is classified into two types viz. advertising channel PDU and data channel PDU. When one of the advertising channels is being used, i.e. when BLE node is working in advertising mode, the link layer PDU must conform to advertising channel PDU. On any other channel, i.e. when data is beign transferred in a connection event, the data channel PDU must be used. Both PDUs have same structure, i.e. they both consist of a 16 bit header at the beginning and the rest of the PDU consists of a payload, however, the bitwise structure of the advertising channel header is different from the data channel header.

There are 7 different types of advertising payloads used in different scenarios. Figure 4.3 lists all the payload types. This section only explains the ones that are used in the multi-hop routing implementation in this project. ADV_NONCONN_IND is non-connectable unidirectional advertising payload. This type is used for broadcasting messages. The payload consists of 6 octets of the advertiser device address and up to 31 octets of advertising data. This is used for route requests in this project. Connectable unidirectional advertising, ADV_IND, has the same payload structure. Although, this type of advertising can be used to establish connection, we use it for transfer of route replies due to its capability to carry advertising data. Connection on the other hand, can also be made using ADV_DIRECCT_IND, i.e. connectable directed advertising. This type of advertising is designed specifically for making connections quicker than ADV_IND. The payload for ADV_DIRECT_IND consists of device addresses of the advertiser (6 octet) and the device to which the advertisement is directed (6 octet) also known (as initiator as it initiates connection event using connection request). Thus, directed connectable advertising payload always has 12 octets of payload. When a device receives the directed connectable advertising PDU, it replies the advertiser with connection request, CONNECT_REQ. A connection request is a type of payload which is responsible for synchronization of clocks, channel hopping sequence and access address for making connections. The CONNECT_REQ payload is 34 octets in size out of which 12 octets are made up of device addresses of the advertiser and the initiator similar to ADV_DIRECT_IND. Rest of 22 octets comprise of LLData fields.

The fields of LLData dictate the parameters related to timing, channel hopping, access

PDU Type	Packet Name
1	ADV_IND
2	ADV_DIRECT_IND
3	ADV_NONCONN_IND
4	SCAN_REQ
5	SCAN_RSP
6	CONNECT_REQ
7	ADV_SCAN_IND

Figure 4.3: Types of advertising channel PDU [2]

address etc. in the connection event. Here, AA stands for access address, which indicates a unique access address for that particular connection. A connection can be established only when both the participating devices change their access addresses from advertising access address to AA mentioned above. CRCInit is used as initial value of CRC in the connection event. Next four fields indicate the timing constraints such as transmit window size and transmit window offset, used to calculate when to listen for new data in the connection. Interval indicates connection interval duration and latency indicates slave latency which allows slave to keep the connection active even if there is no data from sender for some amount of time. 5 octets of ChM describe the channel map of 40 channels where each bit represents a single channel. Along with hop count (5 bit Hop field), ChM is used to calculate next data channel to hop. By sharing his information through connection request, same channel hopping can be ensured. A device can transfer data PDU only when it is in connection state with another device.

The data channel PDU can have either actual data to be transferred from upper layers or it can consist of control data. The data channel PDU header consists of the single bit fields SN (Sequence Number) and NESN (Next Expected Sequence Number) which together act as an acknowledgement mechanism. Another single bit field dubbed as MD (More Data) is used to check whether to close the connection event or not. Other fields length and LLID describe the length and the type of data, respectively. Thus, once a connection is formed, the actual data can be exchanged till MD is set to 0.

The multi-hop routing algorithm developed in this project makes use of advertising channels for route discovery and connection events to transfer data from one node to another. Thus, this method maintains BLE standards and philosophy while bending the utility in innovative ways to achieve multi-hop communication.

Chapter 5

Routing Algorithm for Multi-hop Communication

5.1 Overview

Multi-hop relaying helps to improve the connectivity in WSNs. To achieve the relaying, paths need to be discovered from the source to the destination. This section describes the strategy used for finding routes to distant nodes in the network and then the strategy used for actual data transfer. The routing mechanism used to achieve multi-hop relay is reactive in nature. This means that the paths to destination are not known a priori but are discovered as needed. The routing protocol is inspired from one of the most popular reactive routing protocol, AODV [15] and closely follows its request-reply mechanism. The choice of reactive routing algorithm is due to high overhead in finding routes using advertising mode in BLE as well as based on the intended application of the multi-hop network viz. indoor or industrial automation where the nodes mostly are non-mobile and path once established can be used many times till one of the constituent nodes die out.

Before the data can be transferred to the desired destination, it is necessary to find route from source to destination. This is done by a route request-route reply mechanism. A node seeking route to the destination broadcasts a route request which is captured by its neighbors. The neighbors propagate the broadcast further if they do not have information about the route to the destination. Once request reaches a node which knows route to the destination or which is the destination itself, route reply is sent as unicast. When source receives the route reply, it stores the route in the routing table to be used for data transfer mechanism. All the nodes in the reverse path of route reply also update their routing table for this destination. This mechanism is explained further in sections routing table and route discovery.

Figures 5.1 and 5.2 describe the functionality at each node for sending and receiving. They mainly describe how RREQ-RREP functionalities work, which are described in the subsection route discovery and also gives little insight on the data transfer mechanism. Figure 5.1 describes the steps taken at each node for transmission of the data. If the

entry of destination is already present in routing table, either as neighbours or as possible destination through a particular neighbour, the steps to connection establishment and data transfer are taken. If there is no entry of the destination in the routing table, route discovery is initiated with RREQ broadcast. The functionality for reception in advertising mode is depicted in Figure 5.2. It lists the different steps to be taken when different kind of advertisements are received. If directed advertising is received, the node should reply with a connection request whereas if RREQ is received, it needs to be forwarded or if the node is the destination, it needs to reply with a RREP to the source destination populating reverse path to the destination on the way.

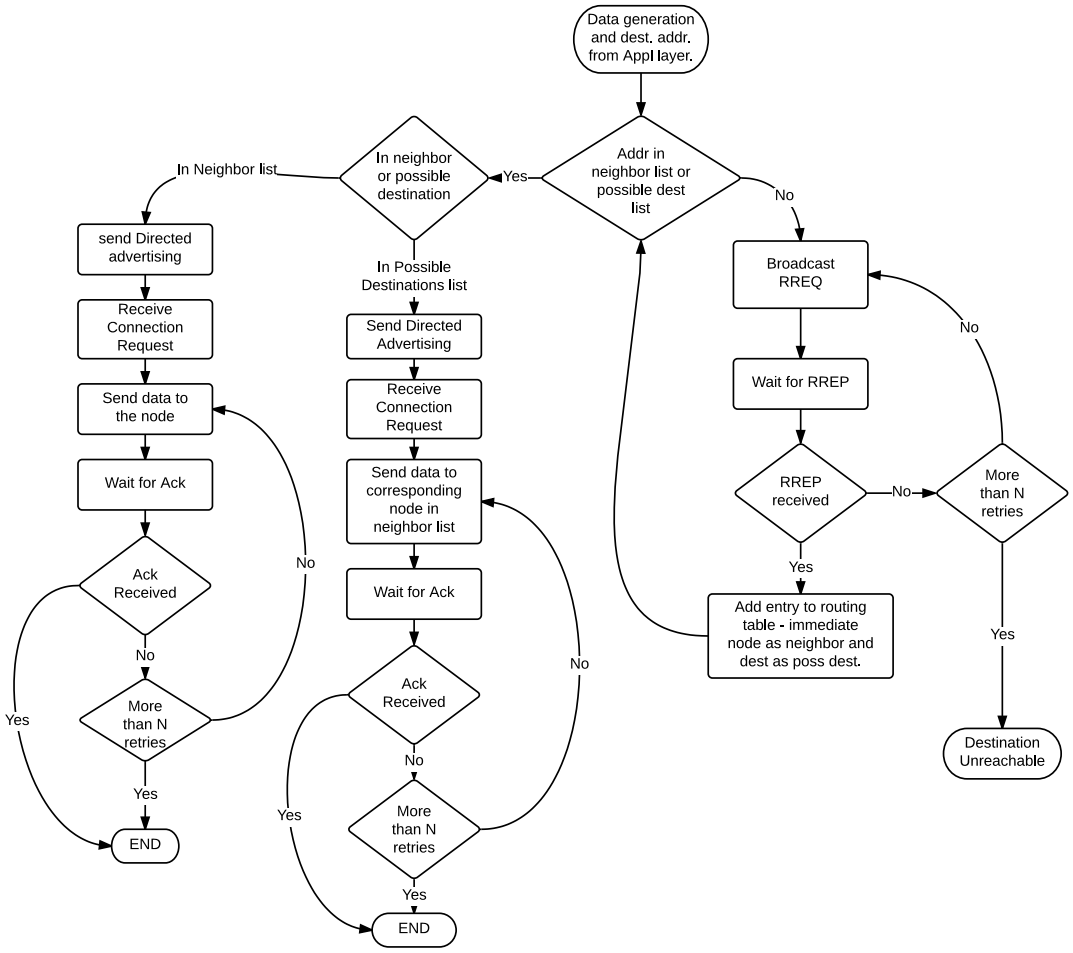

Figure 5.1: Transmit Functionality at a single node

Once a route is established, the source node is ready to send the data. In case of BLE, data transfer takes place with the help of connection events. A connection is established

between neighboring nodes for data transfer. This connection establishment takes place with the help of connection request which provides essential information such as timing constraints and access address to establish the connection. Once the connection is established, the data is transferred between the neighbours. Once the neighbour receives the data and connection event is closed, it repeats the procedure with its neighbor to forward the data. Thus the data transfer in BLE consists of series of connection events through the path discovered by routing mechanism. This mechanism is explained in detail in section 5.4.

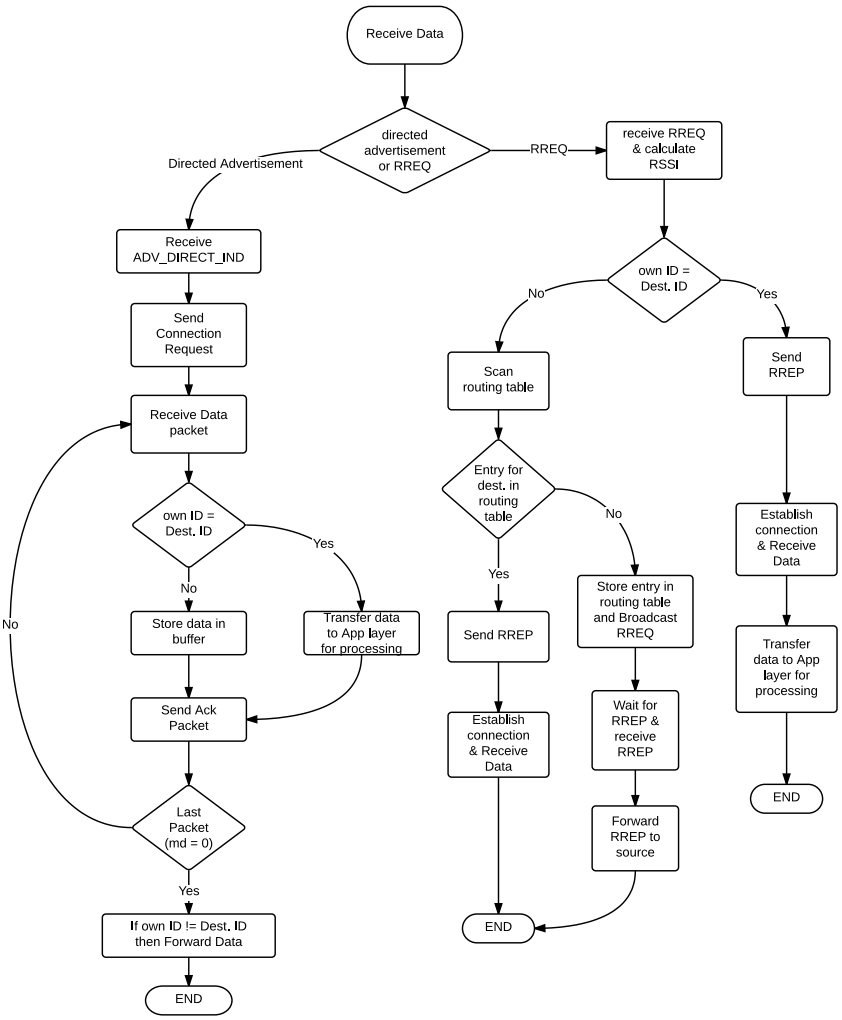

Figure 5.2: Receive Functionality at a single node

5.2 Routing Table

This module is responsible for maintaining records of neighboring nodes and the possible destinations through them. The routing table is implemented as a list where each node has three entries namely neighbor ID, LQI value and a list of possible destinations through that particular neighbor. This structure of routing table differs from the AODV routing table structure in few aspects. As this protocol design is intended for indoor small networks, the sequence numbers are not present in the routing table. Sequence numbers in the original AODV are used for loop avoidance, but as the protocol is designed for small indoor networks, these loops can be avoided and sequence numbers can be omitted to reduce the size and complexity of the route request-route reply mechanism. Also, to make the routing table lightweight, rather than having a new entry for each destination, we club all the destinations from one neighbor in one entry and the destination is added in the list of possible destinations. The functions to search, add and replace a particular entry based on the node ID in either neighbor field or in the possible destination field is needed to manipulate this routing table. The Figure 5.3 shows the basic structure of the routing table.

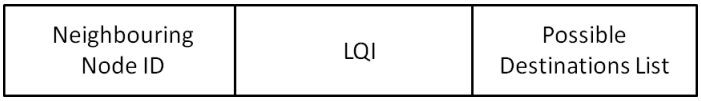

Figure 5.3: Routing table structure

5.3 Route Discovery

To find a route to the neighbors and the possible destinations, we use the route request-route reply mechanism. Here, to minimize the traffic of control signals, we use reactive routing i.e. we only search the route to the node when we need to send some data to it. The structure of route request-reply packet is designed based on the information needed to maintain the entry in the routing table. The structure is shown in Figure 5.4. The packet consists of five fields. Destination address is the network address of the node to which route needs to be found. Source address specifies the address of the node initiating the route discovery. Flag is used to indicate whether the packet is used for route request or route reply (1 for RREQ and 0 for RREP). LQI provides information about link quality which can be used for manipulating transmission power during data transfer. Finally, the relay field is used to indicate the number of hops the data needs to take to reach the destination from the source. This field gives time to live for the route reply in terms of number of hops taken to find the route.

When a node intends to send data to a particular node, it first has to find a route to that particular node (destination node). The source node first searches its routing table for entry of the destination in either neighbors or possible destination node IDs through

Bits: 16	16	1	3	2
Destination Address	Source Address	Flag	LQI	Relay

Figure 5.4: Request-Reply Packet Structure

the neighbors. If no entry is found, route discovery is initiated. For RREQ, flag is set to 1 indicating its a RREQ and LQI and relay fields are set to 0 in the RREQ packet. The packet is then sent as a broadcast. The neighbors receive this broadcast message and check their routing table for an entry of the destination. If no entry is found, the broadcast is repeated by the neighbors. When RREQ is received, the entry of the sending node is added to the routing table. This might lead to a problem where RREQ travels in a loop and the node which broadcast the RREQ might receive it again from the next node broadcasting it causing looping and unnecessary flooding of the network. This can be avoided by having a record of entertained route requests in the node and avoiding modifications and resending due to a route request which is already entertained previously. This flooding and loop avoidance mechanism is not incorporated in this work and can be considered as future work for this implementation.

Once RREQ reaches the destination node or if some intermediate node has entry of the destination in its routing table, route reply (RREP) packet is generated. The packet structure is same as that of RREQ but this time flag is set to 0 and LQI is set to the link quality indication value calculated from RSSI of the received RREQ. The relay field is initially set to 0 and is incremented as RREP propagates towards destination. When RREP reaches the destination, entry is added to the routing table against the immediate node sending the RREP and destination node address is stored in the possible destinations list. This is a typical mechanism used for route discovery using RREQ-RREP.

5.4 Data Transmission - Reception

Once the route entry is found in the routing table, the node is ready to send the data to the destination. In BLE, this is done by establishing a connection event to the relevant neighboring node with destination node in its possible destinations list. This connection event allows both the nodes to act as master-slave pair and thus to transfer the data.

The node initiating the data transfer acts as a slave (advertiser) and sends directed advertisement (ADV_DIRECT_IND) to the receiving node which acts as a master (initiator). The initiator, on receiving directed advertising, sends a connection request (CONNECT_REQ) which specifies parameters like connection event interval, access address for authentication of the connection, among other parameters used for channel hopping and synchronization. Once CONNECT_REQ is received, the advertiser adapts to parameters given in connection request and starts sending data. Since the access address is unique for a particular connection, only the connected node can exchange the data. In one connection event, a node can send several data packets and each packet is responded by an

acknowledgement by the initiator. The acknowledgement mechanism is managed by two bit-fields called sequence number and next expected sequence number in the data header. Once all the data is sent, the connection is closed when timeout occurs.

Once connection is closed, the intermediate node is ready to forward the data to next node. It then searches the destination address in its routing table. The node entry should be present in the routing table as it must be updated during route discovery session. Then, the neighbor establishes the connection with the next node fetched from route table which leads to the destination or in case of only one hop, the destination itself. The data is thus forwarded to the relevant node in a series of the connection events.

Thus, as can be seen, the data transfer in this BLE multi-hop is performed by a series of connection events between the nodes in the routing path rather than end to end connection set-up. The data is also forwarded in terms of large chunks rather than single packets to reduce the number of expensive connection establishment events. This feature, which is ideal for peer-to-peer communication, might slow down the multi-hop communication.

5.5 Implementation of Multi-Hop Routing

The multi-hop routing mechanism explained in the previous section needs to be tested with implementation on actual hardware. Thus, an experiment was conducted on custom PAN radio boards developed in-house at IMEC. The aim of this experiment was just to confirm the functionality of the routing and data transfer mechanism i.e. to confirm that with some changes, BLE can be used for multi-hop data transfer. This section describes the set-up and outcome of the implementation done on the hardware.

For multi-hop link using BLE, the routing algorithm is implemented over the physical and link layer. The link layer needs to be manipulated properly for using it in multi-hop communication. The implementation is done on a Cortex-M0 ARM processor [23] available on the PAVO 2.2 board, a custom hardware developed at IMEC. Sourcery CodeBench Lite baremetal (without OS) toolchain [7] is used for the development of the software. Segger j-Link JTAG debugger [17] is used for programming and debugging whereas serial communication takes place via Teensy v3.0 [1]. A list of libraries developed for these particular boards are used and the routing algorithm is built over them. The libraries over which we develop the routing are as follows:

libdbb_ble.a and libdbb_ctrl.a - These libraries support the digital baseband hardware and drivers for the same.

libsrr_bsp.a and libsrr_drv.a - These libraries are responsible for the board drivers and board support packages.

libllc_ble.a - This library describes the link layer and provides LLC_API for the development of upper layers.

The routing and multi-hop data transfer is implemented over the provided LLC API. This LLC API is analogous to the link layer in OSI model [20] whereas NW API described

[1] https://www.pjrc.com/store/teensy3.html

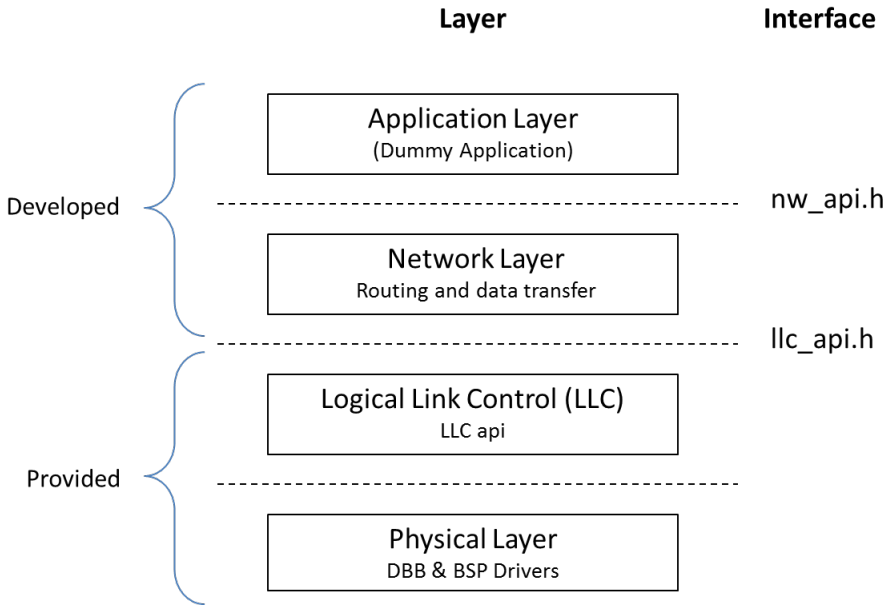

Figure 5.5: Different layers in the application

later in this section acts as a network layer. The NW API provides an interface 'nw_api.h' which can be used by application layer. Application layer is completely independent from the network layer and the only requirement is that it must be able to inject data into the network layer. This implementation uses a dummy application which injects the dummy data to the network layer at fixed time interval. Figure 5.5 visualizes the implementation as OSI model.

LLC API

Here, we discuss the link layer API on the top of which the routing is implemented. This API is developed to support BLE standards over the physical layer which comprises of drivers for the board. The implementation of the LLC can be divided into 6 subsections based on the functionality of each function in it.

Setup - The setup contains functions which are used for setting up and initializing the LLC. These are used only once at the start of the program.

Protocol Timing - This section has functions which describe various timers. We can set values to different timers such as LLC_TIMER_RX_TIMEOUT which sets given value to a timeout for receiving the data after which the receiver will stop listening to the channel.

Time Stamp - Time Stamp section has functions used for getting timestamps at different times such as current timestamp or timestamp of last transmitted and received messages etc.

Control - This part covers various functions used for miscellaneous control functionality such as getting RSSI value, setting channel for channel hopping, setting access address

for data communication and setting communication channel.

Transmit - This section contains functions which are related to transmission of data and callback function for interrupt when the transmission is complete.

Receive - The receive section provides functions which describe receiving of data such as receive start or set receive buffer along with callback function for interrupt when reception is complete.

NW API

NW API is responsible for routing and data transfer to the destination. It is built on top of LLC API and uses LLC API functions to send packets to peers. In a sense of networking stack, LLC API acts as a link layer and NW API acts as a network layer over which application is built. In this project we use a simple application which periodically injects data into the network layer for transmission to a particular destination node.

The NW API consists of 3 modules namely nw_util or network utilities which consists of helping functions for conversion and comparison, nw_mgmt or network management which manages the routing table and the main module called nw_data or network data which manages and manipulates the data like transmission-reception of the data, storing data in the buffer and routing.

Network Management - This module is responsible for management of routing information. It consists of a routing table and functions to manipulate the same. The routing table is defined as a linked list of routing entries. Having a routing table as a linked list allows better memory management and avoids unnecessary pre-allocation of resources. The routing table entries contain 3 fields. The first entry is the address of the neighboring node which can be communicated directly without any hop. The second field is the Link Quality Indicator (LQI) which reflects quality of the link between two neighbouring nodes. To reduce the number of bits representing LQI, the values are partitioned into 8 levels and appropriate level is assigned to the entry based on RSSI values in RREP packet. The RSSI value is calculated at the neighbouring node based on the RREQ packet signal strength. The LQI field can be used to adjust the transmission power in data transfer to reduce the power consumed. The third field is list of nodes that can be reached through that neighboring node. This list, labelled as the Possible Destinations List, is also implemented as a linked list so that it is easy to add or modify the list of reachable nodes. The structure of the routing table is shown Figure 5.6.

The network management module is divided in two parts. The table search functions section has functions to search the routing table. The functions can search the whole routing table for a particular node address in neighboring nodes as well as in possible destination lists of each node. The table modification functions section has entries to add or update entries to the routing table.

Network Data - The network data module is the main module which has functions for route request-route reply (RREQ-RREP) mechanism as well as for data transmission. This module has 5 subsections namely callback functions, routing related functions, transaction related functions, entry point function and helping functions. The first section,

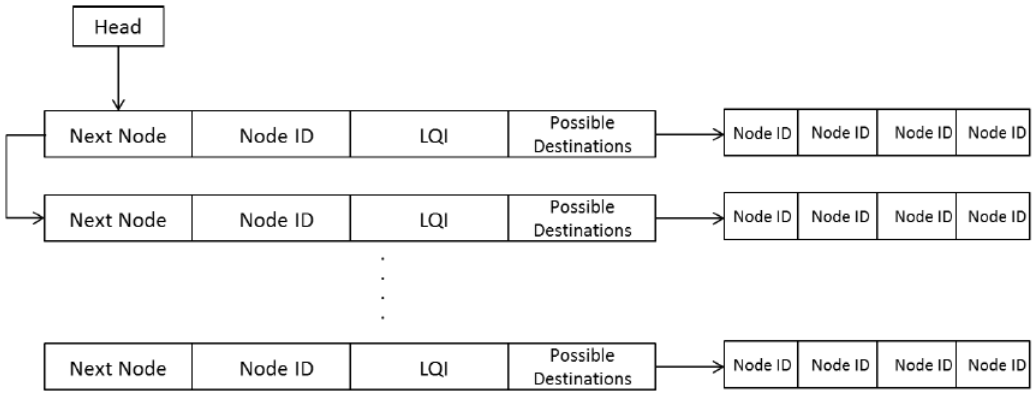

Figure 5.6: Structure of routing table

callback functions, contains functions to be called on receiving an interrupt for end of data transmission or reception. The callback functions for data are also responsible for managing acknowledgement(ack) and negative acknowledgement(nack) mechanism. This is done with the help of bit-fields called sequence number (tSN) and next expected sequence number (nESN) present in the data packet header as per the BLE standard. When receiving data, nESN is incremented by 1 (toggled between 0 and 1) in case of valid data. Also, data is considered valid if tSN and nESN bits in receiving packet header are equal. When sending data to the initiator the tSN bit is incremented by 1 in case of new data is being transmitted. The valid acknowledgement is identified by different values of tSN and nESN (ack) in acknowledgement packet. If they are equal (nack), the packet needs to be retransmitted.

The next section, routing related functions, contains functions responsible for RREQ-RREP mechanism. The RREQ is sent as a broadcast using nonconnectable BLE advertising (ADV_NONCONN_IND). The RREP is sent using nondirected connectable advertising (ADV_IND). This section includes functions to populate the RREQ packets and RREP packets according to the BLE standard. It also includes the functions to manage the received RREQ and RREP which basically are used to decide the forwarding of RREQ and RREP and to add the entries to the routing table if necessary. Finally, it includes the function to actually sending and reception of RREQ-RREP packets.When data needs to be sent from one node to another, the node first searches for the destination node entry in its routing table. If no entry is found, then node relies on RREQ-RREP mechanism.

Third section, transaction related functions, contains all the functions which are involved in establishing the connection and sending data to the neighboring node. In any data transfer in BLE, when a node wishes to send data, it needs to send advertisement packet to relevant neighbor. This is achieved by using directed advertising (ADV_DIRECT_IND). Once the intended peer receives directed advertisement, it responds with a connection request (CONNECT_REQ). This CONNECT_REQ contains information like connection event interval, access address for authentication etc for the peer. The node which sends CONNECT_REQ goes into the Initiator state in LLC. The node receiving CONNECT_REQ

i.e. the node which sent directed advertisement, goes into the Advertiser state in LLC and sets the parameters it received in the connection request. When this is done, the connection is said to be set up.

Once the connection is set up, the nodes are ready for data communication. When first packet (ack) is received by advertiser, the connection is said to be established. The data is transferred in CONTINUOUS mode provided by LLC API. The continuous mode allows us to automate the transaction once started, with appropriate Inter Frame Spacing (IFS). The data transfer takes place packet-wise and each packet reception is responded by acknowledgement packet from initiator with ack-nack mechanism as described above.

The next section act as entry point into NW API. These function are provided in the interface of the API which can be called from application. The application injects data into the network layer. The data injected is detected by main duty cycle function and is then processed based on destination address.

5.6 Experimental Evaluation

It was observed that though multi-hop communication is feasible using BLE, it is a complicated process as multi-hop communication in BLE is essentially a series of connection events or peer-to-peer data transfers. Rather than transferring the data packet by packet from source to destination, one has to repeatedly make and close connection events increasing the overhead and complexity of implementing multi-hop communication. The connection based data transaction provides security and robustness in peer-to-peer communication whereas it turns out to be a bottleneck for implementation of multi-hop data transfer. Thus, it can be said that even though it is possible to have multi-hop communication using BLE, it proves to be not a fruitful approach due to overheads in end-to-end data transfer and complexity of the implementation. This is mainly because BLE standard dictates that it is purely a peer-to-peer communication protocol designed for low data rate requirements.

The aim of this implementation mainly was to test the feasibility of multi-hop communication over BLE. We were able to implement and successfully test individual components such as route detection, routing table management and individual data transfer but the complete integrated solution was not successfully tested. This was due to strict timing constraints of BLE for peer-to-peer communication, which could not be met for multi-hop data transfer. Due to limited amount of time and boards available, these timing constraints could not be met to make the implementation completely compliant with BLE standard. This led us to testing the buffer allocation policies, explained in the next chapter, in a high level simulation environment.

Chapter 6

Buffer Management Strategy

6.1 Overview

As seen in the chapter 5, BLE uses connection events to transfer the data from one node to another. Therefore, to transfer data over multiple hops, a series of connection events needs to take place. Here a node which has received the data needs to store it until it is able to make a connection to another node. This means a BLE node needs to store considerable amount of data before it can forward the same. As the nodes in wireless sensor networks are constrained in aspects of memory and processing power, one needs to take into account the limited amount of memory available for such communication. In such scenario, a node needs to be designed in such a way so as to manage sent data.

BLE is designed in such a way that a node can handle only one active connection at a time. If receiving node provides the whole available buffer to only one node, the other nodes willing to send data to the same node do not get chance to deliver their data causing starvation of nodes and biased allocation of resources. Thus, while receiving data from multiple nodes, a node needs to manage its buffer space in such a way that each node gets chance to forward its data and is not starved in case each node needs to forward large amount of data.

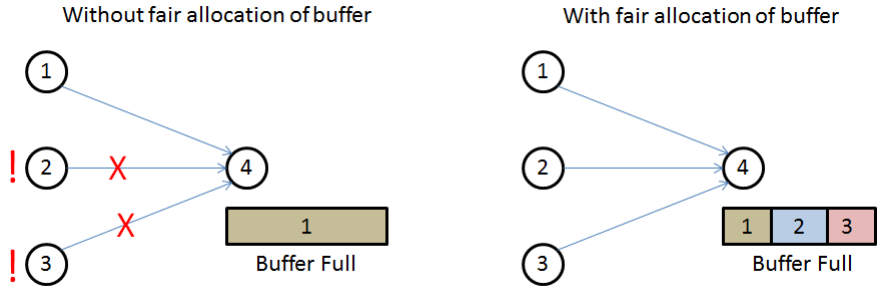

Figure 6.1: Straightforward vs fair allocation of buffer

Figure 6.1 describes the scenario and need of buffer management algorithm in such

scenarios. As illustrated in the left side of the figure, if the data to be sent from each node is large enough, complete buffer space will be utilized for storing data of one node. The other nodes on the other hand will not get any buffer space left to forward their data. In such cases, these nodes either will forward data which will be dropped at reception or will be denied of making connections to the receiving node. In both cases only one of the nodes will be able to forward the data while others starve. On the other hand, as illustrated in the right-hand side figure, with the help of appropriate buffer management policy, fair allocation of available buffer space can be facilitated, which will increase fairness in the network.Therefore, to manage multiple connections in a complicated, multi-hop network where nodes have limited amount of memory space and to efficiently forward data with fair chance to each participating node, a good buffer management policy is necessary.

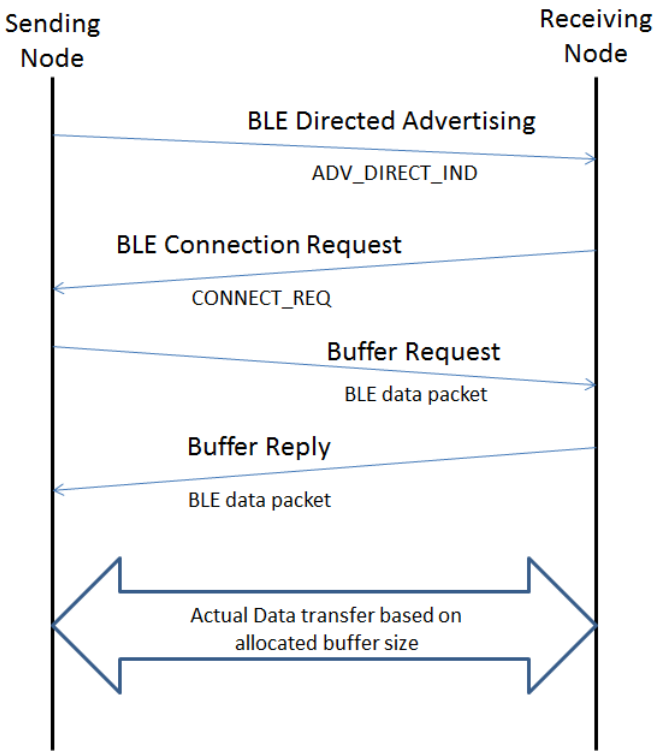

Figure 6.2: Sequence diagram for data transfer when buffer management policies are used

In this chapter, we introduce two buffer management strategies which allocate the available buffer space to incoming connection based on different criteria. Both the algorithms allocate the buffer based on information exchanged in handshaking signals just after establishing connection. The sequence diagram in Figure 6.2 describes the overall sequence followed in the system for data transfer. This diagram assumes the route is already established using routing algorithm explained in previous chapter and only describes the sequence from making connection to the end of data transfer. Once the connection is established, the node willing to send data asks for the size of buffer space the receiving node can offer. This is done with buffer request-buffer reply mechanism where we exchange the

requested (by the node willing to send) and allocated (by the receiving node) buffer sizes respectively. After these handshake signals are exchanged, data transfer takes place. The size of data transferred is equal to the allocated buffer sizes communicated via buffer reply to the node willing to send.

6.2 Buffer Allocation Based on Average Number of Connections

In this method, the buffer is partitioned based on the number of active connections with the receiving node. Once the network is running, each node stores the entries of all the nodes directly communicating with it in a table, called allocation table henceforth. As the node runs through the cycles of sending and receiving data, it calculates the average number of entries in the allocation table. This is done by keeping the track of previous average and number of connection events handled so far. The initial average is set as a default value N over which average of the subsequent iterations is calculated. Thus, if there are fewer entries in allocation table constantly over time, the average converges to less than N. If a node has fixed number of active connections over time, then the average will converge to that particular number and will allocate the available space in the buffer into those many connections equally.

Rather than assigning the full calculated buffer space, we assign a fraction of it (70%) to the requesting node. This mechanism is used to avoid assigning full buffer if there is only one active connection over time. In this case, if there is another connection due to sporadic data generation at some node, it can be quickly accommodated i.e. the remaining fraction can act as a reserve space. This mechanism is efficient in transfer of data where the number of connections do not frequently change over time. This is because, in such case, once the average value converges to number of active connections, it will stay constant without depending on the type and source of data. One of the shortcomings of this method of allocation can be slow response to changes in the network topology and number of active connections due to sporadic data generation. But this shortcoming can be ignored in our case of networks with non-mobile nodes as the application of this work is intended to be indoor and industrial automation where there is negligible change in the network topology over time.

Algorithm 1 describes the workflow of this buffer allocation strategy. ReqSize represents the size requested by the node willing to send data via buffer request. The allocated buffer size is calculated at the receiving node based on the algorithm explained below. The receiving node then replies the allocated size (AllocatedSize) for the particular node via buffer reply.

This algorithm calculates the average number of entries in allocation table over time. Each node stores average calculated till that connection event. As the time passes, the average converges to the number of nodes trying to connect to the receiving node. The value of avg stabilizes over time if the number of nodes trying to send data stays constant.

Algorithm 1 Buffer allocation based on average number of connections

1: Initialize $avg_1 = 6$, $num_1 = 1$ & $resfactor = 0.3$ at the beginning
2: **loop**
3: Receive connection request (CONNECT_REQ)
4: Receive buffer request (BufferReq) & store requested buffer size (ReqSize)
5: Calculate sum
$sum = avg_n \times num_n + length(AllocationTable)$
6: Increment num
$num_{n+1} = num_n + 1$
7: Calculate avg_{n+1}
$avg_{n+1} = \frac{sum}{num_{n+1}}$
8: **if** $avg_{n+1} \geq 1$ **then**
9: $avg_{n+1} = ceil(avg_{n+1})$
10: **else**
11: $avg_{n+1} = round(avg_{n+1})$
12: **end if**
13: Calculate allocated size
$TempAllocatedSize = \frac{AvailableSpace \times (1 - resfactor)}{avg_{n+1}}$
14: $AllocatedSize = min(ReqSize, TempAllocatedSize)$
15: **if** No allocation table entry for requesting node id ($nodeId$) **then**
16: add entry ($nodeId$,$ReqSize$,$AllocatedSize$) to allocation table
17: **else**
18: update $AllocatedSize$ in the corresponding entry against $nodeId$ in routing table
19: **end if**
20: Send buffer reply (BufferRep)
21: Receive data of $AllocatedSize$ amount
22: Update allocation table entry for requesting node id with
$ReqSize_{n+1} = ReqSize_n$ - $AllocatedSize$
23: Delete allocation table entry for requesting node id if $ReqSize_{n+1} = 0$
24: **end loop**

6.3 1-by-k Buffer Allocation Scheme

This method is similar to *1/k-buffer* solution proposed by Shigang Chen and Na Yang in [5]. Their paper focusses on assigning *1/k* times available space to each connecting node. We modify this method to adapt to the way data is transferred in BLE multi-hop communication.

In the method we propose here, an incoming connection request is assigned *1/k* times available buffer space. The value of k is initialized to default value in the beginning. If the same node starts another connection event, we reduce value of k by 1 if the number of entries in the allocation table is less than initial default value of k i.e. assign bigger portion of available space to that particular node. If the number of entries in allocation table is more than initial value of k, its value is increased by 1 i.e. smaller portion of available space is assigned to that particular node. The value of k is increased in such a way that it does never exceed the number of entries in allocation table. Thus, in this method, the value of k tries to converge to the current number of entries in the allocation table. In this method as well, we reserve some portion of the buffer (30%) for congestion and deadlock avoidance. As the buffer entry is updated each time the data is received, allocation entry is removed from allocation table if the data transfer is complete for the particular node. In such case, if the same node wishes to transfer the data again after some time, it will again be assigned *1/k* times the available buffer space because there will not be any entry in allocation table for that node. This helps in keeping the allocated buffer sizes in sync with the active connections to the receiving node. In this method, we combine the received data from the same node over time (if it is not already forwarded) and forward it further based on available buffer space at next hop node. This would help in reducing number of connection events if the next hop node has larger free buffer space. The algorithm is described as follows.

Algorithm 2 describes the workflow of this buffer allocation strategy. ReqSize represents the size requested by the node willing to send data via buffer request. The allocated buffer size is calculated at the receiving node based on the algorithm explained below. The receiving node then replies the allocated size (AllocatedSize) for the particular node via buffer reply.

This algorithm assigns a fraction of available buffer space (*1/k*) to the node willing to send the data. The value of k is varied so as to assign smaller or larger fraction of available buffer space to a particular node based on the state of the buffer and number of times a node tries to transfer the data to the receiving node. This algorithm converges the value of k to the number of connections entertained by the particular node at that moment. Since the value of k is incremented or decremented by 1 at a time, this algorithm is more sensitive to the changes in number of connections and does not stabilize over time as opposed to the buffer allocation based on averge number of connections.

Algorithm 2 1-by-k buffer allocation mechanism

Initialize $resfactor = 0.3$ at the beginning
loop
  Receive connection request (CONNECT_REQ)
  Receive buffer request (BufferReq) & store requested buffer size (ReqSize)
  Calculate $ThresholdSpace = MaxBufferSize \times resfactor$
  if *AllocationTable* is not empty **then**
    if entry exists for requesting node id (*nodeId*) in allocation table **then**
      if $AvailableSpace > ThresholdSpace$ **then**
        if $length(AllocationTable) < k_n$ **then**
          $k_{n+1} = k_n$ - 1
        else if $length(AllocationTable) > k_n$ **then**
          $k_{n+1} = k_n + 1$
        else
          $k_{n+1} = k_n$
        end if
      else
        $k_{n+1} = k_n$
      end if
    else
      $k_1 = 6$
    end if
  else
    $k_1 = 6$
  end if
  Calculate allocated size
    $TempAllocatedSize = \frac{AvailableSpace \times (1 - resfactor)}{avg_{n+1}}$
    $AllocatedSize = min(ReqSize, TempAllocatedSize)$
  if No allocation table entry for requesting node id (*nodeId*) **then**
    add entry (*nodeId*,*ReqSize*,*AllocatedSize*,*k*) to allocation table
  else
    update *AllocatedSize* and *k* in the corresponding entry against *nodeId* in routing table
  end if
  Send buffer reply (BufferRep)
  Receive data of *AllocatedSize* amount
  Update allocation table entry for requesting node id with
    $ReqSize_{n+1} = ReqSize_n$ - *AllocatedSize*
  Delete allocation table entry for requesting node id if $ReqSize_{n+1} = 0$
end loop

Chapter 7

Performance Evaluation and Results

In chapter 6, we discussed theory behind buffer allocation strategies. The performance of these developed algorithms needs to be evaluated with concrete experiments. This chapter explains the experiments conducted. We conducted a set of experiments to compare the performance of different buffer allocation policies introduced in the previous chapter. These experiments were aimed at determining better algorithm among Average Number of Connections buffer allocation and *1/k* buffer allocation in terms of end-to-end latency for data transfer. We developed a high level simulator in MATLAB to simulate the functionality of buffer allocation and conducted tests on different topologies ranging from simple data aggregation topology consisting 4 nodes to complex topology consisting tens of nodes. The following section describes the simulation model developed for conducting these experiments. Then we next discuss the result obtained from various tests conducted in this simulation.

7.1 Simulation Model

In this simulation, we were concerned with the end-to-end latency of the whole data as well as allocation of buffer at relaying nodes, therefore the simulator was designed to model these aspects rather than simulating the transfer of each frame in BLE data transfer. Also, each node was defined only to model the buffer space in the node and other details like processing time and power consumption was ignored. The data transfer was modelled only to calculate the delay in data transfer, thus we decided to develop a high level model. We modelled the behaviour of lower layers with simple probabilistic models that allow a fast simulation with accurate enough results. Though the simulation could be made more accurate by implementing the lower layers in detail, it would have taken much more time to simulate the models of all the layers in detail with not much increase in the accuracy as the end-to-end delay in this case mainly depends on the allocated buffer size and number of connection events taken to transfer the complete data. Therefore, high level simulation model was designed focussing more on the buffer allocation algorithms while keeping the emulation of nodes simple.

The paths were assumed to be static and also it was assumed that nodes do not die out. So, once the paths were found using routing algorithm in the setup phase of the network, they did not change. The assumption was based on the intended application areas of this work such as industrial data logging and indoor automation. As the nodes in such applications are usually fixed and have constant power supply, it is safe to assume static, non-dying paths. This assumption enabled us to reduce the complexity of the simulation by running the routing algorithm only once at the beginning of the simulation. We used the same routing algorithm explained in prior chapters to find the routes based on neighbouring nodes and possible destinations through those nodes. Apart from setting up the routes, the initial network setup included populating the data to be sent at the sources as well as initial states of the buffer allocation mechanism.

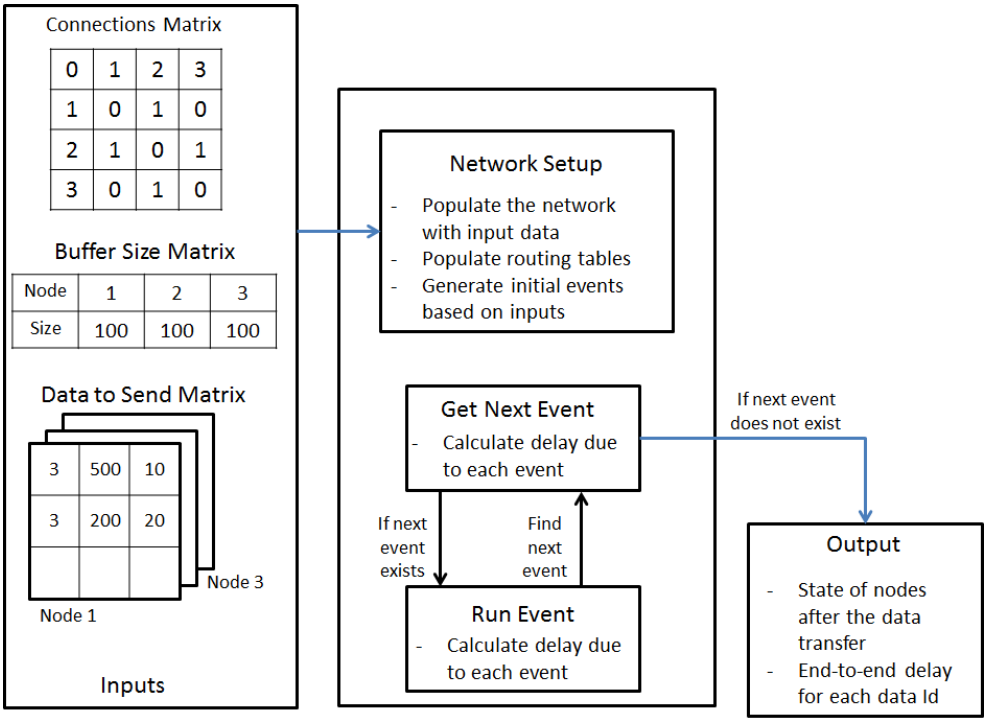

Figure 7.1: Block diagram describing input, process and output of the simulator

The simulator was designed as an event driven simulator, meaning, rather than running continuously over time and checking for events, the simulation jumps from event to event in chronological order. Three matrices namely the connection matrix, which depict connections between neighboring nodes, the buffer space available at each node and a matrix providing information about generated data i.e. its size, destination and generation time, act as input. This mechanism helps in providing a flexible topology as an input and also facilitates handling of different size of data generation at different times in each source node. The block diagram in Figure 7.1 illustrates the input, processing and output in brief.

The events include events to generate the data (GenerateData) based on the given

inputs as well as events to manage the transfer of data once the data is generated. The list of events to manage data transfer consists sub-events such as ConnectReq, BufferReqStart, BufferReqEnd, BufferRepStart, BufferRepEnd and events for the actual data transfer namely DataTxStart, DataTxEnd and DataRxStart, DataRxEnd. Figure 7.2 describes the workflow of the simulator which is explained in detail in the following text.

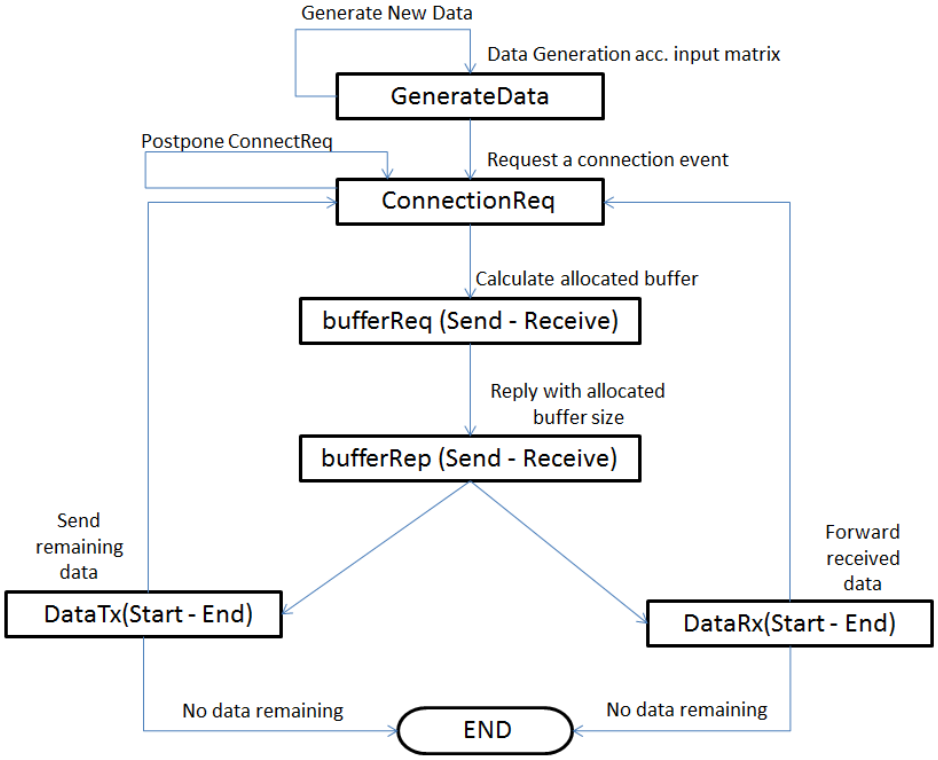

Figure 7.2: Workflow of simulator based on the event types

The ConnectReq event models setting up connection event between peer nodes which include sending directed advertising to the receiving node and the receiving node replying with a connection request. As the node can entertain an incoming ConnectReq event only if it is not in communication with another node, a busy flag was assigned to each node. A node could accept the ConnectReq only if its busy flag is 0. This prevented a node from having more than one connections simultaneously. The node could entertain only a single communication at a time as per BLE protocol. If the node receiving ConnectReq is found busy, i.e. its busy flag is set to 1, the node sending it would postpone the connectReq by 5 ms. This was done to emulate the re-attemt of setting up connection event by the node in case of failure to do so when the receiving node is busy. If the node accepts connection request, it models the setting up of the connection event. The time for this is modelled using uniformly distributed random multiple of 0.625 ms in the range of advertising interval. Since we were more interested in allocation of buffer space, the advertising interval was set to maximum of 1 sec and minimum of 20 ms. A successful ConnectReq event would trigger BufferReq event representing the buffer request handshake signal.

The BufferReq and BufferRep events acts as handshake signals between the peers for exchanging the size of data that can be accomodated by the receiving node. Buffer request is sent by the node willing to transfer the data indicating the amount of data it needs to transfer. This is simulated by assigning the size of data to be sent as requested buffer size in the BufferReqSend event. The reception of buffer request at the receiving node is modelled as BufferReqReceive event where the size of allocated buffer is calculated based on the buffer allocation strategy used. The allocated buffer size is communicated to the sending node via buffer reply signal. This is simulated with the help of BufferRepSend and BufferRepReceive events at either of the nodes. The node willing to send is assigned with the allocated size in BufferRepReceive event which in turn triggers events for data transfer at both the nodes.

The events for data transfer model the time taken to transfer the data based on the size of allocated buffer and taking into account the data rate of BLE and a probability of packet error rate. We include a factor representing packet drops or invalid acknowledgements. We set this factor to 0.1 i.e. the calculated time for data transfer is divided by 0.9 to accommodate delay due to resending of the data packets. Connection event for forwarding the data is triggered at the end of the data tranfer at the receiving node while connection event is triggered at the sending node if there is still some data remaining to be transferred. This completes one cycle of events and continues till there are no events left in the event queue of any node.

7.2 Simulation of Buffer Allocation Policies

The developed simulation model was used to test the performance of two buffer allocation strategies. We considered two topologies for testing and comparing these strategies. Each of the topology was tested for two scenarios. The first topology represents data aggregation. This means many nodes try to send data to single sink node thus causing bottleneck at the relay nodes close to the sink. This is the basic building block in any complex topological scenario. The basic topology was also selected so as to confirm the proper functionality of the simulator. The second topology is a more complex topology which represents a network of 25 nodes resembling closly to real indoor/industrial networks. In this work, only non-mobile network topologies are tested. This is dictated by the intended applications of the work, i.e. indoor-industrial data logging or automation where once the network is deployed, it does not usually change the topology. All nodes are assumed to be active throughout the experiment.

Topology 1

This topology represents data aggregation which is typical characteristic in any realistic network. Data aggregation means data getting concentrated at a particular node, which is typically the case at sink nodes. Figure 7.3 shows the structure of the topology. It is evident that there are three paths namely, 1-3-4-6, 2-3-4-6 and 5-4-6, which converge at

different nodes. The nodes in red i.e. nodes 1,2 and 5 are source nodes whereas node in black i.e. node 6 is a sink node. These simulations were run for both buffer management strategies and then their performance was compared. We tested this topology for two scenarios.

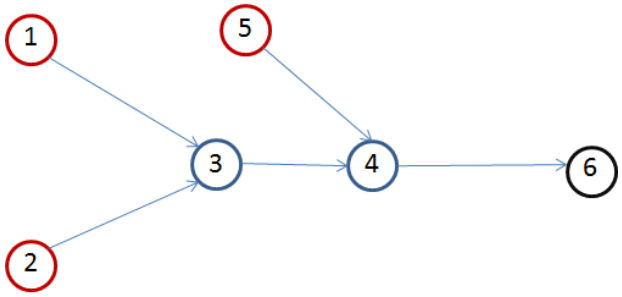

Figure 7.3: Topology of the first experimental setup (Topology 1)

First experiment consisted continuous data transmission in which each node sent data continuously to the destination. In this case, each node had 10kB of buffer whereas it produced 50 kB of data every 200 sec. Figure 7.4 shows the behaoviour of both the buffer allocation strategies. The red marker indicates buffer allcoation based on average number of connections and the blue marker indicates *1-by-k* buffer allocation strategy. This convention is followed throughout the experiment. The plot shows average, maximum and minimum end-to-end delays over all data items generated at each source node. The plot also shows standard deviation of the end-to-end delays over the same data set.

As can be seen here, the latency for buffer allocation based on the average number of connections is lesser than that for *q-by-k* buffer management. This can be justified as the allocation algorithm based on average number of connections 'remembers'the average number of connections whereas *1-by-k* allocation resets the value of k once the complete data is transferred as this value of k is based on the newness of the data and the connection.

The plot in Figure 7.5 shows the performance of both the algorithms for sporadically generated data. In this particular case, nodes 1 and 5 generate data at continuous intervals simillar to previous experiment, but node 2 only generates data sporadically at certain times. This is simulated by generating few data at node 2 at sporadic intervals.

Here as well, the allocation based on average number of connections performs better in terms of end-to-end latency. It can be observed that end-to-end latency for node 5 is reduced. This is because, now, there are less number of nodes willing to send data to node 4 and thus, node 4 can allocate more buffer to node 1 and 5. It can also be seen that the maximum end-to-end delay for node 1 and 2 is high compared to average end-to-end delay. This is the case when data is generated simultaneously at node 1 and 2. This causes node 3 to allocate less space to both the nodes causing spike in the end-to-end delay.

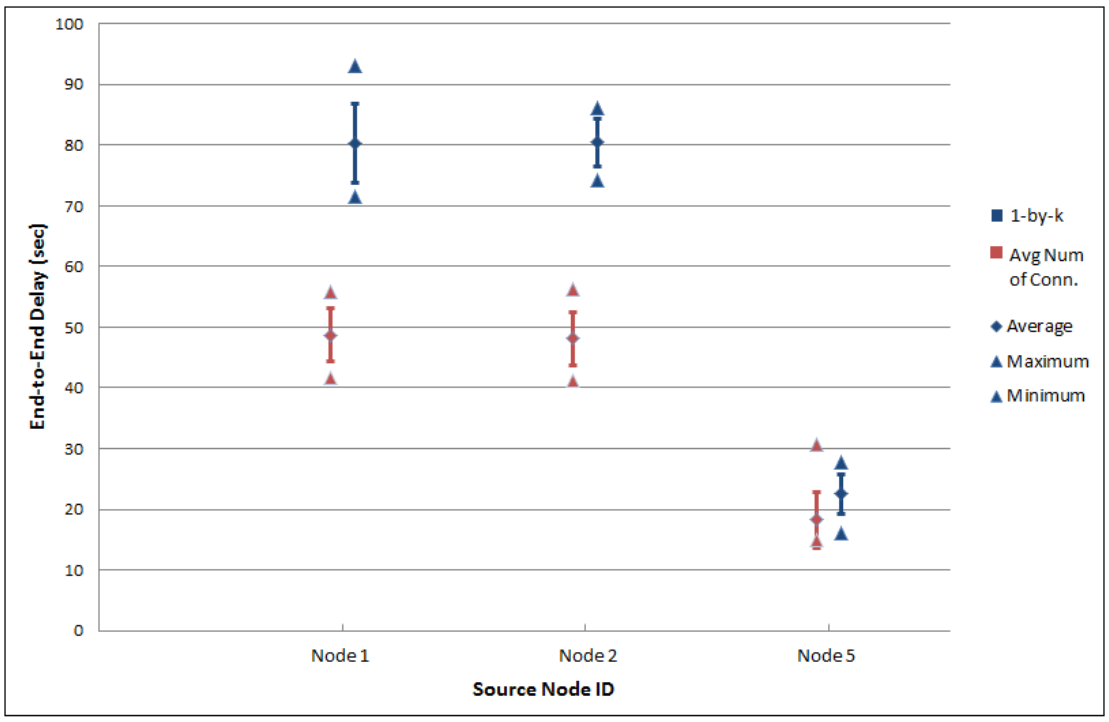

Figure 7.4: Performance of algorithms for continuous data generation in Topology 1

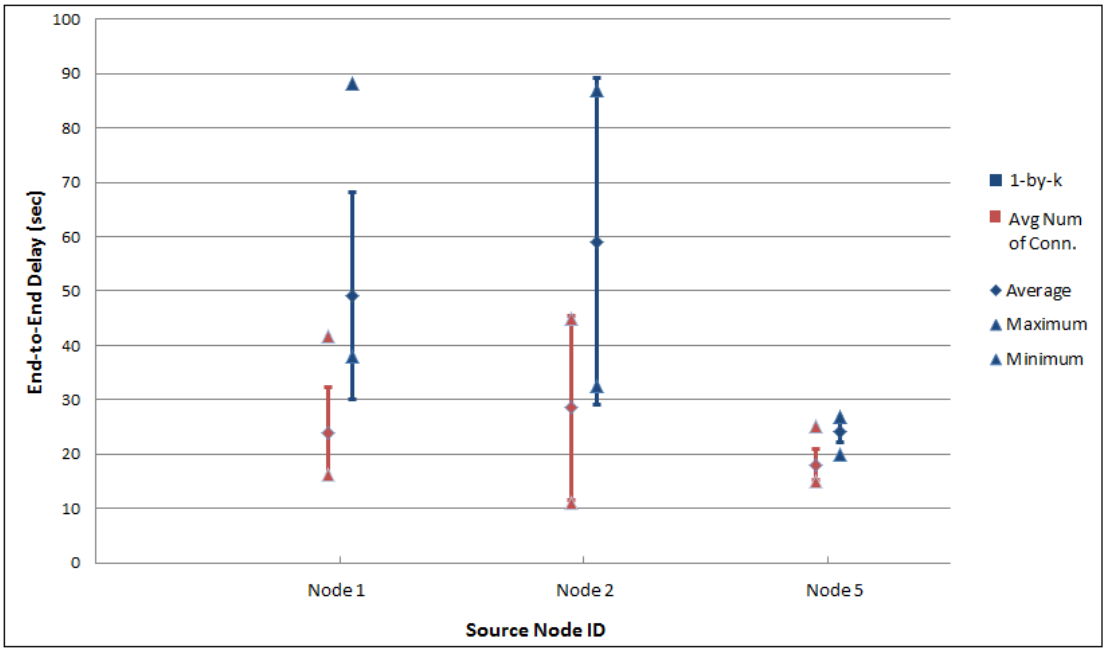

Figure 7.5: Performance of algorithms for sporadic data generation in Topology 1

Topology 2

The second topology is a complex topology consisting of 25 nodes with each path crossing the other paths. This topology represents data aggregation as well as cross-communication between the nodes. With having more than one connection through many nodes, we can analyze the way buffer management policies perform under heavy load circumstances. There are 6 paths in the network which are listed as follows. The topology structure can be seen in 7.6.

Path1 - 7-6-5-4-3-1 (black)
Path2 - 2-3-4-11-12-13 (dark blue)
Path3 - 8-9-10-21-22-23 (green)
Path4 - 14-15-10-11-12-13 (yellow)
Path5 - 17-16-21-20-19-18 (light blue)
Path6 - 24-5-4-11-10-21-22-25 (red)

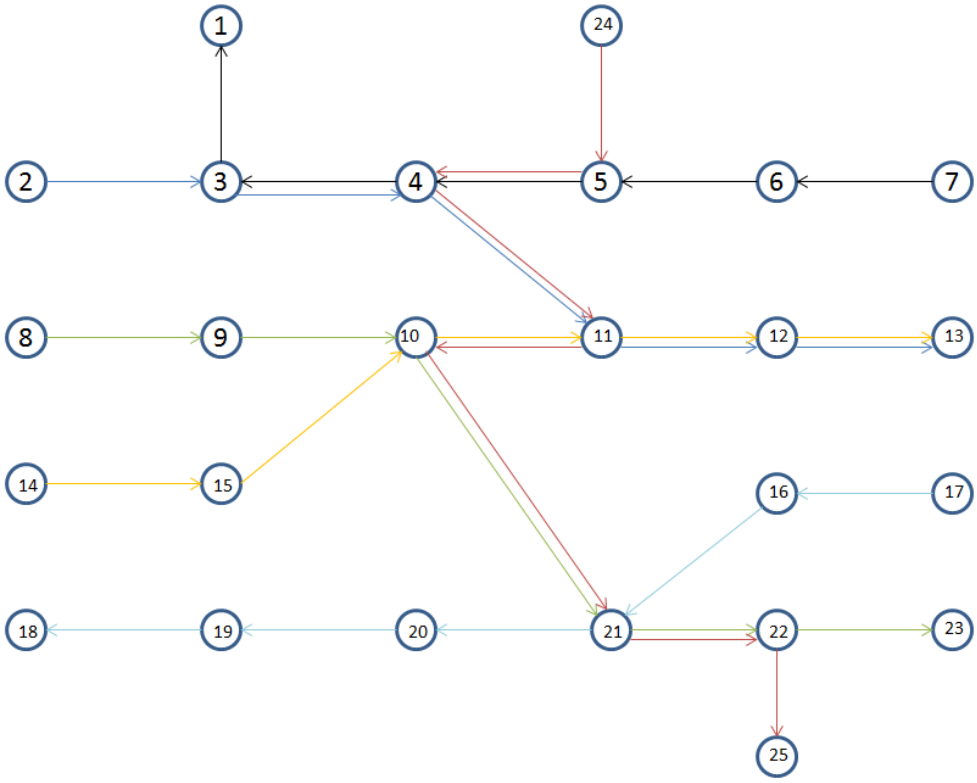

Figure 7.6: Topology of the second experimental setup (Topology 2)

The first experiment on this topology consisted of continuous data generation at each source node. Thus in this case, each path simultaneously sent the data towards different destinations. As there are many paths crossing each other, many nodes have to entertain the data coming from different paths and manage the buffer space in such a way so as not to let the network saturate. This scenario can be an example of an emergency case where

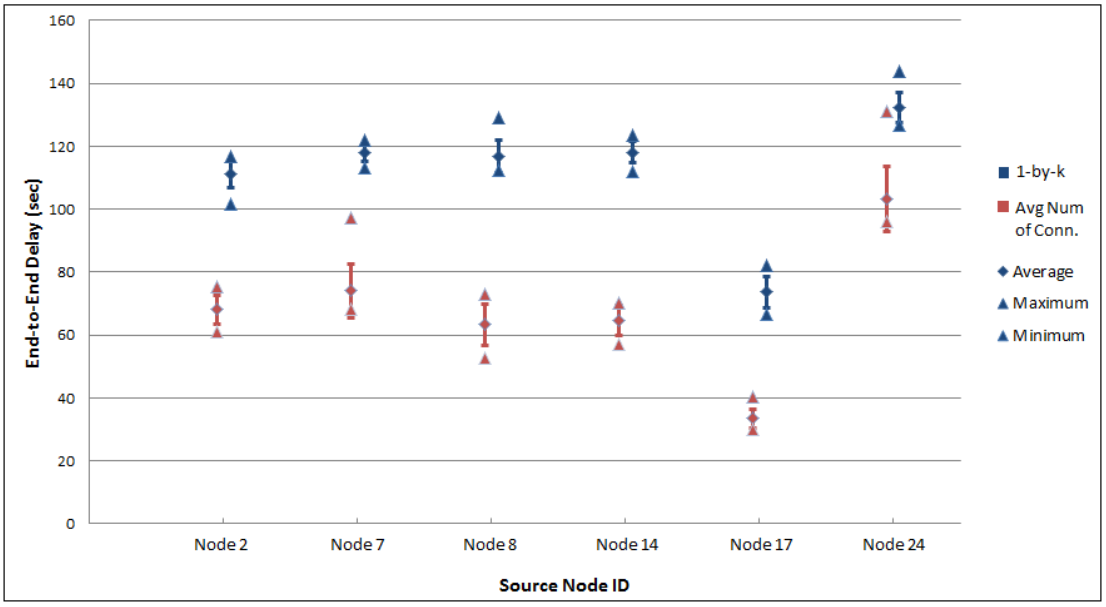

Figure 7.7: Performance of algorithms for continuous data generation in Topology 2

all nodes start sending data at the same moment, causing the network to be saturated.

Figure 7.7 depicts the performance of both the buffer allocation strategies when there is continuous data generation at all the source nodes. As the network gets congested due to high traffic, the end-to-end delay of data at each node increases. It can be observed here too, that *1-by-k* buffer allocation method performs worse as compared to buffer allocation based on average number of connections. But on the contarary, it is observed that the values of end-to-end delay do not deviate much from the average value. This is observed because with all the nodes sending data simultaneously, the netowek reaches steady state and thus end-to-end delay does not deviate much. Path 6 has the highest number of hops and thus has highest end-to-end delay. As can be seen, due to higher maximum delay, the standard deviation has high value whereas minimum value is close to the average value. On the other hand, as Path 5 is the least resistance path, it shows least end-to-end delay.

The other scenario consist of 3 nodes continuously generating data, same as the continuous topology, and 3 nodes generating sporadic data at intermediate times. In this test case, nodes 7, 14 and 24 generate data sporadically at different intervals and nodes 2,8 and 17 generate data at continuous intervals of 200 sec. The plot in Figure 7.8 shows the performance of both the buffer algorithms in such setting. The generation time for node 14 and 24 some times coincide with the data generation of nodes 7,14 and 24. This is the reason for the high maximum end-to-end delay at these nodes. The nodes generating data continuously are also affected by these nodes and the data transfer along that path is slowed down. On the other hand, node 7 always generates data at different times than the other source nodes and thus, does not have to share the buffer space and ontermediate nodes. This results in quickest data transfers. As path 6 has more hops than other paths, it can be observed in both the scenarios that it has higher value of end-to-end latency. In

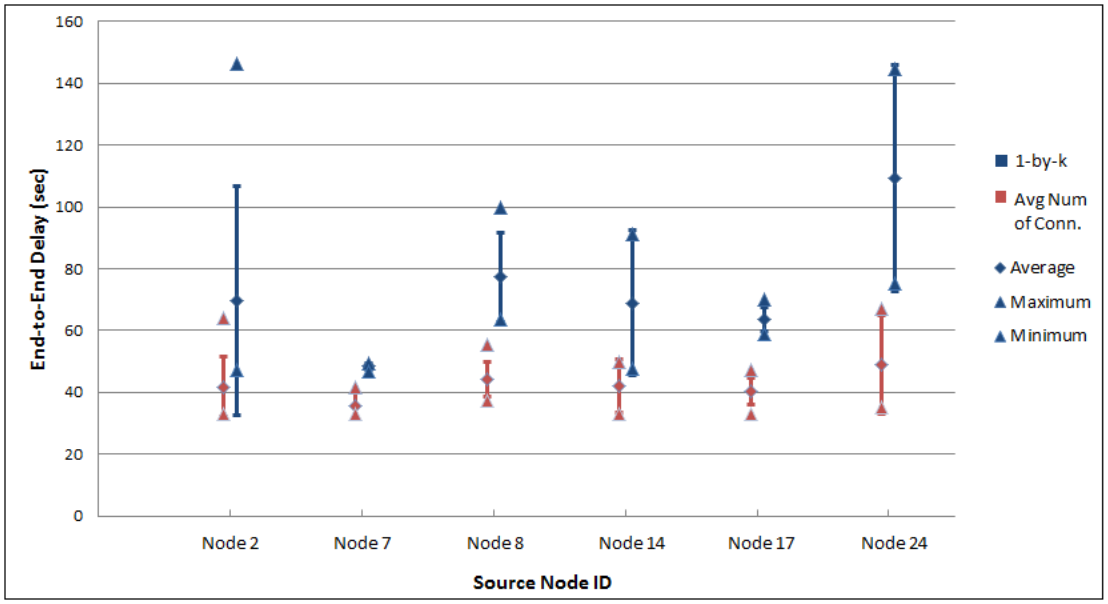

Figure 7.8: Performance of algorithms for sporadic data generation in Topology 2

general, it can be said that higher number of connection events result in higher end-to-end latency in the path.

It can be inferred that buffer allocation based on average number of connections performs better and gives lower end to end delays in different topologies and data generation scenarios. However, the latency for transferring 50 kB of data over 6 to 8 hops with relay nodes having 10 kB memory space is in range of tens to hundresds of seconds which is high delay for many practical purposes even in low data rate networks. This is due to high latency involved in establishing the connection in BLE. Thus, it also can be inferred that BLE is not a very good choice for multi-hop data transfer.

Chapter 8

Conclusions and Future Work

The main objectives of this work are to investigate the possibility of multi-hop communication over Bluetooth Low Energy (BLE) by implementing the same on the custom hardware developed at IMEC and to introduce an efficient buffer allocation strategy for resource constrained sensor nodes in such networks. It provides a fair buffer space allocation to participating nodes while trying to minimize end-to-end latency for data transfer in the network.

The implementation of multi-hop communication was done on PAVO v2.2 boards available at IMEC. The routing and data transfer mechanism was developed as a 'network layer'over BLE physical and logical link control layers. It was observed that though multi-hop communication is feasible using BLE, it is a complicated process to use BLE for such communication mainly because BLE standards dictate that it is purely a peer-to-peer communication protocol designed for low data rate requirements. We were able to implement and successfully test individual components such as route detection, routing table management and individual data transfer but the complete integrated solution remains yet to be tested due to limited amount of time available.

One of the important future work related to the implementation of a multi-hop BLE network is to perform integration testing with different topologies and test the system as a whole. Also, to have successful data communication over provided lower layer APIs, we needed to relax the timing constraints. Thus, another work for future is to profile and optimize both, LLC API and NW API to remove the delays and uncertainties in timing and to make the implementation more compliant with the BLE standards.

As far as we know, this was the first attempt to provide buffer management strategy for nodes in multi-hop BLE networks. As described in the related work, both works on multi-hop communication over BLE, described in [9] and [12], do not focus on buffer management at individual nodes and assume the nodes have sufficient amount of available memory for such communications. Thus to have a fair comparison, we introduced two buffer management strategies namely 'buffer allocation based on average number of connections'and '*1-by-k* buffer allocation strategy'. The experiments were conducted as a MATLAB simulation scenario for various topologies. The network topologies with non-mobile nodes were chosen because the work is aimed at applications such as indoor and industrial data logging

or sensing where nodes do not move once deployed in the network. It was observed that buffer allocation based on average number of connections performed better and provided smaller end-to-end delays in all the tested cases. This observation can be justified as the allocation based on average number of connections 'remembers'the number of connections over time whereas the value of k in *1-by-k* allocation starts at default value for each new connection.

As the simulator modelled to test the effects of buffer allocation strategies, the implementation of simulator can be termed as high level simulation. The simulator did not implement individual nodes with their complete BLE protocol stacks. This would introduce some inaccuracies as packet drops and effects of phenomena such as interference at radio layer were ignored. Therefore, a more detailed modelling of BLE nodes for such simulations can be one of the works to be conducted in future.

Bibliography

[1] Jon T Adams. An introduction to ieee 802.15.4. In *Aerospace Conference, 2006 IEEE*, pages 8–pp. IEEE, 2006. 1

[2] SIG Bluetooth. Bluetooth specification-core version 4.0. *Jun-2010.[Online]. Available: https://www. bluetooth. org/docman/handlers/downloaddoc. ashx*, 2010. 1, 11, 12

[3] Eugene Chai, Mun Choon Chan, and Akkihebbal L Ananda. Coverage aware buffer management and scheduling for wireless sensor networks. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, pages 100–108. IEEE, 2006. 7

[4] Prof. Jitendranath K. M. Sundeep Kumar Challa Kiran, Babu TS Manoj. Congestion control using adaptive buffer flow managements in wsn. 7

[5] Shigang Chen and Na Yang. Congestion avoidance based on lightweight buffer management in sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 17(9):934–946, 2006. 7, 27

[6] N Golmie, O Rebala, and N Chevrollier. Bluetooth adaptive frequency hopping and scheduling. In *Military Communications Conference, 2003. MILCOM'03. 2003 IEEE*, volume 2, pages 1138–1142. IEEE, 2003. 2

[7] Mentor Graphics. Sourcery codebench lite edition. *May-2011. [Online], Available:http://www. mentor. com/embeddedsoftware/ sourcery-tools/sourcerycodebench/ editions/lite-edition/*, 2011. 18

[8] Jose A Gutierrez, Marco Naeve, Ed Callaway, Monique Bourgeois, Vinay Mitter, and Bob Heile. Ieee 802.15. 4: a developing standard for low-power low-cost wireless personal area networks. *network, IEEE*, 15(5):12–19, 2001. 6

[9] Marko Haatanen et al. Self-organizing routing protocol for bluetooth low energy sensor networks. 2012. 6, 7, 38

[10] Dynastream Innovations Inc. Ant message protocol and usage. *Apr-2014. [Online], Available: www.thisisant.com/resources/ant-message-protocol-and-usage/*, 2014. 1

[11] Nicole Lee. Bluetooth 4.0: What is it, and does it matter? *Jul-2014. [Online], Available: http://reviews.cnet.com/8301-19512_7-20116316-233/bluetooth-4.0-what-is-it-and-does-it-matter/*, 2011. 9

[12] Konstantin Mikhaylov and Jouni Tervonen. Multihop data transfer service for bluetooth low energy. In *ITS Telecommunications (ITST), 2013 13th International Conference on*, pages 319–324. IEEE, 2013. 6, 7, 38

[13] Ramesh Nagarajan. Threshold-based congestion control for the ss7 signaling network in the gsm digital cellular network. *Vehicular Technology, IEEE Transactions on*, 48(2):385–396, 1999. 7

[14] BBC News. Bluetooth rival unveiled by nokia. *4-Oct-2006.[Online], Available: http://news.bbc.co.uk/2/hi/technology/5403564.stm*, 2006. 1

[15] Charles Perkins, Elizabeth Belding-Royer, Samir Das, et al. Rfc 3561-ad hoc on-demand distance vector (aodv) routing. *Internet RFCs*, pages 1–38, 2003. 13

[16] connectBlue AB Rolf Nilsson. Bluetooth low energy technology - the optimal solution for wireless sensors and actuators. *Jul-2014. [Online], Availabla:http://www.connectblue.com/press/articles/bluetooth-low-energy-technology-the-optimal-solution-for-wireless-sensors-and-actuators/*, 2012. 2

[17] ARM Segger, J-Link. Api,. *Users guide of the J-Link application program interface (API), Version*, 3. 18

[18] Hnin Yu Shwe, Wei Peng, Haris Gacanin, and Fumiyuki Adachi. Multi-layer wsn with power efficient buffer management policy. *Progress In Electromagnetics Research Letters*, 31:131–145, 2012. 7

[19] Bluetooth SIG. Technical considerations for bluetooth smart application developers. *May-2014. [Online], Available:https://developer.bluetooth.org/DevelopmentResources/Pages/Bluetooth-Smart-Optimizations.aspx*, 2014. 10

[20] William Stallings. *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards*. Macmillan Publishing Co., Inc., 1987. 18

[21] NIST-FIPS Standard. Announcing the advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197, 2001. 2

[22] Chieh-Yih Wan, Shane B Eisenman, and Andrew T Campbell. Coda: congestion detection and avoidance in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 266–279. ACM, 2003. 8

[23] Joseph Yiu. *The Definitive Guide to the ARM Cortex-M0*. Elsevier Inc., 2011. 18