

Submitted in part fulfilment for the degree of BEng.

Real-Time Communication Guarantees over Mote Runner

Jonathan Derrick

1 May 2018

Supervisor: Leandro Soares Indrusiak

Word count: 14,099 (using Microsoft Word's word count)

Page Count: 45 (using Microsoft Word's page count)

Of which the title page, abstract, acknowledgements, table of contents, bibliography and all appendices (provided for completeness) are omitted.

Abstract

The main issues faced by Wireless Sensor Networks within industry are introduced as well as two commonly used communication protocols. A generic specification of data transfers is defined and the project centers around the specification and implementation of a new *Time-Division-Multiple-Access* based protocol, *LikeWHART* to satisfy it. Investigations into timeslot optimization given a defined payload size as well as *LikeWHART*'s integrity are conducted. Based on the experimental data, there is no mathematically representable relationship between the payload size and its required timeslot size – only that in general an increase in payload size requires an increase in the timeslot of the network. The management of and response to external interference by *LikeWHART* is shown to be minimal. The *Fixed Priority Scheduling* and *Smallest Transfer First* orderings are introduced to the reader and are given their own response-time models for use within *LikeWHART* based on network parameters. The concept and value of the superframe *Synchronisation Constant* is introduced and found through experimentation. *Fixed Priority Scheduling* and *Smallest Transfer First* are compared with the former being shown as well generalized to larger sets of data transfers with constrained temporal requirements due to its linear growth in response times compared to that of *Smallest Transfer First*'s exponential growth of values. The proposed response time models are shown to be accurate when compared to measured data from the instance of a network.

Statement of Ethics

One investigation was purposefully attempting to disrupt the communications of a functioning wireless sensor network. This was performed within a simulated environment to avoid interfering with the functioning networks or devices of the University Department and was conducted purely to demonstrate the integrity of the network rather than how to disrupt it. It is believed that the reader will not use this technique to intentionally disrupt any operational wireless sensor networks.

There were no other ethical implications to be considered for this project.

Acknowledgements

To my fantastic project supervisor Leandro Soares Indrusiak who has always been on hand to guide me through this subject and has given me so much drive and determination to do my best this year. I have thoroughly enjoyed our weekly meetings. To my close family who have always supported me and endured the 5 hour drive far too many times. To Glandular Fever for completely destroying my chances of graduating with a first and still making life difficult 18 months on.

1 Table of Contents

2	INTRODUCTION	8
3	LITERATURE REVIEW	9
3.1	WIRELESS SENSOR NETWORKS	9
3.1.1	<i>Protocol.....</i>	9
3.1.2	<i>Interference.....</i>	9
3.1.3	<i>Collisions.....</i>	10
3.2	EXECUTION PLATFORMS FOR MOTES.....	10
3.2.1	<i>TinyOS.....</i>	11
3.2.2	<i>IBM Mote Runner.....</i>	11
3.2.3	<i>Comparisons.....</i>	11
3.3	WIRELESS SENSOR NETWORK PROTOCOLS.....	12
3.3.1	<i>IEEE 802.15.4.....</i>	12
3.3.2	<i>ZigBee.....</i>	13
3.3.3	<i>WirelessHART.....</i>	14
3.3.4	<i>Comparisons.....</i>	15
3.4	REAL-TIME SCHEDULING	16
3.4.1	<i>Fixed Priority Scheduling (FPS).....</i>	17
3.4.2	<i>Response Time Equation.....</i>	17
4	SPECIFICATION	19
4.1	THE PROBLEM & CASE STUDY	19
4.2	GOALS.....	20
5	IMPLEMENTATION DESIGN	21
5.1	CHOICE OF EXECUTION PLATFORM.....	21
5.2	CHOICE OF NETWORK PROTOCOL TYPE	21
5.3	LIKEWHART	21
5.3.1	<i>Synchronisation.....</i>	22
5.3.2	<i>Avoiding Internal Interference.....</i>	22
5.3.3	<i>Managing External Interference.....</i>	22
5.3.4	<i>Ensuring Delivery with Packet Acknowledgements.....</i>	22
5.3.5	<i>Frame Structure.....</i>	23
5.3.6	<i>Slot Tables.....</i>	23
6	TIMESLOT MOTIVATION & PROCESS DESIGN.....	24
6.1	WHY FIND THE OPTIMAL TIMESLOT?	24
6.2	VARYING THE PAYLOAD SIZE.....	25
6.3	EXPLORING THE RELATIONSHIP	25
7	SCHEDULING MOTIVATION & POLICY DESIGN	26
7.1	WHY SCHEDULE THE SUPERFRAME?	26
7.2	THE SYNCHRONISATION PERIOD	26
7.3	IS IT SCHEDULABLE?	26
7.4	FIXED PRIORITY SCHEDULE (FPS)	27
7.5	SMALLEST TRANSFER FIRST (STF).....	27
7.6	RESPONSE TIME EQUATIONS	27
7.7	SCHEDULING ORDER	28

8	LIKEWHART IMPLEMENTATION.....	29
8.1	TIMESLOTS, SUPERFRAMES AND OFFSETS	29
8.2	SYNCHRONISATION	30
8.3	PACKET ACKNOWLEDGMENT	30
8.4	CHANNEL HOPPING	31
8.5	USE OF LEDS	31
9	EXPERIMENTAL PRACTICES & DESIGN	32
9.1	USE OF SIMULATION ENVIRONMENT	32
9.1.1	<i>Limitations</i>	32
9.2	HARDWARE AND SOFTWARE	33
9.3	TIMESLOT SIZE OPTIMIZATION	33
9.4	FINDING THE SYNCHRONISATION CONSTANT	33
9.5	RESPONSE TIMES	34
9.6	EXTERNAL INTERFERENCE	34
10	TESTING OF IMPLEMENTATION.....	35
11	RESULTS	37
11.1	TIMESLOT OPTIMIZATIONS.....	37
11.1.1	<i>10 Byte Payload</i>	37
11.1.2	<i>20 Byte Payload</i>	37
11.1.3	<i>30 Byte Payload</i>	37
11.1.4	<i>40 Byte Payload</i>	38
11.1.5	<i>50 Byte Payload</i>	38
11.2	COMPUTATION OF THROUGHPUTS.....	38
11.3	SYNCHRONISATION CONSTANT	38
11.4	THEORETICAL RESPONSE TIME COMPUTATIONS	39
11.4.1	<i>Fixed Priority Scheduling Response Times</i>	39
11.4.2	<i>Smallest Transfer First Response Times</i>	39
11.5	MEASURED RESPONSE TIMES	39
11.5.1	<i>Fixed Priority Scheduling Response Times</i>	39
11.5.2	<i>Smallest Transfer First Response Times</i>	40
11.6	EXTERNAL INTERFERENCE	40
12	DISCUSSION	41
12.1	IMPLEMENTATION	41
12.2	TIMESLOT OPTIMIZATION	41
12.3	SCHEDULING	42
12.3.1	<i>Synchronisation Constant</i>	42
12.3.2	<i>Fixed Priority vs Smallest Transfer First</i>	43
12.4	EXTERNAL INTERFERENCE	44
12.5	GOAL EVALUATION	45
13	CONCLUSIONS.....	47
14	FURTHER WORK.....	49
14.1	NEAR FUTURE	49
14.2	CONSIDERABLE FUTURE	49
15	BIBLIOGRAPHY	51
16	APPENDIX A: DATA TRANSFER GENERATOR.....	53

17	APPENDIX B – NETWORK MANAGER CODE	54
18	APPENDIX C – NETWORK DEVICE CODE.....	56
19	APPENDIX D – TESTING SCREENSHOTS	60

2 Introduction

Wireless Sensor Networks (WSNs) are increasingly being used for remote monitoring in a variety of different scenarios [1] [2]. However, the communicating devices within WSNs have limited (battery) power and memory resources and therefore protocols devised to manage these systems must balance the resources available whilst also providing stringent security and integrity guarantees about the data it handles [3]. Currently, there is no defined standard for WSN communication protocols and as such there are many variants to choose from when creating a new network.

In this project we develop our own standardized protocol by taking inspiration from two commonly chosen protocols for WSN management and comparing their operating methodologies. As WSNs generally conform to real-time monitoring systems, the basic methods of scheduling and response time analysis are also briefly considered, which could be applied to a WSN to produce models to give real-time guarantees about its operation. Having considered the current options within the field of WSN protocols, we shall begin to design and implement a new WSN protocol within IBM's *Mote Runner* [4] execution platform based on the functionality of what we have found. Having formed the operating methodologies into a working implementation we shall then go onto assess the integrity and operation of the network to determine whether it is fit for purpose for use within industry.

To test the efficiency of the new protocol the analysis shall be divided into three key aspects. First the optimization of the network's parameters shall be investigated within a simple network using the protocol to provide a solid base on which to conduct the other portions of our analysis. We shall then build models which describe the response times for data transfers scheduled within the protocol and will compute and compare these to actual measured operational values, based on a specification of randomly generated data-transfers that need to be satisfied. The final investigation shall then be into the integrity of the network by conducting a series of tests involving external interference and observing how the protocol responds in order to assure successful delivery of transmissions to their destination.

Having completed this analysis on our new protocol a review into the accurateness of the response time models, optimal network parameters and the management of external interference shall be conducted. Conclusions shall then be drawn on the effectiveness of the protocol and its implementation and further works shall be suggested that could be carried out to further refine the protocol's specification.

3 Literature Review

3.1 Wireless Sensor Networks

A wireless sensor network (hereafter WSN) is a collection of interconnected devices which measure some physical property within their environment and communicate using a wireless connection. This network of connected devices/nodes communicate with a central, *sink node/base station* via a radio wave based communication methodology [5].

WSNs consist of anything between a few and several hundred devices, with a tight constraint on their available resources; in order to keep manufacturing costs low [5]. This class of devices, known as motes, are characterised by their restrained resources, typically running on an 8 or 16-bit microcontroller with between 1-2KB of memory and 16-48KB of non-volatile storage [3].

The simple operation of any sensor node with a WSN is to sense a change in environment and send it to the sink node/base station, which acts as an interface to the host system. There are numerous ways of defining how these nodes communicate, but due to their resource constraints and factors that can affect normal operation, specialised protocols must be used to ensure efficient operation of the network [5].

3.1.1 Protocol

A protocol defines a set of rules and procedures that devices must follow in order to communicate over some network. They “govern the end-to-end” [6] communication of data between nodes (devices) of a network.

3.1.2 Interference

When two devices communicate wirelessly, they transmit and receive data within the same electromagnetic frequency range, known as a channel. Transmissions can be heavily affected by interference which can be unintentional: from other nearby networks, natural thunderstorms, microwave ovens; or intentional: from hackers trying to maliciously intercept the transmission data. [7]

Channel hopping is an automated process whereby communications vary the frequency band on which they transmit over time. There are two variations of this: 1) *Reactive*: only when there is an issue detected on the channel, such as lack of transmission acknowledgement, is the channel changed to another; or 2) *Proactive*: transmissions are made on differing channels every time, chosen pseudo randomly [7].

By changing channels proactively, it makes the wireless network less-susceptible to jamming attacks from prospective hackers, making the network more robust and safer in an industrial environment. There are also implementations whereby channels that are known to have heavy interference are blacklisted, meaning that they are not able to be chosen for use, again improving the end-to-end reliability of communications. [7]

3.1.3 Collisions

Whilst ensuring that no external factors can influence the transmissions within a wireless network is important, it does not deal with issues of internal interference, caused by its own transmissions. When two transmissions occur at the same time on the same channel, they overlap with each other, and the result is a garbled signal – a message that is not useful to any device on the network. A *collision* is said to have occurred [8] .

This is illustrated in figure 1, which shows a simple network with three devices A, B and C with their respective transmission ranges, represented by colour co-ordinated circles. Since each network device is contained within the other's transmission circles, it is almost certain that collisions will take place when transmissions occur simultaneously on the same channel.

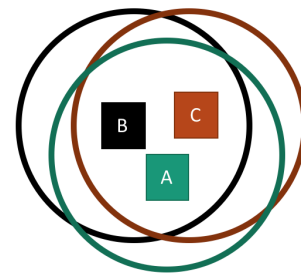


Figure 1 Overlapping transmission ranges

When a collision occurs, the frames involved must be retransmitted at a later period in time, thereby reducing the overall efficiency of the network [8]. In a time-sensitive sensor network, collisions can cause deadlines to be missed which can subsequently lead to a catastrophic event.

The network protocol is responsible for managing its own internal interference, using a scheme which prevents communications from overlapping with each other. This basic functionality can be done within a number of ways and we shall consider two separate protocols in 3.3 that manage it with distinct approaches.

3.2 Execution Platforms for Motes

When developing for motes, the loading and deletion of programs is a near impossible task without the help of tools that have a high level of abstraction, “shield(ing) the application from (the) hardware details” [3]. This can be provided by one of numerous virtual machines or operating systems that simplify the development process, hence reducing time and costs.

3.2.1 TinyOS

TinyOS is an operating system for use with low power wireless devices and runs entirely on an event driven programming model. Developed in nesC, which has a C-like syntax, it works on the basis that applications are made of two constructs, Components and Interfaces. Components handle both the specification of functions the application uses (from a library) and implements itself and also their implementations. The interfaces are two-way sets of functions that are either provided by or used by a component. Components are linked together using the interfaces, forming an application [9].

3.2.2 IBM Mote Runner

IBM's Mote Runner [4] is an efficient, stack-based architecture, virtual machine for sensor networks and small embedded devices which have a limited number of resources to use. It supports event driven programming using any strictly typed high level language, although the main support is for Java and C# through a comprehensive toolchain which automatically compiles into typed bytecode for execution. Typed bytecode is used to remove the overhead required to lookup the type of the data on the program stack and by design, allows any statically typed programming language to be implemented within the virtual machine [3]. The virtual machine gives an abstraction from the hardware in order to allow development to be easier and more cost effective, leading to more maintainable code.

High level languages are of course, used to having vast quantities of volatile memory to utilise and so the converting tool into bytecode for the virtual machines restricts or prohibits the use of certain features of the language. Some features are not supported at all - such as threads and floating-point arithmetic - simply due to the memory requirements. Other features, such as varying integer sizes, are allowed but semantically different to their operation in usual circumstances - integers are mapped by the compiler to smaller sizes, such as 64 bits being mapped to 32 bits. These castings are done automatically by the converter, meaning that the user does not need to cast variables manually [3].

The virtual machine makes heavy use of the delegate construct, which is supported natively within C#, but not within Java, and so this is a keyword that is introduced the external libraries and compiler specifically for use on the mote [3].

3.2.3 Comparisons

Both TinyOS and Mote Runner provide execution platforms which have been optimised for use with low power devices with wireless sensor networks in mind. Whilst both provide support for high level programming languages, the flexibility

of Mote Runner allows the user's choice of any strictly typed language, whereas TinyOS only allows the use of nesC, a comparatively lower-level language. Mote Runner also provides a browser based simulation environment for development to avoid constant reloading of software onto the physical motes, with a significant amount of debugging tools, as well as its plugins for the *Eclipse* IDE.

3.3 Wireless Sensor Network Protocols

Wireless Sensor Networks (WSNs) are used heavily within industry and have a vast amount of applications for real time systems such as nuclear power plants. Wireless Fidelity (Wi-Fi) is a common household name when it comes to wireless networks, but as a protocol, it does not concern itself with the low power abilities of (potentially battery operated) motes, nor does it have inbuilt integrity such as channel hopping [10], providing reliable message delivery. The Bluetooth technology is designed with mobile devices in mind, and as such it can meet the requirements of low power usage as well as provide data integrity with channel hopping. However, Bluetooth also has a drawback in the fact that it is only designed with low range, fixed topology, peer-to-peer networks, restricting to an extent what our sensor network could do [10].

Two predominant protocols that are designed with low power consumption and large topologies in mind are the ZigBee and WirelessHART protocols, which both fundamentally make use of the IEEE 802.15.4 specification for part of their protocol stack, operating on the unrestricted 2.4 GHz radio band, just like Bluetooth [10].

3.3.1 IEEE 802.15.4

IEEE 802.15.4 is an industry standard defining a subsection of the Medium Access Control (MAC) layer and complete Physical layer of Low-Rate Wireless Private Area Networks (LR-WPANs). Although not originally designed for WSNs, its key standpoint is "low-rate, low-power consumption and low-cost wireless networking" [9], which is perfect for application within the field of Wireless Sensor Networking. A number of WSN protocols rely heavily upon IEEE 802.15.4 to define part of their network protocol stack.

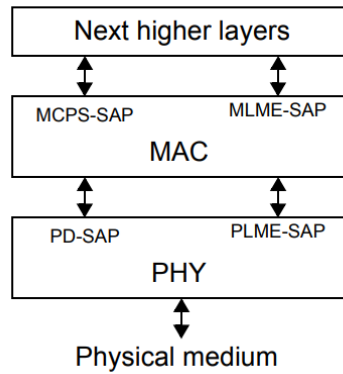


Figure 2 The IEEE 802.15.4 Protocol Stack [11]

The Physical Layer is in charge of data transmission and reception within one of 27 channels offered at range of frequencies bands situated 2.4GHz to 915MHz. It also implements the selected operational mode, as defined by the Personal Area Network (PAN) Coordinator in the network: 1) *Beacon-enabled mode*: where beacons are broadcast over time to time-synchronise all of the nodes, using super frames to define broadcast times; or 2) *non Beacon-enabled mode*: whereby the MAC layer undertakes Carrier Scheme Multiple Analysis with Collision Avoidance (CSMA/CA) operation; both of which are considered and an example protocol introduced later in this section [9].

3.3.2 ZigBee

Originating in 1998, ZigBee is a rapidly evolving protocol which specifies a “low-power, low-cost, low-complexity” [12] methodology for device-to-device and also personal area networks. Employing a five-layer protocol stack based upon the Open System Interconnection (OSI) model, it uses the IEEE 802.15.4 standard to define the operation of both the Medium Access Control (MAC) and Physical layers [9]. It supports the star, mesh and cluster-tree (a special case of mesh) network topologies. [12]

Within ZigBee networks, there are three different classes of devices: 1) *Coordinator*: Responsible for initialising and configuring the network, acting as a router thereafter; 2) *Router*: Associated with the coordinator and is responsible for all of the multi-hop routing of messages; 3) *End-Device*: Completely disjoint from other devices in the network, they cannot route and will only request or submit data for transmission [9].

ZigBee uses the non-beacon enabled, Carrier Scheme Multiple Analysis with Collision Avoidance (CSMA/CA) scheme for transmission scheduling [13], which aims to prevent collisions from occurring. When a device’s network layer receives

a packet to be sent through the protocol stack, the device begins to listen to the channel. If the channel is free, with no other transmission occurring, then the mote shall transmit its packets. If the channel is not free, the device waits for a random period of time, the *backoff factor*, before trying to transmit again. This process is repeated for as long as is required for the channel to be free for transmission [14].

The coordinator maintains the order of broadcasts and super frame transmission within the network of established devices. Each transmission is scheduled such that no two transmits will overlap, thereby preventing collisions of data between nodes in the receiving vicinity [15].

3.3.3 WirelessHART

WirelessHART [10] is a table-based protocol which uses beacon-enabled mode, employing a Time-Division Multiple Access (TDMA hereafter) approach to ensure “collision free and deterministic communication” [10]. Based on the original HART technology for wired networks, it employs a five-layer communication model along with a central network manager to manage the routing and communication scheduling. Being able to handle external interference, 16 channels are available to transmit on, with channel hopping providing plenty of redundancy both in response to transmission difficulties and also pre-emptively, with pseudo-random channel hopping employed. [10]

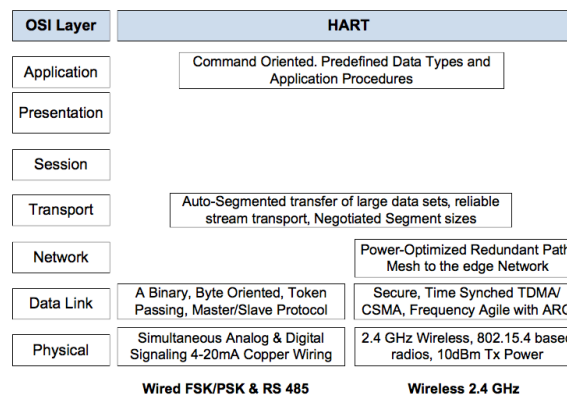


Figure 3 WirelessHART Network Architecture, based on the (wired) HART communication protocol [10]

The Physical layer employs the IEEE 802.15.4 standard, operating over the 16 channels numbered 11 to 26, with a 5MHz gap between each channel. The Data link layer is time synchronised, and a 10ms time slot is defined, with consecutive time slots grouping to form a superframe. Superframes define whether each device should transmit/receive/route other packets/remain idle at any time, and their period is determined by the total sum of length of the member slots. [10]

Each transmission within a timeslot is defined by the vector :

{frame id, index, type, source address, destination address, channel offset} [10]

The frame id and index terms define the specific superframe and time slot from which the transmission is occurring, whilst the channel offset describes which channel the transmission shall take place on. WirelessHART automatically deals with channels that continuously are disrupted by interference, employing channel blacklisting, whereby the offending channels are removed from the active channel table of each device, meaning it shall not be used for any future transmissions. [10]

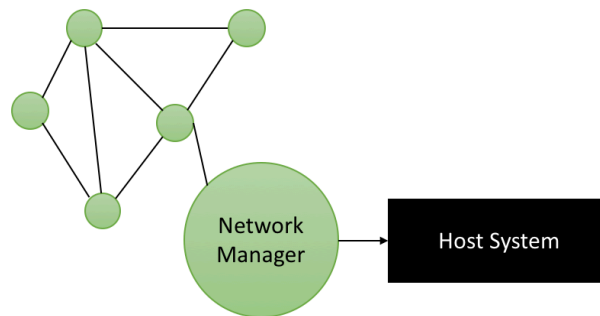


Figure 4 A simple WirelessHART network topology with 5 nodes and a sink node

As previously mentioned, WirelessHART is a table based protocol, with each network device maintaining a list of super frames (defining their behaviour at any time), neighbouring nodes and routing options as well as the previously mentioned active channels table. All of these tables are consistently kept up to date via periodic broadcasts of the most recent tables from the network manager [10].

3.3.4 Comparisons

Reliability is of primary importance when any protocol is to be used within industry, and by comparing ZigBee with WirelessHART it is easy to see how it lacks this. ZigBee's lack of channel hopping capability leaves open a huge vulnerability, from both unintended and intentional (malicious) interference from external sources, leading to messages not being delivered. As the network grows in size of devices and there are more dropped messages due to outside interference, more retransmissions will occur, reducing the overall throughput of the network [13]. This key vulnerability is addressed in an updated version of the protocol, ZigBeePRO [16] but is of course already dealt with in WirelessHART.

Path diversity is another issue with ZigBee, meaning that if a route between two devices is broken, there is no other option but to discover new routes. This increases the delay of data transmission until a new path is established, and in networks with particularly unreliable routes, "route-discovery will eventually

consume all bandwidth available” [13]. With no bandwidth available to fulfil actual transmissions of data, the sensor network shall cease to operate in any beneficial capacity which is not helpful and potentially unsafe in an industrial environment.

WirelessHART does not encounter many of the issues associated with ZigBee. Due to its use of channel hopping and graph routing, it is able to automatically solve its own issues whilst continuing operation, proving itself to be an entirely robust and fit-for-industry methodology. Whilst its use of TDMA requires synchronization throughout the network, it allows for devices to operate in low power mode between their assigned timeslots to preserve battery life, compared to ZigBee’s CSMA/CA which requires them to be in a permanent ON state – since the motes have no predetermined time to transmit [13].

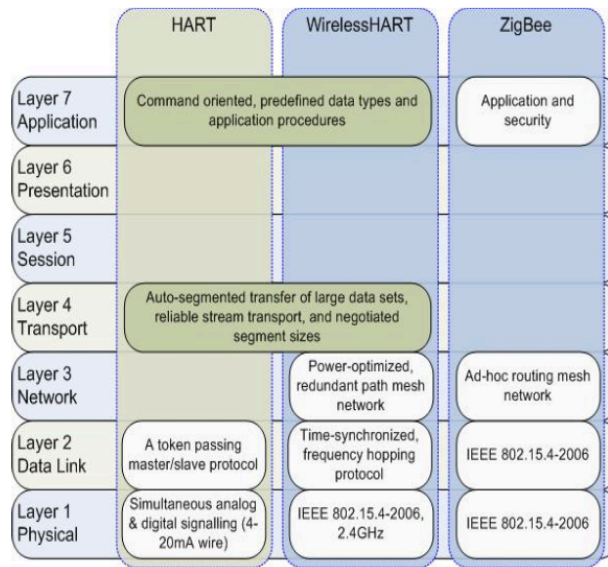


Figure 5 Comparison of (HART), WirelessHART and ZigBee architectures [13]

ZigBee’s specification [15] defines to a large extent, the operation of every part of the system, leaving little to no scope for decisions by the user, whereas WirelessHART provides multiple opportunities to define the exact priorities of the network through its network manager [13].

In summary, WirelessHART as a wireless sensor network protocol is a much more versatile and robust methodology for industry when compared to ZigBee, although both of them have similarities, such as the use of the IEEE 802.15.4 Physical layer and OSI model base.

3.4 Real-Time Scheduling

Having considered a few communication protocols, another important point to be made is that WSNs are real time systems in which transmissions much meet

deadlines in order to avoid potentially catastrophic failure. The host system which depends upon sensor measurements to produce a response to a stimulus depends upon the data arriving. In order to best determine how long it will take for a transmission to complete, the notion of response time analysis is used to describe the absolute worst-case amount of time that a task will take to be completed. Ordering transmissions such that they meet their temporal requirements (deadline and or period) is an important part to ensuring that a system can deal with an input and produce a response within a deterministic amount of time. Therefore, we must consider both scheduling policies (ordering of transmissions) and also response time analysis (determining how long until completion) as a means for making guarantees about the real time communications within our WSN. [17]

3.4.1 Fixed Priority Scheduling (FPS)

There are a number of priority scheduling policies that can determine an execution order of tasks based on a set of parameters. Fixed priority scheduling is a method which applies a static priority to each task pre-runtime based on their temporal requirements (not their importance of completion). The values assigned for each priority vary between literature some using 1 to represent the highest priority whilst others use it to represent the lowest. Each task is then in turn executed in order of decreasing priority. [17] There are two ways we can assign a fixed priority:

Rate Monotonic Priority Ordering (RMPO) is an optimal priority assignment scheme which assigns unique priorities based upon the period of the tasks. A task with a shorter period (more frequently occurring) is therefore assigned a higher priority to a comparatively larger period task [17].

Deadline Monotonic Priority Ordering (DMPO) is a fixed priority assignment scheme which assigns unique priorities based upon the deadlines of the tasks. A task with an earlier (*relative*) deadline (must be completed sooner) is therefore assigned a higher priority to a comparatively later deadline. Whilst RMPO is considered optimal DMPO can be proved to be optimal based on the transformation of priorities from an ordering created by another FPS scheme [17].

DMPO and RMPO are equivalent orderings when the deadlines of the task set are implied from their periods; where the deadline is equal to the period. In which case, either ordering shall produce the same result. When deadlines are constrained by an upper bound of their period $D \leq T$ then DMPO and RMPO produce different orderings [17].

3.4.2 Response Time Equation

The Response-Time equation is a method for fixed priority scheduling analysis which can demonstrate whether a set of tasks using some priority ordering method is guaranteed to meet its deadlines. As such the response time equation is a

necessary and sufficient (exact) test which determines if a set of tasks is schedulable or not. The (worst case) response time of a task is defined as the maximum amount of time taken for the system to complete its execution. [17]

Using the standard definitions R_i for the response time, C_i for the task's worst case execution time and T_i for the task's period, we get the response time equation:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (1)$$

The most basic form of the response time equation [17]

For each task, the worst case response time is defined as its worst case execution time as well as any interference from tasks executing that are of a higher priority. The equation is solved by solving a recurrence relation with the final response time being compared to the deadline of the task [17].

if $R_i \leq D_i$ then schedulable; else fail

Each response time must be no greater than its deadline in order for it to be schedule. This is trivial since if a task takes longer than its deadline to produce a response within a task set then it is clearly not schedulable. [17]

4 Specification

4.1 The Problem & Case Study

The scope of this project is to design, implement and analyze a new wireless sensor network protocol using inspiration from current literature. Our goal is to make this as reliable and efficient as possible and as such we define a case study of a mock network which we shall deploy using our protocol and then undertake significant analysis on it.

Our case study in Table 1 is a randomly generated (by Appendix A) specification of 10 data transfers of varying sizes between 5 network devices labelled A to E, each with a fixed priority (1 is highest) which must be satisfied. We shall use this specification to schedule and analyse the response times of the data transfers and compare them to the theoretical values of models proposed later in this report. We shall also conduct optimisation of network parameters and integrity analysis using this mock network to ensure that the network runs at the highest throughput possible (little to no idle time) and demonstrate that our protocol can successfully manage and respond to external interference.

We begin by making a simplifying assumption whereby all the nodes defined in this hypothetical network are within transmission range of each other, and as such no routings through the network between the source and destination nodes are needed. All communicating devices shall therefore only act as a transmitter or receiver. We also assume that each data transfer's period (and implied deadline) is equal to the total time taken for all data transfers to be completed such that none need to be repeated before all have been completed.

ID	Source	Destination	Size (bytes)	Priority
1	B	E	43	1
2	A	D	33	2
3	C	E	24	3
4	B	C	12	4
5	D	B	45	5
6	D	C	25	6
7	B	E	14	7
8	A	B	11	8
9	D	A	18	9
10	B	C	32	10

Table 1 – The Case Study: The randomly generated data transfer specification

The table defines the data transfers that are required for our network. From this, we shall generate a transmission schedule based on some scheduling policy that satisfies this table. Our aim shall be to optimise the parameters of the network as well as the superframe schedule, such that it provides the best performance.

It's important to note the difference in the terminology used for the rest of this report to avoid confusion: *Data Transfer* refers to a single row of this table and its values; *Transmission* refers to the physical passing of some or all of the required payload to the destination node, based on the parameters of the network.

4.2 Goals

The end goal of this project is to implement a protocol which from using a scheduling policy, 5 nodes and the specification in table 1 can produce a self-contained network that successfully completes the required data transfers in a sensible order.

Before we get invested in the analysis and evaluation of the network and scheduling policy, it's important that we define some goals that will be able to assess the generic implementation of the defined protocol. By taking knowledge from the literature review and combining it with the limited problem specification we are able to define 10 general and implementation-relevant goals.

1. All data transfers listed in the problem specification must be satisfied
2. The implemented protocol must protect itself from internal interference
3. The implemented protocol must protect itself from external interference
4. All data transfers must be completed within the period of the superframe
5. Data transfers should be scheduled in order of priority
6. The implementation should have a low fault rate
7. The implementation should allow parameters to be altered easily
8. A model should be produced which can compute the response times of the data transfers
9. Performance measures should be defined such the functioning network can be assessed
10. The network should be optimized such that it provides the best performance possible

5 Implementation Design

Having considered the generic requirements and goals of the implementation that we wish to produce we shall begin by designing the protocol that we shall use within our WSN to satisfy the specification of data transfers that need to be completed. This gives us a basis for our optimization, response time and integrity analysis later in the project.

5.1 Choice of Execution Platform

Having considered both TinyOS and Mote Runner as execution platforms within the literature review, Mote Runner seems the most sensible option. With its integration within the Eclipse IDE, as well as a fully-fledged simulation environment within which to run programs on motes, it provides all of the tools that will make development a quick and also an easier process compared to that in TinyOS. It also allows development in the comparatively higher, high level language Java, which is much more sophisticated than using a C-like language within TinyOS.

Mote Runner's mote simulator environment will also allow for all analysis to be conducted within a controlled environment, rather than using physical motes which would be susceptible to uncontrollable amounts of external interference and factors such as the distance between the nodes will be easier to maintain. By controlling all of the parameters within a single environment our experimental results shall prove to be more reliable.

5.2 Choice of Network Protocol Type

Based on the problem description and the two protocols that have been considered within the literature review, it is clear that the requirements of the project deem that a WirelessHART-style protocol should be used to implement this network. Whilst ZigBee is a perfectly acceptable methodology it allows transmissions to take place whenever the communication medium is free, unlike WirelessHART which predefines when and to where transmissions are to take place, much like the problem posed. By using a WirelessHART style protocol we can guarantee that all of the data transfers are satisfied within a specified time window – unlike ZigBee which could lead to none-completion of some transfers as they all compete for bandwidth. We shall call this new protocol LikeWHART.

5.3 LikeWHART

Having decided upon our inspiration for our new network protocol with which to satisfy the data transfer specification, it is now important to define the full specification of LikeWHART and its operation.

5.3.1 Synchronisation

LikeWHART will take the same principle from WirelessHART, in that an extra mote shall be added to the 5 motes of our network to act as the dedicated network manager which shall maintain the synchronisation of all of the devices.

A simple beacon-based synchronisation method shall be used, whereby the network manager node will make a broadcast to signify the start of every superframe. Each network device shall store a list of offsets for each superframe, in which it is their turn to transmit or receive. Superframes must be short enough that synchronisation broadcasts are frequent enough to keep devices in sync with each other, but also long enough such that broadcasts do not substantially affect the overall progress of the system.

5.3.2 Avoiding Internal Interference

Each data transfer within the specification shall be undertaken by one or more transmissions on the network, defining a period in which the transmission and acknowledgement shall occur with no other transmissions taking place. By defining the strict timing constraints on what transmission will occur when and where shall prevent two transmissions from overlapping with each other causing internal interference. In the event that a device is out of sync with the rest of the network a method of packet receipt acknowledgement shall be defined which ensures successful delivery of the message.

5.3.3 Managing External Interference

The possible internal interference of the network is to an extent, predictable and can be solved by a rigorous definition of what device is to transmit and when. External interference is entirely random and can come from different sources with various effects over time.

In order to lessen these effects, LikeWHART shall employ a pseudorandom channel hopping structure. Each superframe shall have a hard-coded transmission channel that it should take place on (out of the 16 available channels) defined pre-runtime. Only the network manager will have knowledge of this list and as such at the start of each superframe it shall transmit the next superframe's channel as part of the synchronisation payload. If a message fails to reach its destination – that is, no acknowledgement is received - then the message shall be scheduled to be sent at the next available timeslot for the device.

5.3.4 Ensuring Delivery with Packet Acknowledgements

To ensure transmission delivery we employ a method of packet acknowledgement. When a message is sent from a device A to a device B its receipt by the destination does not end the communication period. Device A now awaits an acknowledgement that confirms the transmission has reached its destination. If

this acknowledging packet does not arrive within a specified period, then the message is deemed to have not reached its destination, and it is queued to be sent in the next available timeslot for that device.

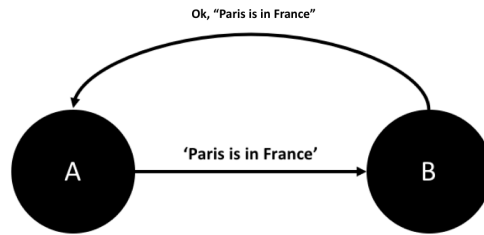


Figure 6 Simple communication between A and B, with acknowledging response

LikeWHART shall use a timeslot length which allows for the transmission of data and receipt of the acknowledgement packet to be done within the same period of time given the topology of the network. This timeslot size should be defined based on the environment and other parameters of the network, such that it gives the best performance.

5.3.5 Frame Structure

Since LikeWHART is based off of WirelessHART, and Mote Runner uses the IEEE 802.15.4 standard for its radio communications. We shall therefore use the structure of the packet/frame from the standard's specification.

5.3.6 Slot Tables

Each of the motes shall keep track of its own offsets from the start of each superframe and know its function at that timeslot - either transmitting or receiving. Offsets into the frame shall be calculated with a timer interrupt, with the time between the device's timeslot being the idle time of the device, where it can enter low-power mode to preserve battery life; awoken by the time-triggered interrupt.

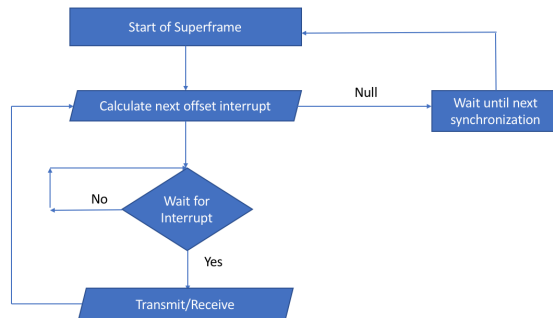


Figure 7 Basic Recurring Operation of Network Devices

Each device shall keep a track of its own slot table only, which shall be hard coded before the network is initialised – the network manager does not need to broadcast this information at the start as in WirelessHART. This means that LikeWHART's supported network topologies are completely static – that new nodes can't join the network whilst it is operational. The code for each superframe slot would have to be changed such that it gave bandwidth to the new device after which the network could be restarted.

6 Timeslot Motivation & Process Design

6.1 Why find the Optimal Timeslot?

Finding the optimal timeslot within a superframe-based network is important because it is a key component to making the network as efficient as possible. However, there is no single optimized timeslot for all data transfer requirements. The timeslot size is optimized uniquely to the network's parameters, especially the size (or payload) of a transmission. We can use various measures of efficiency in order to find this best timeslot size.

The packet failure rate is one of the most important of these measures which describes the percentage of packets that fail to reach their destination – which we measure through the number of missing packet acknowledgements.

$$\text{Packet Fail Rate} = \frac{\text{Number of unreceived acknowledgements}}{\text{Total number of transmissions}} \quad (2)$$

A higher packet failure rate (with no other external factors) indicates that the timeslot size in use by the network is too small for the payload size of transmissions. As the packet failure rate increases, the number of successfully delivered packets reduces until it reaches 0 at which rate no progress is being made by the network. The repercussions of lost packets don't just mean that the packet failure rate is high it also means that retransmission of data has to occur wasting time and reducing the progress of the network as a whole.

As a combination with the packet fail rate being minimized, we want to maximize the use of bandwidth within the network. If the timeslot is too big for the amount of data being sent then the packet fail rate will trivially be 0. However, there will be large amounts of periods within the network's operation where nothing is happening due to the transmission finishing, but the timeslot not ending for a while. This idle time is wasted bandwidth and could be used by other data transfers to make progress themselves. This period of nothing happening reduces the overall throughput of the network.

$$\text{Network Throughput (bytes ms}^{-1}\text{)} = \frac{\text{Total Number of Bytes Sent}}{\text{Superframe length}} \quad (3)$$

If a greater amount of time is taken to complete the same set of data transfers then the network's throughput (bytes per millisecond) shall be lower. Hence in order to optimize the size of the timeslot, we want to maximize the network's throughput and minimize the packet failure rate.

6.2 Varying the Payload Size

Since there is no one single defined timeslot for all payload sizes (because in theory a larger transmission payload requires a longer amount of time), we shall optimize schedule and analyze 5 networks as initially defined by their maximum transmission payload. We base this on the range of the data transfer payloads given in the data transfer specification:

10 Bytes	20 Bytes	30 Bytes	40 Bytes	50 Bytes
----------	----------	----------	----------	----------

By increasing the (maximum) payload size of any given transmission, it allows for more data to be transferred within fewer transmissions. An increase in payload size leads to a reduction in the number of required transmissions. We define this number of transmissions required for a particular data transfer of size W_i on a network with maximum transmission payload size τ as:

$$\left\lceil \frac{W_i}{\tau} \right\rceil \quad (4)$$

To find the optimal timeslot for a given payload size, we shall define a superframe which constantly transmits packets of that size. By then making small changes in the size of the timeslot from an initial value shall enable us to find the optimal value, where the packet failure rate is 0% for the smallest possible timeslot length.

6.3 Exploring the Relationship

To generalize the specification and make our optimization analysis applicable to any give set of data transfers under the LikeWHART protocol, we shall explore the relationship between the payload size and its respective optimized timeslot size.

We shall begin with the hypothesis that the relationship between the timeslot size and payload size of the network is linear, such that a doubling of the (maximum) payload size requires a doubled timeslot size to accommodate it. By exploring this relationship in full we'll be able to make accurate predictions in the future about the required timeslot size for a previously unseen payload size within a LikeWHART network.

7 Scheduling Motivation & Policy Design

7.1 Why schedule the superframe?

Choosing the order of transmissions within a superframe is an important decision for a network in terms of whether the temporal requirements or priorities of the data transfers are addressed. In our LikeWHART network, we shall be considering a static scheduling policy which doesn't change as the network operates, such that the superframe's schedule of transmissions is designated pre-runtime.

In our specification, we assume that the deadline of all data transfers is implied as the same as the period (the length of the superframe). As we change the size of the payload the number of required transmissions will change, hence defining a deadline for each transfer would mean deadlines would be excessive for the higher payload sizes to meet the needs of the lower payload sizes. Hence, we assume that deadlines are the same as the period (the length of the superframe) to remove that particularly awkward definition of values within this analysed specification. However the models and analysis that shall be presented extend to constrained deadlines ($D_i \leq T_i$).

By building different transmission orderings of the superframe we can attempt to improve upon the average completion rates of data transfers, thus providing a more efficient network that completes data transfers quicker on average. We can compute the time taken to complete each data transfer (the response time) by using a simple model adapted from the response time equation for each of the different models. We define δ to be the size of a single timeslot within the network – the value that we will have just optimized for specific payload size τ .

7.2 The Synchronisation Period

At the start of any superframe there is a period in which all nodes are awaiting the broadcast from the network manager that signifies the start of the next timeslot, synchronising them. The value of this waiting period is hypothesised to be a constant value – containing the time in which the nodes listen, receive the network manager's broadcast, process it and begin the next superframe in unison. We shall define this as the *Synchronisation Constant* λ , which we shall include within our scheduling analysis as it contributes towards the response time.

7.3 Is it Schedulable?

Based on a fixed size superframe that consists of a number of timeslots N , we define a simple utilization bound for determining whether a set of data transfers is schedulable, where each data transfer has a period equal to that of the superframe:

$$\sum_{W_i \in W} \left\lceil \frac{W_i}{\tau} \right\rceil \leq N \quad (5)$$

Even though a set of data transfers meets this constraint, they still may not be schedulable if they have constrained deadlines that are not met by their response times. Therefore this test is necessary but not sufficient, but it is exact for implied deadlines equal to that of the period of the superframe.

7.4 Fixed Priority Schedule (FPS)

In our specification we defined a static (pre-runtime) priority for each data transfer. The fixed priority scheduling methodology shall order the superframe such that data transfers are completed in order of decreasing priority (where 1 is the highest). This is a general methodology that shows any fixed priority assignment policy such as deadline monotonic in more complicated data transfer tables than our specification will work; ones with constrained instead of implicit deadlines. The schedule of the superframe shall not move onto the next data transfer until the current one is completed, regardless of the number of transmissions that it takes.

7.5 Smallest Transfer First (STF)

Each data transfer is given a predefined number of bytes that it must send to the destination, and as such there is an inherent ordering of the transfers in increasing order of size. For this scheduling policy, we shall ignore the priorities assigned to each transfer (pretending that they are not known) and just consider the transfers on their relative ordering in transfer size. The schedule of the superframe shall not continue to the next data transfer of equal or larger transfer size until the current one has been completed, regardless of the number of transmissions that it takes.

7.6 Response Time Equations

We shall be measuring the effectiveness of both scheduling policies by considering their response times for each data transfer. We can produce response time equations for both of the policies by adapting the general response time equation as previously discussed [17].

To adapt the equation into a form for both STF and FPS, we must replace terms with time-slot related instances. We first replace the computation time C with the number of timeslots required to complete a data transfer - the maximum number of transfers for a given payload size. We combine this with the length of the timeslot to produce a value in milliseconds that is the required transmission time.

$$C_i = \left\lceil \frac{W_i}{\tau} \right\rceil \delta \quad (6)$$

We have made a generalizing assumption for our specification that simplifies the scheduling problem in that all data transfers have a period equal to that of the length of the superframe. Since this means that the interference calculating term

is constant as the response time will never exceed the superframe period (unless not schedulable) we drop the term which computes the number of executions of an interfering task.

$$R_i = \left\lceil \frac{W_i}{\tau} \right\rceil \delta + \sum_{j \in hp(i)} \left\lceil \frac{W_j}{\tau} \right\rceil \delta \quad (7)$$

Since there is no preemption within our scheduling model too since no single data transfer will occur before all other have been completed we only compute the maximum response time of all preceding data transfers that have occurred i.e. the response time of the previous data transfer. Finally we add in the synchronization constant which gives us the final form:

$$R_i = \left\lceil \frac{W_i}{\tau} \right\rceil \delta + \max_{j \in hp(i)} \left(\left\lceil \frac{W_j}{\tau} \right\rceil \delta, \lambda \right) \quad (8)$$

Where we compute the previous response time or the synchronisation constant as interference if this is the first data transfer to complete. We can now use this to specify the two response time equations for our scheduling policies:

$$R_i = \left\lceil \frac{W_i}{\tau} \right\rceil \delta + \max_{j \in hp(i)} \left(\left\lceil \frac{W_j}{\tau} \right\rceil \delta, \lambda \right) \quad (9) \quad R_i = \left\lceil \frac{W_i}{\tau} \right\rceil \delta + \lambda + \max_{j \in W_i < W_j} \left(\left\lceil \frac{W_j}{\tau} \right\rceil \delta, \lambda \right) \quad (10)$$

For Fixed Priority Scheduling

For Smallest Transfer First

Both of these equations have been built through construction from the original response time equation and as such we deem them to be proved. For instances where the deadlines of the data transfers are not implicitly defined to the length of the superframe like their periods, these equations deem a set of data transfers to be schedule so long as $R_i \leq D_i$ for all data transfers.

To evaluate STF and FPS, we shall compute and compare the response times of both policies to decide upon the method that achieves the smallest average response time. By comparing these, we shall be able to decide upon the on average best scheduling policy for a given set of data transfers. We shall deploy the scheduling policies to each of the optimized networks and measure the actual response times for each of the data transfers and therefore be able to compare theoretical and actual values and therefore the accuracy of the model.

7.7 Scheduling Order

From our scheduling policies, we define the order of data transfers (referenced by their assigned IDs) within the single superframe to be:

FPS	1	2	3	4	5	6	7	8	9	10
STF	8	4	7	9	3	6	10	2	1	5

8 LikeWHART Implementation

The majority of the implementation of LikeWHART was routine with only a few hurdles presented by the restrictions of what parts of the Java language are supported by the execution platform.

Two generic source codes were created: one for the Network Manager (Appendix B) and the other for any Network Device (Appendix C). The Network Manager is the more simplistic code using a timer-based interrupt to send out the synchronisation pulse in predefined intervals which also contains the next superframe's transmission channel. The Network Device code is a skeleton upon which a transmission/reception schedule is added to describe the functionality of a specific node within the network.

8.1 Timeslots, Superframes and Offsets

The original idea was to represent the timeslots assigned to each device within a superframe as a multidimensional array of the following structure:

$$\{\{\text{offset, function, destination}^1\}, \dots\}$$

However multidimensional arrays are not supported in Mote Runner due to the datatypes that the virtual machine utilizes. Hence this had to be done using a single dimension array for each attribute. Whilst this wasn't a major hurdle to overcome, it did effect the general development process and means that for each superframe, several arrays are required to do the job of a single, multidimensional array. To keep track of which timeslot the device is responding to, a simple counter variable is included which is used as an index to retrieve the correct information from the data structures.

Rather than wasting the limited resources of the motes, the destination array only consists of values for the actual transmissions, with no null values for receiving periods as specified in the planned structure. We therefore use a dedicated counter as an index into this array to determine the target address of the transmission, and increment it only when a transmission occurs.

Each device is hardcoded with these arrays before the creation of the network, along with information about the length of a single timeslot, and of each superframe. It is also assigned a unique device address (for use with the radio) on the network. This means that devices automatically ignores packets not addressed to them as per the IEEE 802.15.4 specification. This made the implementation of packet handling easier meaning that we didn't have to manually cater for packets being received that weren't addressed to that device.

¹ Null by default, only set to a destination if that timeslot's function is transmission

8.2 Synchronisation

Ensuring that all of the devices on the network remain in sync with each other is the most important part of the implementation, as it directly influences the amount of internal interference that can occur. The network manager oversees this aspect by sending out a broadcast at the start of every superframe which is spaced at regular intervals using a timer-based interrupt. When the network device receives the broadcast message, they know to begin the next superframe.

Ensuring that the first superframe after the network is created is synchronized is the most important stage of setting up the network. All of the network devices are hardcoded to listen for a broadcast for an extended period of time on their initialization in order to wait for that first pulse from the network manager.

Having received the synchronisation pulse, all of the network devices calculate how long away their first operation (transmit/receive) is by checking the first value of their offsets (*SLOT_TABLE*) array. They then create a timer-based interrupt for that duration, entering low power mode whilst awaiting the interrupt request. The process for calculating the time between one operation and the next is a little more complex, since we must consider how far into the superframe we already are:

```
tslot.setAlarmBySpan(Time.toTickSpan(Time.MILLISECS,  
                                     (SLOT_TABLE[OFFSET_INDEX] -  
                                     SLOT_TABLE[OFFSET_INDEX-  
                                     1]))*TIME_SLOT_LENGTH);
```

The function *Time.toTickSpan* is a simple function that maps an amount of time into a number of clock cycles such that it can be measured on the mote. The first of the two parameters, *Time.MILLISECS*, describes the units of the value we are inputting (seconds is another option). The second parameter is where we calculate the actual amount of time we need to wait to do our next operation, which computes the difference between the two offsets (the one we have just done and the one which is next) which gives us the number of timeslots within the superframe we need to wait. We then multiply this by the network's fixed timeslot length in order to determine the time (in milliseconds) that we must therefore wait. In the case that we have completed the final operation of the superframe, we compute the difference between the length of the superframe in total (a pre-runtime defined constant) and the offset of the final operation, giving us the length we must sleep before listening for the next synchronisation pulse.

8.3 Packet Acknowledgment

I ran into problems when trying to use Mote Runner's incomplete documentation and it provided little assistance to using the IEEE 802.15.4 packet acknowledgement methodology. As such I was unable to determine whether a packet had or had not been acknowledged. Since development of the virtual machine ended in

September 2016 [4] and the limited online resources did not prove helpful, I was unable to implement this desired functionality of the system. Therefore, there is no acknowledgement phase for transmissions between two nodes within LikeWHART it is assumed that they have successfully reached their destination.

8.4 Channel Hopping

The proposed channel hopping methodology was well-implemented with the Network Manager maintaining the list of channels for use of each superframe and broadcasting the next channel as part of its synchronisation broadcast. The only issues that stalled parts of developing this functionality was through the lack of documentation which didn't list all of the available functions for using with the radio. One key undocumented piece of functionality which was missing was the function which stopped the radio such that the channel could be changed – this was only discovered by browsing the code of a GitHub Repository [18].

The functionality is implemented quite simply. The Network Manager maintains a list of channels that shall be used in turn for each distinct superframe. It send the synchronisation packet which includes the channel number that shall be used by the next superframe. The receiving devices then make a note of this and switch to it at the end of the superframe, ready for the next synchronisation.

8.5 Use of LEDs

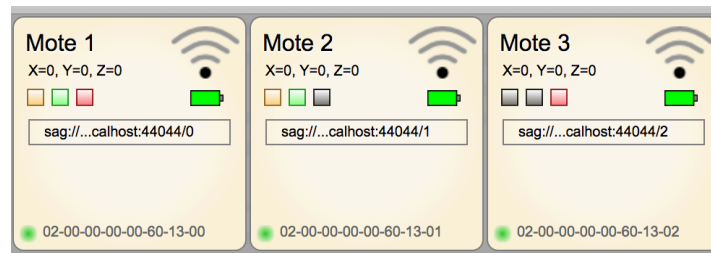


Figure 8 - Example LED output: Network Manager (Left) and two Network Devices

A schematic was defined for the main motes whereby the mote's onboard LEDs are used to provide feedback to an observer (example figure 8) on what is actually occurring/what stage the individual motes are at. The network manager mote simply alternates all LEDs being on/off to signify the start of a new superframe.

As the size of the timeslot decreased, the usefulness of the LEDs as a debugging/observation tool became limited since they are only on for a fraction of a second at most before changing (based on the timeslot size being used).

Meaning		
		Green
		Yellow
		Red
		Idle

9 Experimental Practices & Design

Having considered why optimization of the timeslot's size and scheduling the superframe correctly is important as well as introducing our response time models and a few hypotheses, it is now vital that we define the exact methodologies of each investigation in order to make them reproducible to the reader.

9.1 Use of Simulation Environment

Experiments are all about isolating the dependent variable such that only changes to the independent variable are allowed to effect it and for that reason I've chosen to complete all of the experiments within the Mote Runner simulation environment. This allows no external factors to influence the testing environment, such as unwanted external interference, fluctuations in battery power or distance between the nodes. Also as a matter of ethical considerations, by using the simulation environment this removed the chances that investigations would cause any external interference to other operating networks within the University's Department building.

Whilst ensuring that LikeWHART is able to perform within a variety of environments is important, it is important to measure a baseline on the performance which is not influenced by its external surroundings. The lack of packet acknowledgement within our implementation presents an issues in measuring the success rate of packets for our analysis. Hence, an alternative has been employed through the use of Mote Runner's *logger* (debugging) system library within its simulator. The main functionality of this will be that devices produce a logger output when they receive a packet, giving us pseudo packet acknowledgement.

By logging the number of received transmissions we are able to assess the network's packet fault rate as if the logger outputs are packet acknowledgements.

```
logmsg:      (hal_log) Packet Recieved  
mote:        02-00-00-00-00-F9-F0-01  
msgtype:     log-ev
```

Figure 9 An example custom "Packet Received" logger output from a mote

9.1.1 Limitations

Whilst the use of the simulator does present many advantages in isolating the experiments from external factors it presents significant limitations. The simulator is effectively a software implementation of the motes and as such it does not have to deal with the actual operation of the hardware. In reality, the experimental values obtained will most likely be too generous as mote-specific factors such as the heating effect of an electric current on hardware (increasing hardware delays) are not going to be influencing the results that we obtain. Essentially, the simulator does not emulate the operation of the motes, it simulates how they work using a model that may not be completely accurate. Regardless of how well it attempts to

model the nodes our results will most likely be skewed. We shall take this into account when considering the results obtained.

9.2 Hardware and Software

Since all of the experiments are to be conducted within the simulation environment, it is important to remove any unforeseen factors that could affect the overall performance of the nodes and as such influence the measurements that are taken. Hence, the hardware that the experiments take place on is kept the same as well as the software used.

All experiments were conducted on a 2013 MacBook Pro model, using a 2.4 GHz Intel Core i5 with 4 GB of 1600 MHz DDR3 RAM running macOS Sierra version 10.12.6 (16G29). For the simulation environment, Beta version 17.1.8 (December 2014) of Mote Runner was used within Eclipse Oxygen.2 Release (4.7.2). Each experiment was undertaken on the Iris nodes.

9.3 Timeslot Size Optimization

We shall conduct our timeslot optimization on a network consisting of 3 nodes: 2 network devices and a single network manager (providing the synchronisation). 10 transmissions shall be scheduled between the two nodes within a single superframe with the logger output providing a determinism on whether or not the packet has reached its destination – “Packet Received” or “None Received” shall be outputted for each period where a packet is expected by either node. The packet fault rate for the superframe shall be calculated using the number of “None Received” outputs as the count of unacknowledged packets.

We shall optimize the timeslot length for each specified transmission payload size, starting at 10ms for the 10 byte payload (based WirelessHART’s [10]). The optimal timeslot shall have been found when its packet fault rate is 0 and decreasing the timeslot by 1ms causes the fault rate to begin to grow. If the fault rate does not grow i.e. it remains at 0 then the optimal timeslot has not yet been found.

9.4 Finding the Synchronisation Constant

To find the synchronization constant λ we shall need to measure the time between the end of a superframe and the receiving of the synchronization pulse from the Network Manager by a device. We shall do this by using the logger to output at these two events occurring. The logger by default includes the (local) time within the simulator whenever an output is created and as such we can use the difference between these two times to find the value of the (presumed) constant.

id:	c6e	id:	c75
time:	0:00:55.316'943	time:	0:00:55.340'729
severity:	3	severity:	3
module:	MAPP	module:	MAPP
wtime:	569a7922f213e	wtime:	569a7922f2a6a
logmsg:	(hal_log) Superframe Ends	logmsg:	(hal_log) Sync Recieved
mote:	02-00-00-00-00-68-5A-01	mote:	02-00-00-00-00-68-5A-01
		msgtype:	log-ev

Figure 10 Example Sync Received and End of Superframe Logs showing their times

The experimental setup for this investigation shall consist of a single network device and network manager with the superframe running its course. We shall then be able to compute the synchronisation value from the logger times. We will do this over all of the optimised timeslot values to ensure that the value is truly constant.

9.5 Response Times

Having computed the optimized timeslots based on payload size as well as the synchronisation constant, we shall then compute the theoretical response times for each combination under FPS and STF using the models defined previously. We shall then run the two superframe schedules of the data transfer specification within the Mote Runner implementation of LikeWHART and measure their actual response times again using the logger outputs times. This will consist of 5 network devices (representing A to E) and also a network manager. We will then be able to compare the theoretical response times computed with the actual measured values which shall provide an insight into the accuracy of our response-time models, as well as determine the (on average) better performing scheduling policy.

9.6 External Interference

For measuring LikeWHART's integrity against external interference, we shall use the same setup as that used within the timeslot optimization – 2 devices and a network manager – for the setup of the network, using 10 transmissions between the two devices per superframe. A fourth external node shall be introduced that runs a simple program that broadcasts on each channel (1 through 16) at a parameter-based rate of change.

The timeslot of the main network shall run at the maximum of all optimized timeslots. We shall change the rate of the external node's channel switching in order to see its effect on the packet failure rate of the main network. It is hypothesized that, as the rate of channel changing increases (more frequent changes), the packet fault rate shall increase as the external inference generating device has a higher chance of broadcasting on the same channel as the network.

10 Testing of Implementation

Several parts of the implementation were tested using the following devised test strategy all of which are important to be met. Supporting evidence in the form of screenshots from the simulator where possible are provided in Appendix D – cross referenced to the Test ID.

1	When a network device is transmitting, the correct LED is illuminated	
Expected: The Red LED should illuminate		Red LED illuminates
2	When a network device is waiting to receive, the correct LED is illuminated	
Expected: The Yellow LED should illuminate		The Yellow LED illuminates
3	When a network device has received a transmission it was expecting, the correct LEDs are illuminated.	
Expected: The red and green LEDs should illuminate accompanied by a “Packet Received” logger output		The red and green LEDs illuminate
4	When a network device has received the synchronization broadcast the correct LED is illuminated	
Expected Result: The green LED should illuminate		The green LED illuminates
5	When the network manager sends a synchronization pulse, the correct LEDs are (de)illuminated	
Expected Result: The Red Green and Yellow LEDs should illuminate/extinguish in alternating synchronization period.		All 3 LEDs illuminate at the start of one synchronization period and extinguish at the start of the next.
6	When a network device has received the synchronization pulse it produces a logger output	
Expected Result: a “Sync Received” logger output should be generated		“Sync Received” is outputted to the logger
7	When a network device receives a packet it produces a logger output	

Expected Result: a “Packet Received” logger output should be generated		“Packet Received” is outputted to the logger
8	When a network device fails to receive an expected packet it produces a logger output	
Expected Result: A “Nothing Received” logger output should be generated		“None Received” is outputted to the logger
9	Transmissions are acknowledged with a transmission back to the sender	
Expected Result: A “Packet Acknowledged” output should be generated within the logger.		Actual: Packet Acknowledgements are not implemented and as such there is no “Packet Acknowledged” output within the logger

11 Results

11.1 Timeslot Optimizations

11.1.1 10 Byte Payload

Timeslot (ms)	Packet Fault Rate (%) of Superframe					
	1	2	3	4	5	Max
10	30	20	20	30	20	30
15	10	10	10	10	10	10
20	0	10	10	10	0	0
25	0	0	0	0	0	0
24	0	0	0	0	0	0
23	10	10	0	0	10	10

11.1.2 20 Byte Payload

Timeslot (ms)	Packet Fault Rate (%) of Superframe					
	1	2	3	4	5	Max
24	10	10	0	10	10	10
30	0	0	0	0	0	0
27	0	0	0	0	0	0
25	0	0	0	0	0	0

11.1.3 30 Byte Payload

Timeslot (ms)	Packet Fault Rate (%) of Superframe					
	1	2	3	4	5	Max
25	0	0	0	0	0	0
24	10	0	0	10	0	10

11.1.4 40 Byte Payload

Timeslot (ms)	Packet Fault Rate (%) of Superframe					
	1	2	3	4	5	Max
25	0	0	10	10	0	10
26	0	0	0	0	10	10
27	0	0	0	0	0	0

11.1.5 50 Byte Payload

Timeslot (ms)	Packet Fault Rate (%) of Superframe					
	1	2	3	4	5	Max
27	0	0	0	0	0	0
26	0	10	10	0	0	10
25	0	10	10	10	0	10
24	10	0	0	10	0	10
23	0	10	20	0	0	20

11.2 Computation of Throughputs

Payload (bytes)	Transmissions Required	Timeslot Size (ms)	Superframe Length (ms)	Throughput (bytes/ms)
10	32	24	768	0.33
20	18	25	450	0.57
30	14	25	350	0.73
40	12	27	324	0.79
50	10	27	270	0.95

11.3 Synchronisation Constant

Timeslot Size (ms)	λ (ms)					
	1	2	3	4	5	Avg.
23	24	27	26	25	24	25
25	22	24	25	28	21	24
27	23	25	24	27	25	25
						25

11.4 Theoretical Response Time Computations

11.4.1 Fixed Priority Scheduling Response Times

Payload (bytes)	Theoretical Response time (ms) of each data transfer										
	1	2	3	4	5	6	7	8	9	10	Avg.
10	145	241	313	361	481	553	601	649	697	793	483
20	100	150	200	225	300	350	375	400	425	475	300
30	75	125	150	175	225	250	275	300	325	375	228
40	79	106	133	160	214	241	268	295	322	349	217
50	52	79	106	133	160	187	214	241	268	295	174

11.4.2 Smallest Transfer First Response Times

Payload (bytes)	Theoretical Response time (ms) of each data transfer										
	8	4	7	9	3	6	10	2	1	5	Avg.
10	73	121	169	217	289	361	457	553	673	793	371
20	50	75	100	125	175	225	275	325	400	475	223
30	50	75	100	125	150	175	225	275	325	375	188
40	52	79	106	133	160	187	214	241	295	349	182
50	52	79	106	133	160	187	214	241	268	295	174

11.5 Measured Response Times

11.5.1 Fixed Priority Scheduling Response Times

Payload (bytes)	Actual Response time (ms) of each data transfer										
	1	2	3	4	5	6	7	8	9	10	Avg.
10	145	242	312	360	483	551	601	649	697	798	484
20	101	153	203	224	302	351	378	403	426	479	302
30	75	125	149	178	225	250	279	300	330	378	229
40	80	108	134	164	214	241	268	298	324	352	218
50	52	79	106	130	155	187	215	241	268	275	171

11.5.2 Smallest Transfer First Response Times

Payload (bytes)	Actual Response time (ms) of each data transfer										
	8	4	7	9	3	6	10	2	1	5	Avg.
10	74	121	169	217	287	361	457	553	673	793	371
20	52	75	100	125	175	225	275	325	400	475	223
30	49	73	97	122	150	173	225	275	325	375	187
40	48	79	105	135	161	187	214	241	295	349	182
50	45	74	104	130	155	182	210	238	266	293	170

11.6 External Interference

(With a network timeslot of 27ms.)

Channel Hop Timeslot (ms)	Packet Failure Rate (%) of Superframe					
	1	2	3	4	5	Max
27	0	10	0	0	0	10
13	10	0	0	10	0	10
10	20	30	20	10	10	30
5	40	20	40	20	10	40

12 Discussion

12.1 Implementation

Through the basic testing methodology provided our implementation of LikeWHART within Mote Runner proved to be successful, but aspects which we defined within the specification did not come to fruition due to the limitations of the execution platform. The main loss of functionality – packet acknowledgement presents a huge concern over whether the choice of execution platform was completely justified and that the clear disadvantages of its development being discontinued and its lack of complete documentation were overlooked.

12.2 Timeslot Optimization

The timeslot optimization process was fairly laborious to begin with in order to discover the optimal timeslot for the 10 byte payload (24ms) but this was quickly overruled as a general trend was found. As we can see from the results for the 20 and 30 byte payloads and also the 40 and 50 byte payloads, the optimal timeslot converged on the same point – 25ms and 28ms respectively. Our hypothesis that the relationship between payload and timeslot size being linear was quickly disproved as timeslots recorded a small change of 1ms when doubling the payload from 10 to 20 bytes. In reality, the growth of the optimal timeslot size was incredibly slow.

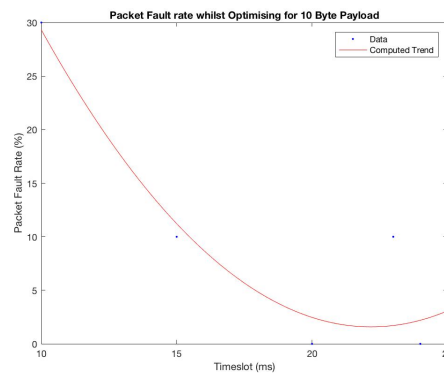


Figure 11 Change in packet fault rate whilst optimising the 10 byte payload

Figure 11 shows how important it is to find the optimal timeslot and to not use one which is too low. The fault rate based on the curve from the data points of the 10 byte payload optimization shows this importance as the packet fault rate grows exponentially as the timeslot size drifts lower than the optimal. This is an important phenomenon to observe when defining any new network and trying to “guess” which timeslot is right to use.

Using the optimised timeslot values and the data transfers previously specified, we were able to compute the throughputs (11.2) for each of the 5 payload sizes and compare them to each other. The trend found is that by increasing the payload size the throughput of the network increases with the 50 byte payload almost giving a

throughput of 1 byte per millisecond! Whilst this trend seems quite steady it is important to consider the role of the optimised timeslot size within it. If, for example an arbitrary timeslot size of 50ms was used for all 10 of the data transfers then the throughputs would be significantly lower than as seen here.

Without further decreasing the number of transmissions required for the payload size of 50 bytes by some method other than increasing the payload size (since this would make no difference), its throughput represents the upper bound for this data transfer specification – since no transfer exceeds the 50 Byte requirement.

We can conclude from the data we obtained during our optimisations that the trend between the payload and optimal timeslot length does require an increase as payload size increases, but it is not anywhere near what we hypothesised it would be. In reality a doubling of the payload size lead to only an increase in the timeslot of 1ms (from 10 to 20 bytes).

12.3 Scheduling

12.3.1 Synchronisation Constant

Computing the Synchronisation constant was relatively easy by using the logger output and this provided us with an average value of 25ms across the 3 different optimized timeslots from earlier. Tests across varying timeslot durations did not find this to be dependent on the value and it remained at an average of around 25ms for all timeslot values as previously optimized. This confirmed the conceptual belief that the synchronisation constant is in fact, a constant regardless of the timeslot size.

Within the synchronisation constant timeframe is contained the period of the network manager changing to the new channel as defined by the previous broadcast, finding the next channel to use and then broadcasting to all devices. The value of the synchronisation constant at 25ms has been found to be the approximate size of all of the timeslots and as such it is deemed to be a major contributor to the overall response time of any data transfer.

Although the values recorded vividly fluctuated between 24 and 28ms, it is of my belief that this was down to the methodology used to measure the value rather than an indication that the value is not constant. Throughout all methods of testing the logger's output has been found to fluctuate both in frequency and also has been observed to lag behind the actual visual simulation of the motes and their LEDs.

12.3.2 Fixed Priority vs Smallest Transfer First

Having found the optimal timeslot for each payload size and computed our network's LikeWHART synchronization constant (an average of 25ms) we were able to apply our response time analysis using both the Smallest Transfer First and Fixed Priority Scheduling policy models.

Having computed both sets of response times it's first of all noticeable that the maximum response time from either scheduling policy is the same for the final data transfer using the same payload size. This is because the number of transmissions required by the data transfers is the same, hence they shall take the same amount of time – throughput therefore being the same as well, since they are using the same timeslot sizes. The same can be said for the STF and FPS response times with a payload size of 50 since each data transfer only takes 1 transmission - order doesn't matter.

Just observing the results shows how STF provides much better average response times to those of FPS. This is down to the fact that STF can complete the earlier data transfers quicker since they are of a smaller size – hence a lower total sum of all response times. We can see how the synchronisation constant plays a large part in the response times for each data transfer – for example the first transfer using the 50 byte payload transmissions are almost double the optimized timeslot.

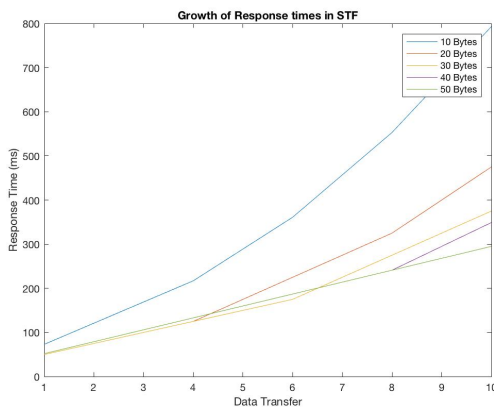


Figure 12 - Growth of STF Response Times

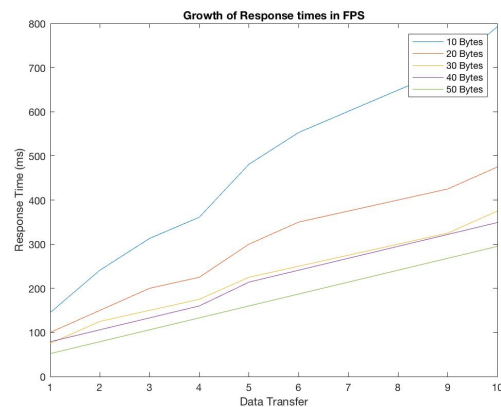


Figure 13 - Growth of FPS response times

Comparing STF and FPS in terms of the growth of their response times shows how the orderings tend to act with sets of data. On the left, STF shows how the response times take a more exponential growth compared to that of FPS which shows a linear growth. This is expected from STF since as the data transfers go on the transfer size increases and as such the respective response time does too. This growth leaves doubt as to whether in the context of data transfers with constrained rather than implied deadlines will all be deemed schedulable – since the growth of the response times is extravagant. FPS is quite simplistic in its linear growth since there is no

ordering of the data transfer based on their size properties and the assignment of priorities in our transfer specification is generally randomly distributed.

Having considered the theoretical values we can compare these to the actually measured ones. Generally, the averages of all response times for a given payload size are of the same value (to within a few ms) of the computed theoretical ones. The fluctuations in the actual response time values can be labelled as down to two reasons based on whether they increase or decrease:

The logger as a method for timing the duration for things to happen is a largely inaccurate method and heavily depends upon the efficiency of the simulation environment to produce an output. As such, the recorded values can be expected to fluctuate slightly within a few milliseconds because of the recording apparatus used.

Even though the theoretical response times are larger than the actual recorded values, this doesn't mean that the model is wrong. The response time calculated gives us the worst case time taken – i.e. the transmission takes up the duration of the timeslot. For most of the data transfers, as the maximum payload size increases they will not need the entire duration of the timeslot to send their data since they are of size smaller than the payload the timeslot is optimised for. Hence they shall complete sooner, giving a smaller actual response time.

12.4 External Interference

The results presented from our investigation into LikeWHART's managing of external interference are hardly surprising. It shows that as the frequency of channel changing (how often channels are changed) increases, the greater the packet fault rate within our LikeWHART network. This is due to the relationship between the timeslot size of our network and the frequency at which the external node is changing channels. However the results aren't consistent and this is down to the fact that is very much a case of what channels in what order were being used by the main network.

In the case of changing channel every 27ms the interference changes channel every time a new timeslot is occurring and as such it has a 1 in 16 chance to cause interference to a single transmission. As the rate of channel changing increases we can see how the packet failure rate increases since the likelihood of broadcasting on the correct channel increases – up to approximately a 5 in 16 chance when we use the 5ms channel hopping period for our interference generator.

If, in an extreme example the external node is changing channels at the rate of $1/16^{\text{th}}$ of the size of the main network's timeslot, then it is essentially guaranteed to cause any transmission to fail to avoid the interference – since all 16 channels can be broadcast on during a single timeslot of the superframe (giving a 100% packet fault rate). We can therefore provide no guarantees that LikeWHART can

cope with external interference if the source of it is chosen carefully. But this is reliant on the external device being able to switch channel and transmit within the limited amount of time given to it.

12.5 Goal Evaluation

Having completed the implementation and testing phase, it is important to assess which goals as set out in specification chapter were met and which were not:

Goal		Comment
All data transfers listed in the problem specification must be satisfied	Green	These can be scheduled within the implementation code using the structure defined in the implementation
The implemented protocol must protect itself from internal interference	Green	Using a TDMA approach we have defined which nodes are allowed to transmit when and all devices are kept in sync with each other through the use of a beacon broadcasting method.
The implemented protocol must protect itself from external interference	Red	Although a method to attempt to lessen the effects of external interference is implemented it still produces packet faults when an external node is producing interference and as such LikeWHART doesn't protect itself from external interference.
All data transfers must be completed within the period of the superframe	Green	All of the defined data transfers can be scheduled and completed within a single superframe based on our timeslot and orderings.
Data transfers should be scheduled based on some scheduling policy	Green	We have defined and tested two different policies for scheduling data transfers for use in LikeWHART
The implementation should have a low fault rate	Green	Ignoring the external interference difficulties we have successfully optimized the timeslot sizes such that the packet fault rate is 0 for all of them.
The implementation should allow parameters to be altered easily	Yellow	If the user understands the parameters and where to edit them in the code as well as being able to use the custom Mote Runner builder before loading

		the assemblies onto the motes then this goal is met. But for an average person this is not met.
A model should be produced which can compute the response times of the data transfers		We have successfully defined two different scheduling policies and their response time analysis as well as proving their validity using real response time measurements.
Performance measures should be defined such the functioning network can be assessed		We defined two measures that were used to compare different timeslots and payload sizes – throughput and packet fault rate.
The network should be optimized such that it provides the best performance possible		We successfully optimized the timeslot size for the network to give the best throughput for varying payload sizes.

13 Conclusions

This project has undertaken the task of specifying, implementing and testing of a new Wireless Sensor Network protocol technology, LikeWHART, based upon some of the already available choices. A data transfer specification was created that presented a simplified version of the scheduling problem faced by real-time wireless sensor networks within industry and our task was to create an implementation which could handle it. We produced two scheduling policies as well as their associated response time models: Smallest Transfer First (STF) and Fixed Priority Scheduling (FPS) based upon commonly used techniques. The implementation of LikeWHART closely followed our defined technical specification but we encountered issues with implementing the integrity-assuring packet acknowledgements due to the lack of documentation and discontinued support within the Mote Runner community. However, our implementation of the protocol fulfilled all other technical aspects and provided a reliable foundation on which to perform our investigations and analysis.

Having completed optimizations as well as some scheduling and integrity analysis we now have a functioning implementation of a specification of data transfers using the LikeWHART protocol within Mote Runner. Our results have shown how the relationship between different parameters and performance measures are all related and have to be considered in order to produce the best performing network given an arbitrary set of data transfers.

The timeslot size required for a network completely depends upon the maximum payload size possible for a transmission and has to be chosen such that the packet fault rate is 0 to avoid the loss of any data. This value should be chosen as the minimum size before the fault rate begins to increase in order to maximize the throughput of the network - send as much data in a period of time as possible. No explicit relationship has been found in order to produce a general formulae for computing the required timeslot size for a LikeWHART network. Hence in implementing any new LikeWHART network the selection of timeslot must be one of the first aspects to be resolved through investigations not-unlike those pursued in this report.

It has been deemed that in order for an increase of the payload size to cause an increase in throughput of the network two conditions must be met: 1) The new payload size must reduce the total number of transmissions needed; and 2) The new payload size should be large enough to significantly reduce the number of transmissions such that the increase in timeslot size does not lead to a larger superframe period. By meeting both of these conditions the throughput of the network should increase upon an increase in the size of the maximum payload.

We've seen how our two transfer ordering policies are able to order the transmissions of a superframe such that data transfers are completed in the

smallest amount of time possible; ordering by temporal or size of transfer requirements. Smallest Transfer First over our specification proved to give faster results with a lower average response time. However Fixed Priority Scheduling has been shown to have a more favorable linear growth in response times meaning that over larger sets of data it shall provide the better average response times as Smallest Transfer First follows a more exponential growth. We have also seen how our models have been proven to be true for response time calculation by comparing the computed theoretical with actual measured values within the simulation environment. We have therefore obtained one of our goals in being able to describe the real time guarantees of a communication within our LikeWHART network.

Data integrity of any WSN protocol is an important aspect that needs be considered and dealt with in order to produce a reliable network. With the lack of packet acknowledgement in the final implementation, ensuring that transmissions reached their destination regardless of the interference they potentially could undergo is even more important. The method of changing channel every superframe did not work in the fact that an external device was easily able to disrupt the network's transmissions.

The goals that were initially set out at the start of this project have been generally well met but some have been missed or only partially covered due to the inferior choice of execution platform at the start of this project.

To conclude, as an implementation and protocol, LikeWHART has been shown to not be capable of running faultlessly within a real-world environment where it could be subjected to masses of external interference – both intended and unintended. Its limitations as a static, non-multi-hop network make its use within any situation questionable and as such it needs to undergo a complete redesign in terms of its specification to provide use within real-time monitoring applications. However, we have successfully developed and analyzed models and also experiments which have demonstrated the functionality of the network and allowed us to determine its uselessness as a protocol.

14 Further Work

LikeWHART and the analysis presented in this project are comparatively basic compared to the well-developed research into wireless sensor network protocols such as WirelessHART. Much more work needs to be done in order for it to be deployable within an industry context but in a short period of time this is unfeasible. A lot of the issues discovered with LikeWHART were caused by issues with the implementation. Hence we break the further work down into the near future (within a few weeks) and considerable future (over the course of a few months).

14.1 Near Future

If only a small amount of time was available to continue the project the current implementation would not be added to due to the limitations of the execution platform and testing that would be required. Focus would be placed onto the analysis (including generating more data for the experiments already undertaken) including moving it onto the physical motes rather than using the simulator environment as well as creating response time models which allow for data transfers of varying periods.

More analysis on scheduling policies would be undertaken – whilst our analysis of both fixed priority and smallest transfer first were thorough (based on simplifying assumptions) there are numerous avenues which were left unexplored. Most importantly the technique of message packing would be explored for scheduling transmissions based on a defined network payload size. This would reduce if not remove the issue of idle time when large payload sizes are used for smaller data transfers as multiple data transfers can be (partly) packed into the same transmission, increasing the overall throughput of the network and potentially reducing the number of transmissions significantly. We would also create models which used data transfers that had constrained deadlines and potentially periods of less than that of the overall superframe. The timeslot/payload relationship would be continued to be explored and extended to find and prove a general model that defines the required timeslot size for a given payload size within a LikeWHART network.

14.2 Considerable Future

The most fundamental change to the implementation given a large amount of time to do so would be to change the execution platform. Mote Runner is a widely out of date and an unsupported platform and this showed through the lack of documentation that described the use of packet acknowledgments as well as omissions of basic radio-managing functions. Hence the execution platform severely influenced the final implementation. Recreating the implementation on a

more commonly used and supported platform would be the first step before making any major additions to the functionality of the protocol.

Once the change had been completed more robust external interference coping techniques would be introduced to the specification such as using blacklists to avoid channels that cause too high of a packet interference rate as well as the actual implementation of packet-acknowledgement. The supported topologies would also be widened such that data transfers could be routed through motes to destinations outside the source's transmission range. This would make LikeWHART both a more robust network to unwanted interference and also be able to support multi-hop routing such that networks could be spread over a larger geographical area. The relevant analysis and response time models would also be completed.

15 Bibliography

- [1] "How wireless sensors could help fight forest fires," BBC Click, 9 November 2017. [Online]. Available: <http://www.bbc.co.uk/news/av/technology-41847619/how-wireless-sensors-could-help-fight-forest-fires>. [Accessed 12 April 2018].
- [2] E. Ferro, V. M. Brea, D. Cabello, P. López, J. Iglesias and J. Castillejo, "Wireless sensor mote for snail pest detection," in *SENSORS, IEEE*, Valencia, Spain, 2014.
- [3] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich and I. Romanov, "Mote Runner: A Multi-Language Virtual Machine for Small Embedded Devices," in *Third International Conference on Sensor Technologies and Applications*, 2009.
- [4] [Online]. Available: <https://www.zurich.ibm.com/moterunner/>. [Accessed 2017].
- [5] A. Förster, "Introduction to Wireless Sensor Networks," John Wiley & Sons, Inc., 2016.
- [6] "What are Network Protocols?," [Online]. Available: <https://www.techopedia.com/definition/12938/network-protocols>. [Accessed December 2017].
- [7] V. Navda, A. Bohra, S. Ganguly and D. Rubenstein, "Using Channel Hopping to Increase 802.11 Resilience to Jamming Attacks," in *26th IEEE International Conference on Computer Communications*, 2007.
- [8] A. S. Tanenbaum, *Computer Networks*, Upper Saddle River, New Jersey: Prentice-Hall International, 1996.
- [9] A. Cunha, A. Koubâa, R. Severino and M. Alves, "Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS," in *IEEE International Conference on Mobile Adhoc and Sensor Systems*, 2007.
- [10] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, M. Nixon and W. Pratt, "WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008.
- [11] "IEEE 802.15.4, Standard for Low-Rate Wireless Networks," 2015.
- [12] B. Chen, M. Wu, S. Yao and N. Binbin, "ZigBee Technology and Its Application on Wireless Meter-reading System," in *IEEE International Conference on Industrial Informatics*, 2006.
- [13] T. Lennvall, S. Svensson and F. Hekland, "A Comparison of WirelessHART and ZigBee for Industrial Applications," in *IEEE International Workshop on Factory Communication Systems*, 2008.
- [14] October 2008. [Online]. Available: <http://searchnetworking.techtarget.com/definition/CSMA-CA>. [Accessed December 2017].

- [15] "ZigBee Specification," December 2014. [Online]. Available: <http://www.zigbee.org/download/standards-zigbee-specification/>. [Accessed December 2017].
- [16] "ZigBee PRO Specification," December 2014. [Online]. Available: <http://www.zigbee.org/download/standard-zigbee-pro-specification-2/>. [Accessed December 2017].
- [17] A. Burns and A. Wellings, *Analysable Real-Time Systems Programmed in Ada*, Fifth Edition, 2016.
- [18] H. Normak, "EMBS Mote Runner," 29 November 2013. [Online]. Available: <https://github.com/henrinormak/embs-moterunner>. [Accessed 12 March 2018].

16 Appendix A: Data Transfer Generator

```
import java.util.Random;
public class GenerateSchedule {

    public static void main(String args[]){

        char[] nodes = {'A','B','C','D','E'}; // The
5 nodes of the network
        int priority = 1;
        for(int i=1; i<=10; i++){

            //Randomly choose the values
            Random rand = new Random();
            int from = rand.nextInt(5);
            int to = rand.nextInt(5);
            int bytes = rand.nextInt(40) + 10;
//Between 0 and 40, so add 10 to make between 10 and 50

            while(from == to){
                //If the source destination are
the same, regenerate the from and to values.
                from = rand.nextInt(5);
                to = rand.nextInt(5);
            }

            System.out.println("Transmission from:
" +(nodes[from])+" to: " +(nodes[to])+" of " + (bytes) + "
bytes" +" with priority " + priority);
            priority++;

        }

    }

}
```

17 Appendix B – Network Manager Code

```
package implementation;
import com.ibm.saguaro.system.*;

public class NM {

    //NETWORK CONSTANTS
    private static byte XMIT_CHANNEL = 3; //The channel to start on
    private static int TIME_SLOT_LENGTH = 1000; //THE ACTUAL TS SIZE
    private static int SUPERFRAME_LENGTH = 10; // THE NUMBER OF TIMESLOTS IN
    THE SF
    private static byte[] CHANNELS = {3,2,3}; //The list of channels we
    switch between - can be any from 1 to 16
    private static int channel_counter = 0;;
    private static int bcast_counter = 1;

    //Radio Setup
    private static byte[] BCAST = new byte[16]; //The broadcast packet
    private static Radio radio = new Radio();
    private static byte panid = 0x01; //NM's address

    //Lights Setup
    private static boolean lights = false;

    //Timer Setup
    private static Timer btimer;

    static {

        //Open the radio and set it up on the fixed channel
        radio.open(Radio.DID,null,0,0);
        radio.setChannel(CHANNELS[channel_counter]); //set the initial
        channel

        radio.setPanId(panid, true);

        //Prepare the addresses of the broadcast
        BCAST[0] = Radio.FCF_BEACON;
        BCAST[1] = Radio.FCA_SRC_SADDRIRadio.FCA_DST_SADDR;
        Util.set16le(BCAST, 3, 0xFFFF); //PAN is set to broadcast for all
        of these messages, so all the mote will bother listening to them
        Util.set16le(BCAST, 5, 0xFFFF); //Broadcast
        Util.set16le(BCAST, 7, 0x22);
        Util.set16le(BCAST, 9, 0x01);

        //Setup the timer for the interrupts for broadcasts
        btimer = new Timer();
        btimer.setCallback(new TimerEvent(null){
            public void invoke(byte param, long time){
                NM.broadcastPulse(param,time);
            }
        });
        btimer.setAlarmBySpan(Time.toTickSpan(Time.MILLISECS,
        SUPERFRAME_LENGTH * TIME_SLOT_LENGTH));

    }
}
```

```

    public static void broadcastPulse(byte param, long timer){

        radio.setChannel((byte)CHANNELS[channel_counter]); //Set the
channel to this broadcast's next one
        channel_counter++; //Increment

        //Turn on or off the leds dependent on their current state
        if(lights) {
            LED.setState((byte)2,(byte)0);
            LED.setState((byte)1,(byte)0);
            LED.setState((byte)0,(byte)0);
            lights = false;
        }else {
            LED.setState((byte)2,(byte)1);
            LED.setState((byte)1,(byte)1);
            LED.setState((byte)0,(byte)1);
            lights = true;
        }

        if(bcast_counter == CHANNELS.length){
            bcast_counter = 0; // wrap back around to the start of
the channels list
        }

        if(channel_counter == CHANNELS.length){
            channel_counter = 0; // wrap back around to the start of
the channels list
        }

        //It's time for a broadcast i.e the start of the next superframe;

        BCAST[11] = (byte)CHANNELS[bcast_counter]; //Something to
identify the bcast
        bcast_counter++;
        //Prepare the bcast frame for transmit
        radio.transmit(Device.ASAPIRadio.TXMODE_CCA, BCAST, 0, 16, 0);
//Transmit the broadcast

        btimer.setAlarmBySpan(Time.toTickSpan(Time.MILLISECS,
SUPERFRAME_LENGTH * TIME_SLOT_LENGTH));

    }
}

```

18 Appendix C – Network Device Code

```
package implementation;

import com.ibm.saguaro.system.*;
import com.ibm.saguaro.logger.*;

public class NetworkDevice {

    //the network
    private static byte BCAST_CHANNEL = 3; //The broadcast channel we're
using for the network
    private static int TIME_SLOT_LENGTH = 1000; //Length of the timeslot
    private static int SF_LENGTH = 10;
    private static int MAX_PAYLOAD = 50;
    private static byte NEW_CHANNEL = BCAST_CHANNEL; //The channel that this
node should begin listening to for the first sync
    private static byte FUTURE_CHANNEL = 1;
    private static byte NEW_CHANNEL_TO_SET = 3; //The channel shall be the
next one we listen on (as defined by the superframe bc)
    private static boolean STARTUP = true;
    //This device's network details
    private static byte panid = 0x11;
    private static byte address = 0x11;

    //This device's transmission schedule
    private static int[] SLOT_TABLE = {1,2,3,4,5,6,7,8,9};
    private static int[] SLOT_FUNCTION = {2,1,2,1,2,1,2,1,2}; //1 = Transmit,
2= Recieve
    private static byte[] DESTINATIONS = {0x12,0x12,0x12,0x12};

    //private static int[] F1_SLOT_CHANNEL = {};
    private static int OFFSET_INDEX = -1; //Set it to -1 so we can keep track
    private static byte[] msg = new byte[(12+MAX_PAYLOAD)]; //Dynamic payload
size for when we change the payload for optimisations
    private static int DEST_INDEX = 0;

    //Radio Init
    static Radio radio = new Radio();

    //Timer Init
    private static Timer tslot;

    static {

        //Open and setup the radio
        radio.open(Radio.DID, null, 0, 0);
        radio.setPanId(panid, false);
        radio.setShortAddr(address);
        radio.setChannel(BCAST_CHANNEL);

        //Setup the basic frame parameters
        //Destination ADDRESS?

        msg[0] = Radio.FCF_DATA;
        msg[1] = Radio.FCA_SRC_SADDR|Radio.FCA_DST_SADDR;
        Util.set16le(msg, 7, panid); //This device's
        Util.set16le(msg, 9, address); //This device's
    }
}
```



```

        //Fill the packet payload with some dummy data
        for(byte i = 11; i < MAX_PAYLOAD + 11; i++){

            msg[i] = (byte)(i + 15);

        }

        //Setup the recieved frame callback
        radio.setRxHandler(new DevCallback(null){
            public int invoke (int flags, byte[] data, int
len, int info, long time){
                return NetworkDevice.handleFrame(flags,
data, len, info, time);
            }
        });

        ///The timer for offsets
        tslot = new Timer();
        tslot.setCallback(new TimerEvent(null){
            public void invoke(byte param, long time){
                NetworkDevice.timeSlot(param, time);
            }
        });

        //Start listening in the network
        // Put radio into receive mode
        radio.startRx(Device.ASAP, 0,
Time.currentTicks()+Time.toTickSpan(Time.MILLISECS, TIME_SLOT_LENGTH *
SF_LENGTH)); //listen for a V long time, since if we miss the first broadcast we
might got out of sync

    }

    private static int handleFrame(int flags, byte[] data, int len, int info,
long time){

        if(data == null){
            //The receiev period ended but we didn't get
anything.

            Logger.appendString(csr.s2b("None Recieved"));
            Logger.flush(Mote.WARN);
            return 0;
        }else if(data[11] <= 16) { //IF this is a broadcast i.e.
is it defining a superframe?? in which case it's sent a channel

            FUTURE_CHANNEL = data[11];
            ledsOff();
            LED.setState((byte)1,(byte)1); //Turn on the
green LED

            OFFSET_INDEX = 0;
            Logger.appendString(csr.s2b("Sync Recieved"));
            Logger.flush(Mote.WARN);

            tslot.setAlarmBySpan(Time.toTickSpan(Time.MILLISECS,
SLOT_TABLE[OFFSET_INDEX] * TIME_SLOT_LENGTH)); //Listen until the first offset
        }else{
            //This is just a generic message
            //Say that we've recieved it

```

```

        Logger.appendString(csr.s2b("Packet Recieved "));
        Logger.flush(Mote.WARN);
        LED.setState((byte)1,(byte)1); //Packet recieved,
set GREEN led

    }

    return 0;
}

public static void timeSlot(byte param, long time){
    ledsOff();

    if(OFFSET_INDEX == SLOT_TABLE.length){
        //End of superframe, go back to listening again
        // Put radio into receive mode for length of a
superframe, just incase we're a bit out
        OFFSET_INDEX = -1; //For purposes of relistening if needs
be.

        radio.startRx(Device.ASAP, 0,
Time.currentTicks()+Time.toTickSpan(Time.MILLISECS, (SF_LENGTH *
TIME_SLOT_LENGTH)));

    }
    else {

        //Do the operation for this offset
        if(SLOT_FUNCTION[OFFSET_INDEX] == 1){
            //Transmit a frame
            LED.setState((byte)2,(byte)1);
            Util.set16le(msg, 5, DESTINATIONS[DEST_INDEX]);
//The destination address we're sending this to
            Util.set16le(msg, 3, DESTINATIONS[DEST_INDEX]);
//The destination address we're sending this to
            radio.transmit(Device.ASAP|Radio.TXMODE_CCA, msg,
0, 61, 0); //Transmit the message to the destination

        }
        else if(SLOT_FUNCTION[OFFSET_INDEX] ==2){
            //Recieve a frame, so just listen
            LED.setState((byte)0,(byte)1);
            radio.startRx(Device.ASAP, 0,
Time.currentTicks()+Time.toTickSpan(Time.MILLISECS, (SF_LENGTH *
TIME_SLOT_LENGTH)));
        }
        OFFSET_INDEX++;
        if(OFFSET_INDEX == SLOT_TABLE.length){
            //No more operations to do, so just listen for
the next superframe

            //Open and setup the radio

```

```

                                radio.stopRx(); //stop the radio - this is more
a bug fix
                                radio.setChannel(FUTURE_CHANNEL); //change the
channel to the next one
                                int endOffFrame = (SF_LENGTH * TIME_SLOT_LENGTH) -
(SLOT_TABLE[OFFSET_INDEX -1] * TIME_SLOT_LENGTH); //How long to wait

                                tslot.setAlarmBySpan(Time.toTickSpan(Time.MILLISECS, endOffFrame));
                                }

                                tslot.setAlarmBySpan(Time.toTickSpan(Time.MILLISECS,
(SLOT_TABLE[OFFSET_INDEX] - SLOT_TABLE[OFFSET_INDEX-
1]))*TIME_SLOT_LENGTH); //Setup the timer again for next offset
                                }

                                public static void ledsOff(){

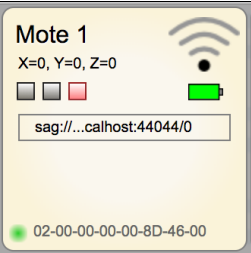
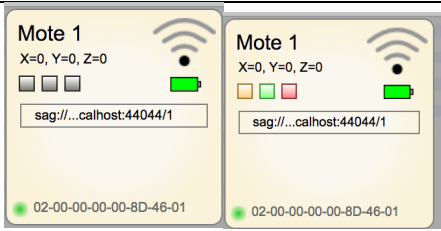
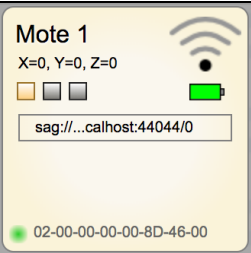

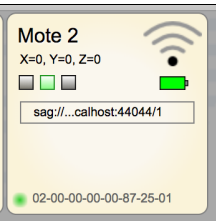
                                LED.setState((byte)0, (byte)0); //LED OFF
                                LED.setState((byte)1, (byte)0); //LED OFF
                                LED.setState((byte)2, (byte)0); //LED OFF

                                }

                                }

```

19 Appendix D – Testing Screenshots

Test One 	Test Five 
Test Two 	Test Six <pre> id: c75 time: 0:00:55.340'729 severity: 3 module: MAPP wtime: 569a7922f2a6a logmsg: (hal_log) Sync Recieved mote: 02-00-00-00-00-68-5A-01 msgtype: log-ev </pre>
Test Three 	Test Seven <pre> logmsg: (hal_log) Packet Recieved mote: 02-00-00-00-00-F9-F0-01 msgtype: log-ev id: 856 time: 0:00:43.930'434 </pre>
Test Four 	Test Eight <pre> time: 0:01:04.669'840 severity: 3 module: MAPP wtime: 56afa52fc03c1 logmsg: (hal_log) None Recieved mote: 02-00-00-00-00-34-FA-02 </pre>
Test Nine No output in the logger.	