3-2021

Simulating a Mobile Wireless Sensor Network Monitoring the Air Force Marathon

Matthew D. Eilertson

SIMULATING A MOBILE WIRELESS SENSOR NETWORK MONITORING THE AIR FORCE MARATHON

THESIS

Matthew D. Eilertson, Captain, USAF

AFIT-ENG-MS-21-M-031

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-031

SIMULATING A MOBILE WIRELESS SENSOR NETWORK MONITORING THE AIR FORCE MARATHON

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

Matthew D. Eilertson, B.S. E.C.E.
Captain, USAF

March 2021

AFIT-ENG-MS-21-M-031

SIMULATING A MOBILE WIRELESS SENSOR NETWORK MONITORING THE AIR FORCE MARATHON

THESIS

Matthew D. Eilertson, B.S. E.C.E.
Captain, USAF

AFIT-ENG-MS-21-M-031

Abstract

This thesis explores the feasibility of deploying a mobile Wireless Sensor Networks (WSN) to the Air Force (AF) Marathon in support of Air Force Research Laboratory (AFRL) research of sensor and networking infrastructure in denied or degraded environments. The marathon is a unique opportunity where large crowds, limited infrastructure, and emergency response teams intersect for a chaotic and contested environment. It is anticipated that if runners are equipped with wirelessly networked sensor devices, then sensor data can be collected through a mesh network and a limited number of gateways.

WSNs contain hundreds of devices called motes requiring significant investment to build and deploy. Therefore, in this research, a simulation called MarathonSim is developed in the Objective Modular Network Testbed in C++ (OMNeT++) Discrete Event Simulator to test the performance of a mobile WSN without committing extensive resources. The simulation is similar to a previous work called MarathonNet, but it includes IEEE 802.15.4 physical and link models and routing protocols. A full factorial design using numbers of runners, transmission powers, and flood versus Ad hoc On-Demand Distance Vector Routing (AODV) routing protocols is tested to determine factor effects, interactions, and estimates for Packet Delivery Ratio (PDR) to a central database, average end-to-end delay of application packets, and average power consumed per mote through the marathon.

The experiment results show flood routing delivers >50% of packets for 7 out of 15 trials and >75% for two trials. The AODV trials performed poorly due to a flaw with the module implementation not handling link breakages. Average delay varied from 0.11 to 7.2 seconds between 25 runners and 125 respectively but had no statistically significant changes due to transmission power. Finally, the average power consumed per node increased across all three factors but appears especially sensitive to additional runners due to increased receptions and transmissions.

By developing a modular simulation, this research assists other efforts into testing and prototyping mobile WSNs in new applications and demonstrating the feasibility of a runner-based WSN.

Table of Contents

Page

List of Figures

List of Tables

List of Acronyms

Abbreviation		Page
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Network	44
AF	Air Force	iv, 2
AFRL	Air Force Research Laboratory	iv, 2
ANOVA	Analysis of Variance	62
AODV	Ad hoc On-Demand Distance Vector Routing	iv, 4
BAN	Body Area Network	14
BER	Bit Error Rate	37
BI	Beacon Interval	16
CAP	Contention Access Period	16
CCA	Clear Channel Assessment	17
CFP	Contention Free Period	16
CSMA/CA	Carrier-Sense Multiple Access with Collision Avoidance	16
CSP	Collaborative Sensor Processing	13
CTS	Clear To Send	22
CUT	Component Under Test	51
DARPA	Defense Advanced Research Projects Agency	13
DoD	Department of Defense	23
DSDV	Destination-Sequenced Distance Vector	24
DSN	Distributed Sensor Network	13
DSR	Dynamic Source Routing	24
DSSS	Direct Sequence Spread Spectrum	14
EMS	Emergency Medical Services	2
FFD	Full Function Device	14
GIG	Global Information Grid	13

Abbreviation		Page
GTS	Guaranteed Time Slot	16
IDE	Integrated Development Environment	58
IEEE	Institute of Electrical and Electronics Engineers	3
IETF	Internet Engineering Task Force	20
INET	Information Network	26
INI	Initialization	26
IoT	Internet of Things	6
IP	Internet Protocol	20
ISM	Industrial, Scientific, and Medical	14
LLN	Low-power Lossy Networks	6
LoRa	LongRange	81
LoRaWAN	Long-Range Wide-Area Network	2
LR-WPAN	Low-Rate Wireless Personal Area Network	13
MAC	Medium Access Control	6
MANET	Mobile Ad hoc Network	24
MEMS	Micro-Electro-Mechanical Systems	8
NED	Network Description	26
OLSR	Optimized Link State Routing	23
O-QPSK	Offset Quadrature Phase-Shift Keying	14
OMNeT++	Objective Modular Network Testbed in C++	iv, 3
OSI	Open Systems Interconnection	26
PAN	Personal Area Network	4
PDR	Packet Delivery Ratio	iv, 55
PER	Packet Error Rate	37

Abbreviation		Page
PSDU	Physical Service Data Unit	35
QoS	Quality of Service	25
RERR	Route Error	20
RF	Radio Frequency	10
RFC	Request For Comments	20
RFD	Reduced Function Device	13
RREP	Route Reply	20
RREP-ACK	Route Reply Acknowledgement	20
RREQ	Route Request	20
RTS	Request To Send	22
SD	Superframe Duration	16
SFD	Start of Frame Delimiter	35
SoC	System on a Chip	4
SUN	Smart Utility Network	14
SUT	System Under Test	51
TTL	Time To Live	20
UDP	User Datagram Protocol	20
USAF	United States Air Force	2
UWB	Ultra-Wideband	14
WLAN	Wireless Local Area Network	3
WPAFB	Wright-Patterson Air Force Base	2
WSN	Wireless Sensor Networks	iv, ix

SIMULATING A MOBILE WIRELESS SENSOR NETWORK MONITORING THE AIR FORCE MARATHON

I. Introduction

1.1 Motivation

The sounds of cheering at the starting line faded nearly an hour ago and squinting into the distance, number 341 steadily grows beyond John's reach. He had passed John around mile five and proved a grueling pacer for the past fifteen minutes. Maybe it is pride that pushes John to try to keep up during his first marathon but the effort is exacting a terrible toll on his body. Each stride feels like his feet are dragging through sand and the sound of his ragged breath and thunderous footfalls echo around the nearly empty street. He gives a quick shake of his head to try to focus on his former partner as 341 is seemingly enveloped in a welcoming shade up ahead. John wistfully hopes it is a lone cloud passing through the clear blue sky, but it lingers too long and continues to darken. The road itself seems to be disappearing in the unnatural gloom and suddenly the world falls out from under him. The moment of panic washes away with a wave of pain as he crashes against the pavement. Stars of red and blue fill his vision as he struggle to catch his breath. Unable to stand or breathe, help is unlikely. However, those blue and red stars were not his imagination. An ambulance is already screaming up the road. Each flash of their lights met by a twin blinking from a sensor device clipped to John's runner bib.

While life-threatening events such as this are rare during a marathon, the sheer number of participants results in numerous injuries requiring Emergency Medical

Services (EMS). The annual Chicago and Boston Marathons each receive more than 30,000 participants where 400-600 runners require treatment at medical tents and up to 100 are transported to hospitals [1]. Recognizing injury, notifying, and dispatching Emergency Medical Services (EMS) is critical to preventing a tragedy. WSNs offer event coordinators the ability to monitor participant health and event security in real-time by equipping runners with sensor devices connected in a mesh network.

1.2 Problem Statement

The Air Force (AF) Marathon at Wright-Patterson Air Force Base (WPAFB) is an annual event where 10,000 athletes engage in a trek from the National Museum of the United States Air Force (USAF) through the interior of the base [2]. It is a unique opportunity to test emerging sensor and networking technologies while assisting participant safety and installation security. For example, advances in smart wearable technologies have led to smart shirts, shoes, watches, socks, and more becoming commercially available to monitor the wearer's health, location, and activities [3–6]. Additionally, networking standards such as IEEE 802.15.4, Bluetooth Mesh, and Long-Range Wide-Area Network (LoRaWAN) have simplified development of low-power networked devices. Equipping marathon participants with networked sensors to form a WSN can enable life-saving monitoring and simultaneously provide a test bed for further mobile WSN research and applications.

Despite the opportunity presented and technology being available, no such WSN application has been tested in a marathon scenario. In order to research movement patterns and sensor data fusion techniques, the Air Force Research Laboratory (AFRL) Sensor Directorate is exploring development of a WSN, or other infrastructure, to be deployed to the AF Marathon. However, WSNs are large-scale application-specific systems requiring multidisciplinary development efforts. Supporting that effort, this work seeks to answer whether a mobile WSN is feasible for the marathon scenario through simulating the performance of the network under varying parameters such as number of runners, transmission range, and routing protocols.

1.3 Hypothesis and Research Goals

This work hypothesizes that if equipped with mesh networked sensors, there exists a minimum quantity of runners such that the majority of sensor data can be delivered to a central database using multi-hop routing and a limited number of stationary gateways acting as data sinks.

The goals of this research are:

- Measure the capability of a marathon WSN to deliver sensor data to a central database.
- Analyze the interaction between transmission range and quantity of devices on network performance.
- Establish energy requirements of WSN devices.
- Evaluate routing protocols used in the WSN.

1.4 Approach

A simulation of the AF Marathon, called MarathonSim, is created in the Objective Modular Network Testbed in C++ (OMNeT++) Discrete Event Simulator with the INET Framework [7, 8]. Marathon runners are simulated to follow the actual route using average marathon finish times. Each runner is equipped with a sensor device, called a mote, to relay data throughout the course. Each mote contains an Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 Wireless Local Area Network

(WLAN) interface modeled after the Nordic Semiconductors nRF52840 System on a Chip (SoC) [9, 10]. At existing marathon infrastructure points, 19 gateways are deployed and act as data sinks for the WSN by transferring wireless communications to a wired network. Data is generated by the motes at regular intervals and routed using Ad hoc On-Demand Distance Vector Routing (AODV) and Managed Flooding. The simulations are conducted in a full factorial design varying the number of runners, mote transmission power, and routing protocol. Finally, the simulation data is collected and analyzed for viability and performance.

1.5 Assumptions and Limitations

The following assumptions are used in designing this simulation:

- Marathon finish times approach a normal distribution, allowing approximation of runner velocities.
- Motes are associated in a single IEEE 802.15.4 Personal Area Network (PAN) before the start of the simulation.
- Internet infrastructure is available to connect gateways and introduces negligible performance impact compared to wireless communications.
- Processing time and energy consumed by sensor and microprocessor operation is negligible compared to radio operations.

The following limitations are placed to ensure manageable scope:

- The motes do not produce real sensor data, just payloads for transmission.
- All wireless communication takes place in a single channel in the 2.4 GHz range.
- Only IEEE 802.15.4 non-beacon mode is simulated.

1.6 Research Contributions

This research contributes to the field of WSN research, specifically through the development of MarathonSim, a modular OMNeT++ simulation. MarathonSim tests the feasibility of mesh networks in a mobile scenario through varying physical and protocol factors. Additionally, MarathonSim promotes mote prototype development by modeling performance based on commercial devices.

1.7 Thesis Overview

This thesis is organized into six chapters. Chapter 2 contains a background on WSNs, a description of the IEEE 802.15.4 protocol and comparable protocols, a highlight of two routing protocols utilized in this work, and a summary of related research. Chapter 3 presents the simulation environment and modules used to develop the marathon scenario. Chapter 4 specifies the parameters and metrics used to conduct the marathon experiments. Chapter 5 analyzes the data collected from the trials. Lastly, Chapter 6 concludes this work by summarizing the work conducted, discussing the limitations of the simulation, and highlighting possibilities for future work.

II. Background and Literature Review

2.1 Overview

This chapter provides a background on the fundamental concepts and technologies used in this research. First, it introduces WSNs and Internet of Things (IoT) along with example applications to set the stage for this work. Next, it provides information on the physical and Medium Access Control (MAC) layers of IEEE 802.15.4 wireless protocol and briefly covers comparable standards. Then the following section describes the AODV and managed flooding routing protocols for mesh networks. Lastly, it summarizes research related to this thesis.

2.2 Wireless Sensor Networks and Internet of Things

2.2.1 General

WSNs are a class of Low-power Lossy Networks (LLN) defined by distributed resource-constrained devices sensing their physical environment and collaborating wirelessly with other nodes to maintain the network [11]. These networks originated from a drive in technology to bridge the gap between the physical and digital realms. In bridging the gap, the intent is to replace human-driven data input with an "Ambient Intelligence" for human interaction [12,13]. This idea may remind the reader of IoT and rightly so; a world filled with everyday items networked and exchanging data over the Internet enhancing user experiences is the goal of IoT. While WSN predate the concept of IoT by several decades, they are an integral part of current and future IoT by providing physical sensing of, and interactions with, their environment [14].

Most WSNs tend to take on a similar form of many small, wireless, resource-constrained devices, called motes, scattered around a physical environment and few resource-rich data sinks. Figure 1 depicts this general WSN architecture. The motes are scattered around a sensing field, collecting and relaying data back to the main-powered data sink over low data rate networks. The data sink acts as a boundary gateway, delivering the sensor data over the Internet to a server running various applications. The end users can then access their applications, fueled by the data collected by the WSN. Typically, the motes are homogeneous devices organized into a star, mesh, or clustered-hierarchy structure depending on the application and number from a handful to potentially thousands [15].

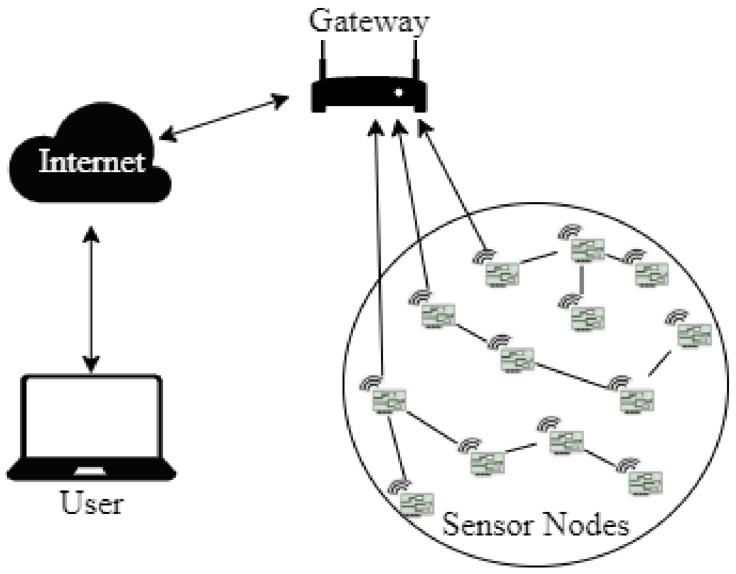


Figure 1: General WSN Architecture [16]

Motes, similar to embedded systems, are designed to be consumable, application specific, physically small, low power, and low cost per device [17]. While the design can vary greatly, the basic components are depicted in Figure 2. The five core elements are a controller, a sensor/actuator, a radio, memory, and a power supply. A controller is necessary to process data and execute code. A sensor or actuator allows interfacing with the physical world. The memory stores programs and data. The radio transmits and receives communication to and from the network. A power supply enables operation away from power infrastructure.

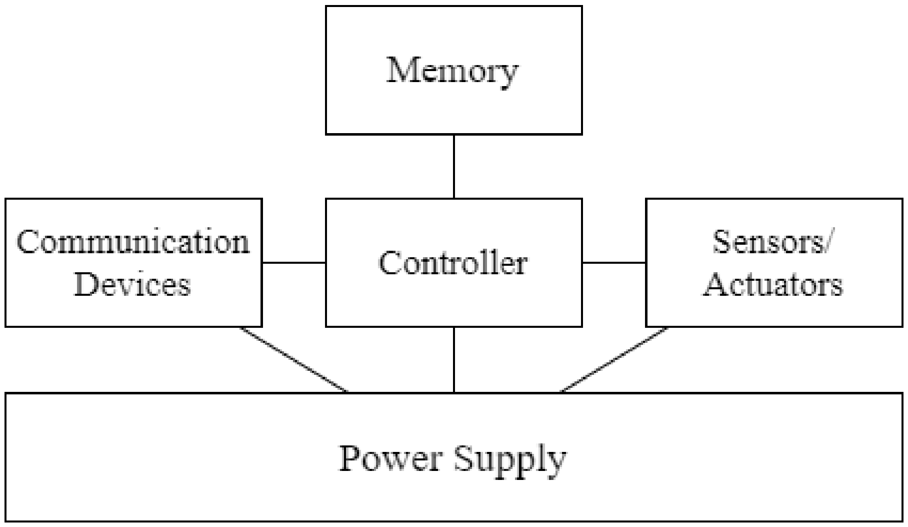


Figure 2: General Mote Architecture [12]

2.2.2 Mote Design Considerations

Of the five main components of a sensor mote, wireless communication through the radio and consumption of the power supply are focuses of most research. The trend in Micro-Electro-Mechanical Systems (MEMS) towards cheaper, more powerful, and efficient electronics relaxes the processing and memory constraints of motes. Instead, the comparative inflexibility of transceiver performance and power supplies force these two components to be brought to the forefront of mote design [18–20].

2.2.2.1 Power

While options exist for energy harvesting like solar panels, size and cost limitations prohibit most motes from utilizing them. Without energy harvesting, the WSN ceases to function when the network partitions due to failing motes. Simple solutions such as adding larger batteries or supplying additional motes to replace depleted motes are materially expensive. Therefore, WSN longevity depends on reducing the consumption of energy through management of the motes' operational states.

The basic functions of the mote are sensing, processing, and relaying data. Each of these functions consume energy depending on the state of the operation. The sensor module is usually either on, or off resulting in discrete states of energy consumption expressed as

$$E_{sens} = T_{on}P_{on} + T_{off}P_{off} \tag{1}$$

where the total energy consumed by a sensor is the time spent in each state multiplied by the power used in each state. Similarly, processors consume energy in running, idling, sleeping, and other modes shown by

$$E_{proc} = \sum_{i=1}^{m} P_{proc\_state}(i)T_{proc\_state}(i) \tag{2}$$

where for m total operating states of the processor, $P_{proc\_state}(i)$ is the power consumption in state i and $T_{proc\_state}(i)$ is the time spent in that state. Maximizing time spent in a sleeping or hibernating state is desired because a processor typically consumes <10 μA of current versus tens or hundreds of milliamps in idling or running states. Lastly, the energy cost of relaying data can be expressed as

$$E_{radio} = \sum_{j=1}^{n} P_{radio\_state}(j)T_{radio\_state}(j) \tag{3}$$

where for n total operating states of the transceiver, $P_{radio\_state}(j)$ and $T_{radio\_state}(j)$ represent the power consumption and time spent in state j. Combining (1), (2), and (3) provides a general form of

$$E_{total} = E_{sens} + E_{proc} + E_{radio} \tag{4}$$

representing the sum of energy usage in a single mote [21]. Overall consumption is used to determine a motes efficiency by determining the joules-per-bit cost of performing tasks. Energy costs can be reduced by maximizing time in low consumption states, but the cost of receiving and transmitting, E_{relay}, is the largest contributor to overall energy costs by three orders of magnitude [21].

A key mechanism in reducing communication costs is synchronizing neighboring motes to relay communications and minimize time spent receiving and transmitting unnecessarily [22]. The percentage of time in active radio states to inactive states is known as the duty cycle of the radio; in some references only transmissions are considered active, but this work considers both receiving and transmitting to be active states. Without synchronization, the duty cycle of the mote is close to 100% because the transceiver must remain on to receive all communications from neighbors in order to maintain network connectivity. In a perfectly synchronized network, motes can maintain reliable communications and achieve low duty cycles by transmitting and receiving only during allotted times and powering down otherwise.

2.2.2.2 Wireless Communication

While designing custom Radio Frequency (RF) transceivers for an application is an option, most requirements are serviceable by modern commercial RF transceiver modules. Mote design then comes down to selecting a transceiver that meets the specifications of the application such as transmission range, data rate, carrier frequency, modulation methods, or energy efficiency. These design factors can impact the others; for example, limited power supplies necessitate lower transmission power resulting in reduced range.

A useful tool in estimating communication range is a simplified link budget expression

$$P_{Rx} = P_{Tx} + G_{Tx} + G_{Rx} - L_{PL} \tag{5}$$

where received power P_{Rx} is equal to the transmitted power P_{Rx}, plus antenna gains G_X, minus path-loss L_{PL}. Transmitted and received power are measured as dBm (decibel-milliwatts), which is power relative to 1mW. Path loss, measured in dB, is the loss of power as the waveform propagates over distance [12]. The ITU Recommendation P.1411-9 for short-range outdoor line-of-sight path loss is

$$L_{LoS} = L_{bp} + \begin{cases} 20\log_{10}\left(\frac{d}{R_{bp}}\right) & \text{for } d \leq R_{bp} \\ 40\log_{10}\left(\frac{d}{R_{bp}}\right) & \text{for } d > R_{bp} \end{cases} \tag{6}$$

where d is propagation distance in meters, and R_{bp} is the break point distance in meters given by

$$R_{bp} = \frac{4h_1h_2}{\lambda} \tag{7}$$

where h_1 and h_2 are the heights of the transmitter and receiver in meters, and λ is the wavelength in meters. Also, from (6), L_{bp} is the path loss at the break point defined as

$$L_{bp} = \left|20\log_{10}\left(\frac{\lambda^2}{8\pi h_1h_2}\right)\right| \tag{8}$$

with the same parameters as (7) [23].

Using (5) and (6), an estimate can be made on transmission range between two transceivers. If, for a projected distance and transmission power, the received power exceeds a receiver's sensitivity, then the transmission may be successful.

More sophisticated link budget expressions can account for more physical phenomena, but the general rule with transmission range is in order to double the range, four times the transmission power is needed [12]. When the energy cost of increasing range may be considered too high for power-limited motes, a designer may opt for multiple intermediary motes with shorter transmission ranges for each device. This method, called Minimum Transmission Energy routing, has been shown to not always be efficient. Reception costs are not negligible. For each intermediary node, a flat reception cost is incurred and can outweigh the cost of a single long-range transmission [20].

2.2.3 Applications

As a technology envisioned to supplement existing application areas, WSNs are used wherever sensors are useful but difficult to permanently install. Interestingly, the surge in IoT interest has enveloped the field of WSN, causing the term to be used almost interchangeably, especially in commercial settings. Until recently, WSNs required proprietary communication protocols on custom hardware resulting in slow adoption of the technology focused on industrial uses. However, new network protocols like Thread, Bluetooth Mesh, and LoRaWAN have reduced the development burden, bringing sensor network capabilities to the consumer market.

2.2.3.1 Disaster Management

Disaster management operations are commonly cited as an application area for WSN as both predictors of events and supporting relief responses. As a predictor, sensors embedded in areas prone to fires, floods, earthquakes, and others work to detect disasters as part of an early warning system. Additionally, when disasters do occur, fixed infrastructure may become inoperable and limit emergency response. Sensor networks, attached to responders or deployed by them, can enable monitoring of assets in the disaster zone independent of existing infrastructure [24, 25].

2.2.3.2 Agriculture Monitoring

Agricultural monitoring through cellular IoT is predicted to improve crop yields by enabling continuous data to be collected on weather effects, soil condition, and presence of pests. The continuous monitoring allows farmers to make informed decisions in the usage of limited resources such as water and fertilizers as well as reducing wasted manpower by remotely monitoring crop health. Additionally, sensor data enables the introduction of advanced automated farm equipment, reducing manpower needs. Expensive all-in-one sensor suites have been available for years, but recently, amateur platforms using Arduino and Raspberry Pi are being utilized by small farmers and in the developing world [26, 27].

2.2.3.3 Military Surveillance

Battlefield monitoring and proximity defenses have been a continuous focus of WSNs for nearly four decades. One of the earliest proposals suggested utilizing a "Distributed Sensor Network (DSN)" of acoustic sensors in perimeter defense to alert anti-aircraft systems to the presence of hostile low-flying aircraft [28]. The Defense Advanced Research Projects Agency (DARPA), through the SenseIT program, investigated the challenges and feasibility of data fusion and Collaborative Sensor Processing (CSP) with experiments in classification of troop and vehicle movements [29]. Military applications continue to be investigated in support of the expansion of the Global Information Grid (GIG) with sensors integrating current and future systems.

2.3 Wireless Protocols

2.3.1 IEEE 802.15.4

2.3.1.1 General

The IEEE 802.15.4 standard, hereafter referred to as 802.15.4, defines the physical and link layer specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). These networks are often referred to as LLNs and are characterized by resource constrained devices communicating over a variety of links. The main objectives for these networks are low complexity, low cost, reliable data transfer, reasonable lifetime, and flexible protocols [30]. In 802.15.4, devices can be either Reduced Function Devices (RFDs) supporting bare minimum functions and single-hop routing or

Full Function Devices (FFDs) capable of additional network demands such as network coordination and multi-hop routing.

The target of the standard is IoT connectivity, but the latest standard, IEEE Std 802.15.4-2020, also includes modulation schemes and frequency bands for more specific applications such as medical Body Area Networks (BANs), Smart Utility Networks (SUNs), and Ultra-Wideband (UWB) devices [9]. Most commercially available 802.15.4 transceivers operate on the 868/915/2450 MHz Industrial, Scientific, and Medical (ISM) bands at rates up to 250 kbps with Direct Sequence Spread Spectrum (DSSS) Offset Quadrature Phase-Shift Keying (O-QPSK) modulation, among others.

2.3.1.2 Topology

The flexible design of 802.15.4 allows networks to operate in many structures such as star topology or peer-to-peer topology as shown in Figure 3. In both structures, the network contains at least one FFD functioning as the PAN coordinator. In the star network, all communication is between the PAN and members of the network. In the peer-to-peer network, communication can occur between any FFD-FFD or FFD-RFD pair in the network provided they are within radio range and previously associated with the PAN coordinator.

The base topology can be extended through higher-layer functions to more complex structures such as a tree or cluster tree in Figure 4. The trees are structured using parent-child associations between FFD coordinators within the same PAN. In this structure, communication only occurs between parents and children. A cluster-tree allows FFDs to maintain their previous PAN association while establishing their own independent PANs.

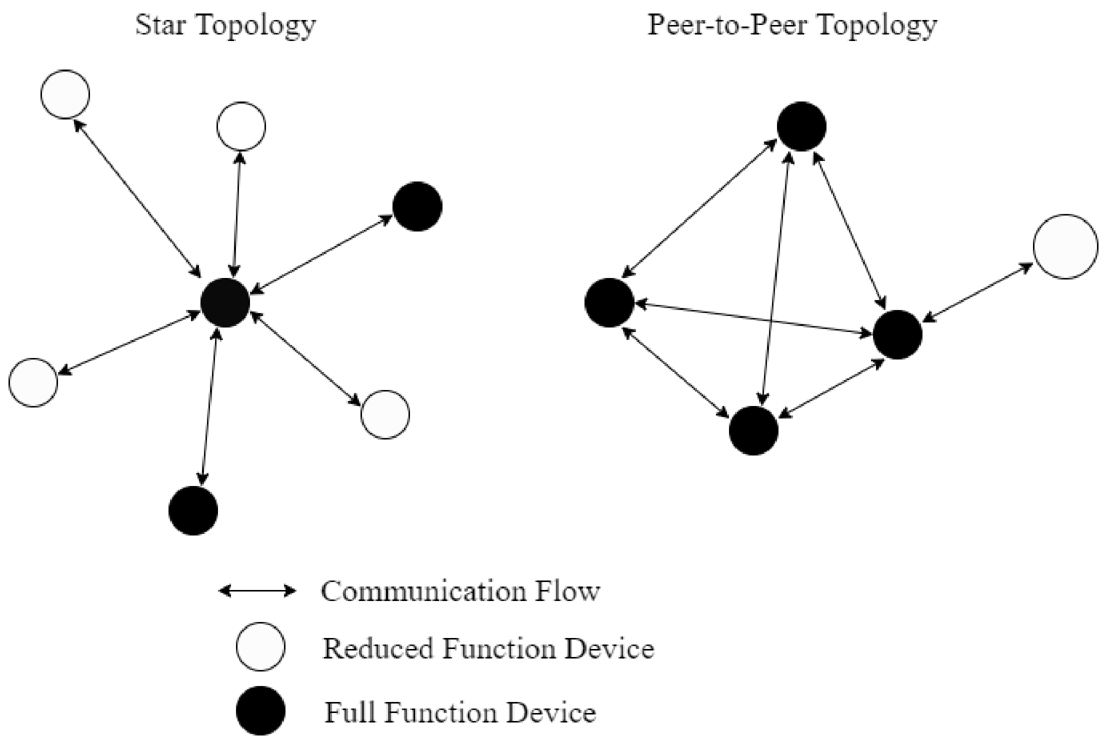


Figure 3: Star and Peer Topology for IEEE 802.15.4 Networks [9]

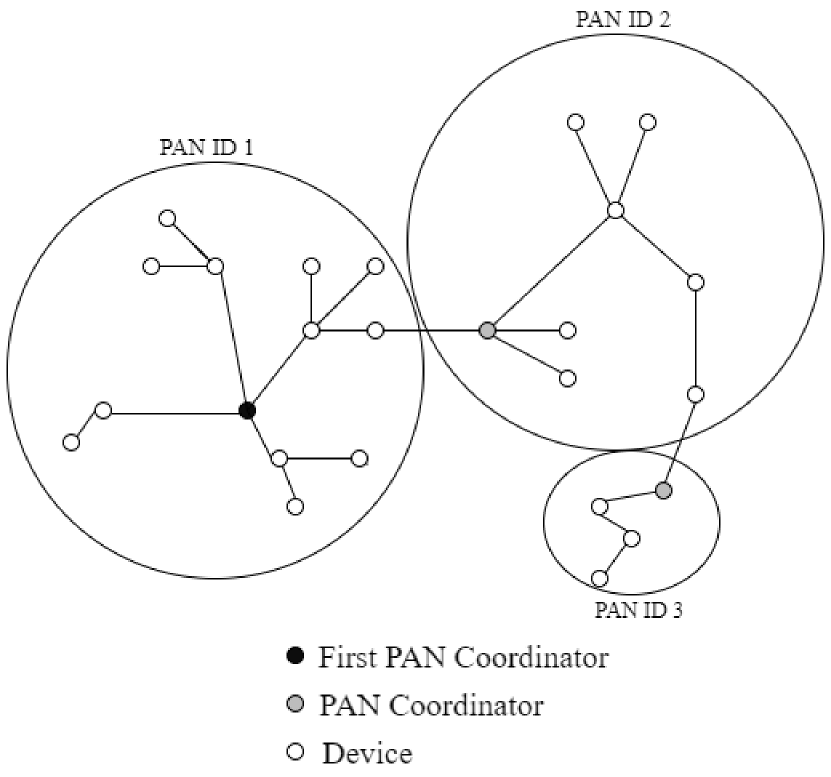


Figure 4: Cluster Tree Topology [9]

2.3.1.3 Medium Access Control

In addition to the topology, the PAN coordinator selects between beacon-enabled and non-beacon operating modes. The selection of operating mode controls how the devices associate with the PAN and share access to the medium for transmissions.

In the beacon-enabled mode, the coordinator broadcasts a superframe, enclosed by beacon frames, shown in Figure 5. The beacon frame synchronizes member devices, allows association of new devices, and describes the structure of the enclosed superframe. Upon entering the Contention Access Period (CAP), nodes are able to access the medium using slotted Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) for data transmission and association. Additionally, nodes can request a Guaranteed Time Slot (GTS) to be reserved in the next superframe during the Contention Free Period (CFP). The superframe ends with an inactive period during which nodes can sleep to preserve power. The intervals for the Superframe Duration (SD) and Beacon Interval (BI) periods are determined by the PAN values of *macBeaconOrder* and *macSuperframeOrder* which are transmitted within the beacon. This mode is available for both star and peer-to-peer topologies, but peer-to-peer topologies require advanced beacon scheduling from delegated coordinators.

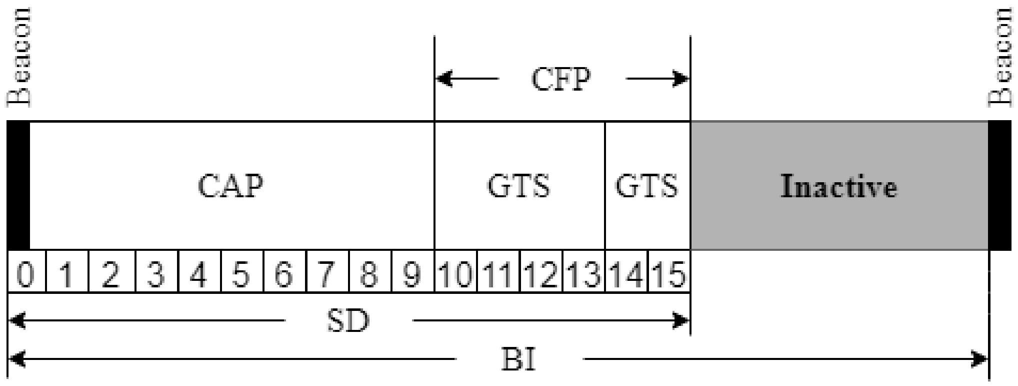


Figure 5: Superframe Structure Used in Beacon-Enabled Mode [9]

In the non-beacon mode, the PAN coordinator does not broadcast beacons unless a beacon is requested for association purposes. Without beacons, the member nodes must remain active at all times, draining power, unless lost transmissions are tolerated to allow asynchronous sleep states. When transmitting, nodes access the medium using unslotted CSMA/CA. Non-beacon mode in a peer-to-peer topology is intended to support mobile devices and ad hoc networks where synchronization with a coordinator is unreliable or unnecessary.

The algorithm for 802.15.4 CSMA/CA is shown in Figure 6. In both slotted and unslotted, the algorithm initializes *NB*, number of backoffs, to 0 and *BE*, back-off exponent, to *macMinBe*, minimum backoff. Slotted CSMA initializes *CW*, the contention window, and locates the backoff period boundary within the current slot before starting its randomly selected delay between $[1,(2^{BE}-1)]$ backoff periods. A backoff period is defined as 20 symbols of the utilized modulation method. Both modes continue delay countdowns regardless of activity on the channel. At the end of the delay, a Clear Channel Assessment (CCA) is performed, a success in the slotted case decrements CW and repeats CCA until CW periods of open channel. In the unslotted case, a successful CCA leads to an immediate transmission. Failure in either case requires an increment of the *BE*, *NB*, and selection of a new delay. Unlike other CSMA/CA implementations, failed acknowledgements do not trigger a re-transmission and instead are left to the frame's source service to handle.

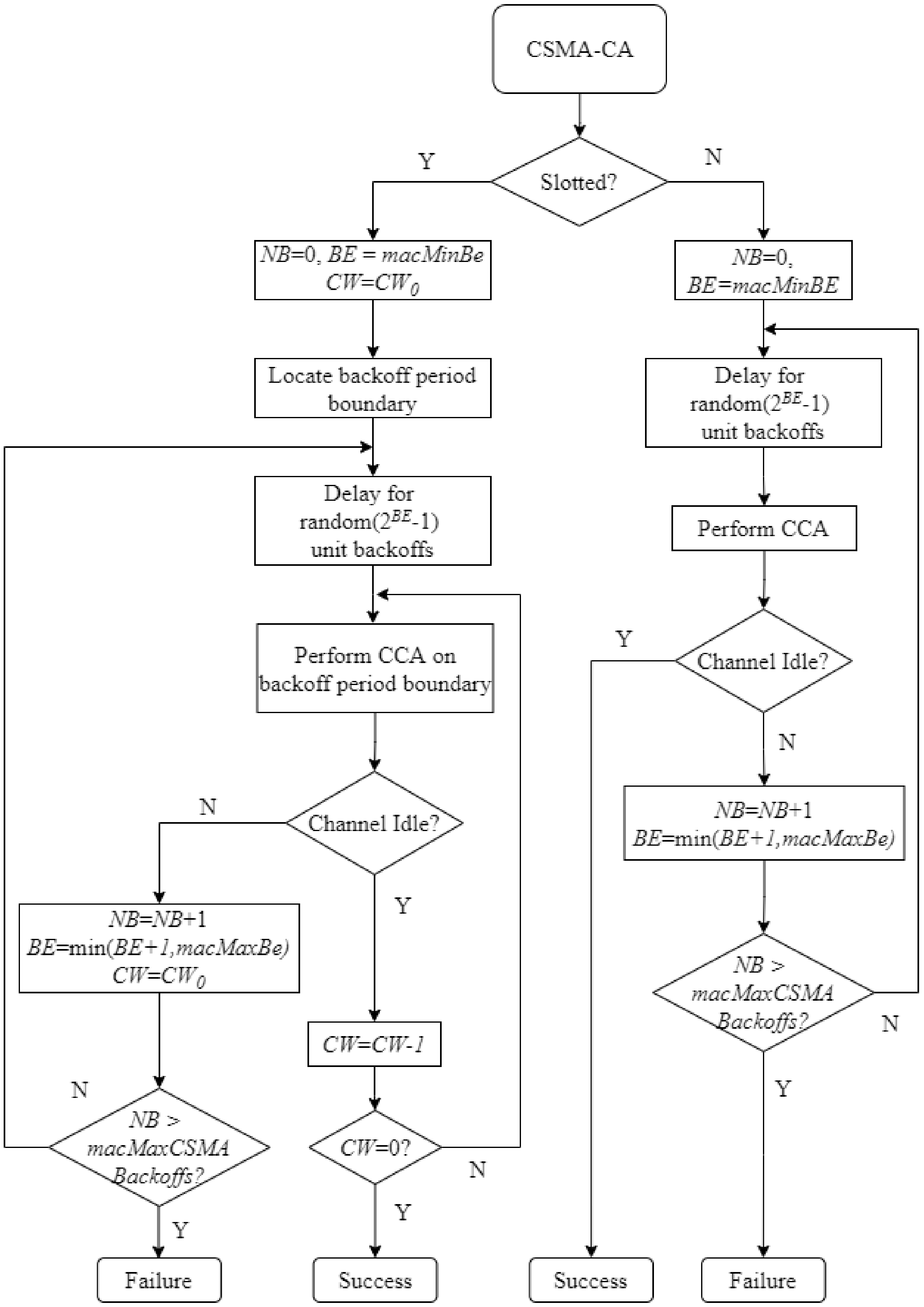


Figure 6: Slotted and Unslotted CSMA/CA Algorithm for 802.15.4 [9]

2.3.2 Alternative Protocols

While this research focuses on 802.15.4, alternative wireless technologies such as Bluetooth, LoRaWAN, and Wi-Fi were considered. Each technology is utilized in various IoT applications such as smart homes, industrial monitoring, personal health devices, and smart utilities. Bluetooth is most similar to 802.15.4 by providing short-range, low-energy mesh modes, but it is a full protocol stack standard. WiFi is near synonymous with wireless communication, but it provides higher data rates than needed for WSNs at high energy costs. Admittedly, the new 802.11ah amendment, referred to as ”Wi-Fi HaLow”, operates in the sub-GHz frequency band and boasts >10 Mbps data rates with ranges of over 1 km and multi-year battery life; however, commercial hardware is limited at this time [31]. Similarly, LoRaWAN, another sub-GHz standard, operates with low energy over 10 km but at much lower data rates.

2.4 Routing Protocols

2.4.1 General Categories

Routing protocols determine the methods by which network devices share information about the network in order route traffic between sources and destinations. Protocols are categorized as using link-state or distance-vector algorithms. Additionally, in ad hoc networks, routing protocols can be further categorized as reactive or proactive methods which determines whether routes are discovered as needed or actively discovered and maintained respectively.

Link state protocols flood the network with their local connectivity information for each router to construct their own table based on the received data. Each node performs some form of Dijsktra’s shortest path algorithm and sends link updates when changes are detected [32]. This method converges and responds to change quickly as updates are sent to all nodes but can be more difficult to manage.

Distance vector protocols implement Bellman-Ford's algorithm by iteratively distributing routing tables to neighbors [32]. Hop count is used as the metric for evaluating shortest paths to any particular destination. Iterative updates are slow to converge and do not respond well to topology changes. However, implementation is simple, requiring little expertise to manage.

2.4.2 Ad Hoc On-demand Distance Vector

AODV is a reactive distance-vector algorithm intended for use by mobile nodes in an ad hoc network. The Internet Engineering Task Force (IETF) proposed AODV as an experimental Request For Comments (RFC) in 2003 and the Zigbee Alliance's used it as the base for the Zigbee mesh routing protocol. The following is an abridged description of the protocol, a full version is available on the IETF website [33].

AODV operates over User Datagram Protocol (UDP) and Internet Protocol (IP) protocols using Route Request (RREQ), Route Reply (RREP), Route Error (RERR), and Route Reply Acknowledgement (RREP-ACK) message types. When a route to a new destination is needed or the current route in the routing table is no longer valid, a RREQ message is broadcasted to nodes within range. If any receiving node has a valid route more recent than indicated in the RREQ, then the node sends a RREP unicasted back to the requestor. If the receivers do not have a route, they will rebroadcast the RREQ while the Time To Live (TTL) is still valid. Nodes remember RREQ and originator pairs in case the RREQ is rebroadcasted back to them, in this case, the node will ignore the request.

Nodes monitor the links in their active routes to detect link breaks. When a link break occurs, such as by a failure to receive an acknowledgement of a unicast packet, a RERR message is generated and sent to nodes that have used the broken link previously. All nodes affected by the link breakage invalidate the route if it is still active.

Routes can also be invalidated if they have expired. When a RREP for a route is being returned, and when RREQ are received, nodes immediately add or update the current route to the sender in their routing table and set the *Lifetime* field of the route to the *ACTIVE_ROUTE_TIMEOUT* parameter. This field, along with *Destination Sequence Number* are used to maintain the freshness of routes. Once a route expires, it is marked for deletion.

Additional proactive route maintenance functions using *HELLO* messages are not used in this work.

2.4.3 Managed Flooding

Managed flooding is a basic broadcast routing method used by Bluetooth Mesh. In multi-hop mesh networks, broadcast routing utilizes multi-paths to deliver packets to the destination without predetermining a route as each receiving node relays the packet if it is not the intended recipient. There are two methods to restrict message rebroadcasting, the TTL of the message and node caching. The TTL of a message decrements after each reception, guaranteeing the broadcast will eventually stop and has a maximum value of 127. The node caching method adds messages to a list and compares newly received messages to the list before rebroadcast. If the message has been seen already, it is ignored, if not it will rebroadcast.

Wasted bandwidth from duplicated transmissions is directly related to the node density, or neighborhood size, of a network segment. In the best case of a series of N nodes connected in a chain, each node transmits a single packet per message but receives a rebroadcast from each neighbor. In the worst case where all N nodes are neighbors, each node receives the initial transmission then rebroadcasts for N transmissions and N receptions. For ad hoc networks, the performance costs introduced by duplicate broadcasts and collisions is somewhat countered by multi-path routes reaching the destination reliably without routing protocol overhead.

2.5 Related Research

This work builds on the MarathonNet system proposed by Pfisterer et al. in 2006 [34]. MarathonNet simulated a mesh network deployed to monitor runners at a marathon. The marathon scenario used runner data from the Hamburg marathon in 2005 to develop an interpolation of times along the track. They placed between 4 and 20 base stations, incremented two at a time, at equidistant spaces through the race to act as data sinks. A fixed quantity of 500 simulated runners carried networked sensors with ranges between 50-300 meters and attempted to maintain connectivity between data sinks. For trials with transmission ranges under 100 meters, less than 50% of devices maintained connectivity throughout the race. Trials with transmission ranges greater than 150 meters resulted in a majority of devices maintaining connectivity for the majority of the course. In their recommendation, a transmission range of 200 meters and 8 base stations enables 80% connectivity for 80% of the race. Additionally, they noted that transmission range affected connectivity more than additional base stations beyond 10.

However, MarathonNet used a custom simulator called 'Shawn' which is no longer available and publications do not address several assumptions made by the authors. First, the simulation does not model packet transmissions; it only monitors distance between nodes as the factor of connectivity. Secondly, it does not consider routing mechanisms required for an ad hoc network which could impact connectivity. Additionally, in predicting throughput capacity, the authors assume a Request To Send (RTS) and Clear To Send (CTS) mechanism is available to guarantee packet delivery which research has been shown to be unreliable in ad hoc networks and even increases congestion [35].

In their thesis from the Naval Postgraduate School in 2004, Bach and Fickel performed a feasibility analysis on 802.11 and 802.15.4 wireless networks within the context of the Department of Defense (DoD) GIG [36]. While the definition has changed over time, the GIG conceptually encompasses the interconnection of DoD information systems collecting and processing data to control equipment or services and inform warfighters or policy makers. The analysis included an overview of ad hoc routing protocols, mesh architectures suitable to the GIG, and experiments to gauge maturity of the technologies. In the overview of routing protocols, the authors concluded no single protocol would be suitable for GIG applications, and instead, proposed a hybrid implementation of several protocols for different parts of the GIG. They segmented the GIG into fixed-mesh, mobile-mesh, and sensor-mesh architectures. The fixed-mesh architecture represented a semi-permanent wireless infrastructure providing a "last mile" connection to wired Internet. The mobile-mesh architecture represented the idea of networked warriors operating outside of permanent wireless infrastructure. The sensor-mesh architecture represented low data rate, low power, ubiquitous intelligence informing other systems in the GIG. The experiments included trials of fixed and mobile mesh Wi-Fi networks formed by between 5 and 10 laptops using AODV, Optimized Link State Routing (OLSR), and other protocols. The distance between laptops varied between 40 and 100 meters through the trials. To test reconfiguration properties, latops walked out of range of their neighbors. OLSR proved to have lower latency than AODV with similar reconfiguration performance, but its proactive behavior decreased throughput. The authors concluded that the mesh networks performed adequately during the trials, but the technology needed to mature due to numerous failures during setup of the trials.

Kumar et al. conducted an analysis of several Mobile Ad hoc Network (MANET) routing protocols under three different mobility models. The simulations tested AODV, Dynamic Source Routing (DSR), and Destination-Sequenced Distance Vector (DSDV) routing protocols. Simulated nodes moved according to Random Group, Manhattan Grid, and Freeway mobility models. Additionally, number of nodes and node velocity varied across trials from between 10 and 50 nodes moving between 10 and 40 meters per second. The authors measured end-to-end delay and routing overhead for each protocol; in nearly all cases, AODV outperformed the others as the size of the network grew but performed poorly at high velocities [37].

2.6 Background Summary

This chapter introduces the concept of WSNs and IoT and some example application areas. It provides information on the 802.15.4 protocol and briefly highlights comparable standards. Then, an overview of AODV and managed flooding routing protocols is described since they are used in this thesis. Finally, related research is highlighted. However, outside of MarathonNet, very limited research has been conducted in this area. This research proposes a WSN implementation using mobile 802.15.4 mesh motes.

III. MarathonSim Design

3.1 Overview

This research develops an OMNeT++ simulation, called MarathonSim, to test the performance of a mesh LR-WPAN of sensor motes carried by runners during a marathon. The simulation models the intended use case of sensors monitoring the health and location of runners and transmitting the data through multi-hop routing to an EMS Hub where event coordinators can observe the participants. This is an extension of MarathonNet in Section 2.5 by modeling data transmission, power consumption, and routing with 802.15.4 capable motes and gateways using AODV and flooding protocols.

While MarathonNet concluded that marathon-wide mesh network connectivity is feasible, it lacked predictions of any Quality of Service (QoS) metrics such as delays and packet delivery reliability. Any attempt to develop a real marathon mesh network needs to be informed by expected performance for a given a set of hardware parameters and protocols. Therefore, MarathonSim simulates data transmissions using the 802.15.4 protocol to account for contention access and collisions affecting performance. Power consumed during transmissions and reception is measured to provide insight into energy efficiency. AODV and flooding routing protocols are utilized to model ad hoc network routing from two existing technologies, Zigbee and Bluetooth Mesh respectively.

This chapter introduces the structure of an OMNeT++ simulation, and it then describes the sensor motes, gateway, and marathon environment implementations of MarathonSim.

3.2 OMNeT and INET

The simulation is implemented in the OMNeT++ simulator and uses models from the Information Network (INET) framework [7, 8]. OMNeT++ is an open-source discrete event simulator used for modeling communication networks and other systems. The simulator can be extended by various frameworks and libraries. The most common extension is the INET framework which implements, among other things, the Open Systems Interconnection (OSI) stack, mobility models, and radio wave propagation.

There are four main components of an OMNeT++ simulation shown in Figure 7. At the lowest level is the *simple* module which describes a single functional component in a Network Description (NED) file and implemented in accompanying C++ files. The NED file uses the NED language to describe the module's *parameters*, *gates*, and event *signals*. These characteristics are used by the module's C++ implementation to define functionality such as the *initialize()* and *handleMessage()* behaviors. Simple modules can be connected and grouped to form *compound* modules where *gates* are connected through *channels*. Compound modules can use other compound modules as submodules, allowing hierarchical nesting of arbitrary depth. At the highest level is the *system* or *network* module which is a compound module itself. The last component is the Initialization (INI) configuration file, named `omnetpp.ini` by default, which defines runtime parameters and execution of the simulation.

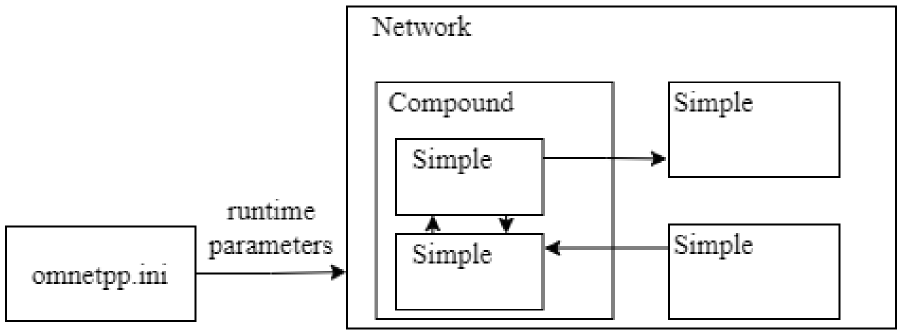


Figure 7: OMNeT++ Simulation Architecture [38]

3.3 System Summary

An illustration of MarathonSim is provided in Figure 8. At the highest level, the simulation consists of runners equipped with sensor motes producing data, a marathon route the runners follow, and infrastructure gateways placed along the route connected to an EMS Hub. As runners are moving through the course, the data produced is relayed from runner to runner until a gateway is reached and sent to the Hub. The simulation includes a radio medium model to describe the path loss of the radio waves. The Internet infrastructure is assumed to be available and acts as an ideal connection to the EMS Hub from the gateways.

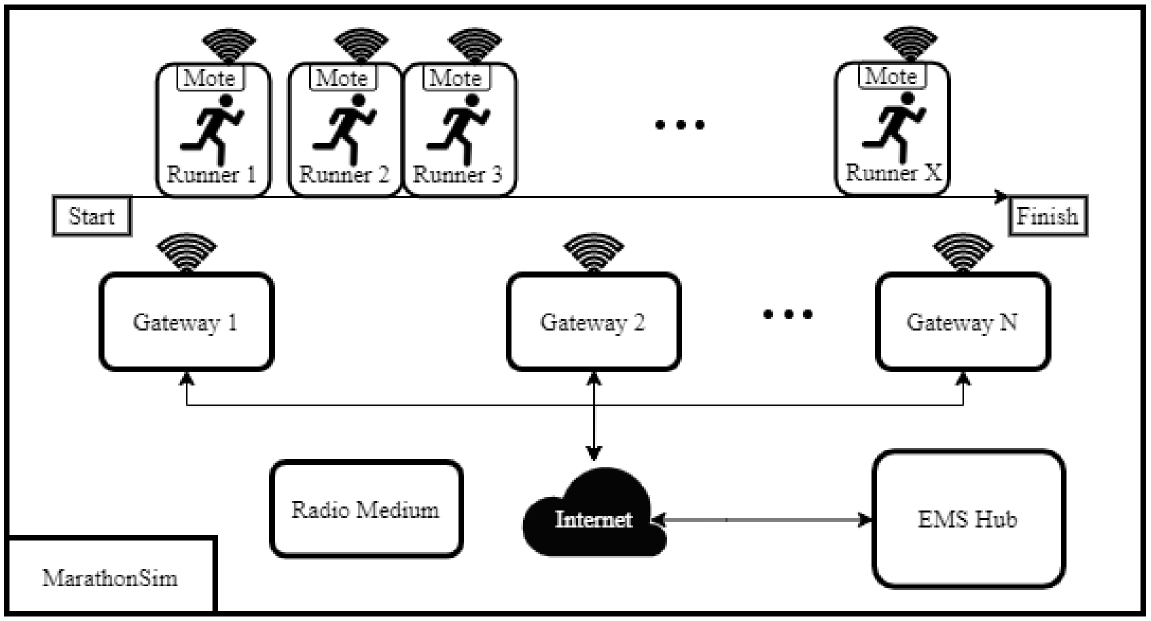


Figure 8: MarathonSim System Diagram

3.4 The MarathonSim Network

The `MarathonSim.ned` network module in Appendix A describes the system submodules and connections. The submodules reflect the entities shown in Figure 8 with a few minor changes. The runners and motes are combined into `host` submodules. The Internet infrastructure is modeled with idealized connections from each gateway to a `connectingSwitch` submodule connected to the EMS Hub. Additionally, the `configurator` submodule initializes IPv4 addresses and default routes necessary for AODV.

Without an INI file, MarathonSim is incomplete because the configuration file defines an instance of the network by specifying submodule types and parameters of each instance such as number of hosts in the race. MarathonSim's INI file in Appendix B is organized to construct instances of the MarathonSim network through composition of configuration sections. The following sections describe MarathonSim in more detail.

3.4.1 The Marathon Environment

MarathonSim uses the route ran by participants in the AF Marathon at WPAFB. The course, shown in Figure 9, is 42,195 meters long and includes 17 hydration stations and 8 co-located medical tents with the exception of the standalone finish line medical tent [2]. These points of temporary marathon infrastructure, plus one at the starting line, are used as locations of the gateways. These locations are selected as high likelihood placements in a real scenario while enjoying some degree of strategic deployment covering multiple overlapping race segments. Future work may include optimizing placement of gateways.

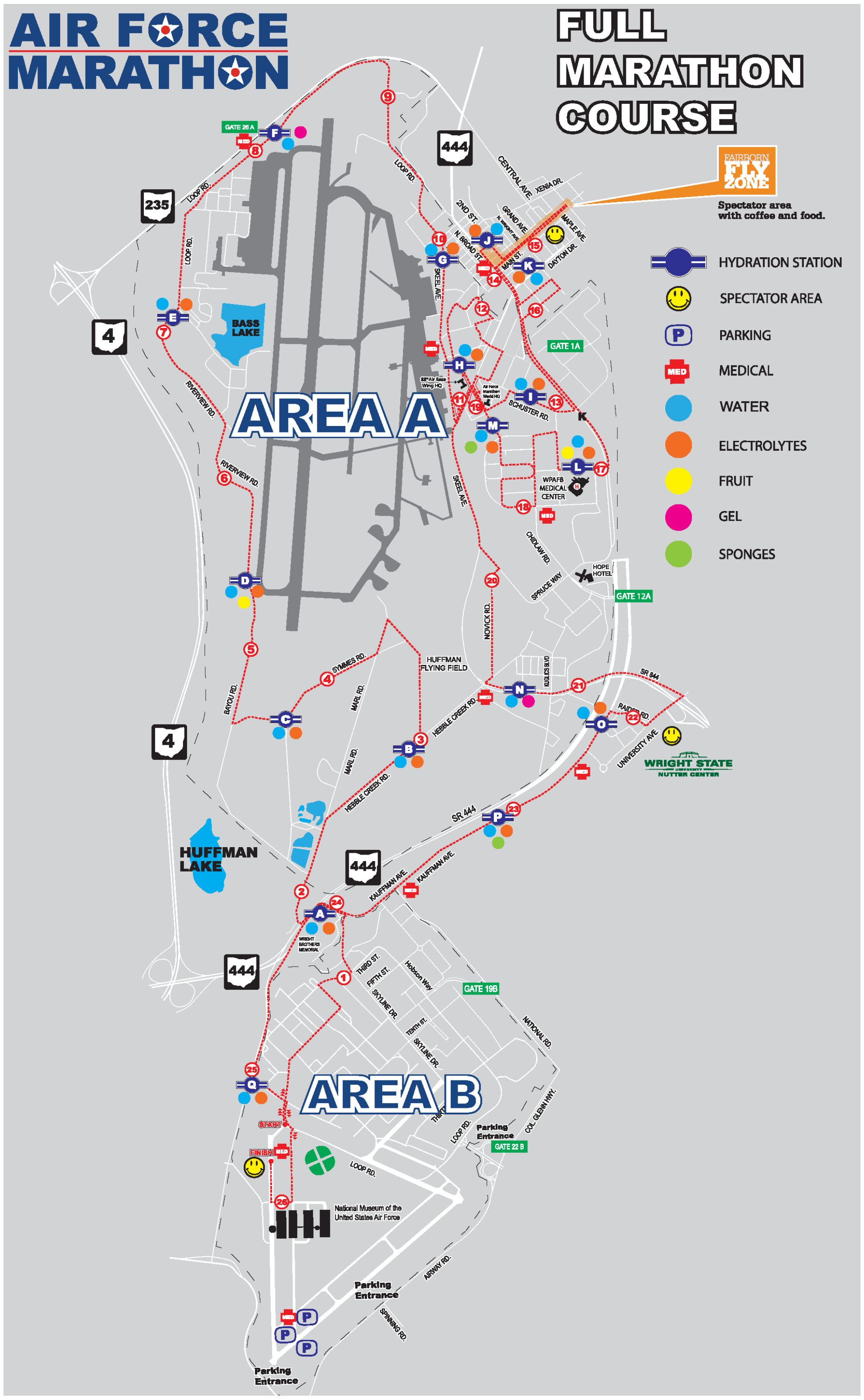


Figure 9: Full Marathon Route [2]

The simulated marathon route shown in Figure 10 is translated to the OMNeT++ environment by a Python script called `waypointScript.py` in Appendix C. First, the course is manually traced using a photo editing tool called Krita [39]. From the resulting trace, linear segments approximating the path are created by selecting waypoints and recording the associated pixel coordinates. A total of 67 waypoints and the 19 gateway positions are manually written to input text files `marathon\_wapoints .txt` and `gateways.txt` respectively. The script takes the pixel coordinates and calculates (X,Y) cartesian plane positions in meters. These positions are used in the INI file under `[Config LimitedGateways]` to place the gateways and mobility tracefiles to control mote mobility. The resulting route is 41,368 meters long, 0.5 miles short of a full marathon, but it is sufficient for the simulation. The loss of accuracy is due to approximating curves with straight lines. A more accurate representation would require additional waypoints to better approximate curves. However, future efforts should pivot to using mapping software to supply more exact route waypoints.

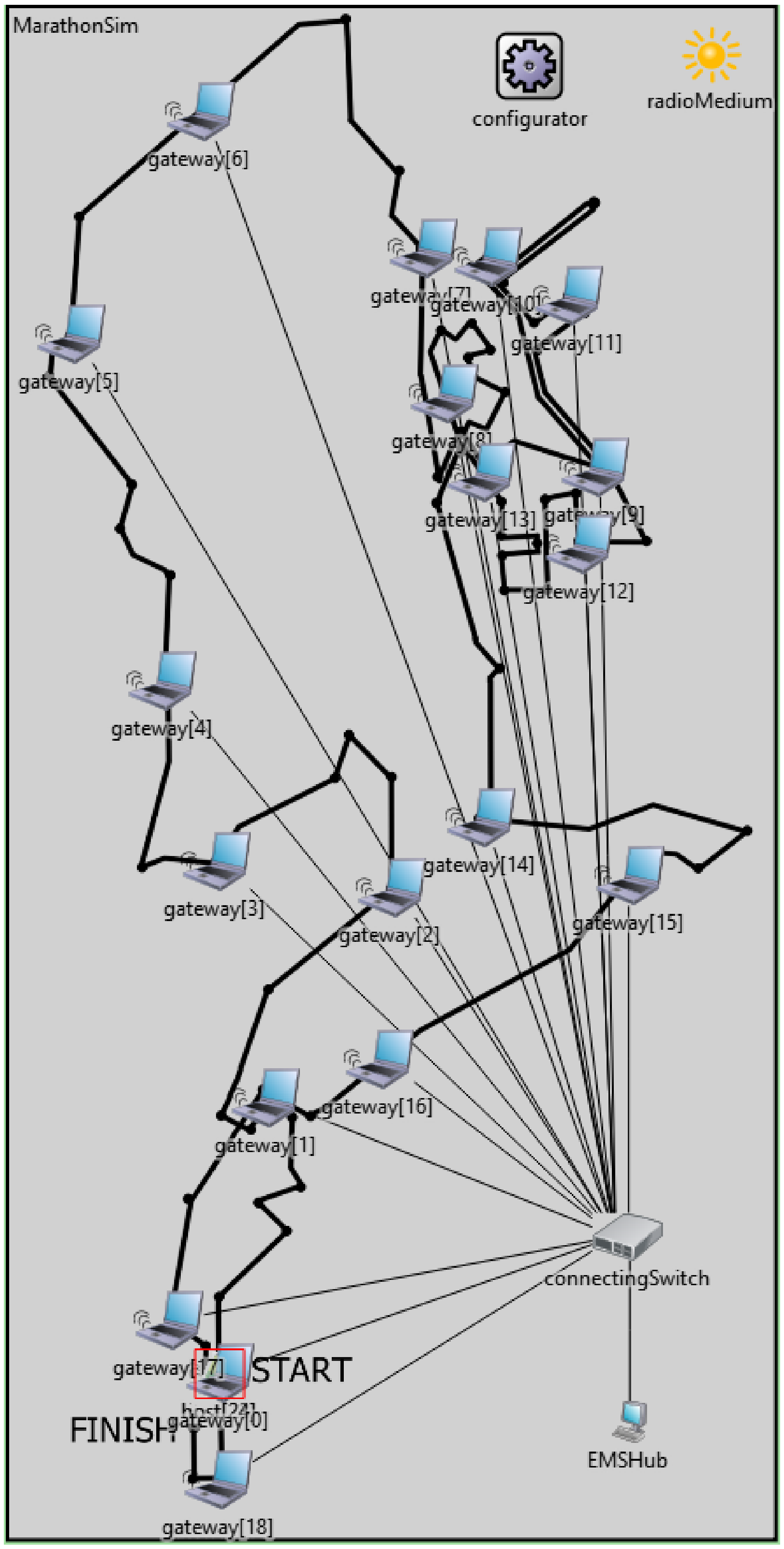


Figure 10: Simulated Route with Gateways Co-located with Marathon Infrastructure

3.4.2 Sensor Motes and Runners

The sensor motes and runners are combined into a single compound model depicted in Figure 11. The model is defined in `marathonSensorBase.ned` and extends existing INET models providing the layers shown by assigning the submodules of each layer. At the physical layer, the mote consists of an 802.15.4 transceiver, power submodules, and a runner mobility model. Above that is the link layer containing the 802.15.4 MAC model running in non-beacon mode. The network layer allows a modular assignment of either AODV or Managed Flood routing. The transport layer is only used when the mote uses AODV as it requires UDP datagrams. Finally, the application layer contains a data source application which emulates the production of sensor data. The following sections describe the model in more detail.

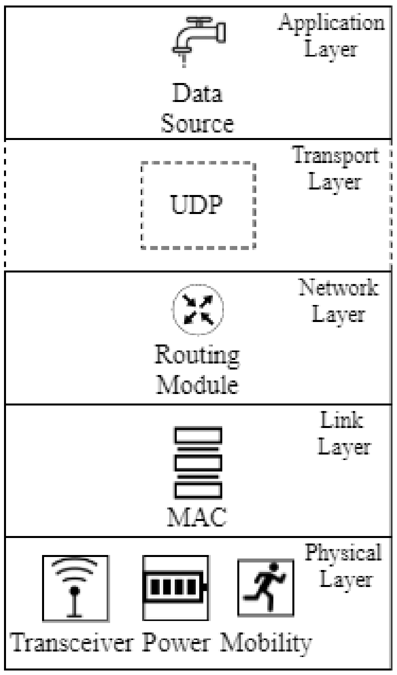


Figure 11: Mote Architecture

3.4.2.1 Mote Physical Layer

The physical layer of the motes is responsible for signal transmission and reception, power storage and consumption, and mobility. The performance of the radio is based on the physical characteristics of the nRF52840 SoC in Table 1. The nRF52840 is representative of modern SoCs available for LR-WPANs by supporting variable transmission power, low power consumption, and multiple communication protocols such as Bluetooth, Thread, 802.15.4, and proprietary 2.4 GHz protocols.

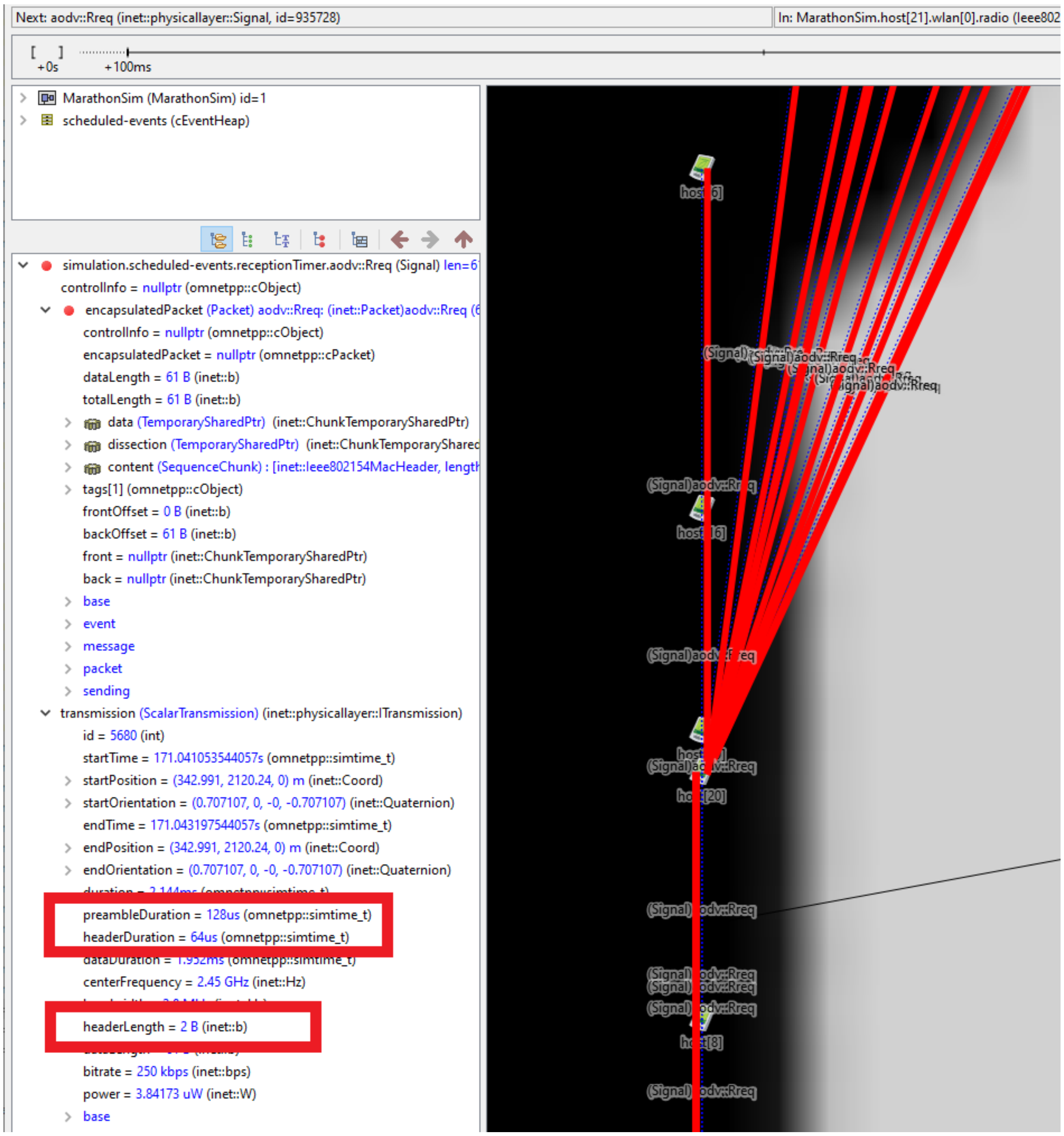
Table 1: Nordic Semiconductors nRF52840 Operating Attributes

Attribute	**Value**	**Description**
Transmission Power	0, 4, 8 dBm	The power output of the transmitter signal
Transmission Current Consumption	4.8, 9.6, 14.8 mA	Current consumed by the transmitter during transmissions with output powers 0, 4, 8 dBm
Receiver Sensitivity	-100 dBm	Minimum power required for signals to possibly be received
Receiver Current Consumption	4.6 mA	Current consumed by the receiver during receptions

Transceiver The motes use the INET `Ieee802154NarrowbandInterface.ned` module containing a `Ieee802154NarrowbandScalarRadio.ned` submodule which defines the physical reception and transmission of data packets and transceiver states. Through multiple levels of inheritance, this module contains a transmitter, a receiver, and an antenna modeling 802.15.4 physical layer specifications. The majority of the default parameters are not modified for this simulation.

The transmitter and receiver are set to `Ieee802154NarrowbandScalarTransmitter.ned` and `Ieee802154NarrowbandScalarReceiver.ned` respectively. These modules inherit from INET modules that implement dozens of convenience functions and parameters. They are set to operate on a single channel centered at 2450 MHz because the INET model does not support channel scanning and selection. However, DSSS O-QPSK modulation is implemented, allowing a 250 kbps data rate at a symbol rate of 62.5 thousand symbols per second, or 62.5 kilobaud (kBd).

Figure 12 shows an example of a transmission by a mote in MarathonSim. When the transmitter generates a transmission, it begins with a 4 byte preamble followed by a 1 byte Start of Frame Delimiter (SFD), and then the 1 byte physical header; the remainder of the transmission contains the payload, or Physical Service Data Unit (PSDU), up to a maximum of 127 bytes. The example transmission also shows the duration of the transmission of each component to be correct based on the 250 kbps data rate. The preamble and SFD plus header complete in 128 microseconds and 64 microseconds respectively.


```
Next: aodv::Rreq (inet::physicallayer::Signal, id=935728)
In: MarathonSim.host[21].wlan[0].radio (Ieee802
+0s    +100ms

> MarathonSim (MarathonSim) id=1
> scheduled-events (cEventHeap)

v simulation.scheduled-events.receptionTimer.aodv::Rreq (Signal) len=6
    controlInfo = nullptr (omnetpp::cObject)
  v encapsulatedPacket (Packet) aodv::Rreq: (inet::Packet)aodv::Rreq (6
      controlInfo = nullptr (omnetpp::cObject)
      encapsulatedPacket = nullptr (omnetpp::cPacket)
      dataLength = 61 B (inet::b)
      totalLength = 61 B (inet::b)
    > data (TemporarySharedPtr) (inet::ChunkTemporarySharedPtr)
    > dissection (TemporarySharedPtr) (inet::ChunkTemporaryShared
    > content (SequenceChunk) : [inet::Ieee802154MacHeader, length
    > tags[1] (omnetpp::cObject)
      frontOffset = 0 B (inet::b)
      backOffset = 61 B (inet::b)
      front = nullptr (inet::ChunkTemporarySharedPtr)
      back = nullptr (inet::ChunkTemporarySharedPtr)
    > base
    > event
    > message
    > packet
    > sending
  v transmission (ScalarTransmission) (inet::physicallayer::ITransmission)
      id = 5680 (int)
      startTime = 171.041053544057s (omnetpp::simtime_t)
    > startPosition = (342.991, 2120.24, 0) m (inet::Coord)
    > startOrientation = (0.707107, 0, -0, -0.707107) (inet::Quaternion)
      endTime = 171.043197544057s (omnetpp::simtime_t)
    > endPosition = (342.991, 2120.24, 0) m (inet::Coord)
    > endOrientation = (0.707107, 0, -0, -0.707107) (inet::Quaternion)
      duration = 2.144ms (omnetpp::simtime_t)
      preambleDuration = 128us (omnetpp::simtime_t)
      headerDuration = 64us (omnetpp::simtime_t)
      dataDuration = 1.952ms (omnetpp::simtime_t)
      centerFrequency = 2.45 GHz (inet::Hz)
      [illegible]
      headerLength = 2 B (inet::b)
      [illegible]
      bitrate = 250 kbps (inet::bps)
      power = 3.84173 uW (inet::W)
    > base
```

Figure 12: Example Transmission Showing Proper Modeling of Timing and Physical Protocol

Receiver sensitivity and transmitter power are assigned according to the nRF52840 parameters in Table 1. Since these parameters represent a specific instance of a device, they are assigned in the INI file described in Appendix B rather than the NED file.

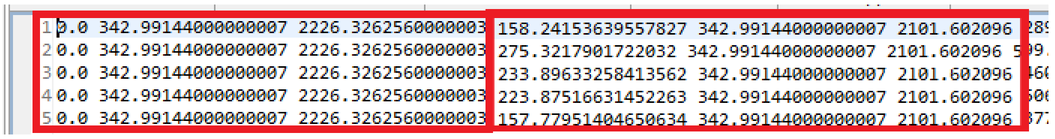
The reception of a transmission is possible if the received power is greater than the nRF52840 receiver sensitivity of -100 dBm. However, the reception is not guaranteed to be successful. Every reception is processed through receiver's error module to calculate a Packet Error Rate (PER) based on the modulation scheme's Bit Error Rate (BER). A random number is then generated and compared against the PER to determine a successful reception.

Power The mote module contains both power supply and power consumer modules to represent battery capacity and consumption. As noted in Section 2.2.2.1, operating the transceiver consumes more energy than any other mote operation. Therefore, only transceiver power consumption is considered and uses the nRF52840 consumption characteristics in Table 1.

Battery capacity is fulfilled through the `SimpleEpEnergyStorage.ned` module. At the start of a simulation, a `nominalCapacity` is assigned for maximum capacity, in Joules, which is also the default `initialCapacity`. The CR2032 lithium coin cell battery is an expected power source for small sensor motes and has a useful capacity of 2400 Joules [40]. In the worst case, if a transceiver is constantly transmitting for the 6 hour marathon, it is expected to consume only $0.0148\ A * 3.0\ V * 21,600\ sec = 959.04$ Joules. Therefore, motes are not expected to become exhausted during the marathon.

Power consumption is fulfilled through the `StateBasedEpEnergyConsumer.ned` module. This module functions similar to the equations in Section 2.2.2.1. As the transceiver transitions to sleep, idle, busy, transmitting, or receiving, a different energy consumption is applied for the duration of the state. The module uses milliwatts rather than milliamps, so a conversion is needed from Table 1 using $Power = current * voltage$ with the 3.0 V operating voltage of the nRF52840. Again, as a specific instance of a device, these parameters are assigned in the INI file.

Mobility The mobility of the motes is controlled through the `BonnMotion .ned` mobility module. BonnMotion describes the movement of an object through a series of non-punctuated (T X Y) tuples in a text file. Each tuple shown in Figure 13 represents the arrival of an object to point (X,Y) at time T. Each line of the file describes the motion of one object. Therefore, line 1 is assigned to mote 1, and describes starting at coordinates (342,2226) in meters and moving to (342, 2101) after 158 seconds. BonnMotion is based on the a mobility generation tool of the same name that generates mobility scenario trace files, but the generator is not used in this work [41].



```
1 0.0 342.99144000000007 2226.3262560000003 158.24153639557827 342.99144000000007 2101.602096
2 0.0 342.99144000000007 2226.3262560000003 275.3217901722032 342.99144000000007 2101.602096
3 0.0 342.99144000000007 2226.3262560000003 233.89633258413562 342.99144000000007 2101.602096
4 0.0 342.99144000000007 2226.3262560000003 223.87516631452263 342.99144000000007 2101.602096
5 0.0 342.99144000000007 2226.3262560000003 157.77951404650634 342.99144000000007 2101.602096
```

Figure 13: Example BonnMotion Tuples from Trace File

Since the route is very specific, trace files are created using `waypointScript.py`. The marathon route waypoints are used to generate a line in the trace file for each mote depending on the runner's randomly selected characteristics. As MarathonNet observed, and is seen from other sources, marathon runner finish times are close to normally distributed such that the random variable $X \sim N(266, 59)$ describes finish time in minutes [42]. In the absence of access to sample runner data, a runner's finish time is selected from X in `waypointScript.py` and divides the route length to get a velocity. However, a constant velocity for all runners is unrealistic; therefore, running behaviors are assigned randomly according to Table 2. These four behaviors include constant pace runners, interval runners, frequent walkers, and drop outs which are not accounted for in finish times alone.

Table 2: Running Behaviors

Runner Behavior	**Percentage Assigned**	**Description**
WALK_RUN	20	20% chance to walk segments
DROP_OUT	7.5	0.5% chance to drop out at each segment
WALKER	10	40% to walk after run, 80% to run after walk
RUNNER	62.5	Consistent pace throughout race

The behaviors in Table 2 adjust a runner's state as segments are completed. For each segment of the course, the segment distance and runner velocity are used to calculate an arrival time at the end of the segment. All runners start in a running state using their assigned velocities. Depending on their behavior, a runner changes their state between running, walking, or dropped out after each segment. For a WALK_RUN runner, at each segment the runner has a 20% chance to walk the segment, but it resumes running at the following segment. A DROP_OUT runner has a 0.5% at each segment to halt running and remain in place for the remainder of the marathon. A WALKER has a 40% chance to walk after each segment, and they return to running with an 80% chance at each segment. The RUNNER behavior maintains their velocity throughout the race. At each segment, all runner velocity varies by +/- 10% to introduce additional movement of runners in relation to each other. The resulting (T X Y) tuples are written to a default output file and must be copied to the `omnetpp.ini` configuration file. The number of runners generated is configurable through command line arguments such as:

```
python waypoint_script.py --num_runners=X
```

3.4.2.2 Mote Link Layer

The link layer is responsible for the validation and acknowledgement of data frames, PAN association, beacon operating mode, and the MAC protocol. The motes utilize the non-beacon operating mode due to their constant mobility changing the network structure especially when passing gateways. Additionally, as runners change positions or move out of range of each other, the motes' receivers must stay active to receive and relay communications reliably.

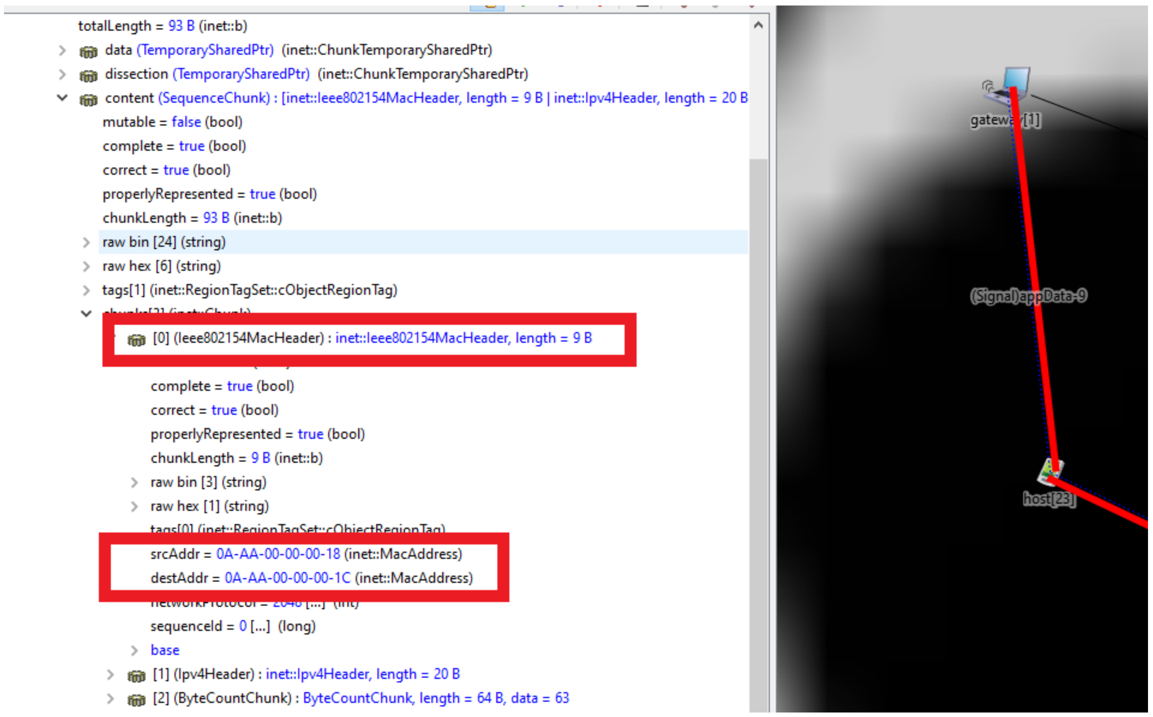
IEEE 802.15.4 Interface The `Ieee802154NarrowbandInterface.ned` module encapsulates the link and physical layers through the `mac` and `radio` submodules. The `mac` submodule is assigned the `Ieee802154NarrowbandMac.ned` module type to implement the unslotted CSMA/CA algorithm in Figure 6. Unfortunately, this module does not perform PAN association, management, or addressing. The lack of complete PAN management means the simulation must assume motes are associated prior to the start of the marathon.

Despite the CSMA/CA mechanism, transmission collisions are a concern. The default backoff exponent is `macMinBE=3`, meaning only $[0, 2^3 - 1]$ backoffs are available for nodes to select for delays. For sufficiently high traffic or quantity of nodes, collisions become likely. To decrease the likelihood of collisions, the `macMinBE` could be increased at the cost of higher average delays. For this system, collisions are dangerous because both routing mechanisms discussed in Section 3.4.2.3 rely on broadcast transmissions, which are not acknowledged under 802.15.4. The lack of acknowledgment messages means if a broadcasted transmission collides, it is irrecoverable. Other research has shown that to achieve a 90% probability of successful transmission under CSMA/CA requires

$$n < \frac{2^{BE}}{4} \tag{9}$$

where n is the number of devices attempting to transmit and BE is the backoff exponent [43]. For up to 125 motes attempting a transmission simultaneously, a `macMinBe=9` is necessary to achieve 90% success on the first attempt. However, a high density of motes is only expected at the start of the marathon when runners are together. Fast and slow runners move out of range within 5 minutes of the start of the race, reducing device density. This minimum backoff exponent, `macMinBE=9` , introduces an average single-hop delay of $\frac{((2^9-1)}{2} * 16\mu sec * 20symbols = 0.08176$ seconds where 16μ and $20symbols$ are parameters of 802.15.4 MAC and bitrate. This exponent could be reduced if collisions are infrequent.

Because motes are assumed to be associated before the simulation starts, addresses are assumed to be known a priori. This abstraction is performed at the network and link layers through the `GlobalArp` submodule allowing instantaneous simulation time address resolution. The module allows traditional 48 bit hardware addressing for destinations addressed by variable names, module names, or address protocol such as IPv4. However, 802.15.4 allows long, 64 bit, or short, 16 bit, addressing resulting in a header length between 9 and 25 bytes. Therefore, INET specifies physical transmission of MAC headers separate from simulation logic. An example is depicted in Figure 14 showing a transmission from `host[23]` to `gateway[1]`. The source and destination addresses are using traditional 48 bit hardware addresses for compatibility with simulation functions, but the actual transmitted header is specified to be 9 bytes long. 802.15.4 MAC headers are a minimum size of 9 bytes for associated, intra-PAN communication.



```
totalLength = 93 B (inet::b)
> data (TemporarySharedPtr)  (inet::ChunkTemporarySharedPtr)
> dissection (TemporarySharedPtr)  (inet::ChunkTemporarySharedPtr)
v content (SequenceChunk) : [inet::Ieee802154MacHeader, length = 9 B | inet::Ipv4Header, length = 20 B
    mutable = false (bool)
    complete = true (bool)
    correct = true (bool)
    properlyRepresented = true (bool)
    chunkLength = 93 B (inet::b)
  > raw bin [24] (string)
  > raw hex [6] (string)
  > tags[1] (inet::RegionTagSet::cObjectRegionTag)
  v chunks[3] (inet::Chunk)
      [0] (Ieee802154MacHeader) : inet::Ieee802154MacHeader, length = 9 B
          complete = true (bool)
          correct = true (bool)
          properlyRepresented = true (bool)
          chunkLength = 9 B (inet::b)
        > raw bin [3] (string)
        > raw hex [1] (string)
          tags[0] (inet::RegionTagSet::cObjectRegionTag)
          srcAddr = 0A-AA-00-00-00-18 (inet::MacAddress)
          destAddr = 0A-AA-00-00-00-1C (inet::MacAddress)
          networkProtocol = 2048 [...]  (int)
          sequenceId = 0 [...]  (long)
        > base
    > [1] (Ipv4Header) : inet::Ipv4Header, length = 20 B
    > [2] (ByteCountChunk) : ByteCountChunk, length = 64 B, data = 63
```

Figure 14: Address Abstraction from INET 48 bit Addressing to 802.15.4 Headers

3.4.2.3 Mote Network Layer

The network layer is responsible for packet forwarding and routing through the network. The 802.15.4 standard does not include specifications for the network layer, so MarathonSim uses methods from two other full stack LR-WPANs protocols, Zigbee and Bluetooth Mesh. These protocols use AODV and managed flooding respectively to route traffic to destinations.

AODV Routing In order to use AODV, the motes specify usage of IPv4 and the `Aodv.ned` routing module. IPv4 has historically been considered impractical for WSNs due to address limitations and protocol overhead, but it has been demonstrated to not be the barrier previously thought as hardware improved [44]. IPv6 is even used by modern IoT networks such as Thread protocol. However, IPv4 is required for the INET implementation of AODV. IPv4 is a built-in layer of INET modules and instantiated by default in the motes.

Within `marathonSensorBase.ned`, the `appRoutingModule` submodule is not defined and requires the INI file to specify the `Aodv.ned` type which executes the AODV protocol. Assigning the module and its parameters takes place within the [Config AODVBase] option. While the majority of the operating parameters remain default values, the INI file sets `deletePeriod=1s` to clear out inactive routes quickly. Experiments with adjusting other parameters such as `activeRouteTimeout`, `netDiameter`, and `ttlThreshold` failed to improve performance consistently.

Managed Flooding Similar to AODV, managed flooding is assigned through the INI config file under [Config FloodingBase]. This section removes the IPv4 layer from the motes and replaces it with a `SimpleNetworkLayer.ned` using the `Flooding .ned` network protocol type functioning as described in Section 2.4.3. The INI file configures flooding parameters. The max number of remembered message-origin pairs and deletion times are set to $n * m$ and 21600 sec respectively. Max entries is a variable parameter based on the number of motes, n, in that instance of the network. It is multiplied by the total messages sent by a mote, m, to remember every message sent by every mote to prevent duplicates being sent to the EMS Hub. This is an unrealistic expectation due to inefficient use of limited memory. However, it does not affect simulated performance, and other methods to detect message duplications at the destination failed.

3.4.2.4 Mote Transport Layer

When AODV routing is used, the transport layer is required to be instantiated because the AODV module uses UDP datagrams. The datagrams introduce an 8 byte header penalty compared to just flooding, but the packet size remains under the 127 octet requirement of 802.15.4. Since 802.15.4 does not include networking or transport layer mechanisms, end-to-end transmissions in MarathonSim are unreliable. The absence of end-to-end reliability illustrates the baseline network reliability and whether further mechanisms are needed.

3.4.2.5 Mote Application Layer

The intended application of the motes is to collect sensor data on the runner's current health and position periodically and transmit it to the EMS Hub. Therefore, each mote contains an `IpvxTrafGen.ned` submodule to generate sensor data during the race. However, data generation is simulated only to test network performance; meaning a payload is needed but not actual sensor data. An example data payload is shown in Table 3 containing a total of 32 bytes across six data fields.

Table 3: Estimated Sensor Payload Size

Data	**Type**	**Size**
Runner_ID	int	4 bytes
Heartrate	int	4 bytes
Temperature	int	4 bytes
Position	double	8 bytes
Seq_No	int	4 bytes
Timestamp	double	8 bytes

As mentioned in Section 3.4.2.2, 802.15.4 sets a maximum frame size of 127 bytes with between 9 and 25 bytes reserved for MAC headers. Therefore, a 'safe' payload must be less than $127 - 25 = 102$ bytes, but devices using the newest standard can handle $127 - 9 = 118$ bytes. Fragmentation is not supported by 802.15.4, so higher layer protocols such as IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) must be used if larger payloads are needed. However, this work limits payload size specifically to avoid fragmentation.

A mote simulates sending sensor data every 60 seconds. It collects one hypothetical sensor measurement every 30 seconds, resulting in 64 data bytes every one minute.

This payload, plus potentially 20 bytes from IP and 8 bytes from UDP, fits within the 'safe' payload size. Changes in runner status such as velocity and heart rate are expected to be low, so the 60 second interval provides a regular update to the central database and reduces the probability of collisions when accessing the medium.

3.4.3 Infrastructure Gateways

The gateways are primitive models of Thread and Zigbee Border Routers [45]. In MarathonSim, the gateways act as the border between the low-rate, low-power 802.15.4 wireless communication of the motes and high-speed Ethernet or fiber backbone of the Internet. For consistency and because the nRF52840 is utilized in Nordic Semiconductors' guide for DIY border routers, the gateways are designed similar to the motes as shown in Figure 15. The physical layer has a transceiver for wireless communication and Ethernet port for connecting to the Internet. The remainder of the layers are the same except for the lack of an application layer.

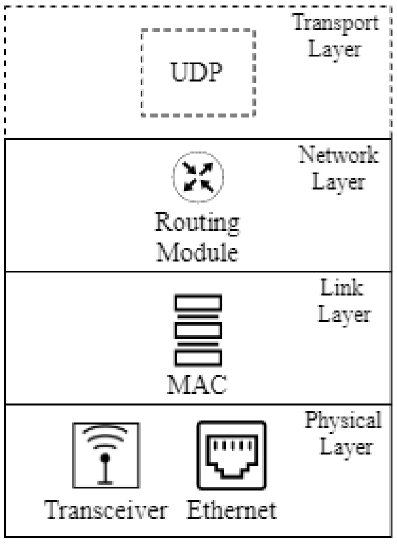


Figure 15: Gateway Architecture

3.4.3.1 Gateway Physical Layer

Transceiver The gateway transceiver is identical to the mote transceiver described in Section 3.4.2.1. It is assigned simultaneously in the INI file.

Ethernet The gateway Ethernet port is connected through a 100 Gbps Ethernet connection to a switch that connects to all other gateways and the EMS Hub. Additionally, the connection is simulated as only 1 meter long to minimize propagation delay. These unrealistic parameters isolate network performance metrics to the wireless components.

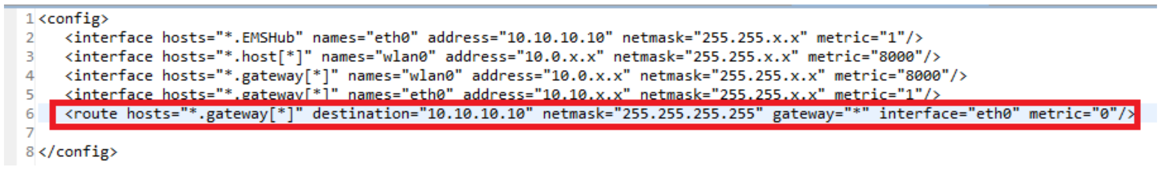
Mobility Despite being stationary, the gateway positions are assigned using the `StationaryMobility.ned` module through the INI file. When running `waypointsScript.py`, another output file called `omnetppini.txt` is generated that contains the INI parameters to assign all gateways to their positions when copied into the [Config LimitedGateways] section.

3.4.3.2 Gateway Link Layer

IEEE 802.15.4 Interface The gateway link layer is identical to the mote link layer described in Section 3.4.2.2. It is assigned simultaneously in the INI file.

3.4.3.3 Gateway Network Layer

The gateway network layer retains an IPv4 layer in both routing methods for transmissions to the EMS Hub, but it utilizes the same modules as the mote network layer described in Section 3.4.2.3. The key difference is the assignment of a static route through the MarathonSim `configurator` submodule. The `configurator` is assigned `flat_network.xml` within the INI that sets the subnets and the static route. All gateways are assigned a static route shown in Figure 16 for transmissions destined for the EMS Hub to be directed out onto the Ethernet interface.



```
<config>
  <interface hosts="*.EMSHub" names="eth0" address="10.10.10.10" netmask="255.255.x.x" metric="1"/>
  <interface hosts="*.host[*]" names="wlan0" address="10.0.x.x" netmask="255.255.x.x" metric="8000"/>
  <interface hosts="*.gateway[*]" names="wlan0" address="10.0.x.x" netmask="255.255.x.x" metric="8000"/>
  <interface hosts="*.gateway[*]" names="eth0" address="10.10.x.x" netmask="255.255.x.x" metric="1"/>
  <route hosts="*.gateway[*]" destination="10.10.10.10" netmask="255.255.255.255" gateway="*" interface="eth0" metric="0"/>

</config>
```

Figure 16: Static Route Assigned in `flat_network.xml`

For AODV operations, the gateways act as intermediary nodes with permanent connections to the EMS Hub. Any mote within range of a gateway should be able to contact the gateway with a RREQ message and receive a RREP with the gateway as the next hop due to the static route. Additionally, the static route cannot be invalidated, so if a mote fails transmissions due to collisions and tries to invalidate the route, the gateway ignores the attempt to invalidate the route.

Flooding operations similarly relies on the static route as a method of moving the transmission from the WLAN interface onto the Ethernet interface. Without the static route and presence of IPv4, the gateway simply rebroadcasts over the WLAN interface. It is expected that a proper border gateway implementation handles the relaying between interfaces.

3.4.4 EMS Hub

The EMS Hub represents the database from which observers access the data produced by the participants' sensors. It is implemented as a `StandardHost.ned` module and connected to the gateways through the `connectingSwitch` with identical channels. The Hub includes a `dataSink` submodule of type `IpvxTrafSink.ned` to collect data transmitted from the motes but has no other function. The EMS Hub is assigned the 10.10.10.10 static IP address through the `configurator` submodule for mote applications and static routes to target.

3.4.5 Network Configurator

The MarathonSim `configurator` uses a network configuration file to manually assign subnets to the wireless interfaces, the Ethernet interfaces, and a specific address to the EMS Hub. As mentioned in Section 3.4.3.3, the `configurator` also assigns a static route used to ensure translation from wireless communication to the wired interface connected to the EMS Hub. In addition to the static route, Figure 16 also shows the assignment of subnets with a corresponding metric used to determine cost of a route. Wireless interfaces are assigned an arbitrarily high metric to force the gateways to respond to RREQ messages with the low metric, static, wired interface route rather than any wireless routes to the EMS Hub.

3.4.6 Radio Medium

The `radioMedium` submodule manages the propagation and delivery of signals to receivers within range of detecting the transmission. The submodule is assigned the `Ieee802154NarrowbandScalarRadioMedium.ned` type to match the radios used by the devices in the network. This model accounts for background noise interference, path loss calculations, and filtering of transmissions to help with runtime complexity.

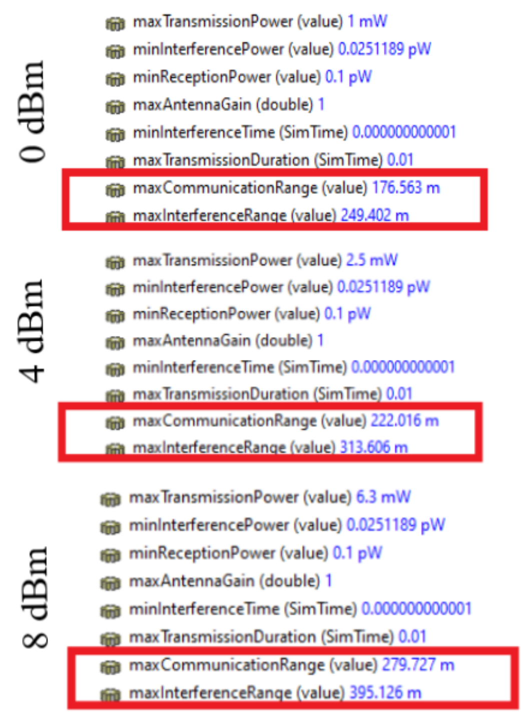
The `radioMedium` submodule uses `BreakpointPathLoss.ned` to model path loss. This module uses the same path loss formula recommended in Section 2.2.2.2, but it simplifies Equation (8) to just Free-Space Path Loss at the break point distance. The break point distance is calculated from (7) as 32 meters with a break point path loss $L_{bp} = 70.33$ dB. Path loss coefficients of 2 and 4 are used from (6).

Motes available to possibly receive a transmission are calculated as those within a maximum interference range. The maximum communication range is shorter and essentially a link budget using the transmission power, receiver sensitivity, and path loss over distance. For the lowest transmission power of the nRF52840, a 0 dBm output power results in a link budget equation of

$$-100dBm = 0dBm - L_{PL} \tag{10}$$

where L_{PL} is the path loss meaning up to -100 dBm can be tolerated. Interference range accounts for transmissions being outside of communication range but still affecting the reception of other transmissions. The INET framework commonly used sensitivity minus 6 dBm as a guideline because there is no rule.

With clear line of sight, the nRF52840 has been reported to successfully transmit 196, 231, and 280 meters at 0, 4, and 8 dBm transmission power respectively using the Zigbee protocol [46]. Simulation trials using `marathonSensorBase.ned` and the `Ieee802154ScalarRadioMedium.ned` resulted in similar, but shorter, ranges shown in Figure 17. The simulated ranges of 176, 222, and 280 meters indicate the INET models or the assigned path loss coefficient parameters underestimate transmission range performance.



```
maxTransmissionPower (value) 1 mW
minInterferencePower (value) 0.0251189 pW
minReceptionPower (value) 0.1 pW
maxAntennaGain (double) 1
minInterferenceTime (SimTime) 0.000000000001
maxTransmissionDuration (SimTime) 0.01
maxCommunicationRange (value) 176.563 m
maxInterferenceRange (value) 249.402 m

maxTransmissionPower (value) 2.5 mW
minInterferencePower (value) 0.0251189 pW
minReceptionPower (value) 0.1 pW
maxAntennaGain (double) 1
minInterferenceTime (SimTime) 0.000000000001
maxTransmissionDuration (SimTime) 0.01
maxCommunicationRange (value) 222.016 m
maxInterferenceRange (value) 313.606 m

maxTransmissionPower (value) 6.3 mW
minInterferencePower (value) 0.0251189 pW
minReceptionPower (value) 0.1 pW
maxAntennaGain (double) 1
minInterferenceTime (SimTime) 0.000000000001
maxTransmissionDuration (SimTime) 0.01
maxCommunicationRange (value) 279.727 m
maxInterferenceRange (value) 395.126 m
```

Figure 17: Max Transmission Ranges for nRF52840 Resulting from 0, 4, and 8 dBm Output Power

3.5 MarathonSim Design Summary

This chapter introduces the basic structure of an OMNeT++ simulation and INET capabilities. Additionally, Section 3.4 presents the components and parameters of MarathonSim. Limited demonstrations illustrate the verification of the functionality of the utilized models.

IV. Methodology

4.1 Overview and Objectives

The simulations conducted in this research test the performance of a mobile WSN in a marathon scenario. The economic burden of testing a prototype WSN of hundreds of motes is too large without performance data to support development. While MarathonNet provided connectivity estimates, it did not simulate full devices. MarathonSim models full-stack motes and gateways to achieve the following objectives:

- Measure the capability of the network to deliver sensor data.
- Analyze the interaction between transmission ranges and quantity of devices.
- Examine energy requirements of mote devices.
- Evaluate performance of AODV and Managed Flooding routing protocols.

The outcomes of these objectives guide development of a prototype WSN by estimating performance and demonstrating possible limitations of the technologies.

4.2 System Under Test

Figure 18 illustrates the System Under Test (SUT) and Component Under Test (CUT). The output metrics, or response variables, include packets delivered to the EMS Hub, end-to-end delay of those packets, and the power consumed by the motes through the marathon. The factors are the inputs intended to affect a change in the response variables, such as the transmission power, routing protocol, and number of motes in the marathon. Constant parameters are the variables that could affect performance but kept constant throughout the trials.

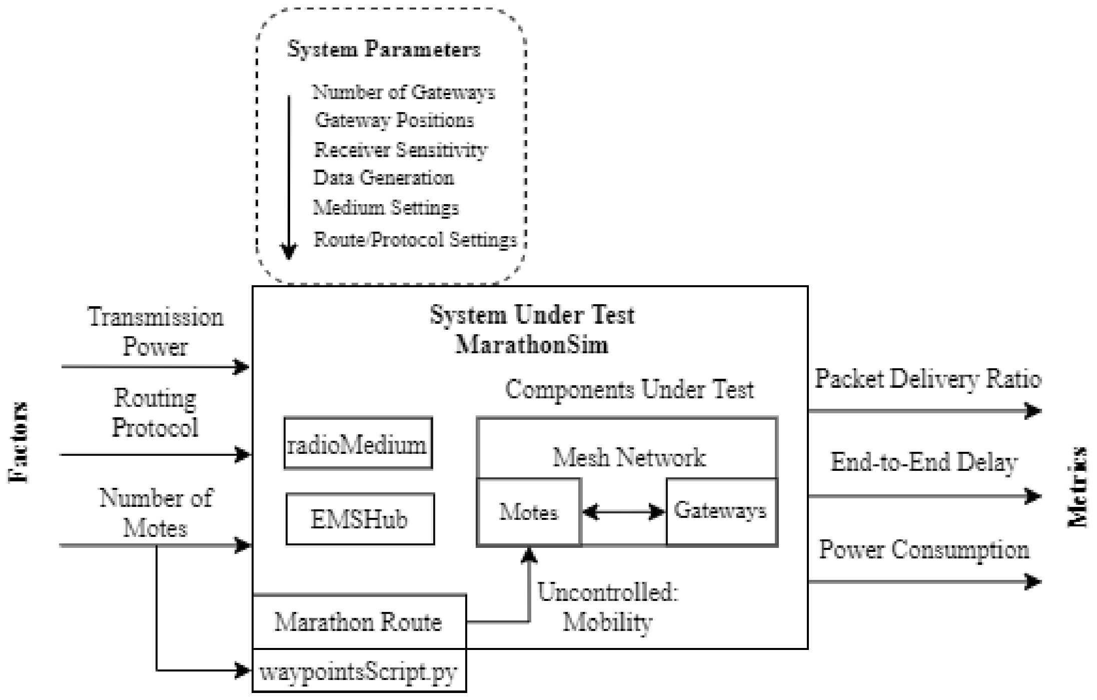


Figure 18: System Under Test and Component Under Test Diagram

4.3 Assumptions and Limitations

The following assumptions are used in executing this experiment:

- Marathon finish times approach a normal distribution, allowing approximation of runner velocities.
- Motes are associated in a single 802.15.4 PAN before the start of the simulation.
- Internet infrastructure is available to connect gateways and introduces negligible performance impact compared to wireless communications.
- Processing time and energy consumed by sensor and microprocessor operation is negligible compared to radio operations.

The following limitations are placed to ensure manageable scope:

- All wireless communication takes place in a single channel in the 2.4 GHz range.

- Only 802.15.4 non-beacon mode is simulated.

4.4 Factors

The trials of this experiment execute a full factorial design by running a trial of each combination of the factor levels listed in Table 4. Transmission power, number of motes, and routing protocol are varied between each trial to induce changes in network performance.

Table 4: Factor Levels

Factor	**Level**	**Description**
Transmission Power	1, 2.5, 6.3 mW	Transceiver output power based on the nRF52840
Number of Runners	25, 50, 75, 100, 125	Quantity of runners equipped with motes running the marathon
Routing Protocol	AODV, Flooding	The method of routing packets throughout the network

The transmission power is the power output from the mote and gateway transceivers during data transmissions. It is a critical design metric for motes in a WSN that impacts network structure and general performance, as well as mote lifetime. Higher power allows greater coverage with fewer devices, but it increases consumption of limited power supplies. The selected levels represent the operating characteristics of the nRF52840 on which the models are based. The change of units from Table 1 is required by the implementation in MarathonSim. The equation to change from dBm to mW is

$$P_{mW} = 1mW * 10^{\frac{P_{dBm}}{10}} \tag{11}$$

where P_{dBm} represents the dBm being changed to mW and P_{mW} is the result.

The number of motes in each trial represent both runners and sensors participating in the mesh network. Similar to transmission power, greater quantities of motes increases coverage potential but introduces additional material cost, bandwidth usage, and network density. Previous work by MarathonNet tested a fixed quantity of 500 motes, but that would require a significant commitment of resources to deploy. Instead, this experiment explores lower numbers of motes from 25 to 125 in order to establish network performance at lower device densities.

The routing protocols of AODV and managed flooding are an approximation of routing used by commercial technologies. Zigbee and Bluetooth mesh are both LR-WPANs with similar physical layer performance such as data rates and communication ranges but utilize different routing protocols.

4.5 System Parameters

Throughout the experiments, some factors are held constant to prevent them from influencing results between trials. These system parameters include both default parameters of INET models used in MarathonSim and configurations set through the INI file. Parameters unique to MarathonSim are shown in Table 5. Default parameters of the INET models such as 802.15.4 protocol parameters are not included.

Table 5: Selected System Parameters

System Parameter	**Value**	**Description**
Number of Gateways	19	19 gateways distributed according to Figure 10
Gateway Positions	See Figure 10	Gateways placed alongside marathon infrastructure
Receiver Sensitivity	-100 dBm	nRF52840 receiver sensitivity
backgroundNoise.power	-110 dBm	Estimated background noise [47]
pathLoss.breakpointDistance	32 m	Distance at which path loss changes based on (7)
pathLoss.l02	70.33 dB	Free Space Path Loss at breakpoint distance
pathLoss.alpha2	4	Recommended path loss coefficient from (6)
receiverReceivingPowerConsumption	13.8 mW	nRF52840 receiver power consumption
wlan.macMinBE	9	Minimum backoff exponent
wlan.macMaxBE	12	Maximum backoff exponent
dataGen Start Time	uniform(1,60) s	Time the application begins producing data
dataGen Packet Length	64 B	Size of the sensor application packet payload
dataGen Interval	60 s	Time between packet generations
AODV Active Route Timeout	3 s	Lifespan of an AODV route before expiring if not used

4.6 Performance Metrics

The metrics are the responses measured from the trials to support the objectives of this experiment and are listed in Table 6.

Table 6: MarathonSim Performance Metrics

Metric	**Unit**	**Description**
Packet Delivery Ratio	%	The ratio of packets sent from all motes to the packets received at the EMS Hub.
End-to-End Delay	seconds	The delay experienced from packet transmission to packet reception at the EMS Hub.
Mean Power Consumption per Node	J	Average power consumed by motes during the marathon.

Packet Delivery Ratio The Packet Delivery Ratio (PDR) is a measure of the ratio of total packets delivered to the EMS Hub and total packets sent by the motes. The ratio is both an indicator of connectivity and congestion. Network segments with high congestion experience collisions and dropped transmissions. Without an application or transport layer mechanism, the PDR represents the best effort provided by the lower layers. Depending on the deployed application, additional QoS services may be needed.

For N runners in a simulation, the PDR is calculated simply as

$$PDR = \frac{Q_{received}}{360 * N} \tag{12}$$

where $Q_{received}$ is the number of packets received at the EMS Hub. The number of runners is multiplied by 360 because each mote produces a data packet every 60 seconds, for the 6 hour marathon.

Mean End-to-End Delay The mean end-to-end delay measures the average duration of time from creation to delivery to the EMS Hub. This QoS metric provides insight into network congestion as motes enter exponential backoff delays when sensing the channel is active. Long delays are indicative of frequent backoffs or many intermediate hops. Mean delay is calculated as

$$meanDelay_{E,E} = \frac{1}{Q_{received}} \sum_{i=1}^{Q_{received}} (T_{received}(i) - T_{generated}(i)) \tag{13}$$

where $T_{generated}(i)$ is the sim time the packet was generated and $T_{received}(i)$ is the time it was received at the EMS Hub.

Mean Power Consumption per Mote The average power consumption of a mote is a classic WSN performance metric measuring the energy efficiency of the battery-powered motes. Most WSN applications are expected to operate for days, weeks, or even years. As shown in Section 3.4.2.1, a CR2032 lithium coin cell battery is more than enough to supply a mote for the duration of a 6 hour marathon. However, the usefulness of this technology to other applications depends on this metric. Mean power consumed per node is calculated as

$$\bar{P}_{mote} = \frac{1}{N}\sum_{i=1}^{N}(P_{timeavg}(i) * 21600s) \tag{14}$$

where for N motes $\bar{P}_{mote}$ is the average power consumed per mote and $P_{timeavg}(i)$ is the time average power consumed by mote i as it transitions from different states over time. It is multiplied by the duration of the marathon to estimate consumption throughout the simulation.

4.7 Uncontrolled Variable

One factor remains uncontrolled in this experiment, the mote mobility. The mobility is described in Section 3.4.2.1 and uses a normally distributed random variable to generate runner velocities that approximate human movement. However, as the number of motes increases between trials, a new set of mobility traces are generated for the new set of motes. The movement of motes through the marathon affects the connectivity of the network and there exists a chance that one trace file has a more clustered group of runners than another set. A larger cluster potentially allows higher connectivity in between gateways. This variable could be eliminated by accessing a database with more granular distance and time data on marathon participants that can also remain constant between trials.

4.8 Experimental Design

This section describes the design of this experiment and the steps performed in its execution. The experiment is a full factorial design to explore the interactions between the selected factors and determine levels that appear to optimize performance. With the factors shown in Table 4, a total of $(3 * 5 * 2) = 30$ trials are conducted without replication. The lack of replication is due to the runtimes of MarathonSim trials with more than 100 runners exceeding 7 days each. The consequences of this is discussed further in Section 4.9.

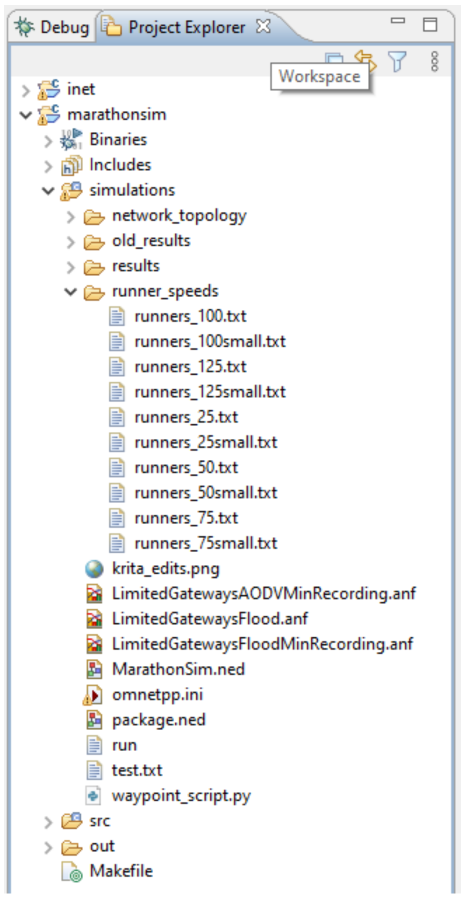
As a simulation for a proposed system, MarathonSim cannot be validated against an operational system or experimental data. The only similar system is the MarathonNet simulation, but the tool is no longer available and uses different parameters and marathon route. Instead, face validity is established from using components modeling existing protocols and devices, inspecting the analytical models for accuracy, and examining animation behaviors mentioned throughout Section 3.4. Face validity is a weak validation, so simulated results should not be taken as predictions of a real marathon WSN. However, the changes in metrics due to factor levels are expected to be representative of the real marathon.

4.8.1 Mobility Trace Generations

The first step in executing the experiment is generating the mobility traces for each number of motes factor level. Using `waypointsScript.py`, trace files are generated by running the command

```
python waypoint_script.py --num_runners=X
```

for each number of motes. The script generates a `runners_X.txt` file that must be copied into the MarathonSim project directory in the OMNeT++ Integrated Development Environment (IDE). All mobility trace files are placed into the `marathonsim /simulations/runner_speeds` directory as shown in Figure 19.



```
> inet
v marathonsim
  > Binaries
  > Includes
  v simulations
    > network_topology
    > old_results
    > results
    v runner_speeds
        runners_100.txt
        runners_100small.txt
        runners_125.txt
        runners_125small.txt
        runners_25.txt
        runners_25small.txt
        runners_50.txt
        runners_50small.txt
        runners_75.txt
        runners_75small.txt
      krita_edits.png
      LimitedGatewaysAODVMinRecording.anf
      LimitedGatewaysFlood.anf
      LimitedGatewaysFloodMinRecording.anf
      MarathonSim.ned
      omnetpp.ini
      package.ned
      run
      test.txt
      waypoint_script.py
  > src
  > out
    Makefile
```

Figure 19: Placement of Mobility Trace Files within Project Directory

4.8.2 INI Configuration

The INI file defines the instances of `MarathonSim.ned` with default values and configurations shown in Appendix B. No changes should be necessary to it, but for executing factors unique to this experiment, the following lines are used

```
*.numHosts = ${n=25, 50, 75, 100, 125}
**.transmitter.power = ${power=1,2.5,6.3}mW
```

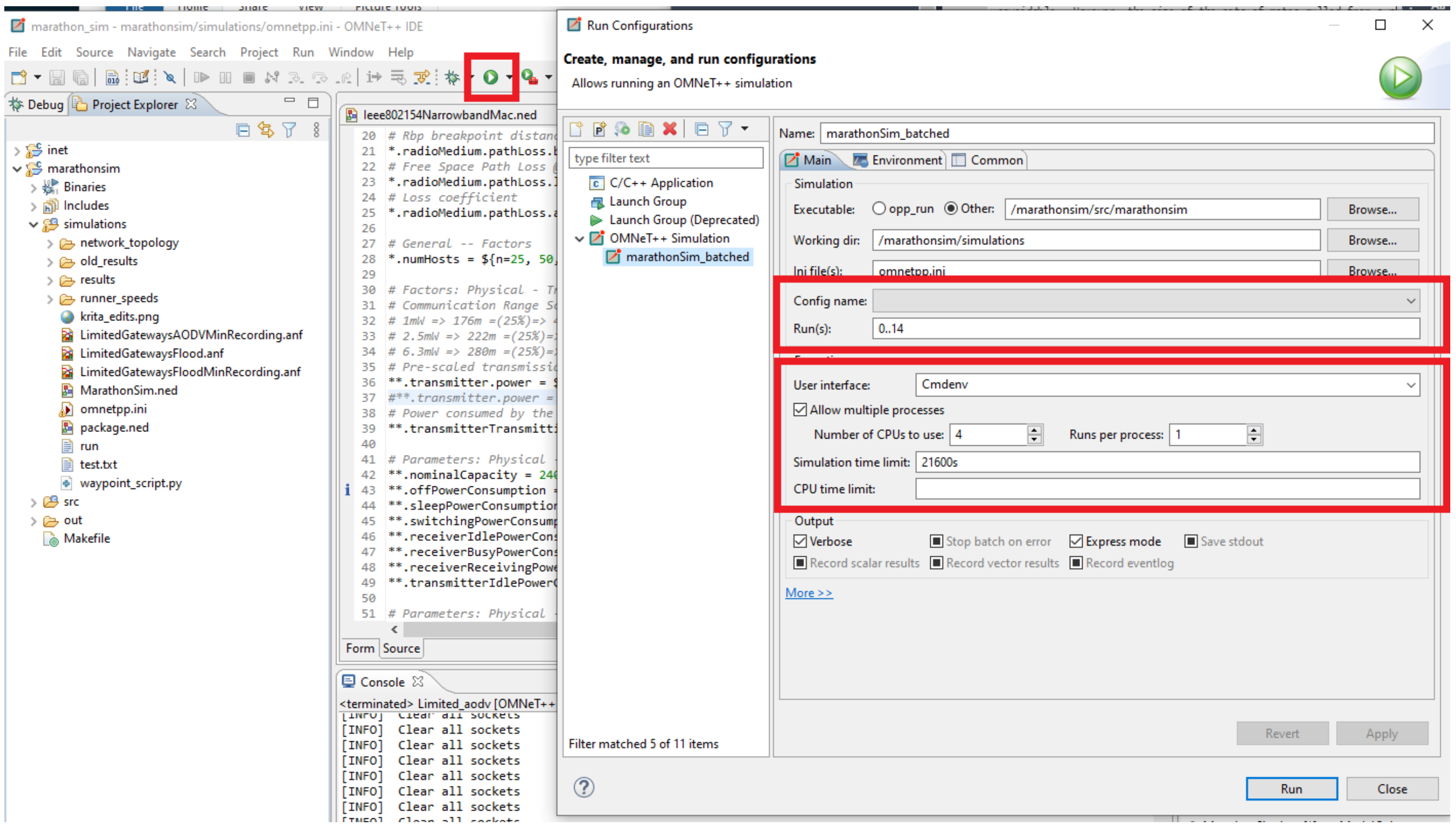
where the first line sets an iterative variable for the number of runners. The second line sets an iterative variable for transmitter power.

4.8.3 Batch Execution

The execution of the trials occurs in batches within the OMNeT++ IDE. Figure 20 shows the interface for setting up a batched run configuration. Clicking the arrow next to the green play button brings up the dialog box to allow a customized batch. The first section under "Simulation" contains the "Config name" and "Run(s)" options. The "Config Name" drop-down box allows selection of the configurations enumerated in the INI file. The first batch using the AODV protocol is set by selecting "LimitedGatewaysAODVMinRecording" and setting "Run(s)" to 0..14. This executes every number of runners and transmission power combination using the AODV protocol. The second batch is executed similarly by selecting "LimitedGatewaysFloodMinRecording" and setting "Run(s)" to 0..14.

Additionally, under the "Execution" parameters, the "User interface" is set to "Cmdenv" for shell execution. Due to limited hardware, only 4 CPU cores are used to execute parallel simulations.

Clicking "Run" begins the batch execution and takes up to 10 days. Progress can be monitored through the command terminals. Each trial's results are written to a scalar and a vector file for single values and timestamped vectors of values respectively.



```
# Rbp breakpoint distan
*.radioMedium.pathLoss.b
# Free Space Path Loss @
*.radioMedium.pathLoss.l
# Loss coefficient
*.radioMedium.pathLoss.a

# General -- Factors
*.numHosts = ${n=25, 50,

# Factors: Physical - Tr
# Communication Range Sc
# 1mW => 176m =(25%)=> 4
# 2.5mW => 222m =(25%)=
# 6.3mW => 280m =(25%)=
# Pre-scaled transmissio
**.transmitter.power = $
#**.transmitter.power =
# Power consumed by the
**.transmitterTransmitti

# Parameters: Physical -
**.nominalCapacity = 240
**.offPowerConsumption =
**.sleepPowerConsumption
**.switchingPowerConsump
**.receiverIdlePowerCons
**.receiverBusyPowerCons
**.receiverReceivingPowe
**.transmitterIdlePower(

# Parameters: Physical -
```

```
<terminated> Limited_aodv [OMNeT++
[INFO]  Clear all sockets
[INFO]  Clear all sockets
[INFO]  Clear all sockets
[INFO]  Clear all sockets
[INFO]  Clear all sockets
[INFO]  Clear all sockets
[INFO]  Clear all sockets
```

Run Configurations

Create, manage, and run configurations

Allows running an OMNeT++ simulation

type filter text

- C/C++ Application
- Launch Group
- Launch Group (Deprecated)
- OMNeT++ Simulation
  - marathonSim_batched

Filter matched 5 of 11 items

Name: marathonSim_batched

Main | Environment | Common

Simulation

Executable: ( ) opp_run (●) Other: /marathonsim/src/marathonsim [Browse...]

Working dir: /marathonsim/simulations [Browse...]

Ini file(s): omnetpp.ini [Browse...]

Config name:

Run(s): 0..14

User interface: Cmdenv

[✓] Allow multiple processes

Number of CPUs to use: 4 Runs per process: 1

Simulation time limit: 21600s

CPU time limit:

Output

[✓] Verbose [■] Stop batch on error [✓] Express mode [■] Save stdout

[■] Record scalar results [■] Record vector results [■] Record eventlog

More >>

Figure 20: Batch Run Configuration Tool and Execution

4.9 Statistical Analysis

In support of the objectives listed in Section 4.1, the metrics in Table 6 are recorded by OMNeT++ for each trial and analyzed using a series of tests in MATLAB. The goals of evaluating network performance, interaction between transmission power and number of runners, and protocol differences are evaluated through statistical comparisons. However, assessment of energy consumption is limited by the assumptions of the simulation and small sample size.

As mentioned in Section 4.8, the experiment trials are conducted without replication due to runtimes exceeding 7 days each for trials with more than 100 runners. MarathonSim includes computationally expensive modeling of signal propagation resulting in longer runtimes as the number of devices increases. As a consequence, the lack of replicates reduces the statistical power of tests conducted. A low power means there is a higher probability of a Type II error which is a failure to reject a false null hypothesis. Incidentally, smaller factor effects are more difficult to detect than if more samples were available as the null hypothesis is more likely to not be rejected. Additionally, with a single observation per factor combination, the predicted performance is vulnerable to being misrepresented by variance and outliers.

Keeping in mind the lack of replicates, a three-way Analysis of Variance (ANOVA) test is conducted for each metric to determine significant factors among transmission power, number of runners, and routing protocol. The higher-order interaction of all three factors is pooled to estimate error, otherwise the degrees of freedom for error would be zero and invalidate the test. The main factors are maintained as well as the second-order interactions.

Results of the ANOVA must be verified by ensuring its standard assumptions are not violated: independence of observations, normality of residuals or errors, and equal variance between factor levels. The independence of observations is evident from simulation trials being isolated. Normal distribution of residuals is tested using the Anderson-Darling test where a non-significant result against a threshold of $\alpha = 0.05$ indicates normality. The equality or homogeneity of variance is verified through the Levene's test where a non-significant result against a threshold of $\alpha = 0.05$ indicates homogeneity.

Following verification of a significant ANOVA result, a post hoc multiple comparison Tukey test is conducted to determine which factor level proved statistically different from the others. The post hoc tests require the same assumptions to be met as ANOVA. All tests are conducted against the null hypothesis of no difference in means with an alternative hypothesis of a difference. A significant difference requires a p-value to be reported between groups such that it is less than a threshold of $\alpha = 0.05$.

While the trials provide weak support for estimating specific levels of performance, they capture statistically significant factors and levels. The predicted performances can guide development of future devices, but they should not inform specific design choices until the simulation is validated through small prototypes.

4.10 Methodology Summary

Chapter 4 provides a breakdown of the methodology for the experiment in this research. A full factorial experiment evaluates WSN performance metrics under varying levels of transmission power, quantity of runners, and routing protocol. Section 4.8 shows necessary commands to set up and execute the experiment in the OMNeT++ IDE. Finally, Section 4.9 specifies how the results are analyzed for significance.

V. Results and Analysis

5.1 Overview

This chapter describes the results of the MarathonSim experiment detailed in Chapter 4. The results are presented in Section 5.2 along with analysis before being summarized in Section 5.3.

5.2 MarathonSim Performance Metrics Analysis

This section presents the results of the data collected on the performance metrics of Section 4.6. Each metric is presented with graphs and statistical analysis. As noted in Chapter 4, the lack of replicates reduce the usefulness of the experiment in predicting performance values. Specifically, the results are vulnerable to Type II errors of failing to reject a false null hypothesis due to low power. Therefore, the bulk of this analysis focuses on trends and effects rather than specific predicted performance data.

5.2.1 Packet Delivery Ratio

The PDR metric serves as a multipurpose measurement of overall network performance. As runners move through the course, the motes lose and regain access to the network depending on available peers and gateways to relay data. A high PDR is desirable and indicates a well-connected network with low congestion allowing continual updates at the EMS Hub. The measurements of PDRs across all trials is shown in Figure 21. Each line represents a combination of routing protocol and transmission power charted over increasing numbers of runners. The AODV trials are printed with dashed lines, while flood trials use a dotted lines.

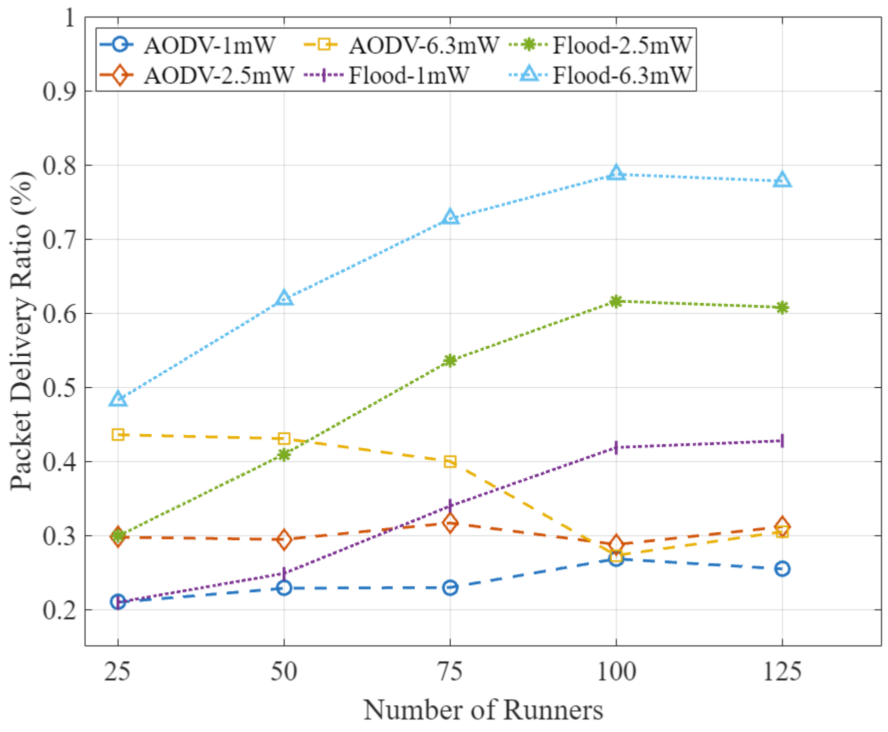


Figure 21: Packet Delivery Ratio by Protocol and Transmission Power over Number of Runners

Before analyzing the results, a minimum level of connectivity is estimated for a single mote running through the course. The minimum connectivity expected can be calculated as percentage of time a single mote is within range of a gateway. Using the simulated route length of 41,368 meters and simulated transmission ranges of 176, 222, and 280 meters, the percentage of time a mote is in range of a gateway is calculated by

$$\frac{(2R) * G_{course} + R * G_{start}}{41368m} = Con_{base}$$

where R is the transmission range for 18 course gateways, G_{course}, and one starting line gateway G_{start}. The baseline connectivity, Con_{base}, is the percentage of the course covered under a given transmission range R. At the starting line gateway, runners move away from the gateway rather than into it so only the radius is used while the remaining 18 gateways, G_{course}, use their full transmission diameters. For 0, 4, and 8 dBm output power, the baseline connectivity is 15.5%, 19.8%, and 25.0% respectively. As seen in Figure 21, each trial of 0, 4, and 8 dBm exceeds the baseline connectivity for the corresponding output power. Each increase in transmission power increases flood protocol PDR by between 10-20% which exceeds the increases in base connectivity between 15.5%, 19.8%, and 25.0%. The highest PDR of 78.7% occurred under 100 runners with 6.3 mW transmission power motes running the flood protocol, more than triple the baseline connectivity of 25.0% for 8 dBm output power.

Unexpectedly, nearly every AODV trial resulted in a PDR less than the corresponding flood trial with the exception of trials with 25 runners. In fact, AODV PDR remained constant, or decreased, as the number of runners increased while the flood trials exhibited consistent growth. With flood trials doubling PDRs between 25 and 125 runners, the lack of improvement in AODV is concerning for a MANET protocol. Upon further inspection, link breakages between motes and gateways failed to trigger a RERR message broadcast. A function within `Aodv.ned` when creating a RERR message calls to `route->getProtocolData()`, which returned static IP route data rather than an `AodvRouteData` unit. This caused RERRs to fail to propagate the network and invalidate routes when moving beyond the range of a gateway.

Additionally, for the AODV trials with 100 runners, the PDRs converged rather than improving and erased the benefits of longer transmission ranges seen in other trials. The results potentially indicate influence from the uncontrolled mote mobility mentioned in Section 4.7; however, the absence of a performance hit in the flood trials means the problem is unique to AODV.

The ANOVA results in Table 7 indicate significant effects from all factors. The p-value for each main factor is less than the significance threshold of $\alpha = 0.05$ which rejects the null hypothesis. The effects from routing protocol may not be as strong as indicated due to the flaws discovered in the AODV implementation, but the transmission power and number of runners effects are statistically significant. The second-order terms of routing protocol and the other factors are also undermined by the results of AODV. However, the second-order interaction of number of runners and transmission power are statistically significant with a p-value of 0.0253.

Table 7: Results of ANOVA on PDR Metric

Source	**Sum Sq.**	**DF**	**Mean Sq.**	**F-stat**	**P**
NumberRunners	0.3759	4	0.0940	16.5496	0.0006
TransmissionPower	1.7212	2	0.8606	151.5594	0.0000
RoutingProtocol	1.5355	1	1.5355	270.4110	0.0000
NumberRunners* TransmissionPower	0.2007	8	0.0251	4.4174	0.0253
NumberRunners* RoutingProtocol	0.5573	4	0.1393	24.5362	0.0002
TransmissionPower* RoutingProtocol	0.1314	2	0.0657	11.5733	0.0044
Error	0.0454	8	0.0057		
Total	4.5673	29			

Testing of the assumptions of ANOVA is shown in Table 8. With no p-value less than the significance threshold of $\alpha = 0.05$, the assumptions of normal distribution and equal variances are not rejected.

Table 8: ANOVA Assumption Tests for PDR

Test Type	**Metric**	**P-Value**
Anderson-Darling (Normality)	N/A	0.9670
Levene's (Variance)	Number of Runners	0.8944
Levene's (Variance)	Transmission Power	0.4114

The PDR, averaged by factor levels, are shown in Figure 22. The isolated averages exhibit an increased PDR as the level of runners and transmission power increases as expected. With such limited samples, averaging across groups assists to visualize the impact of the increasing factor. Unfortunately, the AODV factor misleads partially due its flawed performance.

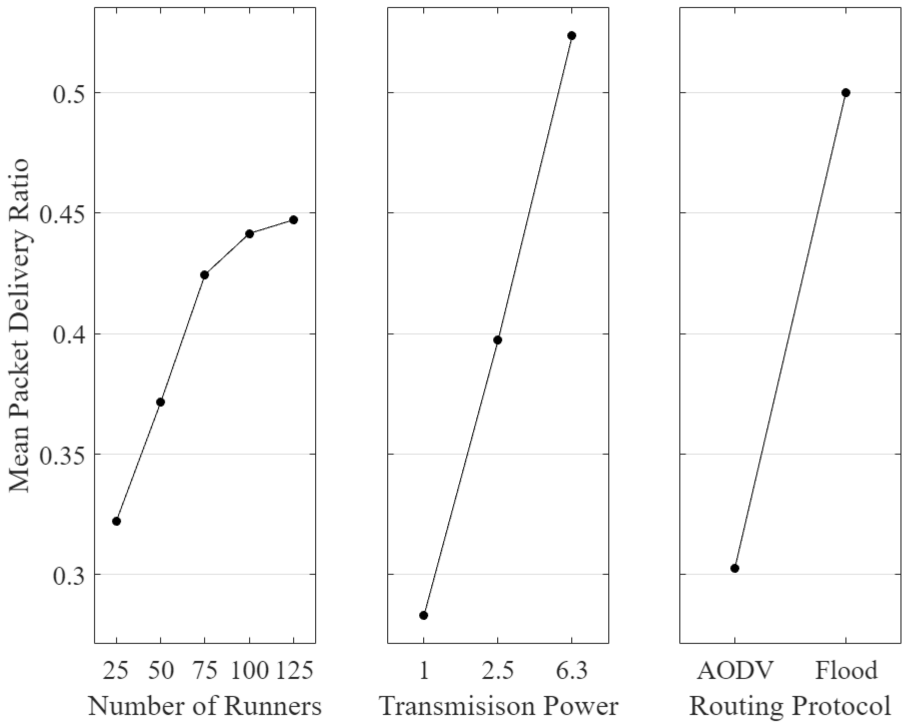


Figure 22: Packet Delivery Ratio by Main Effects

The post hoc Tukey test for the factor levels of runners is shown in Table 9. The p-values that are less than the threshold of $\alpha = 0.05$ represent statistically significant factor levels. Of note, 25 runners is significantly different than all but 50 runners, while only 50 and 125 are different. However, the lack of significant differences among higher numbers of runners implies diminishing returns of PDR. Figure 22 graphically demonstrates the diminishing returns from increased runners.

Table 9: Pairwise Runners Comparison for PDR

Paired Runners	**P-Val**
25-50	0.0865
25-75	0.0024
25-100	0.0015
25-125	0.0009
50-75	0.1126
50-100	0.0646
50-125	0.0320
75-100	0.9931
75-125	0.8802
100-125	0.9831

The post hoc Tukey test for the factor levels of transmission power is shown in Table 10. All the p-values are less than the threshold of $\alpha = 0.05$ demonstrating each level is statistically significant. Unlike the number of runners, increasing transmission power demonstrates no visible diminishing returns. While this experiment focused on the nRF52840, other hardware may provide even higher transmission power, and therefore, range that would enable closer to 100% PDR.

Table 10: Pairwise Transmission Power Comparison for PDR

Paired Transmission Powers	**P-Val**
0 dBm - 4 dBm	0.0001
0 dBm - 8 dBm	0.0000
4 dBm - 8 dBm	0.0001

5.2.2 End-to-End Delay

The mean end-to-end delay metric intends to capture congestion and latency effects especially due to density. At the start of the race, runners are densely packed causing contention congestion, but as the race continues, the space between runners expands. While contention decreases with distance, more intermediate transmissions are needed to relay data which increases overall traffic. The results of the measured delays are shown in Figure 23. Unlike PDR, statistically significant factors are not immediately visually evident.

As a consequence of the high initial minimum backoff exponent for the CSMA/CA protocol mentioned in Section 3.4.2.2, average delay for a single-hop transmission is expected to be dominated by backoff delays. For example, a 127 byte packet would require

$$\begin{aligned} T_{delay} &= T_{trans} + T_{BE} + T_{prop} + T_{proc} \\ T_{delay} &= \frac{127 bytes}{250 kbps} + 0.08 sec \\ T_{delay} &= 0.0040 sec + 0.08 sec \\ T_{delay} &= 0.0840 sec \end{aligned}$$

where $T_{BE} = 0.08$ seconds is the average backoff delay derived in Section 3.4.2.2 and assuming propagation, T_{prop}, and processing delays, T_{proc}, are negligible. However, the results suggest multiple seconds of delay across most treatments and even more under AODV. The flooding protocols appear to report lower average delays despite the likelihood of greater medium access contention.

Despite the error in the AODV protocol execution, part of the increased delay can be explained by functions of the protocol. Whenever a mote originates a RREQ message, the duration it waits for a RREP is dependent on the size of the network. If a RREP is not received, the mote will attempt a RREQ again twice while delaying the data packet in a queue. As seen in Figure 23, the larger networks with more runners increase the AODV trials delays more than flood trials.

It should be noted, two measurements are missing from Figure 23 for trials with 100 runners. Two flood trials with 100 runners, 2.5 and 6.3 mW, reported average delays of 18 and 24 seconds which were deemed outliers and removed. Unfortunately, the simulations with more than 100 motes take 7-9 days with inconsistent completions and so could not be replaced.

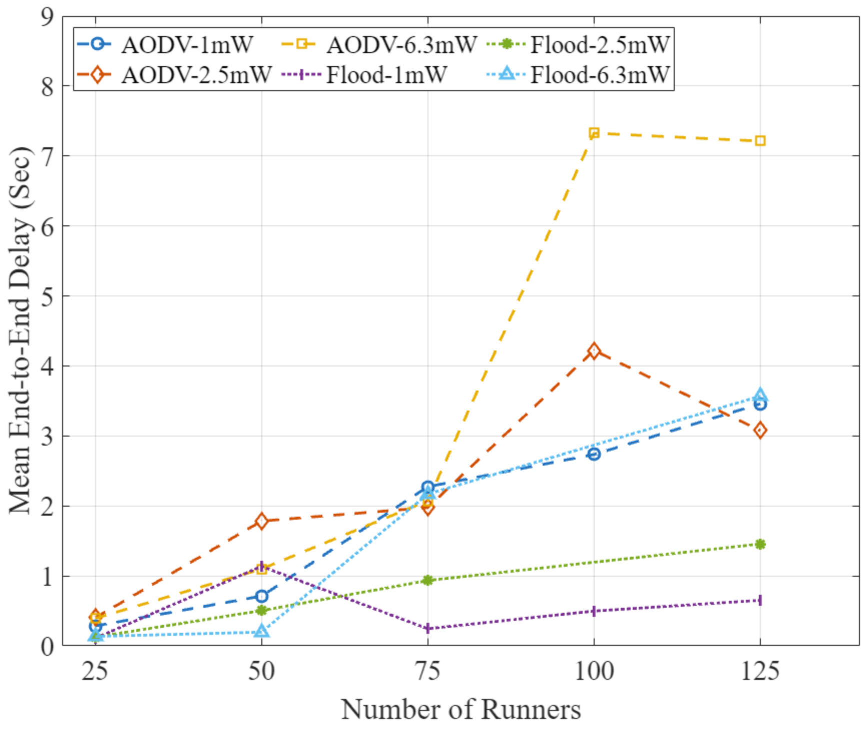


Figure 23: Mean End-to-End Delay by Protocol and Transmission power over Number of Runners

The ANOVA results shown in Table 11 indicate two statistically significant effects, the number of runners and the routing protocol with p-values less than the threshold with 0.0031 and 0.0069 respectively. However, as discussed previously, the routing protocol factor is inconclusive due to the AODV flaws. With more runners participating in the network, more collisions and contention backoff delays occur as more motes try to send their sensor data as well as more traffic is being relayed from peers, therefore increasing the time the channel is busy. While transmission power did trend upwards, it closely failed to achieve statistical significant effects. The failure of significance could be a Type II error where the null hypothesis is not rejected despite the means between groups being different.

Table 11: Results of ANOVA on End-to-End Delay Metric

Source	**Sum Sq.**	**DF**	**Mean Sq.**	**F-stat**	**P**
NumberRunners	5.3980	4	1.3495	14.38	0.0031
TransmissionPower	0.9175	2	0.4588	4.89	0.0570
RoutingProtocol	1.5192	1	1.5192	16.19	0.0069
NumberRunners* TransmissionPower	1.1975	8	0.1497	1.60	0.2931
NumberRunners* RoutingProtocol	0.3403	4	0.0851	0.91	0.5157
TransmissionPower* RoutingProtocol	0.0111	2	0.0056	0.06	0.9431
Error	0.5630	6	0.09383		
Total	12.0903	27			

Testing of the assumptions of ANOVA is shown in Table 12. Neither normality nor equal variance for significant effects are violated due to p-values greater than $\alpha = 0.05$. However, the assumption of equal variance between runner levels is risked due to the removal of outliers.

Table 12: ANOVA Assumption Tests for Delays

Test Type	**Metric**	**P-Value**
Anderson-Darling (Normality)	N/A	0.2685
Levene's (Variance)	Number of Runners	0.8969

The averaged delays by factor levels are shown in Figure 24. The peak in delay at 100 runners is due to the two outliers being removed and overemphasizing the effect of the AODV results.

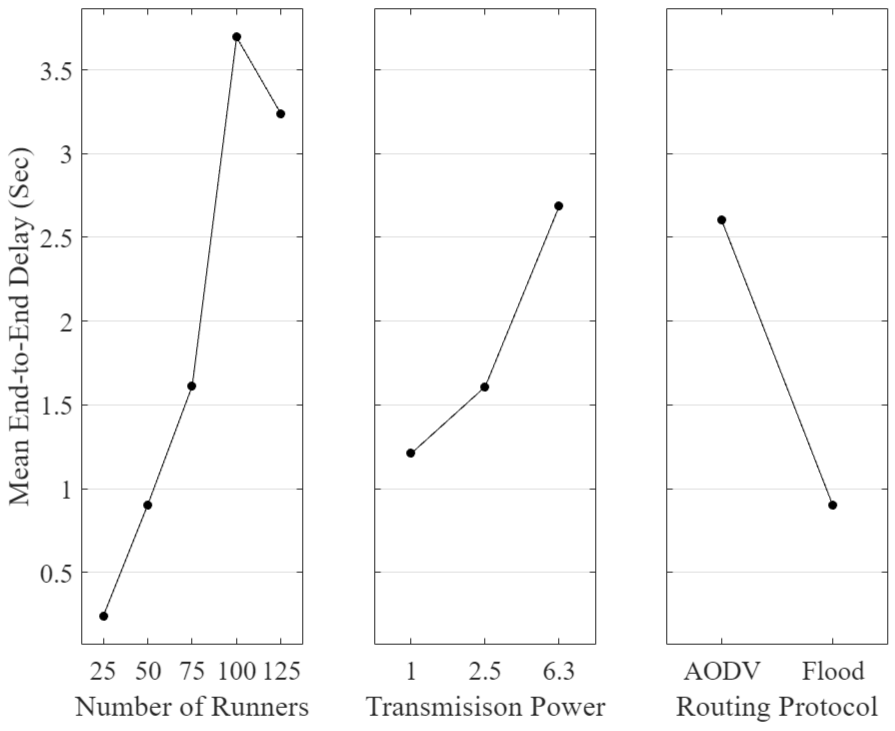


Figure 24: Mean End-to-End Delay by Main Effects
over Number of Runners

The post hoc Tukey test for factor levels of runners is shown in Table 13. Similar to the PDR results, 25 runners is significantly different from all except 50 runners and 100. However, the 100 runners not being significant could be due to the removal of outliers.

Table 13: Pairwise Runners Comparison for Delay

Paired Runners	**P-Val**
25-50	0.2065
25-75	0.0402
25-100	0.0983
25-125	0.0082
50-75	1.0000
50-100	1.0000
50-125	0.2202
75-100	1.0000
75-125	1.0000
100-125	1.0000

5.2.3 Average Power Consumption

The average power consumption per node measures the expected lifespan of motes used in the marathon. Energy exhaustion is not a problem in this scenario, but it may in longer duration operations or after repeated uses. The average energy consumed per mote is shown in Figure 25. The graph shows consistently increasing power consumption as number of runners and transmission power increases as expected.

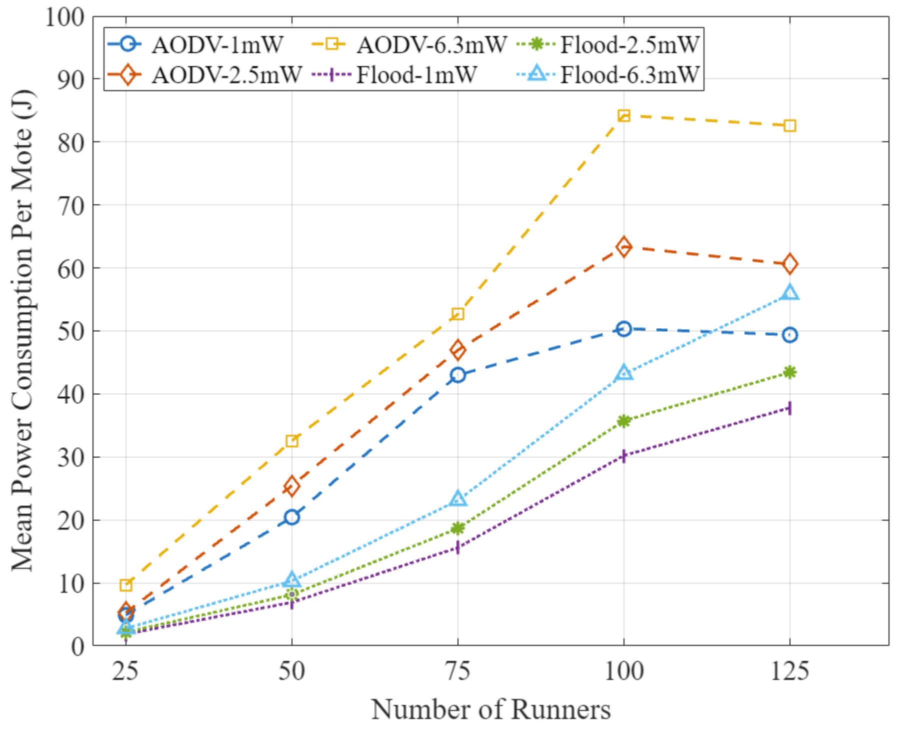


Figure 25: Mean Power Consumption per Node by Protocol and Transmission Power over Number of Runners

Transmission power appears to have less of an effect on power consumption than increased numbers of runners. This is consistent with Sections 2.2.2.1 and 2.2.2.2 which highlighted the energy cost of receiving transmissions and how multi-hop transmission costs can exceed single long-range transmissions. With more motes participating in the network, more devices are transmitting their own data and inflicting reception costs on all neighbors. The reason AODV appears to consume more energy is due to allowing multiple RREQ and uni-cast transmission attempts; motes using flooding only transmit a packet once each.

The ANOVA results for power consumption are shown in Table 14. Given the near linear increase in power consumption over number of runners and discrete separation between transmission powers, it is unsurprising that the factors are statistically significant. The p-values are all near zero, which reject the null hypothesis of no difference between factor levels.

Table 14: Results of ANOVA on Mean Power Consumption Per Node Metric

Source	**Sum Sq.**	**DF**	**Mean Sq.**	**F-stat**	**P**
NumberRunners	11233.8	4	2808.45	354.62	0.0000
TransmissionPower	954.4	2	477.21	60.26	0.0000
RoutingProtocol	2917.1	1	2917.06	368.33	0.0000
NumberRunners* TransmissionPower	428.2	8	53.53	6.76	0.0070
NumberRunners* RoutingProtocol	624.3	4	156.08	19.71	0.0003
TransmissionPower* RoutingProtocol	134	2	66.99	8.46	0.0106
Error	63.4	8	7.92		
Total	16355.2	29			

Testing of the assumptions of ANOVA is shown in Table 15. Since the p-values are not less than the threshold of $\alpha = 0.05$, it appears the assumptions remain intact.

Table 15: ANOVA Assumption Tests for Average Power Consumption

Test Type	**Metric**	**P-Value**
Anderson-Darling (Normality)	N/A	0.5040
Levene's (Variance)	Number of Runners	0.1330
Levene's (Variance)	Transmission Power	0.80016

The average power consumption by factor levels is shown in Figure 26. As expected, with more devices participating in the network, power consumption increases due to the higher amount of traffic regardless of protocol and transmission power.

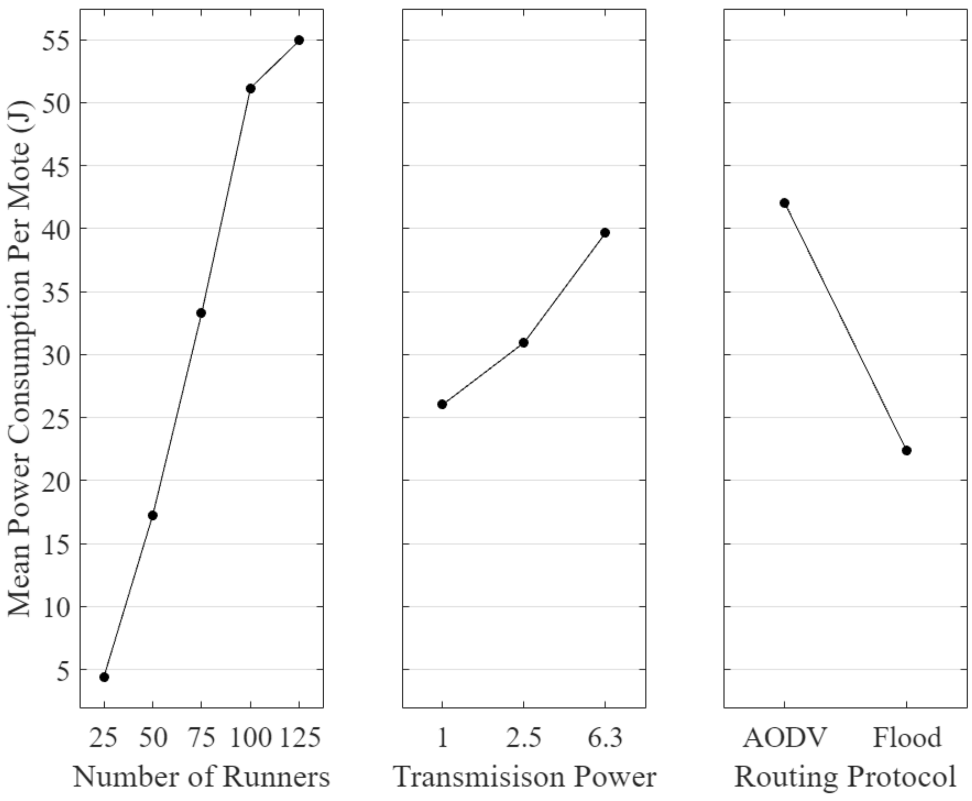


Figure 26: Mean Power Consumption per Node by Main Effect

The post hoc Tukey test for factor levels of runners is shown in Table 16. Each factor level is significant except for 100 runners to 125 runners. This reinforces a trend seen previously where differences in performance between 100 and 125 runners is less pronounced than other factor levels.

Table 16: Pairwise Runners Comparison for Average Power Consumption

Paired Runners	**P-Val**
25-50	0.0003
25-75	0.0000
25-100	0.0000
25-125	0.0000
50-75	0.0001
50-100	0.0000
50-125	0.0000
75-100	0.0000
75-125	0.0000
100-125	0.2325

The post hoc Tukey test for factor levels of transmission power is shown in Table 17. Each level of transmission power is statistically significant from the other which is expected for a metric measuring power consumption. Any advantages of increased range come at the expense of greater power or more expensive hardware.

Table 17: Pairwise Transmission Power Comparison for Average Power Consumption

Paired Transmission Powers	**P-Val**
0 dBm - 4 dBm	0.0110
0 dBm - 8 dBm	0.0000
4 dBm - 8 dBm	0.0003

5.3 Overall Analysis

Unfortunately, the critical flaw discovered in the `Aodv.ned` module potentially invalidates half of the trials conducted. As runners move towards a gateway, data is passed forward through peers to the gateway. However, after a runner passes the gateway and moves out of transmission range, the link breaks, but the mote fails to broadcast the RERR message. Without the RERR message, motes continue to send data through a broken route even though another path to the gateway may exist.

This error affects the results of each metric and undermines the reported statistical significance of routing protocols.

The AODV trials remained in the analysis as the levels of transmission power and number of runners still affected a change in the metrics. While the AODV trials fail to report representative performance of the protocol, factor effects from increased transmission power and number of runners are salvageable in an already small sample experiment.

Despite the flaws of the AODV trials, the results of this experiment are still promising. The flooding protocol proved to be successful at delivering more than 50% of packets in 7 out of 15 trials, and more than 75% in trials with 100 and 125 runners using 6.3 mW transmission power. The results show despite the inefficiencies from duplicated broadcasts; the simulated sensor traffic generated by the motes is light enough that the network is not overburdened.

Additionally, the results show that the performance of the prospective nRF52840 in transmission range and energy consumption is capable of supporting a scenario like the marathon. The 75% PDR attained under flood routing trials averaged between 2.5 and 3.5 seconds of delay. This suggests collisions are infrequent or negligible for delivery, but congestion due to medium access contention may be high from motes entering multiple backoffs when attempting transmissions.

5.4 Results Summary

This chapter presents and analyzes the data collected during the execution of the simulations. Each performance metric is shown and analyzed before an overall assessment of the experiment is presented.

VI. Conclusion

6.1 Overview

This chapter summarizes this work and the results discovered during development of MarathonSim and execution of the experiment. Section 6.2 highlights the conclusions from the experiment results. Section 6.3 restates the contributions made by this work. Section 6.4 discusses the limitations of this work. Finally, Section 6.5 recommends future work to extend upon this effort.

This research succeeded in demonstrating several cases in which the mesh network of motes and gateways can maintain a majority of connectivity throughout a marathon. However, a major flaw in the implementation of the `Aodv.ned` module meant motes did not correctly break links and route RERR messages.

6.2 Research Conclusions

As hypothesized, this research successfully showed that under the conditions of MarathonSim's environment, a minimum of 50 motes equipped with an nRF52840 SoC running 802.15.4 and maximum output power can deliver more than 50% of data through the mesh network. The previous work by MarathonNet showed that using only 6 gateways but 500 runners established a similar connectivity.

The experiment partially fulfilled the goals of this research. The successful hypothesis, and other measurements of PDR and delay demonstrate an ability to predict general capabilities of the WSN to transport data. While both increased transmission power and number of runners led to improved PDRs, the number of runners experienced diminishing returns but affected power consumption more than other factors. Unfortunately, the failure of the AODV trials negate any attempt at evaluating the routing protocols against each other. Additionally, the small samples force performance predictions to be general comparisons between factor levels instead of estimates of potential system performance. It would require larger samples, as well as limited prototypes, to be able to predict specific performance.

Still, the results assist the development of a future WSN. The flood protocol demonstrated no obvious faults which suggests the Bluetooth Mesh protocol to be suitable for development. Average power consumption per node, no matter the configuration, never exceeded the capacity of a simple coin cell battery. As for connectivity, transmission power improved PDR similar to increased runners but at less cost to power consumption and delay. This suggests increasing the number of devices is less helpful with data transport than fewer devices with longer ranges.

6.3 Research Contributions

This research builds upon MarathonNet by redeveloping the proposed marathon mesh network within an OMNeT++ simulation including data transmission modeling, propagation, and MAC. The simulation is also moderately modular by allowing new physical devices to be modelled instead of the nRF52840 to assist with prototype development. By modeling the nRF52840 first, this research provides evidence of the suitability of the nRF52840 or similar device to a WSN application.

6.4 Limitations of Research

MarathonSim in its current state underperforms in a number of ways. The long runtime of large simulations limited experimental runs and, by extension, the statistical analysis. The lack of an existing system to use observational data to validate MarathonSim's performance means estimates cannot suggest exact performance. Additionally, the failure of `Aodv.ned` module left a research goal unfulfilled.

The validation of MarathonSim is based on face validation where valid analytical models and conceptually correct interactions are reviewed rather than tested against true sample data. The uncertainty with face validation is how non-functional modules end up in experimental trials. If even a small prototype existed, then the results could be compared to the simulation to confirm behaviors and allow informed predictions.

The failure of the `Aodv.ned` module undermined the analysis of results. Half of the results came from trials using AODV with true performance metrics likely different than those measured as well as throwing off the estimated overall mean performances. Despite testing for the exact situation on a small scale simulation, the introduction of gateways with static routes caused an error in the module. Fixing the module would require adjustment or translation of an INET static, manual IP route to an `AodvRouteData` type.

6.5 Recommendations for Future Work

- The `Aodv.ned` module should be corrected and retested in the experiments proposed in this work. The flawed functionality of the routing module undermined half of the experiment and failed to compare routing protocols.

- MarathonSim in its current state is very inefficient as evidenced by long runtime and bulk data. If it is to continue being used, the code base should be reevaluated for bloat and reimplement only necessary components.

- Currently, INET does not have a full network stack for MANETs or WSNs. A new full-network stack should be developed for INET alongside a prototype mote to validate the simulation properly.

- This research only simulated 802.15.4, but there are several other competitive wireless technologies. An effort should be made to test a new wireless technology to the scenario to highlight differences. LongRange (LoRa) appears especially promising due to the 10x range potential and the sensors requiring only limited data rates.

Appendix A. MarathonSim NED Files

MarathonSim.ned This file describes the top-level modules and connections of the marathon environment.

```
package marathonsim.simulations;

import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
import inet.node.ethernet.EtherSwitch;
import inet.node.ethernet.Eth100G;
import inet.visualizer.integrated.IntegratedMultiVisualizer;
import inet.physicallayer.contract.packetlevel.IRadioMedium;
import marathonsim.*;

network MarathonSim
{
    parameters:
        int numHosts;  //number of runners in the marathon, set in INI
        int numGateways; //number of gateways in the marathon, set in
            INI
        //@display("bgb=1247.2416,2494.4832;bgi=background/
            marathon_route,s"); //graphical display
        @display("bgb=4988.9664,9977.9328;bgi=background/marathon_route,
            s"); //graphical display
        //Statistic collecting the ratio of packets received at Hub to
            packets sent by all nodes
        @statistic[packetDeliveryRatio](title="PDR"; source=count(
            appPacketReceived)/count(appPacketSent); record=last);
        //Statistic collecting the packet error rate as a function of
            SNIR and ASPK error rate during reception
        @statistic[packetErrorRate](title="Packet error rate"; source=
            packetErrorRate(packetSentToUpper); record=vector,mean,
            histogram);
        //Statistic collecting the total packet transmissions by
            wireless interfaces
        @statistic[transmitPacket](source=(packetBytes(transmitPacket));
            record=sum);

    submodules:
        host[numHosts]: marathonSensorBase {
            @display("i=device/palm2;is=s;p=358.6022,2215.6494");
        }
        gateway[numGateways]: Ieee802154gateway {
            @display("p=465.32904,2322.3762;is=m");

        }
        EMSHub: EmsHub {
            @display("i=device/pc;p=1011.7705,2305.2998;is=s");
            mobility.typename = "StationaryMobility";
        }
        connectingSwitch: EtherSwitch {
            @display("p=1007.50146,2002.1957");
        }
        configurator: Ipv4NetworkConfigurator {
            @display("p=1099.12,314.748;is=s");
        }
        radioMedium: <default("Ieee802154NarrowbandScalarRadioMedium")>
            like IRadioMedium {
            @display("p=1144.1118,76.84333;is=s");

        }
        visualizer: IntegratedMultiVisualizer {
            @display("p=829.336,64.948;is=s");
        }

    connections allowunconnected:
        EMSHub.ethg++ <--> Eth100G { length = 1m; } <-->
            connectingSwitch.ethg++;
        for i=0..numGateways-1 {
            gateway[i].ethg++ <--> Eth100G { length = 1m; } <-->
                connectingSwitch.ethg++;
        }
}

@license(LGPL);
```

MarathonSensorBase.ned The sensor base allows modular assignment to flood or AODV routing method.

```
package marathonsim;


import inet.node.base.ApplicationLayerNodeBase;
import inet.applications.generic.IIpvxTrafficGenerator;
import inet.applications.contract.IApp;

module marathonSensorBase extends inet.node.base.
    ApplicationLayerNodeBase
{
    parameters:
        @display("i=misc/sensor2");
        @figure[submodules];
        //Signal to collect power consumed
        @statistic[powerConsumption](title="Power Consumption"; source=
            powerConsumptionChanged; record=last(timeavg)*21600);

        //Application Layer:
```

```
        int numGens = default(1);  //Number of data generator
            applications

        //Network Layer:
        hasIpv6 = false;
        forwarding = default(true); //Allows node to forward other
            packets
        ipv4.arp.proxyArpInterfaces = default(""); // proxy arp is
            disabled on hosts by default

        //Physical+Link Layers
        wlan[*].radio.energyConsumer.typename = default("
            StateBasedEpEnergyConsumer");
        energyStorage.typename = default("SimpleEpEnergyStorage"); //
            Simple energy source
        numWlanInterfaces = default(1);
        wlan[*].typename = default("Ieee802154NarrowbandInterface"); //
            Sets both 802.15.4 transceiver and mac attributes
        hasStatus = default(true); //Used for lifecyle shutdown/startup
            power management

    submodules:
        //Application Layer Submodules:
        dataGen[numGens]: <> like IIpvxTrafficGenerator {
            @display("p=751.27496,74.024994");
        }
        //Implements the AODV routing module when specified by the
            runtime parameters
        appRoutingModule: <default("")> like IApp if typename != "" {
            parameters:
                @display("p=924.3,75.4");
        }

    connections allowunconnected:
        for i=0..numGens-1 {
            dataGen[i].ipOut --> tn.in++;
            dataGen[i].ipIn <-- tn.out++;
        }
        appRoutingModule.socketOut --> at.in++ if exists(
            appRoutingModule);
        appRoutingModule.socketIn <-- at.out++ if exists(
            appRoutingModule);
}
```

Ieee802154gateway.ned Similar to the sensor base, allows modular assignment to flood or AODV routing

```
package marathonsim;
import inet.node.inet.WirelessHost;
import inet.applications.contract.IApp;

module Ieee802154gateway extends WirelessHost
{
    parameters:
        @display("i=device/wifilaptop");

        //Networking Layer
        forwarding = default(true);

        //Physical and Link Layers
        numWlanInterfaces = default(1);
        numEthInterfaces = default(1);
        wlan[*].typename = default("Ieee802154NarrowbandInterface");


    submodules:
        //Implements the AODV routing module when specified by the
            runtime parameters
        appRoutingModule: <default("")> like IApp if typename != "" {
             parameters:
                 @display("p=924.3,75.4");
        }

    connections allowunconnected:
        appRoutingModule.socketOut --> at.in++ if exists(
            appRoutingModule);
        appRoutingModule.socketIn <-- at.out++ if exists(
            appRoutingModule);

}
```

Appendix B. MarathonSim INI Config File

omnetpp.ini The INI file sets runtime parameters and instances of objects make during simulation.

```
[General]
network = MarathonSim

# Limit simulation to 6h marathon
sim-time-limit = 21600s

# Limits transmissions to only nodes within interference or
    communication range
# Interference Range - Max distance simultaneous transmissions interfere
     at the receiver
# Communication Range - Max distance that received power exceeds
    sensitivity of receiver
*.radioMedium.rangeFilter = "interferenceRange"
#*.radioMedium.rangeFilter = "communicationRange"

# Transmissions are not sent to inactive transceivers
*.radioMedium.radioModeFilter = true

# Background Noise - set to thermal background noise for 2 MHz BW
    channel
# -174 + 10 log (2MHz)
*.radioMedium.backgroundNoise.power = -110dBm

# Rbp breakpoint distance (4*h1*h2/(wavelength=12.5cm))
*.radioMedium.pathLoss.breakpointDistance = 32m
# Free Space Path Loss @ 32m
*.radioMedium.pathLoss.l02 = 70.33
# Loss coefficient
```

```
*.radioMedium.pathLoss.alpha2 = 4

# General -- Factors
*.numHosts = ${n=25, 50, 75, 100, 125}

# Factors: Physical - Transmission Power
# Communication Range Scaling:
# 1mW => 176m =(25%)=> 44m
# 2.5mW => 222m =(25%)=> 55.5m
# 6.3mW => 280m =(25%)=> 70m
# Pre-scaled transmission power. To meet 25% scaling, transmission power
    is reduced using the breakpoint distance formula.
#**.transmitter.power = ${power=1,2.5,6.3}mW # nordic nRF52840 0dBm,4dBm
   ,8dBm 1,2.5,6.3
**.transmitter.power = ${power=0.00384173, 0.00937923, 0.0241444}mW #
   nordic nRF52840 0dBm,4dBm,8dBm 1,2.5,6.3
# Power consumed by the transmitter is not scaled as it is not distance
   dependent.
**.transmitterTransmittingPowerConsumption = ${14.4, 28.8, 44.4 ! power}
  mW #4.8mA(14.4mW), 9.6mA(28.8), 14.8mA(44.40v) for 0,4,8dbm, 3V*i mA
    = p mW

# Parameters: Physical - Power
**.nominalCapacity = 2400J #Arbitrary high capacity to avoid shutdown
   errors
**.offPowerConsumption = 0mW
**.sleepPowerConsumption = 0.001mW
**.switchingPowerConsumption = 0mW
**.receiverIdlePowerConsumption = 0.005mW
**.receiverBusyPowerConsumption = 0.1mW
**.receiverReceivingPowerConsumption = 13.8mW #4.6mA - nRF52840
**.transmitterIdlePowerConsumption = 5mW
```

```

# Parameters: Physical - Mobility
*.host[*].mobility.typename = "BonnMotionMobility"
*.host[*].mobility.traceFile = ${"./runner_speeds/runners_25.txt", "./
    runner_speeds/runners_50.txt", "./runner_speeds/runners_75.txt","./
    runner_speeds/runners_100.txt", "./runner_speeds/runners_125.txt" !
    n}
*.host[*].mobility.is3D = false
*.host[*].mobility.nodeId = -1 # Assigns each node ID to Line Number in
    tracefile (0 -> 0, 1 -> 1 etc)
*.host[*].mobility.updateInterval = 10s #

# Parameters: Physical - Receiver Sensitivity
**.sensitivity = -100dBm # nordic nRF52840

# Parameters: Link Configuration
#achieving 90% successful transmission requires n < BE/4 ; BE=2^x-1
*.host[*].wlan[*].mac.macMinBE = 9
*.host[*].wlan[*].mac.macMaxBE = 12

# Parameters: Network Configuration
*.configurator.addStaticRoutes = false
**.netmaskRoutes = ""
*.configurator.config = xmldoc("./network_topology/flat_network.xml")
**.arp.typename = "GlobalArp"

# General -- Applications Setup
# Application: Mobile Nodes
*.host[*].numGens = 1
*.host[*].dataGen[0].typename = "IpvxTrafGen"
*.host[*].dataGen[0].startTime = uniform(1s,60s)
*.host[*].dataGen[0].destAddresses = "EMSHub"
```

```
*.host[*].dataGen[0].protocol = 200
*.host[*].dataGen[0].packetLength = 64B #2x[ID, heartrate, temp, Seq,
    timestamp]=2x32B=64B
*.host[*].dataGen[0].sendInterval = 60s

# Application: Central Sink
*.EMSHub.numSinks = 1
*.EMSHub.dataSink[0].typename = "IpvxTrafSink"
*.EMSHub.dataSink[0].protocol = 200

# Recording Configs
# Turns off a number of default recorded statistics to reduce runtime.
**.vector-record-eventnumbers = false #Turns off eventnumber recording,
    only used for sequence charts
#**.scalar-recording = false
**.packetErrorRate:vector.vector-recording = true
**.endToEndDelay:vector.vector-recording = true
**.packetDropIncorrectlyReceived:vector.vector-recording = true
**.packetReceived:vector.vector-recording = true
**.wlan[*].mac.queue.queueingTime:vector.vector-recording = true
**energyConsumer.powerConsumption:vector.vector-recording = false
**energyStorage.powerConsumption:vector.vector-recording = false
**energyStorage.residualEnergyCapacity:vector.vector-recording = false
**.radio.radioMode:vector.vector-recording = false
**.receptionState:vector.vector-recording = false
**.transmissionState:vector.vector-recording = false
**.eth[*]**.statistic-recording = false
**.connectingSwitch**.statistic-recording = false
**.wlan[*].mac.queue.packetPopped:vector(packetBytes).vector-recording =
    false
**.wlan[*].mac.queue.packetPushed:vector(packetBytes).vector-recording =
    false
```

```
**.wlan[*].mac.queue.queueLength:vector.vector-recording = false
**.wlan[*].mac.packetDropIncorrectlyReceived:vector(packetBytes).vector-
    recording = false
**.dataGen[*].packetSent:vector(packetBytes).vector-recording = false
**.bitErrorRate.statistic-recording = false
**.minSnir.statistic-recording = false
**.queueingTime:histogram.bin-recording = false
#**.endToEndDelay:histogram.bin-recording = false
**.packetErrorRate:histogram.bin-recording = false

[Config MinRecording]
# Completely turns off vector recording.
# MinRecording REQUIRED for full duration tests with >25 nodes. Result
    files >10GB per trial
**.vector-recording = false
**.hasTcp = false
**energyConsumer.powerConsumption.statistic-recording = false
**energyStorage.powerConsumption.statistic-recording = false
**energyStorage.residualEnergyCapacity.statistic-recording = false
**.radio.radioMode.statistic-recording = false
**.receptionState.statistic-recording = false
**.transmissionState.statistic-recording = false
**.eth[*]**.statistic-recording = false
**.connectingSwitch**.statistic-recording = false
**.wlan[*].mac.queue.packetPopped.statistic-recording = false
**.wlan[*].mac.queue.packetPushed.statistic-recording = false
**.wlan[*].mac.queue.queueLength.statistic-recording = false
**.wlan[*].mac.packetDropNotAddressedToUs.statistic-recording = false
**.wlan[*].mac.packetDropRetryLimitReached.statistic-recording = false
**.wlan[*].mac.packetDropIncorrectlyReceived.statistic-recording = false
**.udp.packetReceived.statistic-recording = false
**.udp.passedUpPk.statistic-recording = false
```

```
**udp**.statistic-recording = false
**ip**.statistic-recording = false
**gateway[*]**.statistic-recording = false
#**.dataGen[*].appPacketSent.statistic-recording = false
**.queueingTime.statistic-recording = false
**.bitErrorRate.statistic-recording = false
**.symbolErrorRate.statistic-recording = false
**.minSnir.statistic-recording = false
**.endToEndDelay:histogram.bin-recording = false
#**.packetErrorRate:histogram.bin-recording = false
**.lo**.statistic-recording = false

[Config AODVBase]
# Parameters: AODV
# Sets the IEEEGateway modules to use AODV routing
*.gateway[*].appRoutingModule.typename = "inet.routing.aodv.Aodv"
# Sets the motes to use AODV
*.host[*].*.routingTableModule = "^.ipv4.routingTable"
*.host[*].appRoutingModule.typename = "inet.routing.aodv.Aodv"
# Default AODV Recommendations
*.*.aodv.activeRouteTimeout = 3s
*.*.aodv.deletePeriod = 1s
# Net diameter is recommended to be the longest path in the network.
# However, large netDiameters delays repeat RREQs when discovery fails
    and
# paths are not expected to be longer than 12 hops. Use default.
*.*.aodv.netDiameter = 35

[Config FloodingBase]
## Parameters: Flooding
# Sets the motes and gateways to use flooding protocol
**.hasGn = true
```

```
*.host[*].hasIpv4 = false
*.host[*].hasIpv6 = false
**.generic.typename = "SimpleNetworkLayer"
**.np.typename = "Flooding"

# Adjust protocol header for application packets. Attaches 'flood'
    header instead of IP.
*.host[*].dataGen[*].networkProtocol = "flooding"
# Sets the max entries in the message history to num of nodes * 360 (
    Every message sent)
*.*.generic.np.bcMaxEntries = ${($n)*360}
# DelTime does not come into effect.  Only when history reaches max.
*.*.generic.np.bcDelTime = 21600s

[Config LimitedGateways]
# LimitedGateways -- Parameters
# Sets the number of gateways and the positions. Generated from
    waypointsScript.py
*.numGateways = 19
*.gateway[*].mobility.typename = "StationaryMobility"
*.gateway[*].mobility.initFromDisplayString = false
*.gateway[*].mobility.initialZ = 0m
*.gateway[0].mobility.initialX = 342m
*.gateway[0].mobility.initialY = 2226m
*.gateway[1].mobility.initialX = 416m
*.gateway[1].mobility.initialY = 1777m
*.gateway[2].mobility.initialX = 623m
*.gateway[2].mobility.initialY = 1434m
*.gateway[3].mobility.initialX = 336m
*.gateway[3].mobility.initialY = 1390m
*.gateway[4].mobility.initialX = 249m
*.gateway[4].mobility.initialY = 1097m
```

```
*.gateway[5].mobility.initialX = 99m
*.gateway[5].mobility.initialY = 530m
*.gateway[6].mobility.initialX = 311m
*.gateway[6].mobility.initialY = 168m
*.gateway[7].mobility.initialX = 673m
*.gateway[7].mobility.initialY = 392m
*.gateway[8].mobility.initialX = 708m
*.gateway[8].mobility.initialY = 628m
*.gateway[9].mobility.initialX = 954m
*.gateway[9].mobility.initialY = 748m
*.gateway[10].mobility.initialX = 779m
*.gateway[10].mobility.initialY = 405m
*.gateway[11].mobility.initialX = 910m
*.gateway[11].mobility.initialY = 467m
*.gateway[12].mobility.initialX = 929m
*.gateway[12].mobility.initialY = 873m
*.gateway[13].mobility.initialX = 769m
*.gateway[13].mobility.initialY = 757m
*.gateway[14].mobility.initialX = 767m
*.gateway[14].mobility.initialY = 1320m
*.gateway[15].mobility.initialX = 1010m
*.gateway[15].mobility.initialY = 1415m
*.gateway[16].mobility.initialX = 604m
*.gateway[16].mobility.initialY = 1714m
*.gateway[17].mobility.initialX = 261m
*.gateway[17].mobility.initialY = 2139m
*.gateway[18].mobility.initialX = 340m
*.gateway[18].mobility.initialY = 2400m


[Config LimitedGatewaysAODV]
extends = AODVBase, LimitedGateways
```

```
# LimitedGatewayAODV -- Visualization
# Visualization: Interfaces
#*.visualizer.*.interfaceTableVisualizer[*].displayInterfaceTables =
    true
# Visualization: Network
#*.visualizer.*.routingTableVisualizer[*].displayRoutingTables = false
#*.visualizer.*.routingTableVisualizer[*].destinationFilter = "
    destination"
## Visualization: Info
#*.visualizer.*.infoVisualizer[*].modules = "*.EMSHub.dataSink[0]"
#*.visualizer.*.infoVisualizer[*].format = "%t"
#*.visualizer.*.infoVisualizer[*].placementHint = "topCenter"
## Visualization: Traffic
*.visualizer.*.networkRouteVisualizer[*].displayRoutes = true
*.visualizer.*.networkRouteVisualizer[*].packetFilter = "**appData**"
*.visualizer.*.networkRouteVisualizer[*].fadeOutMode = "simulationTime"
*.visualizer.*.networkRouteVisualizer[*].fadeOutTime = 10s

[Config LimitedGatewaysAODVMinRecording]
extends = LimitedGatewaysAODV, MinRecording

[Config LimitedGatewaysFlood]
extends = LimitedGateways, FloodingBase
*.visualizer.*.networkRouteVisualizer[*].displayRoutes = true
*.visualizer.*.networkRouteVisualizer[*].packetFilter = "*"
*.visualizer.*.networkRouteVisualizer[*].fadeOutMode = "simulationTime"
*.visualizer.*.networkRouteVisualizer[*].fadeOutTime = 10s

[Config LimitedGatewaysFloodMinRecording]
extends = LimitedGatewaysFlood, MinRecording
```

Appendix C. waypointScript.py

waypointScript.py This script is responsible for making the mobility trace files for motes and stationary mobility/distribution for the gateways.

```
import sys
import getopt
import math
import numpy as np
import random
import scipy.stats
import argparse
from enum import Enum

#Map scaling allows physical dimensions and runner velocities to be
    reduced or increased
MAP_SCALING = 1

#imported pixel coordinates based on 1000x2000 resolution marathon map
X_PIXELS = 1000
X_MILES = 3.1*MAP_SCALING

Y_PIXELS = 2000
Y_MILES = 6.2*MAP_SCALING

x_meters_per_pixel = (X_MILES * 1609.344 ) / X_PIXELS # meters per pixel
    - 1mi=1609.344m
y_meters_per_pixel = (Y_MILES * 1609.344 ) / Y_PIXELS # meters per pixel

waypoints = []
gateways_stations = []
runners = []

```

```
class Runner_Type(Enum):
    WALKER = 1
    WALK_RUN = 2
    RUNNER = 3
    DROP_OUT = 4
class Runner_Status(Enum):
    STARTING_LINE = 1
    WALKING = 2
    RUNNING = 3
    DROPPED_OUT = 4
    FINISH_LINE = 5

class Runner:
    def __init__(self):
        self.runner_type = Runner_Type.WALKER
        self.runner_status = Runner_Status.STARTING_LINE
        self.runner_time = 0

    @classmethod
    def from_type(cls, runner_type, time):
        runner = cls()
        runner.runner_type = runner_type
        runner.runner_time = time
        runner.runner_status = Runner_Status.RUNNING
        return runner


class Marathon:
    def __init__(self, waypoint_file, num_runners):
        self.waypoint_file = waypoint_file
        self.gateways_file = "gateways.txt"
        self.waypoints = []
        self.gateways_stations = []
        self.mean_time = 266 #minutes -- 4 hours 26 minutes
        self.std_time = 59 #minutes -- 59 minutes
        self.num_runners = num_runners
        self.runners = []
        #Semi arbitrary, multiple online references regarding marathon '
            types'
        self.drop_out_rate = 0.17 # 17%
        self.no_walk = 0.65 # 65%
        self.walkers = 0.20 # 20%
        self.pace_walkers = 0.15 # 15%

    #ingests (x,y) pixel tuples from text files and returns them as (x,y
        ) meters
    def read_waypoints(self, waypoints, file_name):
        try:
            with open(file_name, "r") as waypoint_file:
                data = waypoint_file.readlines()
                lines = [ x.strip() for x in data ]
                pairs = [ line.split(',') for line in lines ]
                waypoints = [ tuple([float(int(x)*x_meters_per_pixel),
                    float(int(y)*y_meters_per_pixel)]) for x,y in pairs
                    ]
                #print(waypoints)
                return waypoints
        except:
            print('Error reading file:', waypoint_file)
            exit()

    #uses the waypoints to calculate length of segments between
        waypoints
    def gen_distances(self, waypoints):
        distances = [ math.hypot(i[0]-j[0],i[1]-j[1]) for i, j in zip(
            waypoints[:-1],waypoints[1:]) ]
        #print(distances)
        return [0] + distances

    #Randomly assigns runner behaviors
    def populate_runners(self, num_runners):
        #generate random runner times
        runner_times = scipy.stats.norm.rvs(loc=self.mean_time, scale=
            self.std_time, size=num_runners)

        for i,time in enumerate(runner_times):
            #type_decider = np.random.uniform()

            if i/num_runners <= 0.20:
                runner_type = Runner_Type.WALK_RUN
            elif i/num_runners <= 0.275:
                runner_type = Runner_Type.DROP_OUT
            elif i/num_runners <= 0.375:
                runner_type = Runner_Type.WALKER
            else:
                runner_type = Runner_Type.RUNNER

            self.runners.append( Runner.from_type(runner_type, time) )

    #Takes a runner and 'runs' them through the course. Each segment is
    #divided by runner's velocity based on finish time and behavior.
    #also transitions runner to different speeds (walk/run) or dropout.
    def build_route(self, runner):
        waypoints = self.waypoints
        distances = self.gen_distances(waypoints)
        running_time = 0
        times = []
        route = []
        if runner.runner_time:
            runner_mps = sum(distances)/(runner.runner_time*60)
            #runner_mps = mph_to_mps(runner_mph)
            walking = mph_to_mps(4)*MAP_SCALING
        else:
            print("Runner times not populated")
            return
        for i,(segment,waypoint) in enumerate(zip(distances, waypoints)):

            if runner.runner_status == Runner_Status.DROPPED_OUT:
                route.append(drop_out_point)
                segment_time = 0
                running_time = running_time + segment_time
                times.append(running_time)
            elif runner.runner_status == Runner_Status.WALKING:
                route.append(waypoint)
                segment_time = segment/(sum([walking,runner_mps])/2)
                running_time = running_time + segment_time
                times.append(running_time)
            elif runner.runner_status == Runner_Status.RUNNING:
                route.append(waypoint)
                segment_time = segment/np.random.uniform(runner_mps*.9, runner_mps*1.1)
                running_time = running_time + segment_time
                times.append(running_time)
            else:
                print("Runner status unknown")
                return

            if runner.runner_type == Runner_Type.WALKER:
                if runner.runner_status == Runner_Status.RUNNING:
                    if np.random.uniform() <= .40:
                        runner.runner_status = Runner_Status.WALKING
                elif runner.runner_status == Runner_Status.WALKING:
                    if np.random.uniform() <= .80:
                        runner.runner_status = Runner_Status.RUNNING

            if runner.runner_type == Runner_Type.RUNNER:
                pass #Runners will always run
            if runner.runner_type == Runner_Type.WALK_RUN:
                if runner.runner_status == Runner_Status.RUNNING:
                    if np.random.uniform() <= .20:
                        runner.runner_status = Runner_Status.WALKING
                elif runner.runner_status == Runner_Status.WALKING:
                    runner.runner_status = Runner_Status.RUNNING

            if runner.runner_type == Runner_Type.DROP_OUT:
                if runner.runner_status == Runner_Status.DROPPED_OUT:
                    pass
                else:
                    if np.random.uniform() <= (i*.005):
                        print("Forcing Dropout at segment: ", i)
                        runner.runner_status = Runner_Status.DROPPED_OUT
                        drop_out_point = waypoint
        runner.time_points = times
        runner.route = list(zip(times, route))

    def build_routes(self):
        for runner in self.runners:
            self.build_route(runner)

    def write_route_to_file(self, output_file="default_output.txt"):
        route_string = ""
        try:
            with open(output_file, "w") as out:
                for runner in self.runners:
                    #for time,coords in zip(runner.time_points, self.
                        waypoints):
                    for time,coords in runner.route:
                        route_string = route_string + str(time) + " " +
                            str(coords[0]) + " " + str(coords[1]) + " "
                    route_string = route_string + "\n"
                    out.write(route_string)
                    route_string = ""
        except:
            print("Error writing to output file")
        out.close()

#Uses runner speed and segment distances to calc time to run/walk the
    segment
def calc_times(marathon_waypoints, speed, distances):
    time_lapse = []
    running_time = 0
    for segment in distances:
        segment_time = segment / speed
        running_time = running_time + segment_time
        time_lapse.append(running_time)
    #print(time_lapse)
    return time_lapse
# mph_to_mps:
# converts a single argument, miles per hour (mph), into meters per
    second.
def mph_to_mps(mph):
    meters_per_hour = mph*1609.344
    meters_per_second = meters_per_hour / 3600 #3600 seconds per hour
    return meters_per_second

# minute_miles_to_mps:
# converts a single argument for a pace, minutes per mile, into meters
    per second.
def minute_miles_to_mps(pace):
    seconds_per_mile = pace*60
    miles_per_second = 1/seconds_per_mile
    meters_per_second = 1609.344*miles_per_second
    return meters_per_second


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-n", "--num_runners", help="number of runners
        in marathon scenario", type=int)
    parser.add_argument("--config", help="one or more strings for
        waypoint inputs to be generated for the ini file", nargs="+")
    args = parser.parse_args()

    myMarathon = Marathon('marathon_waypoints.txt', args.num_runners)

    myMarathon.waypoints = myMarathon.read_waypoints(myMarathon.
        waypoints, 'marathon_waypoints.txt')
    myMarathon.gateways_stations = myMarathon.read_waypoints(myMarathon.
        gateways_stations, 'gateways.txt')

    myMarathon.populate_runners(myMarathon.num_runners)

    myMarathon.build_routes()

    myMarathon.write_route_to_file("runners_"+str(args.num_runners)+".
        txt")

    gateways_stations = []
    for i,gateways_point in enumerate(myMarathon.gateways_stations):
        gateways_stations.append(f"*.gateway[{i}].mobility.initialX = {
            math.floor(gateways_point[0])}m\n")
        gateways_stations.append(f"*.gateway[{i}].mobility.initialY = {
            math.floor(gateways_point[1])}m\n")
    try:
        with open("gatewaysini.txt", "w") as omnet_out:
            for line in gateways_stations:
                omnet_out.write(line)
    except:
        print("Error writing omnetconfig: gateways_ini.txt")
    omnet_out.close()


    myMarathon.waypoint_distances = myMarathon.gen_distances(myMarathon.
        waypoints)

    #print("sum of distances", sum(myMarathon.waypoint_distances))

if __name__ == '__main__':
    main()
```

Bibliography

1. G. Chiampas, “Emergency Preparedness in Mass Events,” 2012. Accessed 22 Jun 2020, [Online], Available: http://aims-worldrunning.org/symposium/7th_AIMS_Symposium_George_Chiampas_ppt.pdf.

2. “Air Force Marathon.” Accessed 21 July 2020, [Online], Available: https://www.usafmarathon.com/.

3. “Hexoskin Smart Shirts.” https://www.hexoskin.com/. Accessed 13 Aug 2020, [Online], Available: https://www.usafmarathon.com/.

4. A. Christian and J. Healey, “Gathering Motion Data Using Featherweight Sensors and TCP / IP,” in *IEEE International Symposium on Wearable Computing, Workshop on On-body Sensing*, (Osaka, Japan), 18-21 October 2005.

5. B. M. Eskofier, S. I. Lee, M. Baron, A. Simon, C. F. Martindale, H. Gaßner, and J. Klucken, “An Overview of Smart Shoes in the Internet of Health Things: Gait and Mobility Assessment in Health Promotion and Disease Monitoring,” *Applied Sciences (Switzerland)*, vol. 7, no. 10, p. 986, 2017.

6. B. Reeder and A. David, “Health at hand: A systematic review of smart watch uses for health and wellness,” *Journal of Biomedical Informatics*, vol. 63, pp. 269–276, 2016.

7. “Omnet++.” accessed 1 Feb 2020, [Online], Available: https://omnetpp.org/.

8. “Inet framework.” accessed 1 Feb 2020, [Online], Available: https://inet.omnetpp.org/.

9. *IEEE Standard for Low-Rate Wireless Networks.* IEEE Std 802.15.4-2020.

10. Nordic Semiconductors, “Nordic nRF52840 Product Specification v.1.1,” 2019. Accessed 20 Aug 2020, [Online], Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fkeyfeatures_html5.html.

11. S. Yinbiao, K. Lee, P. Lanctot, F. Juanbin, H. Hao, B. Chow, J.-P. Desbenoit, G. Stephan, L. Hui, X. Guodong, S. Chen, D. Faulk, T. Kaiser, H. Satoh, O. Jinsong, W. Shou, Z. Yan, S. Junping, Y. Haibin, Z. Peng, L. Dong, and W. Qui, “Internet of Things: Wireless Sensor Networks,” Tech. Rep. December, International Electrotechnical Commission, 2014. Accessed 18 May 2020, [Online], Available: http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf.

12. H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks.* John Wiley and Sons Inc., 2005.

13. C. S. Raghavendra, K. Sivalingam, and T. Znati, *Wireless Sensor Networks.* Massachusetts: Kluwer Academic Publishers, 2004.

14. J. A. Manrique, J. S. Rueda-Rueda, and J. M. Portocarrero, “Contrasting Internet of Things and Wireless Sensor Network from a Conceptual Overview,” in *2016 IEEE International Conference on Internet of Things*, pp. 252–257, Institute of Electrical and Electronics Engineers Inc., Dec 2016.

15. E. Callaway, *Wireless Sensor Networks: Architectures and Protocols.* Boca Raton: Auerbach Publications, 2004.

16. M. Miller, “Evolution of Industrial Wireless Sensor Networks,” Nov 2017. Accessed 18 May 2020, [Online], Available: https://www.mwee.com/design-center/evolution-industrial-wireless-sensor-networks.

17. M. May, “Design of a Wireless Sensor Node Platform,” Master’s thesis, University of Waikato, 2012. Accessed 18 May 2020, [Online], Available: https://wand.net.nz/sites/default/files/mm236%20COMP520%20Report.pdf.

18. J. Hill, *System Architecture for Wireless Sensor Networks.* PhD thesis, University of California Berkeley, Berkeley, 2003. Accessed 18 May 2020, [Online], Available: http://www.jlhlabs.com/jhill_cs/jhill_thesis.pdf.

19. N. Kamyab Pour, *Energy Efficiency in Wireless Sensor Networks.* PhD thesis, University of Technology, Sidney, 2015. Accessed 22 May 2020, [Online], Available: https://arxiv.org/abs/1605.02393.

20. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-efficient Communication Protocol for Wireless Microsensor Networks,” in *Hawaii International Conference on System Sciences*, pp. 1–10, IEEE, 2000.

21. H.-Y. Zhou, D.-Y. Luo, Y. Gao, and D.-C. Zuo, “Modeling of Node Energy Consumption for Wireless Sensor Networks,” *Wireless Sensor Network*, vol. 03, no. 01, pp. 18–23, 2011. Accessed 10 May 2020, [Online], Available: https://www.scirp.org/journal/paperinformation.aspx?paperid=3803.

22. C. H. S. Oliveira, Y. Ghamri-Doudane, and S. Lohier, “A Duty Cycle Self-Adaptation Algorithm for the 802.15.4 Wireless Sensor Networks,” in *Global Information Infrastructure and Networking Symposium*, (Trente, Italy), pp. 1–7, 2013.

23. Recommendation ITU-R P.1411-9, “Propagation Data and Prediction Methods for the Planning of Short-range Outdoor Radiocommunication Systems and Radio Local Area Networks in the Frequency Range 300 MHz to 100 GHz,” Tech. Rep. 1411-9, International Telecommunication Union, 2017.

24. D. Chen, Z. Liu, L. Wang, M. Dou, J. Chen, and H. Li, "Natural disaster monitoring with wireless sensor networks: A case study of data-intensive applications upon low-cost scalable systems," *Mobile Networks and Applications*, vol. 18, pp. 651–663, Oct 2013.

25. K.-T. Kim and J.-G. Han, "Design and Implementation of a Real-time Slope Monitoring System based on Uqiquitous Sensor Network," in *25th International Symposium on Automation and Robotics in Construction*, (Vilnius, Lithuania), pp. 330–336, 2008.

26. M. Ryu, S. Choi, J. Kim, J. Yun, T. Miao, I.-Y. Ahn, and S.-C. Choi, "Design and Implementation of a Connected Farm for Smart Farming System," in *IEEE Sensors*, pp. 1–4, 2015.

27. A. Mondal, I. S. Misra, and S. Bose, "Building a low cost solution using wireless sensor network for agriculture application," in *Proceedings of 2017 International Conference on Innovations in Electronics, Signal Processing and Communication, IESC 2017*, pp. 61–65, Institute of Electrical and Electronics Engineers Inc., Oct 2017.

28. R. Lacoss and R. Walton, "Strawman Design of a DSN to Detect and Track Low Flying Aircraft," 1978.

29. G. Mitchell, "Data in a Wireless Sensor Network: A View from the Field," tech. rep., BBN Technologies. Accessed 3 Sep 2020, [Online], Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.416.7640&rep=rep1&type=pdf.

30. J. Vasseur, "RFC 7102 - Terms Used in Routing for Low-Power and Lossy Networks," 2014. Accessed 12 Sep 2020, [Online], Available: https://tools.ietf.org/html/rfc7102.

31. C. Hetting, "White Papers: Wi-Fi HaLow best IoT tech for range, battery life, & breadth of applicability," 2020. Accessed 12 Sep 2020, [Online], Available: https://wifinowglobal.com/news-and-blog/white-papers-wi-fi-halow-best-iot-tech-for-range-battery-life-breadth-of-applicability/.

32. J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*. Pearson, 6th ed., 2013.

33. C. Perkins, E. Belding-Royer, and S. Das, "RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing," 2003. Accessed 18 May 2020, [Online], Available: https://tools.ietf.org/html/rfc3561.

34. D. Pfisterer, M. Lipphardt, G. Buschmann, H. Hellbrueck, S. Fischer, and J. H. Sauselin, "MarathonNet: Adding value to large scale sport events - A Connectivity Analysis," *Proceedings of the First International Conference on Integrated Internet Ad hoc and Sensor Networks, InterSense '06*, p. 12, 2006.

35. S. Ray, J. B. Carruthers, and D. Starobinski, “RTS/CTS-induced congestion in ad hoc wireless LANs,” *IEEE Wireless Communications and Networking Conference, WCNC*, vol. 3, pp. 1516–1521, 2003.

36. E. J. Bach and M. G. Fickel, *An Analysis of the Feasibility and Applicability of IEEE 802.x Wireless Mesh Networks within the Global Information Grid*. PhD thesis, Naval Postgraduate School, 2004. Accessed 4 Jun 2020, [Online], Available: https://apps.dtic.mil/sti/citations/ADA427228.

37. G. Santhosh Kumar, V. Paul, and K. Poulose Jacob, “Impact of Node Mobility on Routing Protocols for Wireless Sensor Networks,” in *International Conference on Sensors and Related Networks*, pp. 480–485, 2007.

38. A. Varga and R. Hornig, “An overview of the omnet++ simulation environment,” 2008.

39. “Krita.” Accessed 1 Apr 2020, [Online], Available: https://krita.org/en/.

40. K. Furset and P. Hoffman, “High Pulse Drain Impact on CR2032 Coin Cell Battery Capacity,” tech. rep., Nordic Semiconductor, Sep 2011. Accessed 27 Jun 2020, [Online], Available: https://www.dmcinfo.com/Portals/0/Blog%20Files/High%20pulse%20drain%20impact%20on%20CR2032%20coin%20cell%20battery%20capacity.pdf.

41. N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, “BonnMotion-A Mobility Scenario Generation and Analysis Tool,” 2010. Accessed 1 May 2020, [Online], Available: http://sys.cs.uos.de/bonnmotion/.

42. E. J. Allen, P. M. Dechow, D. G. Pope, and G. Wu, “Reference-Dependent Preferences: Evidence from Marathon Runners,” *Management Science*, vol. 63, no. 6, pp. 1657–1672, 2017.

43. P. Veeraraghavan, G. Khomami, and F. P. Fontan, “The relation between the probability of collision-free broadcast transmission in a wireless network and the stirling number of the second kind,” *Mathematics*, vol. 6, no. 7, pp. 1–17, 2018.

44. J. W. Hui and D. E. Culler, “IP is dead, long live IP for wireless sensor networks,” in *SenSys’08 - Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems*, pp. 15–28, ACM, 2008.

45. “Thread Border Router – Kirale.” https://www.kirale.com/products/ktbrn1/. Accessed: 2020-10-12.

46. Notsane0, “nRF52840-DK Range Testing With BLE, ZigBee and Thread Protocols at 0, 4 and 8dBm Transmit Power Settings.” https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/nrf52840-dk-range-testing-with-ble-zigbee-and-thread-protocols-at-0-4-and-8dbm-transmit-power-settings, Dec 2018. Accessed: 2020-11-01.

47. “Thermal noise formulas & calculator.” Electronics Notes. Accessed 2 Oct 2020, [Online], Available: https://www.electronics-notes.com/articles/basic_concepts/electronic-rf-noise/thermal-noise-calculations-calculator-formulas.php.

REPORT DOCUMENTATION PAGE				*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 25–03–2021	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE SIMULATING A MOBILE WIRELESS SENSOR NETWORK MONITORING THE AIR FORCE MARATHON	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Eilertson, Matthew D, Capt	**5d. PROJECT NUMBER** 21G532A
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-MS-21-M-031
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RYAA, 2241 Avionic Cir, WPAFB OH 45433-7765, COMM 937-713-8573, Email: Eric.Lam.3@us.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RYAA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

This thesis explores the feasibility of deploying a mobile Wireless Sensor Networks (WSN) to the Air Force (AF) Marathon in support of Air Force Research Laboratory (AFRL) research of sensor and networking infrastructure in denied or degraded environments. A simulation called MarathonSim is developed in the Objective Modular Network Testbed in C++ (OMNeT++) Discrete Event Simulator to test the performance of a mobile WSN. A full factorial design using numbers of runners, transmission powers, and routing protocols is executed to measure Packet Delivery Ratio (PDR) to a central database, average end-to-end delay of application packets, and average power consumed per mote through the marathon. The experiment results show flood routing delivers >50% of packets for 7 out of 15 trials and >75% for two trials. Average delay varied from 0.11 to 7.2 seconds between 25 runners and 125 respectively. Average power consumed per node increased across all three factors but appears especially sensitive to additional runners. The experiments show it is feasible to deploy a WSN to a marathon under the simulated conditions.

15. SUBJECT TERMS

Wireless Sensor Network, IEEE 802.15.4, Flood, Ad hoc On-Demand Distance Vector, Mobile Mesh Network

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Barry E. Mullins, AFIT/ENG
U	U	U	UU	127	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636, ext 7979; barry.mullins@afit.edu