Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2017

Project Title:	**Location-based Routing Algorithms in Mobile Ad Hoc Networks**
Student:	**Sachin Leelasena**
CID:	**00841878**
Course:	**EEE4T**
Project Supervisor:	**Dr J.A. Barria**
Second Marker:	**Dr A. Gyorgy**

Abstract

This project explores the use of Mobile Ad Hoc networks for crowd control, as a response to events in the local area which have caused disruption to transport links. The aim of the project is to identify the most appropriate routing algorithm to use in this Mobile Ad Hoc network. The movement of people leaving a football stadium after a match is modelled and a network amongst the crowd, between devices with Bluetooth Low Energy connectivity, is simulated. The four routing algorithms simulated have significant differences in their design and include both active and reactive routing protocols, allowing for a thorough performance comparison. Performance is measured through two main metrics: Group Transmission Time (GTT) and power consumption. GTT is defined as the time difference between the first and last relevant nodes in the network receiving the message being propagated. Power consumption is used as a secondary metric to assess the suitability of each algorithm, due to the power limited nature of the mobile devices involved in the network. For dense networks, active routing algorithms are found to have slightly lower GTT in absolute terms, at the cost of significantly higher power consumption, compared to reactive routing algorithms.

Acknowledgements

I would like to thank Dr Barria for his advice and guidance throughout this project.

Contents

1 Introduction

Wireless networks typically refer to infrastructure networks where fixed routers and wired gateways are present [1]. Nodes within these networks connect to the nearest base station to facilitate the transmission of data packets. In Mobile Ad-Hoc Networks (MANETs), no fixed routers are present and all nodes are capable of movement. The nodes within these networks can be connected in an arbitrary manner and the connections can change dynamically. This dynamic nature opens up a large number of possibilities for efficient information transfer in situations where infrastructure networks are not suitable. For example, in the event of a natural disaster where infrastructure networks have been damaged, a MANET can be set up quickly by rescue teams to share information about the rescue operation. The mobility of the nodes allows the MANET to span a larger geographical area than an infrastructure network with the same hardware.

Rescue operations are not the only use for MANETs; the use case presented in this project relates to controlling the movement of large groups of people leaving a football stadium after a match. Crowd control after football matches is usually overseen by local police forces to ensure the safety of the crowd. This is resource intensive and can require a large number of police officers to be involved. The widespread adoption of smartphones means a MANET can be constructed within the crowds to relay information, from the police or other sources, in an efficient manner. The use of a MANET is particularly useful since the large number of people in a small area around the stadium tends to overload cellular networks, so information transfer using infrastructure networks can be difficult.

The aim of this project is to identify a suitable routing protocol for a MANET created between smartphones carried by people leaving a stadium. The network will be used to inform the crowd of local area events which may affect their journey home. Local area events include, but are not limited to, fire alerts at London Underground stations or road closures. This will require the creation of a simulation environment where the performance of a range of routing algorithms can be tested. The simulation environment created is the software deliverable for this project and has been provided in electronic form.

The objective comparison of different routing algorithms requires the use of performance metrics. While a large number of performance metrics exist for MANETs, this project focuses on two main metrics. The use of mobile devices requires the MANET to be power limited, so the performance of the different algorithms will be in part measured by the power consumption. Since the aim of the MANET is to propagate information quickly, another performance metric for the routing protocols will be the time taken to propagate information through the network. The methodology used to obtain these performance metrics are discussed in detail in the Analysis and Design and Implementation sections.

This report summarises the background reading done in the Background section, before outlining the objectives and requirements of the project in the Requirements Capture section. The Analysis and Design section gives a high level overview of the system designed to simulate the different routing algorithms including details of the mobility model. The key features of each routing algorithm along with the design process used are also discussed in the Analysis and Design section. The Implementation section then details the low level design of some of the important or more interesting parts of the simulation system created. The Testing section focuses on the functional correctness testing procedure used to ensure the simulation system produced accurate results. The Results section presents and discusses the results obtained for each routing algorithm simulated before making a comparison between them and selecting the best algorithm. The completion of the objectives outlined in the Requirements Capture section are then analysed in the Evaluation section, along with a discussion of the strengths and weaknesses of the simulation system designed. The Conclusions and Further Works section leads on from this and discusses the success of the project and details some modifications or extensions that can be made in the future. A user guide is provided in this report should the reader want to run or modify the simulations.

2 Background

The main focus of this project is the identification of the most suitable routing algorithm for use in a MANET, so this section contains an overview of a number of routing algorithms. The movement of nodes in the network should accurately model the movement of people walking to places of interest, so some potential mobility models are outlined in this section. A number of potential simulation packages and wireless communication technologies are analysed to find the most suitable ones for this project. A brief review of literature regarding evacuation systems is conducted as some concepts may be relevant to this project.

2.1 Proactive Routing Protocols

Proactive Routing Protocols require each node to maintain routing information to every other node in the network. This routing information may be held in a number of tables which are updated periodically or if the network topology changes. Some commonly used proactive routing protocols will be described in this section.

2.1.1 Destination-sequenced distance vector (DSDV)

The destination-sequenced distance vector (DSDV) protocol produces a single path to the destination based on the distance vector shortest path routing algorithm and guarantees loop free routes. Each routing table entry is tagged with a sequence number so nodes can identify stale routes and thus avoid routing loops being formed [2]. A route with a greater sequence number will be picked first [3]. DSDV uses two types of update packet in order to reduce network overhead, known as "full dump" and "incremental" packets. The full dump packets carries all the available routing information whereas the incremental packets only carry information changed since the last full dump. Incremental packets are sent more frequently than full dump packets [4]. DSDV introduces large amounts of overhead to the network due to the periodic update messages so is unsuitable for a large network since a large amount of bandwith is used in the updating process [4].

2.1.2 Wireless routing protocol (WRP)

WRP also generates loop free routes but requires each node to maintain four routing tables: a distance table, a routing table, a link-cost table and a message retransmission list [5]. This introduces a significant memory overhead at each node in the network so does not scale well for large networks. WRP uses hello messages which are exchanged between neighbouring nodes whenever there is no recent packet transmission. This uses a large amount of power and bandwidth in order to keep all the nodes in the network active [4].

2.1.3 Cluster-head gateway switch routing (CGSR)

CGSR is a hierarchical routing protocol where nodes are grouped into clusters. Each cluster is maintained with a cluster-head which is a node that manages all the other nodes within the cluster. The cluster-head controls the transmission medium and all inter-cluster communications go through it.

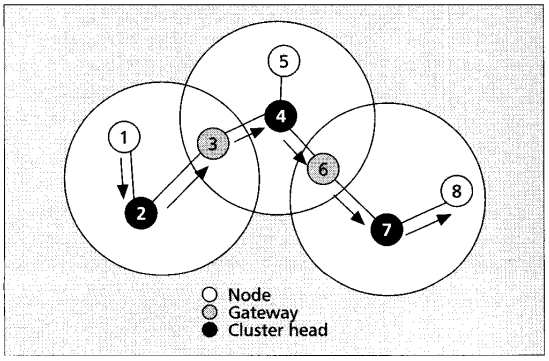


Figure 1: CGSR routing from node 1 to node 8. Figure obtained from [6]

In this routing protocol, each node only maintains routes to its cluster-head, resulting in lower routing overheads compared to flooding routing information through the entire network [4]. However, maintaining clusters incurs a significant overhead since each node needs to periodically broadcast and update its cluster member table [4].

2.1.4 Optimised link state routing (OLSR)

OLSR is a point-to-point protocol based on the link-state algorithm [7]. Each node stores topology information about the network by periodically exchanging link-state messages. OLSR minimises the size of each control message and the number of rebroadcasting nodes used per update by using multipoint replaying (MPR) strategy. During each update, each node selects a set of neighbouring nodes to retransmit its packets. This set is known as the multipoint relays of the node. Any node not in the set can read but not retransmit the packets. To select the MPRs, each node periodically broadcasts a list of one hop neighbours using hello messages [4]. From this list, each node selects a subset of one hop neighbours, which covers all of the original node's two hop neighbours. The MPR set is an arbitary subset of the neighbourhood of a node N, where every node in the two hop neighbourhood of N must have a bidirectional link toward the MPR of N. This allows routes to every destination to be available when transmission begins. A smaller MPR set leads to more optimal routing [7].

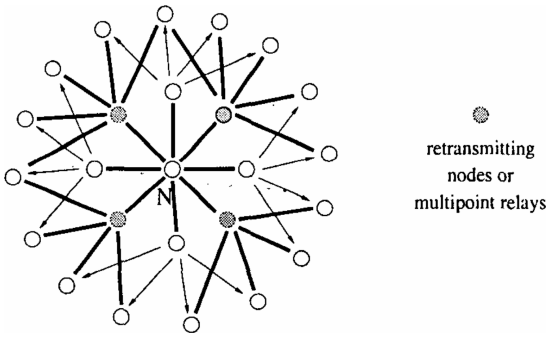


Figure 2: Multipoint Relays for a node N. Figure obtained from [7]

2.1.5 Proactive Routing Summary

Most flat proactive routing algorithms scale poorly due to the high bandwidth and power consumption required for the topology update process. OLSR may scale the best [4] by using multipoint relays to reduce the number of rebroadcasting nodes. This reduces the number of control packets in the network compared to flooding based protocols. CGSR can scale well due to the structure of the network which controls the amount of overhead in the network. This is done by only the cluster-head rebroadcasting control information. However, a highly dynamic network will introduce a lot of overhead to the network (cluster formation and maintenance), making CGSR unsuitable [4].

2.2 Reactive Routing Protocols

Reactive routing protocols aim to reduce overheads and network control traffic compared to proactive routing protocols by determining routes only when required.

In a source routed protocol, the packets contain the source and destination addresses with each intermediate node forwarding the packets according to the information in the header. This removes the need for the intermediate nodes to maintain routing information for each active route. The downside to this is that the protocol doesn't scale well. In a large network, the number of intermediate nodes is large so the probability of link failure can be high. Additionally, increasing the number of intermediate nodes in a route increases the the overhead of each packet as the header is larger.

In hop-by-hop or point-to-point routing, each packet contains the destination and next hop address. This requires the intermediate nodes to use their routing tables to forward the packets to the destination. An advantage of hop-by-hop routing is that the routes adapt to the dynamic nature of the network as

the routing tables can be updated as nodes move. This allows more efficient routes to be used and fewer route recalculations are required. A disadvantage of hop-by-hop routing is the need for each node to store routing information for each active route and beaconing messages may be required to update routing tables based on the behaviour of neighbouring nodes [4].

2.2.1 Dynamic source routing (DSR)

In the DSR protocol, each packet carries the address of every hop in the route from source to destination. This limits the effectiveness of the protocol in large networks due to the large overhead required in each packet. However, DSR may outperform other protocols such as AODV in small to moderately sized networks [4]. Each node can store multiple routes in its route cache, allowing the source node to check for a valid route before initiating route discovery. DSR can therefore be very efficient in a network with low mobility as the routes will be valid longer. DSR does not require periodic beaconing (hello messages) so nodes can enter sleep mode to conserve power which also saves bandwidth [4].

2.2.2 Ad Hoc On-Demand Distance Vector (AODV)

AODV is based on the DSDV protocol described in section 2.1.1. The periodic updates and sequence numbering procedure from DSDV are used but the number of broadcasts are reduced by the routes being created on-demand. Route requests are forwarded to neighbours until the destination or an intermediate node with a route to the destination is reached [8]. AODV uses a similar route discovery procedure as DSR but the packets only carry the destination address instead of the route, giving a lower routing overhead than DSR. The route replies in AODV only carry the destination IP address and the sequence number instead of the address of every node in the route as in DSR. The AODV protocol is adaptable to highly dynamic networks but delays may be experienced during route construction and link failures which trigger route discovery [4]. AODV only works in a network with symmetric links, which can be hard to achieve in a mobile network, whereas DSR can be used with asymmetric links. DSR is also more resilient than AODV when it comes to link failure due to storing multiple routes in the route cache [8].

2.2.3 Light-weight mobile routing (LMR)

The LMR protocol uses flooding to discover its routes. Each node maintains multiple routes to each destination which increases the reliability by allowing nodes to select the next available route without initiating route discovery [4]. Additionally, each node only maintains routing information to their neighbours, avoiding the delays and storage requirements for maintaining complete routes. However, this introduces the possibility of invalid routes which leads to delays in determining a correct route [4].

2.2.4 Temporally ordered routing algorithm (TORA)

The TORA protocol is a highly adaptive loop-free distributed routing algorithm based on the concept of link reversal [6]. During route creation and maintenance, nodes use a "height" metric to create a directed acyclic graph (DAG) rooted at the destination. Links are then assigned a direction (upstream or downsteam) based on the relative height of neighbouring nodes. In this protocol, nodes maintain routing information about adjacent nodes, meaning only a few control messages where the network topology has changed is required to update the routing. This makes TORA suitable for highly dynamic networks with dense nodes. TORA relies on timing as the "height" metric is dependent on the logical time of a link failure and the protocol assumes all nodes have synchronised clocks, such as from the Global Positioning System [6].

2.2.5 Associativity-based routing (ABR)

ABR uses a query-reply technique for route discovery, but route selection in ABR is primarily based on stability. Each node maintains an associativity tick with its neighbours and links with higher associativity are selected for the route [4]. While this may not lead to the shortest route, the route tends to last longer so fewer route rediscoveries are required and more bandwidth is available for data transmission. However, periodic beaconing is required so nodes cannot sleep to reduce power consumption. Another disadvantage is the lack of multiple routes or a route cache so route discovery is required after link failure [4].

2.2.6 Signal Stability Adaptive (SSA)

SSA descends from ABR but routes are selected based on signal strength and location stability rather than an associativity tick. As with ABR, SSA may not pick the shortest route to the destination, but each route will last longer meaning fewer route rediscoveries are required. Unlike DSR and AODV, intermediate nodes in SSA cannot reply to route requests so route discovery can cause long delays. SSA also makes no attempt to repair routes where link failure occurs [4].

2.2.7 Relative distance micro-discovery ad hoc routing (RDMAR)

RDMR minimises routing overheads by calculating the distance between source and destination nodes and limiting each route request packet to a certain number of hops. This allows route discovery to be limited to a small region of the network. The same technique is used for route maintenance when links fail which conserves bandwidth and battery power [4]. RDMR does not require location aided technology such as a GPS to determine routing patterns, but the relative-distance discovery process can only occur if the two nodes have previously communicated. If no previous communication has occurred, the protocol will behave in the same manner as flooding algorithms where route discovery has a global effect [4].

2.2.8 Cluster-based routing protocol (CBRP)

In CBRP, nodes are organised in a hierarchy by grouping them into clusters. As with CGSR, each cluster has a cluster-head which manages data transmission within and between clusters. Since only cluster heads exchange routing information, the control overhead transmitted through the network is significantly lower than flooding methods. However, the hierarchical aspect of the protocol means there is an overhead associated with cluster formation and maintenance [4]. A disadvantage of CBRP is the possibility of temporary routing loops. This can occur due to nodes carrying inconsistent topology information due to long propagation delays. This protocol is therefore suitable for medium size networks with slow to moderate mobility and is likely to perform best when nodes within a cluster remain together [4].

2.2.9 Reactive Routing Summary

Most reactive routing protocols follow similar route discovery and maintenance procedures so will have roughly the same routing cost for a worst-case scenario. For example, RDMR has the same cost as a flooding algorithm if the source and destination nodes have not previously communicated, so is unlikely to be efficient in a highly dynamic network. This will be the case when nodes join the network, but they become more aware of their surroundings as they remain part of the network. DSR initiates a network wide flooding route discovery method when routes expire in the route cache but RDMR uses a more localised route discovery procedure to reduce overhead. Control overhead can also be reduced by picking routes based on stability as ABR (associativity tick) and SSR (signal strength) do. While this may not result in the shortest routes, the increased stability should reduce the frequency of route discovery as links are less likely to fail. This can lead to better performance than shortest path protocols such as DSR but may scale badly since each packet carries the full destination address and the probability of link failure increases with network size. CBRP reduces control overhead by splitting the network into clusters. During route discovery, cluster-heads exchange information rather than every intermediate node; this results in a significant reduction in control overhead compared to flooding algorithms. However, CBRP may have a large amount of processing overhead in a highly mobile network related to cluster formation and maintenance [4].

2.3 Hybrid Routing Protocols

Hybrid routing protocols are both proactive and reactive and attempt to combine the best features of each class of protocol. Hybrid protocols are designed to increase scalability by allowing nodes in a local area to form a backbone to reduce route discovery overheads. This can be achieved by proactively maintaining routes to nearby nodes and using route discovery to determine routes to far away nodes. Most hybrid protocols are zone-based where the network is split into different zones for each node [4]. Nodes may also be grouped into trees or clusters.

2.3.1 Zone routing protocol (ZRP)

In ZRP, each node has a routing zone which defines a range where network connectivity needs to be maintained proactively. For nodes within the routing zone, routes are immediately available. For nodes outside the routing zone, routes are determined reactively using any on-demand routing protocol. ZRP significantly reduces the amount of communication overhead compared to purely proactive protocols [4]. Delays are also reduced by allowing routes to be discovered faster. This speed increase occurs because in order to determine a route from within the routing zone to a node outside it, the routing only needs to travel to a node on the boundary of the destination node's routing zone. This boundary node proactively maintains routes to the destination node so can send a reply to the source with the required routing address in order to complete the route. If the routing zone is large, ZRP can behave as a purely proactive protocol while if the zone is too small, it behaves like a purely reactive protocol so the optimal zone size needs to be determined [4].

2.3.2 Zone-based hierarchical link state (ZHLS)

ZHLS uses a hierarchical structure made up of a node level topology and a zone level topology. The network is divided into non-overlapping zones where each node has a node and zone ID calculated using a GPS. No cluster-head is used to coordinate the data transmission so there is no processing overhead compared to the CGSR protocol. There is also a reduction in communication overheads compared to reactive routing protocols. When the source and destination nodes are in different zones, the source node broadcasts a zone-level location request to the other zones, resulting in significantly lower overhead compared to flooding used in reactive protocols. ZHLS is adaptable to dynamic topologies since only the node and zone IDs of the destination are required for routing, as long as the destination does not move to another zone. This may allow the protocol to scale well to large networks. A disadvantage of ZHLS is that each node must have a preprogrammed zone map so the protocol is not suitable for networks where the geographic boundary is dynamic [4].

2.3.3 Scalable location update routing protocol (SLURP)

SLURP is similar to ZHLS in that nodes are organised into non-overlapping zones. SLURP reduces the cost of maintaining routing information by removing global route discovery by assigning a home region for each node. The home region is determined using static mapping:

$$f(\text{NodeID}) \mapsto \text{regionID} \tag{1}$$

$$f(\text{NodeID}) = g(\text{NodeID}) \mod K \tag{2}$$

Equation 2 shows an example of a mapping function where g(NodeID) is a random number generator using the node ID as the seed and K is the total number of home regions [9]. Since each node has a constant node ID, the function will always calculate the same home region. Each node unicasts a location update message towards its home region which is then broadcasted to all nodes in the home region. To determine the location of any node, each node can unicast a location discovery packet to the home region of the node in question. The source can then send data towards the destination once the location has been discovered. When packets reach the region of the destination node, source routing using DSR gets the packet to the destination node [9]. A disadvantage of SLURP is that it also relies on a preprogrammed zone map like ZHLS.

2.3.4 Distributed spanning trees based routing protocol (DST)

DST groups nodes in a network into trees which contain root nodes that control the tree structure and internal (regular) nodes. Nodes have three states; router, merge and configure and will be in one of the states depending on the task being performed [8]. Route determination can be done by hybrid tree flooding (HTF) or distributed spanning tree shuttling (DST). In HTF, control packets are sent to neighbours and adjoining bridges in the tree whereas in DST, control packets are broadcasted from the source along the tree edges. When control packets reach leaf nodes, they are sent up the tree until it reaches a certain height (shuttling level). Once this level is reached, the control packets can be send down the tree or to adjoining bridges. The main disadvantage of DST is the reliance on the root node to configure the tree as this is a single point of failure [8].

2.3.5 Hybrid Routing Summary

Hybrid routing protocols can scale better than purely proactive or reactive protocols by attempting to minimise the number of rebroadcasting nodes through defining a structure to the network. The structure allows nodes to work together to optimise the route discovery process. Collaboration between nodes can help to maintain routing information for longer, reducing or even eliminating the need for flooding. Hybrid routing protocols also attempt to remove single points of failure and bottlenecks in the network. This can be done by allowing any number of nodes perform routing or data forwarding if a preferred link fails [4].

2.4 Location-based Routing Protocols

2.4.1 Distance routing effect algorithm for mobility (DREAM)

In the proactive DREAM protocol, each node knows its location coordinates using GPS. Nodes periodically exchange coordinates and store them in a location table. Exchanging location information consumes significantly less bandwidth than exchanging link state or distance vector information so it scales better than other protocols [4]. The routing overhead is also reduced by making the update message frequency proportional to mobility and the distance effect. This means static nodes don't need to send update messages.

2.4.2 Location-aided Routing (LAR)

The reactive LAR protocol attempts to reduce the routing overheads in flooding algorithms by using location information. The protocol assumes each node knows its location through GPS. The routing can then be determined by defining a boundary where the route request packets can travel to reach the destination [4]. Alternatively, the route request packets can store the GPS coordinates and only travel in the direction where the distance to the destination decreases. Both methods conserve bandwidth by limiting the control overhead transmitted in the network. In most cases, this will determine the shortest path from source to destination. A disadvantage of LAR is that each node requires a GPS and the protocol may behave similarly to DSR and AODV in highly mobile networks [4].

2.4.3 Geocast Adaptive Mesh Environment for Routing (GAMER)

Geocasting is a variation of multicasting, where the goal is to deliver packets to a group of nodes in a geographical area known as the geocast group [10]. Source routing, forwarding zones and meshes are used in the protocol. Source routing is used to route geocast packets where each packet carries the full route in its header. Forwarding zones are used to route geocast packets by flooding packets in the forwarding zone rather than the entire network, similar to DREAM. GAMER uses a mesh "to establish multiple paths between a source node and geocast regions" [10]. The GAMER protocol is able to dynamically adjust the density of the mesh; high mobility nodes create a dense mesh, low mobility nodes create a sparse mesh. The close proximity of geocast group members means it is less costly to establish redundant paths to a geocast region than to a multicast region. The mesh is created by flooding JOIN-REQUEST packets to the mobile nodes in the geocast region via a forwarding zone which dynamically changes in size which changes the density of the mesh [10]. GAMER has better transmission accuracy compared with non-adaptive, mesh-based geocast protocols [10].

2.4.4 Two-hop Routing

Static wireless sensor networks (WSN) can use 2-hop neighbour knowledge to minimise the number of forwarder nodes in a network to reduce energy consumption [11]. Each forwarder node selects a neighbour node which has the largest number of neighbours and is closest to the destination D to guarantee packet delivery. The forwarding mechanism can be improved using the average distance from two-hop neighbours to D and the probability related to the average [12].

In the TN-CMAD algorithm [12], the forwarder node computes the progress of its one-hop neighbours and the average distance from two-hop neighbours to D. The forwarder node then selects the neighbour node with the minimum two-hop distance from D as the next sender.

In the TN-CRDN algorithm [12], the forwarder node calculates a probability index which reflects the one-hop distance to D and the number of two-hop neighbours. The closest one-hop neighbour of the forwarder node will have the lowest probability p_{avg}. The one-hop neighbour with the most neighbours will have the highest probability p_n. The overall probability index of a neighbour v_x is determined as follows with the node with the highest probability index being selected as the next hop:

$$Pr(v_x) = (1 - p_{\text{avg}}(v_x))p_n(v_x) \tag{3}$$

These two-hop routing algorithms reduce the network overhead and have lower end-to-end delays, while TR-CDN also reduces the number of hops compared to GAMER [12].

2.4.5 Greedy Perimeter Stateless Routing (GPSR)

GPSR uses router position and the destination of a packet to make packet forwarding decisions [13]. GPSR is greedy as it selects the next hop as the node closest to the destination of the packet until the destination is reached. GPSR maintains routing information through the use of periodic beaconing which provides all nodes with positions of their neighbours. If no beacon is received for longer than a time-out period T, the node is assumed to have failed or left the network. An advantage of greedy forwarding is that it only relies on knowledge of immediate neighbours, which can be significantly less than the number of nodes in the entire network. However, only using the position of neighbouring nodes can require packets to temporarily move further from the destination.

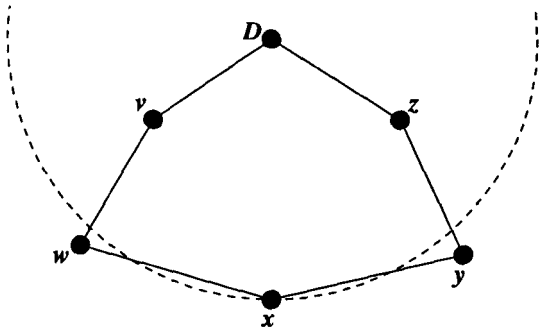


Figure 3: Greedy forwarding failure. x is a local maximum in geographic proximity to D; w and Y are farther from D. Figure obtained from [13]

In this scenario, node x sees a void where no neighbours exist and the destination D is on the other side of the void. In order to route to D, node x employs the right-hand rule to route around the void to a node closer to D than x.

2.5 Mobility Models

A number of different mobility models can be used to simulate the motion of nodes in a MANET. Mobility models typically describe the velocity and direction of nodes amongst other parameters. This section explores some mobility models that may be suitable to describe the motion of people.

2.5.1 Random Walk

In the random walk model, nodes move from their current position to their next position with a fixed velocity and random direction. Subsequent movements are independent of each other i.e. the model is memoryless [8]. At every time interval, the node takes one of eight possible directions, each of which have equal probability. While this is useful for modelling irregular movement, the scenario for this project is concerned with more regular movement of people towards points of interest, so a random walk mobility model is not suitable for a very accurate simulation.

2.5.2 Random Waypoint Model

The random waypoint model (RWP) is one of the most popular synthetic mobility models. Each node in the model moves towards the next waypoint with a constant velocity v, where $v \sim U(0, V_{max})$. The

model also allows for 'thinking times' when nodes reach a waypoint. In the RWP process, consecutive legs are not independent as they share a common waypoint [14]. Additionally, the RWP process is 'time reversible' as a path along waypoints P_0, P_1, ..., P_n is equally likely as the time reversed path P_n, P_{n-1}, ..., P_0 [14]. Once a node reaches a waypoint, it travels in a new direction with probability P or remains in the current direction with probability 1 - P [8]. While this mobility model may be more appropriate than the random walk model for most cases, it is unlikely to be useful for this project as nodes need to move towards fixed points to simulate people walking towards points of interest.

2.5.3 Shortest Path Map Based Movement (SPMBM)

The shortest path map based movement model is a more advanced version of the map based movement (MBM) model. The MBM model places nodes randomly in the map area and moves them along path segments until the end of the road where they turn back, or until an intersection is reached. At an intersection, nodes randomly select a new direction but do not move backwards [15]. SPMBM also starts with nodes in random locations, but selects a certain destination for all nodes and finds the shortest path to the destination using Dijkstra's shortest path algorithm [15]. The map data can contain Points of Interest (POIs) which have a higher probability of being chosen as a node destination than regular places on the map. This mobility model is promising for use in this project as it will give a good approximation for the movement of people towards POIs such as underground stations and bus stops around a stadium. The ONE simulator supports the use of maps which can be used by the SPMBM model to simulate the movement of people.

2.6 Simulation Packages

A range of simulation packages have capabilities to model MANETs, some of which would be more suitable to use than MATLAB for this project.

2.6.1 Opportunistic Network Environment (ONE) simulator

The ONE simulator is designed to simulate delay tolerant networks (DTNs). The software package is written in Java and includes movement modeling, routing simulation, visualization and reporting [15]. The simulator has a number of movement models built in, including the Shortest Path Map Based Movement (SPMBM) model and also allows for mobility traces to be imported from external sources. The movement of nodes towards points of interest (POIs) is supported. The simulator has a GUI where node movement and routing behaviour can be observed, along with a reporting module which allows data about the simulation to be measured. This simulator would be useful for this project if it could be modified to simulate conventional MANET router types.

2.6.2 ns-3

ns-3 is a network simulator for internet systems written in C++ with optional Python bindings. The simulation package provides models for packet data networks and a simulation engine. A lot of documentation and tutorials for ns-3 appear to be available including resources for MANET simulation. However, installation issues have prevented testing of the software.

2.6.3 OMNeT++

OMNeT++ is a C++ simulation library and framework for building network simulators. A number of simulation frameworks already exist which cover MANET protocols and mobility, such as the INET framework. Open source frameworks specifically for MANETs also exist so this simulation package looks promising, but the lack of familiarity with the package is a significant disadvantage compared to MATLAB.

2.6.4 MATLAB

While MATLAB is not specifically a network simulator, a large number of toolboxes are available which can be used to simulate networks. A large number of open source resources also exist for MATLAB which may be useful in the implementation stage of the project. Given the time requirements of the project, the use of a familiar simulation package is likely to be the best course of action. The MATLAB scripts accompanying the MSc thesis by L. Feng [8] have been analysed and the simulation procedure seems

relatively straightforward. Although the mobility model and algorithms for this project are different, a similar simulation framework can be used.

2.7 Wireless Communication Technologies

The nodes in the MANET will need to transmit packets between themselves using existing wireless communication technologies as there is likely no benefit to designing a new protocol for this use case. Designing a new technology is also outside the scope of this project. The main aspects of the technologies to be used are the transmission range, transmission speed and power consumption to make the simulation as accurate as possible. This section will explore the wireless communication technologies available in current smartphones as these devices are the most suitable to build a MANET amongst a crowd of people leaving a stadium. Near Field Communication (NFC) was not considered due to the need for close proximity of devices to transfer information (< 10 cm [16]).

2.7.1 Bluetooth

Bluetooth Low Energy (BLE or Bluetooth 4.x) is a widely used variant of the Bluetooth standard with a range of up to 50m [17]. The technology was developed by the Bluetooth Special Interest Group as a low-power solution so appears suitable for this project. Modern mobile phones have BLE connectivity which allows for the creation of a concentrated network with a large number of nodes. Smartphones typically contain Class 2 Bluetooth devices which have a maximum range of 10m [16]. BLE devices used in Wireless Sensor Networks were found to have a power consumption of 15mW when in transmission mode by Nair et al. [18]. Bluetooth 4.0 enables high-speed operation up to 24 Mbps [16]. While Bluetooth devices generally require pairing before data can be transferred, it is possible to transfer data through the advertising channels only, bypassing the pairing requirement [19]. Therefore, the simulation in this project will make the assumption that pairing is not required to transmit data in the MANET. Schrader et al. investigated the power consumption for the advertising state of BLE devices and found a peak power consumption of approximately 40mW [20].

2.7.2 Wi-Fi Direct

Wi-Fi Direct builds on the IEEE 802.11 infrastructure mode and allows devices to negotiate which will perform the access point (AP) like functionalities [21]. This lets Wi-Fi Direct inherit Quality of Service (QoS), power saving and security mechanisms from the Wi-Fi infrastructure mode. Wi-Fi Direct also has the same range as conventional Wi-Fi connections of up to 200m [22], far greater than the 10m range for Class 2 Bluetooth devices. Wi-Fi Direct is supported by fewer mobile devices than Bluetooth as iOS devices have a proprietary version instead which combines Wi-Fi and Bluetooth connectivity [23]. Using Wi-Fi Direct could produce better performance of the MANET in terms of transmission speed as the network is more sparse, but the simulation would be less realistic; this suggests the use of Bluetooth is better.

2.8 Evacuation Systems

A number of studies focus on the use of MANETs in disaster situations to find survivors or to facilitate evacuations. While these works cover a slightly different area to this project, some overlap exists.

2.8.1 Emergency Rescue Urgent Communications - Evacuation Support System

Okada et al. [24] propose the Emergency Rescue Urgent Communications - Evacuation Support System (ERUC-ESS). This system aims to identify a disaster outbreak point and identify evacuation paths. Each mobile terminal determines its position through the use of GPS or a Local Positioning Service (LPS). These terminals monitor the movement of people communicate with each other using a MANET. The state information exchanged by the terminals allows the disaster outbreak point to be determined. Once this has occurred, appropriate escape routes are searched and a graphical representation for available/unavailable routes are given for evacuation [24]. Each terminal contains two types of map, an Initial Area Map (IAM) and a Post-disaster Area Map (PAM). The PAM is used to find evacuation routes, making use of Dijkstra's algorithm to do so [24]. The experimental results presented suggest the use of ERUC-ESS increases the effectiveness of an evacuation.

2.8.2 Emergency Rescue Evacuation Support System

Hayakawa et al. [25] propose a similar system called Emergency Rescue Evacuation Support System (ERESS). ERESS aims to identify the outbreak of a disaster, estimate the type of disaster and find the most appropriate evacuation path [25]. ERESS also makes use of the Emergency Rescue Urgent Communications (ERUC) system mentioned by Okada et al. [24]. Mobile terminals are equipped with a number of sensors (acceleration, angular velocity etc.) and exchange information with each other through the use of a MANET [25]. A Support Vector Machine (SVM) is used to identify the type of disaster before evacuation routes are determined through the use of Dijkstra's algorithm.

2.9 Background Summary

From the background reading, the most appropriate wireless technology to simulate for this project is Bluetooth Low Energy due to the high adoption rate. The routing protocols to simulate have been identified as Distance routing effect algorithm for mobility (DREAM), Light-weight mobile routing (LMR) and Location-aided Routing (LAR). DREAM was identified due to the low or variable routing overhead which can allow message transmission to be fast. The GPS aspect for both DREAM and LAR is easy to achieve as smartphones usually have GPS and other location functionality. LMR was identified to allow the performance of a flooding based protocol to be tested. Simulating LAR will allow a performance comparison between reactive and proactive location-based routing algorithms.

The most suitable mobility model identified from the background is the shortest path map based movement (SPMBM) model as it can model the movement of people leaving a stadium quite accurately.

While MATLAB may not be the most suitable simulation package in terms of the built in functionality for modelling MANETs, the familiarity with the package is a significant advantage over the other packages outlined in this section. This familiarity along with the time constraint of the project means that MATLAB is the most suitable simulation package to use.

Bluetooth Low Energy will be the wireless communication technology modelled in the simulations due to the widespread support for BLE compared with limited support for Wi-Fi Direct on mobile devices. A peak power consumption of 50mW while transmitting will be used in the simulations as this value is slightly higher than the results of Schrader et al. [20]. This will mean the power consumption results are an overestimate rather than an underestimate.

3 Requirements Capture

The main deliverable for this project is a set of MATLAB scripts used to simulate a MANET amongst people leaving a football stadium after a match. The purpose of the scripts is to identify the most suitable routing algorithm for use in the MANET. Clearly, an objective method of identifying the most suitable algorithm is required. Since the purpose of the MANET is to facilitate the transmission of information quickly within the network, a good performance metric is the time taken for the information to propagate. In this report, **Group Transmission Time** (GTT) will refer to the time taken for all the relevant nodes in the MANET to receive the information. The GTT will be calculated as the time difference between the first and last relevant nodes in a network receiving the information being propagated. A relevant node is one that is heading towards, or in the vicinity of, the local area event being simulated.

If the GTT results for multiple routing algorithms are similar, a secondary performance metric will be required to identify the most suitable algorithm. Since the MANET relies on mobile devices, power consumption is an important consideration. The power consumption of the nodes in the MANET should therefore be measured as part of the simulations.

To allow a comparison based around the performance metrics outlined above, a minimum of two routing algorithms must be implemented and tested. A larger number of algorithms would produce a better result for the most suitable routing algorithm, but the time constraints of the project may limit the analysis of the algorithms if too many are implemented. Similarly, different types of routing algorithm should be implemented and tested to find the most suitable one. To meet this requirement, at least one active and one reactive routing algorithm must be implemented and tested. This requirement will strengthen the choice of best routing algorithm as a good range will have been investigated.

The location based routing algorithms tested should accurately model existing wireless communication technologies where possible. The main aspects that must be modelled accurately are the transmission range and throughput of the wireless communication technology. Section 2.9 identified Bluetooth Low Energy (BLE) as the most appropriate technology to simulate, so the transmission range and throughput of BLE must be accounted for in the network simulations. In this context, throughput refers to the number of transmissions a node can make in a second. The power consumption of BLE could be implemented accurately, but this is a lower priority as power consumption is a secondary performance metric for this project.

The results obtained from the simulations can only be considered to be valid if the mobility model used is accurate to some degree. The most suitable mobility model identified in section 2.9 is the Shortest Path Map Based Movement (SPMBM) model. This model should be implemented as accurately as possible to produce valid results from the simulations. The key aspect of this mobility model is the use of Dijkstra's shortest path algorithm, so this will have to be implemented in the simulations. An existing implementation built in to MATLAB should be used if available. An accurate graph of the simulation area, in terms of distances between nodes, will have to be constructed to allow Dijkstra's algorithm to be used.

3.1 Objectives

Based on the above requirements, a few key objectives have been determined. These objectives will be used to evaluate the success of the project:

1. Simulate people leaving a football stadium and heading towards nearby public transport hubs through the use of a realistic mobility model.
2. Simulate a minimum of two routing algorithms in a MANET created between smartphone users in a local area. This should include at least one active and one reactive routing algorithm.
3. Measure the speed of packet transmission in the MANET. Group Transmission Time (GTT) will be defined as the time difference between the first and last relevant nodes in the network receiving the message being propagated.
4. Identify the most suitable routing algorithm for use in the MANET based on the performance metric of GTT. Power consumption should also be used where appropriate to identify the best routing algorithm.

3.2 Specification

This project **must**:

- Simulate a MANET created using mobile devices with BLE connectivity. The MATLAB scripts used for the simulation of the network are a deliverable for this project
- Simulate the movement of people in a large crowd leaving a football stadium after a match
- Implement and simulate a minimum of two routing algorithms
- Objectively measure the performance of the implemented routing algorithms using GTT
- Analyse the results collected from the simulations and give a recommendation on the most appropriate routing algorithm for use in this network

This project **should**:

- Implement a minimum of one active and one reactive routing algorithm
- Implement the SPMBM mobility model to model the movement of people leaving a stadium after a football match

This project **could**:

- Objectively measure the performance of the implemented routing algorithms using the power consumption of the network
- Accurately model the power consumption of transmission via BLE

4 Analysis and Design

The system designed in this project simulates the MANET in question using a number of different routing algorithms. This section will give a high-level overview of the system design, along with how the system design changed over the course of the project. The design choices considered along with the strengths and limitations of the system will also be discussed.

4.1 Overview

The overall system is designed to simulate a MANET where the nodes are mobile devices with BLE connectivity. These mobile devices will be carried by people leaving a stadium, so the mobility of the nodes in the network is related to the mobility of the crowd. The purpose of the system is to identify the best routing algorithm for use in the MANET, so a minimum of two routing algorithms are required, as per the project objectives. Although more functional network simulators were considered, MATLAB was chosen as the simulation package for this project. The main reasons behind this choice were the flexibility and the familiarity of the package, as highlighted in section 2.9.

The initial design analysis stage highlighted two potential design routes: designing a simulation system from scratch or using existing tools. The use of existing tools has a lower initial cost in terms of development time, but as the system design becomes more refined, the limitations of the tools may hinder development. The specific nature of the simulation required meant that designing a system from scratch was determined to be the best course of action, even when the higher initial development time was considered. Over the course of the project, this decision has allowed for specific routing algorithms to be implemented exactly as required. However, issues have sometimes been slow to fix, due to the lack of resources caused by the specific nature of this bespoke system.

At the initial design stage, the two main components of the system were identified as the mobility model and the routing algorithms. The first design stage was chosen to be around the mobility model, since only one model was required. Initially the plan was to design and implement the mobility model before integrating the routing algorithms into the simulation. However, this approach was not deemed to be suitable for the project as each routing algorithm would have to be integrated into the simulation separately. A more modular approach was pursued, so the same mobility model implementation could be used for each routing algorithm. This had the dual benefits of saving time on the implementation of the system, along with simplifying the correctness testing procedure.

The high-level system flow is described in figure 4.

Figure 4: System high-level design. The initialising step is only done once while the transmission and movement steps loop throughout the simulation period.

The simulation initialisation step sets a number of parameters, including the number of nodes in the simulation, the simulation time and relevant distances for the mobility model. The initial position of the nodes are set, along with information about the local area event being simulated. The packet transmission step is where the main benefits of a modular design approach can be seen. Each routing algorithm was designed to appear as a black box to the rest of the simulation, so different algorithms can be simulated using the same framework. The node movement step causes each node in the simulation to move to a new position if required, based on the SPMBM mobility model. The simulation initialisation only occurs once and the packet transmission and node movement steps loop to simulate the network for a specified amount of time.

The modular approach taken allowed the mobility model to be designed, implemented and tested independently of the routing algorithms. This greatly sped up the design and implementation of the simulation system as it was broken down into smaller sections, where errors in one section did not have a

knock-on effect on others. A disadvantage of this modular approach arose when the different sections of the simulation had to be integrated. During the independent development of the sections, the main focus was on designing components that worked in a system and not much thought was given to optimising the components to work well together. Integration testing took longer than expected as some unexpected behaviour required line by line debugging to identify the cause of a bug. Once the first few components were integrated into the overall system, the design approach used changed to focus more on optimising the interaction between the different components, in order to save time as the project progressed.

4.2 Mobility Model

The Shortest Path Map Based Movement (SPMBM) mobility model was the first component of the system to be designed. Most of the information about this mobility model was found in the work of A. Keranen [15]. The most important aspect of this model is the use of Dijkstra's shortest path algorithm. Since Dijkstra's algorithm is widely used for solving shortest path problems, MATLAB has built in functions that were used in the implementation of the mobility model. A custom implementation would take longer to develop and is likely to be less accurate, so the choice was made to use an existing implementation within MATLAB.

The use of Dijkstra's algorithm requires the generation of a graph, with non-negative values for each edge in the graph, to represent the distance between graph nodes (waypoints) in the simulation area. For this project, an accurate mobility model is required, so accurate distances were required for the graph. Accurate distances require a specific stadium to be used in the simulation area, so Wembley Stadium in North-West London was selected due to the number of nearby transport links. Distances for the graph edge weights were obtained from Google Maps using the *measure distance* tool. This gave a simulation map size of approximately 400 m x 1000 m, but only parts of the area corresponding to roads and paths are accessible to the nodes in the simulation. A slight limitation of this method is that straight line approximations were used for some of the measurements, but the discrepancy in the measurements will not significantly affect the results of the simulation. The decision to use Google Maps distance data was made as it is not feasible to measure the required distances in person.

Once the relevant distances were obtained for the edge weights, the shortest path between two nodes in the graph was obtained. This path was then split into a number of steps where each step corresponds to a time slice. This design choice was made to allow the position of a given node at a given time in the simulation to easily be determined. This choice simplified both the testing of the mobility model and the design of the routing algorithms. The process of splitting the overall path into smaller steps was designed to somewhat account for human walking patterns. Since people don't walk at a constant speed, a random speed for each step is selected from within a specified interval. The upper and lower bounds for the interval were determined from works about preferred walking speeds, including the work of Browning et al. [26] and Samson et al. [27]. This led to a lower bound of 0.6 m/s and an upper bound of 1.8 m/s to account for individual differences in walking speeds. While this adds realism to the simulation, testing and the comparison of results is more complicated as no two simulation runs will be the same. The benefits of the more realistic mobility model were determined to outweigh the drawbacks, so this design choice was kept. The difficulty in comparing results between simulation runs was overcome by running each simulation 10 times and comparing the mean and median results.

The mobility model actually fits into two sections of the high-level design seen in figure 4: the simulation initialisation step along with the node movement step. At the start of the simulation, the position of each node at each time slice is calculated and stored, corresponding to how people will have an idea of the route they will take to public transport hubs in advance. At the node movement stage, the position of each node in the simulation is updated to the next position in the pre-determined route.

If a message packet is received by a node, it reacts by calculating a new route to avoid the local area event. This recalculation step presented an interesting design challenge, as the message packets could be received at any position within the simulation, whilst only waypoints are considered in the graph. One option was to expand the graph to include every position in the simulation area, but this would have been extremely computationally intensive and difficult to implement, due to the size of the area used for the simulation (approx. 400,000 m^2). Another option was to change the path of the node so it moves to the nearest waypoint, before using Dijkstra's algorithm to determine the shortest path to a

new public transport hub. This method was chosen since it is far less computationally intensive, but it does have limitations. Moving towards the nearest waypoint may not be realistic due to safety concerns, for example if the local area event is a terrorist attack such as the one at the Manchester Arena in May 2017. This reduction in the realism of the mobility model was determined to be an appropriate trade-off due to the significant reduction in the complexity of the model.

A limitation of the mobility model exists as the density of nodes is not considered. No limit is imposed on the number of nodes that can be present in a given area, which can potentially lead to unrealistic node densities when considering the movement of people. The use of random velocities within a specified interval means the probability of multiple nodes occupying the same space is low, but this probability increases with the number of nodes in the simulation. The Green Guide to Safety at Sports Grounds [28] indicates a density of 47 people per $10m^2$ as the maximum density allowed within various parts of a stadium. Considering the simulation area of approximately 400,000 m^2, the density of nodes in the simulation is highly unlikely to exceed 50 people per $10m^2$ unless an unreasonably high number of nodes are used in the simulation.

However, ignoring the density of nodes in the simulation may still reduce the realism of the mobility model, due to relationship between the density of people in a crowd and the velocity of the crowd. The works of R.A. Smith [29] and Fang et al. [30] investigate the link between crowd density, flow rate and walking speeds. Fang et al. show there to be a logarithmic relationship between density and velocity, but the mobility model has not been designed to account for this relationship. The decision to ignore this relationship for this project was made to reduce the complexity of the mobility model, as the main aim of the project relates to the routing algorithms used in the simulation. Increasing the accuracy or realism of the mobility model could be part of a future work.

4.3 Routing Algorithms

The second major component of the system is to do with simulating packet transmission via different routing algorithms, to find the most suitable one for the given scenario. At the outset of the design stage, two approaches were identified for the routing algorithms: treat each node in the network as independent objects that react to external stimuli or model the interaction of nodes stochastically. The 'independent object' approach was selected as it allows for a more realistic simulation of the network. The use of stochastic models could have produced significantly different results, leading to invalid conclusions being drawn from this project. Each algorithm followed a common framework and the different behaviours were designed to fit around this framework. The routing algorithms simulated in this project were selected to have a range of different features, to allow for a thorough performance comparison. The routing algorithms are only simulated for transmission between moving nodes in the simulation. Stationary nodes representing public transport hubs transmit message packets to moving nodes within BLE range if required, but the algorithm used, transmission time and power consumption are not considered as part of this project.

4.3.1 Distance Routing Effect Algorithm for Mobility (DREAM)

The DREAM algorithm was designed such that nodes in the network always maintain valid routes to neighbouring nodes. Each node uses a location table to store the position of the other nodes in the network using XY coordinates, but entries in the table are only updated when two nodes are within Bluetooth Low Eneergy (BLE) range of each other. A 'message table' concept is introduced into the DREAM algorithm design so nodes can keep track of which nodes in the network are yet to receive the message packets. The message table simply stores a value for each node in the simulation, indicating whether that given node has received the message packets about the local area event. The entries in the message table can take one of three values: true, false or unknown. The unknown state exists in the message table when two nodes have not yet been within BLE range to exchange update packets. The message table allows for a reduction in power consumption for the network, as message packets are only transmitted to nodes who have not yet received the message. This is a good example of where the decision to design the simulation system from scratch has benefited this project. The addition of a message table may not have been possible using an existing network simulator, so the design and implementation of the DREAM algorithm would have been more complicated.

In the DREAM routing protocol, location information is exchanged in a manner proportional to mobility, so static nodes do not need to transmit update packets. This is a key feature of the protocol so this was included in the design of the DREAM algorithm for this project. The DREAM algorithm is expected to have a significant overhead, as update messages are transmitted even if no message packets are. The use of a message table, in conjunction with update messages being proportional to mobility, should lead to a reduction in the routing overhead. This overhead is an important consideration due to the power limited nature of the mobile devices involved in the MANET. The results of the DREAM algorithm simulation allow the trade-off between routing overhead and always valid routes to be seen.

The transmission of update messages via BLE limits the transmission to nodes within 10 m of each other. This can lead to different location and message table entries across different nodes, which could lead to incorrect information about the network being transmitted. To avoid this, only table entries that are known to be correct are exchanged between nodes in the update messages. This requirement limits the exchange to only updating table entries for the two nodes involved in the transmission and receiving of the update message. While this increases the routing overhead of the algorithm, it avoids errors caused by old information overwriting new information. A possible workaround involves storing timestamps for each table entry, but this could lead to an increased number of packets per update due to the extra information required. For the MANET under consideration, this workaround was deemed to be unnecessary as the key aspect is measuring packet transmission speed, so the limited exchange method was retained.

While BLE transmission is limited by distance, it is also limited by time. Since each time slice within the simulation represents a second in real time, the number of BLE transmissions possible per time slice must be considered. BLE has a minimum connection interval which determines the number of transmissions a single node can make within a second. Since the majority of smartphones are iOS and Android devices, the minimum connection intervals for the two mobile operating systems were considered when designing the routing algorithms. Android devices have minimum connection periods of 7.5 ms (Nexus 4 and Nexus 6P devices), while iOS devices have minimum connection periods of 30 ms (iPhone 6 devices) [31]. Taking the upper bound of 30 ms allows a single node to carry out 33 transmissions in a time slice of one second. However, this assumes multiple transmissions can be carried out without any time delays between them. To be on the safe side, if an average delay of 70 ms is assumed, 10 transmissions can occur per node per second. This value is used in the DREAM algorithm design, but the maximum transmissions per time slice can be varied in the simulation, to see the effect on performance. However, raising the value in the simulation to above 33 transmissions per second may give invalid results.

The limit imposed on transmissions per second means a theoretical minimum value exists for the Group Transmission Time (GTT). This value can be calculated using equation 4 where TPS is the maximum number of transmission per second per node.

$$\text{Minimum GTT} = \log_{(\text{TPS}+1)}(\text{nodes per group}) \tag{4}$$

This theoretical minimum is calculated on the basis that each node that has already received the message packets can transmit the maximum number of packets allowed at all times. While this is unlikely to be true in practice, dense networks can get close to the theoretical minimum value. This equation can be used to give minimum values for the other algorithms simulated too. The minimum values obtained using this equation should be used to verify the validity of the simulation results, as no results should be below the theoretical minimum. If such a result occurs, the simulation scripts will have to be checked for errors.

4.3.2 Reactive Algorithm

This algorithm is referred to as the **'Reactive'** algorithm throughout this report as it was not designed to imitate a specific reactive routing algorithm investigated in the Background section.

Reactive routing algorithms determine routes only when required in order to reduce transmission overheads and power consumption. This algorithm is designed to determine routes when required, but the routes are discarded after one second. Discarding and re-determining routes means only valid routes can be present, so no packet transmission failures can occur due to invalid routes. The downside of this is that the energy intensive process of route discovery is required every second. This algorithm is designed

in such a way to see whether having always valid routes (when required) as part of a reactive routing protocol was beneficial.

The route discovery process for this algorithm can be broken down into two parts: the flooding of route request packets and the replies from nearby nodes. These reply messages are used to construct the routes for transmission. This algorithm is designed so that the flooding process has a significantly higher power consumption than normal packet transmission, to account for the energy cost of flooding the channel with route request packets. If nodes are within BLE range of the node that is determining routes, a reply message may be transmitted. Replies are only transmitted if the receiving node has not received the message packets regarding the local area event. This is determined using a message table, as discussed in the DREAM algorithm section. This design is selected for the algorithm to prevent redundant packet transmissions within the MANET. Reducing redundant transmissions should reduce the power consumption of the algorithm.

Since the route discovery stage is energy intensive, this algorithm is designed to be limited to a single route discovery and transmission cycle per second. The idea behind this is to limit the power consumption of the nodes and to allow performance comparisons between algorithms with single or multiple packet transmissions per second. This limitation means only the first route in the temporary route cache will ever be used, so an argument could be made for only constructing one route. The decision to keep the construction of multiple routes is based on increasing the flexibility of the algorithm, to allow for different network simulations to be run with varying parameters. While this may not be so relevant for this project, this decision increases the scope of future work related to this project. While the reduction in performance increases with the number of nodes in the simulation, the increased time to run the simulations is not a key concern for this project.

This reactive algorithm retains both the location and message tables introduced in the DREAM algorithm, but makes less use of them. Location tables are only updated when nodes send reply messages as a response to receiving a route request packet. The message tables are updated when replying to route request packets, as well as when nodes transmit packets using this reactive routing protocol. The reduced reliance on the location and message tables is due to the process of discovering new routes at each time slice of the simulation. This means storing the location and message status of each node is redundant and would simply increase the overhead of the algorithm. This contrast from DREAM will allow the effect of using routing tables on algorithm performance to be seen.

4.3.3 Reactive Delay Algorithm

A modified version of the 'Reactive' routing algorithm was designed as a new algorithm. This algorithm is referred to as the **'Reactive Delay'** algorithm throughout this report.

The modification made to the Reactive algorithm limits the number of route request flooding stages a node can initiate. This modification is made to limit the power consumption of the Reactive routing algorithm as initial testing showed the power consumption to be higher than expected. While power consumption isn't the primary performance metric for this project, it is still an important consideration as the MANET consists of power limited mobile devices. An algorithm with significant power consumption will be unsuitable as some nodes in the network may cease to exist if the battery on the mobile devices run out.

A blocking period is introduced where nodes cannot initiate route discovery until a specified number of seconds have passed since the last route discovery stage. The blocking period was designed to be flexible to allow for a performance comparison between blocking periods of different lengths. Every time a node initiates a route discovery step, the time stamp is stored. When the node next tries to initiate route discovery, the time difference to the last route discovery stage is calculated and if the time difference is less than the blocking period, the node is prevented from determining new routes. The simplicity of the blocking period design means it can easily be modified as part of a further work.

The rest of this algorithm is the same as the original 'Reactive' routing algorithm discussed in the previous section. Again, the modular design used throughout this project simplified the design of this modified

algorithm as common elements could be reused very easily. This modular design also makes the overall system more resilient to errors, since correctness testing is easier.

4.3.4 Lightweight Mobile Routing (LMR)

Another reactive algorithm used in the system is the Lightweight Mobile Routing (LMR) protocol. The key features of this routing protocol identified in the background section are the use of flooding to discover routes and routes only being maintained to neighbouring nodes in the network. The use of flooding in the 'Reactive' routing algorithm simplified the design of the LMR algorithm as the same framework could be used for both. Maintaining routes only to neighbouring nodes was also straightforward to design since the other algorithms in the system have similar behaviour with regards to maintaining routes.

The LMR algorithm was designed to have a blocking period between route discovery stages because the other algorithms are designed in such a way that ensures routes are always valid before transmission occurs. This blocking period results in the possibility of some routes being invalid, allowing for a performance comparison between the different approaches. An additional performance metric for the LMR algorithm is the Packet Transmission Error Rate (PTER). This rate can be calculated by tracking the number of failed transmissions due to invalid routes, along with the number of successful packet transmissions:

$$\text{Packet Transmission Error Rate}(\%) = \frac{\text{Failed Transmissions}}{\text{Completed Transmissions} + \text{Failed Transmissions}} \times 100 \quad (5)$$

While a 'pure' LMR algorithm maintains multiple routes to destination nodes, the version of LMR designed for this project only maintains a single route to neighbouring nodes. This design choice was made to reduce the overhead time of the algorithm, since the primary aim of the MANET is to transmit message packets about a local area event as quickly as possible. Maintaining only one route is likely to increase the link failure rate, but allows for a comparison between the other algorithms in the system to verify the effect on performance.

This LMR algorithm is designed to discard routes from the route cache once transmission to the relevant neighbouring node is attempted. This design is used to prevent redundant transmissions within the MANET, as packet transmission to nodes which have already received the local area event message reduces the performance of the network. In the event of a failed transmission due to an invalid route, the route will be discarded from the route cache as usual and replaced the next time the node constructs routes to neighbouring nodes. This may not be the most efficient approach for the MANET in question, but the performance comparison between the different algorithms will allow this hypothesis to be proved or disproved.

The limitation on the number of transmissions per node per second is also considered in the design of this LMR algorithm. The default value is set to a maximum of 10 transmissions per node per second, although this value can be modified easily for various simulations. As with the DREAM algorithm, setting this value above 33 transmissions per node per second may give invalid results, since an upper bound of 30 ms per BLE connection interval exists for smartphones.

5 Implementation

This section discusses some of the key aspects of the network simulation system developed for this project. The most important parts of some key functions are given along with detailed explanations of how they work and why these functions are required.

5.1 Overview

As mentioned previously in section 2.9, MATLAB was selected as the most appropriate simulation package for this project. The simulation system designed consists of a number of MATLAB scripts and functions that can broadly be split into two sections: the mobility model and the routing algorithms. The modular design process discussed in the Analysis and Design section sped up the development of parts of the system due to the reuse of common elements. A downside to this modular approach is that some sections of the system were designed without future use in mind, so the implementation may be inefficient or unintuitive to use.

The first step of the simulation is to fetch the parameters used for that particular simulation. The variable parameters are defined in a single location and fetched each time the simulation runs to ensure the same simulation parameters are used for each run. This approach helps to avoid simulation errors, so the results obtained for the different algorithms can safely be compared. Each simulation script has a call to the function `getSimulationParams` which returns a cell array of the relevant parameters. The use of a cell array allows for data of different types to be stored in the same array, unlike a vector or matrix.

Once the simulation parameters have been fetched, each node in the simulation is initialised. A class called `StationaryNode` was created to model the behaviour of stationary nodes in the simulation. Stationary nodes represent public transport hubs such as London Underground stations and bus stops. Since these nodes are stationary, the only parameters that need to be initialised are node id, group and their fixed position in the simulation. Stationary nodes are always assigned a group number of 0 since the group attribute for nodes is only relevant to moving nodes.

Once the stationary nodes have been initialised, the local area event is modelled by simulating a station closure in the simulation area. This is done by setting the `message_to_transmit` attribute to true for a stationary node. When a moving node comes within BLE range of this stationary node, the message propagation will begin using the MANET among the moving nodes.

Moving nodes are then initialised and the mobility model is used to calculate their movement for the simulation. The group attribute for moving nodes is used to assign a route from the stadium to an 'exit point' corresponding to a public transport hub. All nodes within a group have the same start and end points in the simulation, but the individual movements vary due to the nature of the mobility model. Paths are generated by calling the `calculatePaths` function which uses the `SPMBM` function where the Shortest Path Map Based Movement mobility model is implemented.

The `nodes` cell array contains all the different nodes involved in the simulation. This cell array is a compact data structure for storing information about each node in the simulation. This cell array makes accessing attributes on the different node objects straightforward, for example: `nodes{1}.current_position` returns the current position of the node with id 1 in the simulation. The object type in the cell array depends on the routing algorithm being simulated since a new class was written for the implementation of each algorithm. These classes will be discussed in the relevant subsections within this section.

The main part of the simulation is the loop over all time slices in the simulation. Within the loop, there are two main sections: a transmission step and a movement step. For every moving node in the simulation, the possibility of transmission is evaluated. If transmission is deemed to be possible, packet transmission occurs using one of the routing algorithms implemented. Transmission is carried out within a class level function of the class used to implement a specific routing algorithm. The same structure is used for each algorithm which allows the same syntax to be used for each algorithm. An example for transmission between nodes 5 and 6 in the simulation is given below in listing 1:

```
1 [nodes{5},nodes{6}] = nodes{5}.transmit(nodes{6}); % This line is called in the
      simulation script
```

```
2
3  classdef DREAMNode
4     methods
5        function [self,dst] = transmit(self,dst)
6           % Method to transmit a message from node to dst node
7           % NOTE: Code has been removed from this listing for readability
8        end
9     end
10 end
```

Listing 1: Code snippet showing syntax for transmission between two nodes. The function displayed on line 5 shows the transmission function called by the first line

The `transmit` method for simulating the DREAM algorithm is presented in listing 1. The first line is executed in the simulation script and the function is called from the `DREAMNode` class. The code from the `transmit` method in the `DREAMNode` class has been removed in this listing for readability purposes. The code for this method will be discussed in detail later in this section.

After the transmission step, the nodes are moved to the next position in the path determined by the mobility model. The design of the mobility model simplifies this step since the position of a given node at every point in the simulation is generated and stored. Once all nodes have moved (if required to move), the time slice being simulated is incremented and the loop moves back to the transmission step. Once the maximum time for the simulation has been reached, the performance metrics (GTT and power consumption) are calculated.

For dense networks, the simulations can take a significant amount of time to run so an 'early exit' procedure exists. If the `quit_simulation_early` variable is set to true, the simulation loop exits once all the nodes in the test group have received the message packets about the local area event. The performance metrics are then calculated but one measure of power consumption is ignored. This will be discussed further in the section describing the implementation of the performance metrics.

The overall program flow is shown in figure 5:

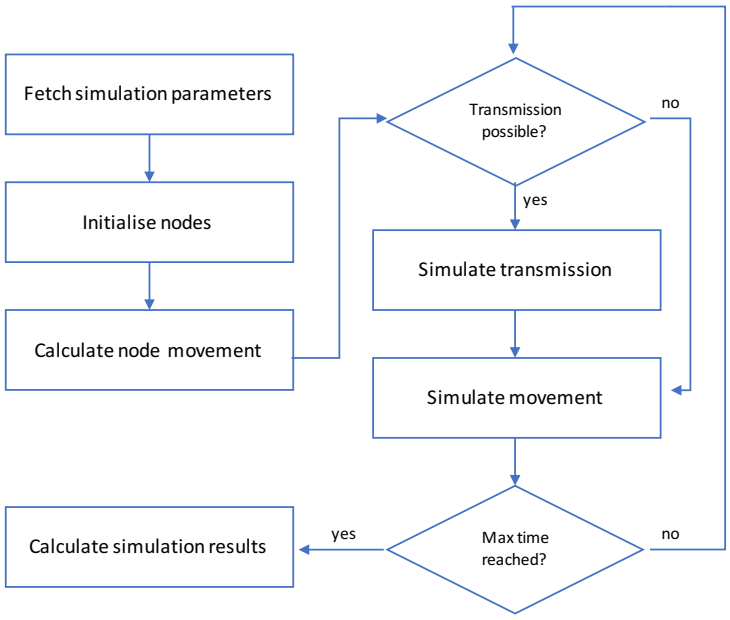


Figure 5: Overall simulation process flow. The *Simulate Transmission* step is different for each routing algorithm used in the system. Only this step varies significantly between the different simulation scripts due to the modular design utilised.

5.2 Shortest Path Map Based Movement (SPMBM) Mobility Model

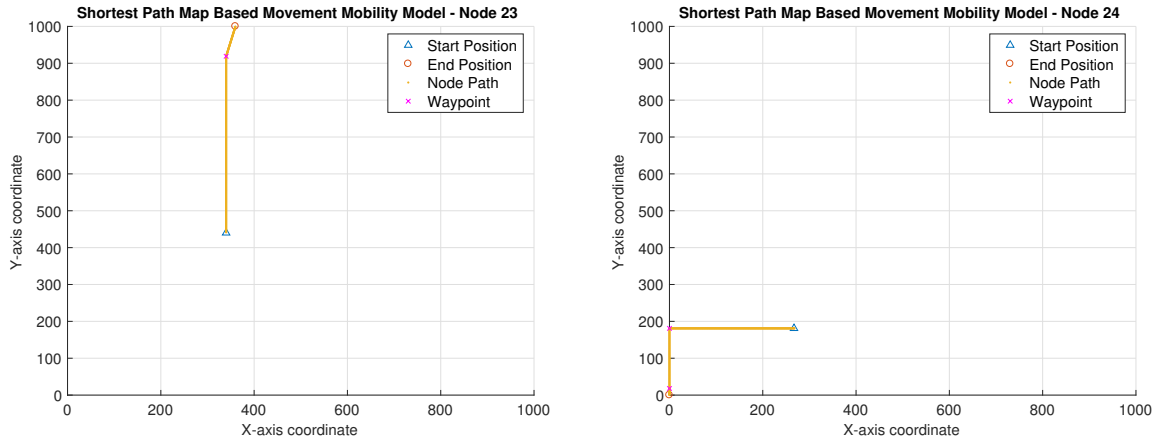
The simulation scripts access the SPMBM mobility model through calling the `calculatePaths` function. This function calls the `SPMBM` function where the mobility model is implemented, along with applying some post processing to the output of the `SPMBM` function.

```
1  function nodes = calculatePaths() % NOTE: Inputs have been removed for readability
2
3     for node = 1+number_of_stationary_nodes:number_of_nodes
4        [start_and_end,waypoints,main_path] = SPMBM();
5        % NOTE: Inputs to SPMBM have been removed for readability
6        overall_path = [start_and_end(1,:); main_path; start_and_end(end,:)];
7
8        % Extend paths to avoid indexing errors
9        while length(overall_path) < max_time
10          overall_path(end+1,:) = overall_path(end,:);
11       end
12
13       % Store path as an attribute of the node object
14       nodes{node}.position = {start_and_end,waypoints,main_path,overall_path};
15
16    end
17
18 end
```

Listing 2: Code snippet showing the key aspects of the `calculatePaths` function. Some function parameters have been removed to make the code snippet more readable.

The `SPMBM` function outputs three matrices, as seen on line 4 of listing 2. The `start_and_end` matrix is a 2x2 matrix containing XY coordinates for the start point and end point of a given node. The `waypoints` matrix has 2 columns but the number of rows depends on the number of waypoints a node passes through on its path. The `main_path` matrix also has 2 columns while the number of rows depends on the number of steps in the path. This matrix contains all the node positions for the overall simulation, apart from the start and end points. These three matrices are used to create the `overall_path` matrix and the `position` cell array which is stored as an attribute of the node object, as seen in listing 2.

The use of the `position` cell array, as opposed to a single matrix for the path of the node, is to allow for the path of a specified node to be plotted easily for testing purposes. Elements 1, 2 and 3 from the cell array can be passed to the `plotSPMBM` function which produces a plot similar to figure 6. These plots came in useful during the development process for the mobility model, as well as aiding the debugging of the overall simulation.



(a) Movement path for node 23

(b) Movement path for node 24

Figure 6: Plots generated by the `plotSPMBM` function for two nodes in a simulation

The post processing within the `calculatePaths` function extends the generated path to avoid indexing errors in the simulation script. This is done by duplicating the final position until the number of rows in the `overall_path` matrix is equal to the maximum simulation time.

The `SPMBM` function calls the `getPath` and `getMovement` functions in order to generate the outputs of the mobility model i.e. the path each node takes. As mentioned in the background section, a key feature of the SPMBM mobility model is the use of Dijkstra's algorithm to find the shortest path between two nodes. The `getPath` function creates an undirected graph using the graph parameters specified by the system. The undirected graph, along with the start and end nodes for the path, are passed as parameters to the `graphshortestpath` function from the Bioinformatics Toolbox. This produces a path from the start node to the end node determined using Dijkstra's algorithm. The `SPMBM` function passes this path to the `getMovement` function to produce the step by step node paths required for the simulation.

```
function [endpoints,waypoints,main_path] = getMovement() % NOTE: Inputs removed
number_of_legs = length(path)-1;
% Generate random speeds within an interval for each time step
leg_speeds = (max_speed - min_speed).*rand(1,number_of_legs)+min_speed;

for leg = 1:number_of_legs
   % Start and end point coordinates
   start_pos = [map_path(leg,1), map_path(leg,2)];
   end_pos = [map_path(leg+1,1), map_path(leg+1,2)];

   current_pos = start_pos;
   speed = leg_speeds(leg); % Get a value from the vector of random speeds

   % Find overall distances in X and Y directions
   % Calculate resultant vector distance
   distance_x = end_pos(1) - start_pos(1);
   distance_y = end_pos(2) - start_pos(2);
   if distance_x == 0
      resultant = distance_y;
   elseif distance_y == 0
      resultant = distance_x;
   else
      resultant = sqrt(distance_x^2 + distance_y^2);
   end

   % Work out angle between resultant vector and X axis
```

```
27      angle = acos(distance_x/resultant);
28      % Calculate movement in X and Y directions for this step
29      movement_x = speed*cos(angle);
30      movement_y = speed*sin(angle) * sign(distance_y);
31
32      % Find new position and append it to position matrix
33      new_pos = [start_pos(1) + movement_x, start_pos(2) + movement_y];
34      node_pos = [start_pos;new_pos];
35
36      while new_pos(1) ~= end_pos(1) || new_pos(2) ~= end_pos(2)
37            % While the end point has not been reached, carry out the same process as above
38            % NOTE: The code for this loop has been removed for readability
39      end
40  end
```

Listing 3: Code snippet showing the key aspects of the `getMovement` function. Some code has been removed from this snippet to highlight only the key aspects.

Listing 3 shows the key aspects of the `getMovement` function. The path generated by the `getPath` function is converted to a path of XY coordinate pairs using the position of each waypoint in the simulation area. The movement for each step is then is calculated using simple geometry by calculating the resultant vector for each step of the path and finding the X and Y direction components. The `leg_speeds` vector contains the random speeds within an interval used for each stage of the path. The X and Y components are calculated using these random speeds and the next position for the node is determined. This new position is then appended to the matrix that stores the path of the node. This process is repeated until a path has been generated to the specified end point. The simple geometry used is sufficient because the waypoints in the simulation have been designed to have approximately straight line paths between them, to make modelling the movement between waypoints simpler.

If packet transmission occurs, the `updatePaths` function is called to generate a new path for the node receiving the message packets. Ideally, this function would calculate the shortest path from the current position of the node to its new end point, but this is difficult to implement, as mentioned in the Analysis and Design section, due to the size of the simulation area. This functionality would require every point in the simulation area to be considered in the graph used to determine the shortest path between waypoints. To get around this, the `move2Waypoint` function is called which identifies the closest waypoint to the node and generates a path to this waypoint. The `SPMBM` function is then used to generate a path from the closest waypoint to the new end point. These path matrices are combined and some post processing is done to prevent indexing errors before the new path is stored as an attribute in the node object. Figure 7 shows an example of a modified path where the node was initially moving to (0,0), but is diverted to (360,1000) as a result of receiving message packets from another node in the network.

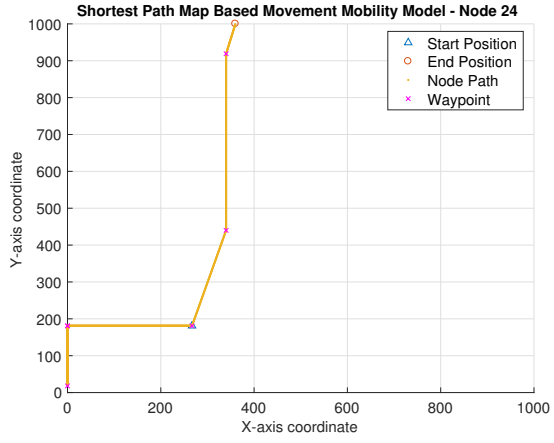


Figure 7: Updated path for a node with an initial end point (0,0) which is updated to an end point of (360,1000).

5.3 Routing Algorithms

Each routing algorithm simulated was implemented using a class to define the behaviour of each node. This approach was taken so the interactions between nodes could easily be modelled by creating an object for each node in the simulation. The class for each routing algorithm defines behaviour for all objects of that type and can be thought of as a template. Each object is an instantiation of the class, allowing for consistent behaviour to be observed regardless of the number of nodes being simulated. This object-oriented approach fits well with the modular design used for the system, as the same framework can be used for the simulation of each algorithm. Each class has common elements, such as functions to check whether nodes are within BLE range and functions to calculate power consumption. The code varies for different algorithms, meaning class level functions are the most appropriate implementation.

At the start of each simulation, each non-stationary node present in the network is initialised by creating an object and setting initial values for some of the attributes on each object. While this initialisation process is common to all algorithms, the actual code varies due to the different attributes and parameters of each algorithm. Each node object is stored in the `nodes` cell array to make the code easier to understand and modify. This object-oriented approach used for the implementation of the routing algorithms makes the extension of this project more straightforward, as a future work can simply build on top of the existing code base rather than reworking it.

The most important sections of each algorithm will be explored further in this section. Where sections of code are reused between algorithms, the implementation is only discussed once.

5.3.1 Distance Routing Effect Algorithm for Mobility (DREAM)

Nodes using DREAM for packet transmissions are described in the `DREAMNode` class. This class can be instantiated and initialised for a simulation as seen in listing 4:

```
1  % Set initial values for each moving node e.g. start point, end point etc.
2  for m = 1:number_of_moving_groups
3     for n = 1:nodes_per_group
4        index = n + (m-1)*nodes_per_group + number_of_stationary_nodes;
5
6        nodes{index} = DREAMNode;
7        nodes{index}.id = index;
8        nodes{index}.group = m;
9        nodes{index}.start_point =
             [map_node_positions(start_node(m),1),map_node_positions(start_node(m),2)];
10       nodes{index}.end_point =
             [map_node_positions(end_node(m),1),map_node_positions(end_node(m),2)];
11       nodes{index}.min_speed = min_speed(m);
12       nodes{index}.max_speed = max_speed(m);
13       nodes{index}.location_table{1,number_of_nodes} = [];
14       nodes{index}.message_table{1,number_of_nodes} = [];
15       nodes{index}.current_position = nodes{index}.start_point;
16
17       % Positions of stationary nodes are fixed and therefore known at start
18       for k=1:number_of_stationary_nodes
19          nodes{index}.location_table{k} = nodes{k}.current_position;
20       end
21    end
22 end
```

Listing 4: Code snippet showing the instantiation and initialisation of nodes for simulating packet transmission using the DREAM algorithm

Line 6 in listing 4 creates a new object by instantiating the `DREAMNode` class and stores it in the `nodes` cell array. Lines 7 to 15 set the initial parameters of the node where `id` and `group` are used as identification attributes for each node. The `start_point` and `end_point` attributes are used to store the XY coordinates for the start and end points for the node object. These points, along with the minimum

and maximum speeds, are used by the mobility model to generate the shortest path between the two for the simulation. The start point, end point, minimum speed and maximum speed depend on the group the node has been assigned as these simulation parameters are defined for the entire group. The `current_position` attribute is initialised to be the start point of the node as it will begin the simulation at the defined start point.

Each `DREAMNode` object has `location_table` and `message_table` attributes where cell arrays are used to store the entries of the two tables. Upon initialisation of the nodes, the two tables are empty but some entries in the location table are filled in the loop between lines 18 and 20 in listing 4. This loop fills in the entries in the location table for the position of the stationary nodes in the simulation, as their locations are fixed and known at the start of the simulation. The message table is left empty as the status of the stationary nodes is not known.

The `index` variable in listing 4 is used to store node objects at different positions in the `nodes` cell array based on their group. The structure of the `nodes` cell array is shown in figure 8.

Figure 8: Structure of the `nodes` cell array where N moving groups are stored

The node objects are grouped by their group number in the `nodes` cell array, but nodes can only be accessed by their id. This is why the `index` variable is required as it determines a node id for each node within a group. Since the first group (group 0) contains only stationary nodes, the number of stationary nodes in the simulation is used as an 'offset' in the `index` calculation. The group number (`m`) of the node being initialised and the position of the node within its group (`n`) are used to generate the node id and the index where the node is stored in the `nodes` cell array. This indexing system is used for all the algorithms in the project for consistency.

```
1  % Moving node to moving node transmission step
2     for src=1+number_of_stationary_nodes:number_of_nodes
3        nodes{src}.packets_transmitted_slice = 0;
4        if nodes{src}.message_to_transmit
5           for dest=1+number_of_stationary_nodes:number_of_nodes
6              if dest ~= src && nodes{src}.packets_transmitted_slice <
                   nodes{src}.max_transmissions_per_sec
7                 if nodes{src}.checkBTRange(nodes{dest}) &&
                      ~nodes{dest}.message_to_transmit
8
9                    % Transmit message to destination node
10                   [nodes{src},nodes{dest}] = nodes{src}.transmit(nodes{dest});
11                   % Increment counter for number of packets transmitted in the current
                        time slice
12                   nodes{src}.packets_transmitted_slice =
                         nodes{src}.packets_transmitted_slice + 1;
13
14                   % Update paths for destination node
15                   nodes{dest} = updatePaths() % NOTE: Parameters removed for readability
16
17                % Update tables if destination node has already received the message
18                elseif nodes{src}.checkBTRange(nodes{dest}) &&
                       nodes{dest}.message_to_transmit
19                   nodes{src}.message_table{dest} = true;
20                   nodes{src}.table_updates = nodes{src}.table_updates + 1;
21                   nodes{dest}.update_packets_transmitted =
                         nodes{dest}.update_packets_transmitted + 1;
```

```
22                  end
23              end
24          end
25      end
26  end
```

Listing 5: Code snippet showing the instantiation and initialisation of nodes for simulating packet transmission using the DREAM algorithm

The main aspect of simulating the DREAM routing algorithm is the transmission step. Listing 5 shows the key aspects of the implementation of the DREAM algorithm. The algorithm is simulated by iterating through every moving node in the simulation and checking whether transmission to every other node in the simulation is possible. This essentially creates a 2D iteration space across the source nodes (`nodes{src}`) and the destination nodes (`nodes{dest}`). Some points in this iteration space are not simulated if the source node has not already received the message packets to allow transmission to other nodes. Similarly, points on the diagonal of the 2D iteration space are not simulated since a node has no need to transmit packets to itself. This 2D iteration space results in dense network simulations taking a significant amount of time to run as the computational complexity of the algorithm is $O(n^2)$.

The second conditional statement on line 6 in listing 5 shows the check implemented to prevent an unrealistic number of packet transmissions per second. As discussed in the Analysis and Design section, the default value for `nodes{src}.max_transmissions_per_sec` is set as 10, but this can be modified in the `DREAMNode` class.

A final condition (line 7) must be met for transmission between two nodes to occur. The function `checkBTRange` returns true if the source and destination nodes are within BLE range (10m) and the destination node must not already have the message. This final check prevents redundant transmissions from occurring to reduce power consumption and increase the speed of message propagation. Transmission is simulated by calling the `transmit` function in the `DREAMNode` class, which updates a number of simulation parameters used for performance tracking, as well as updating the relevant location and message table entries.

Once transmission has occurred, the transmission counter is incremented for the current time slice and the path of the destination node is updated. The `updatePaths` function uses the mobility model to generate a path to a different 'exit point' within the simulation area. Lines 18-22 in listing 5 show what happens if the intended destination node has already received the message from a different source node. This scenario occurs when the intended destination node has not been in BLE range of the source node, so update messages have not been exchanged. This section of the code simulates the exchange of these update messages to update the routing tables of the original source node.

The main class level functions in the `DREAMNode` class are the `transmit` and `checkBTRange` functions. The `checkBTRange` function is used in each routing algorithm class, since the method for checking whether two nodes are within BLE range is the same regardless of algorithm. A simple check is implemented to see if two nodes are less than or equal to 10 m apart in the simulation area at a given time. Listing 6 shows the `transmit` function within the `DREAMNode` class. This function updates the number of packets transmitted and received for the transmitting and receiving nodes respectively, along with updating the message tables of both nodes. The message table updates set the relevant entries in the message table to 'true' since the destination node has now received the message packets. The number of packets and update messages transmitted are used to calculate the power consumption of each node simulating transmission via DREAM.

```
1  classdef DREAMNode
2     methods
3        function [self,dst] = transmit(self,dst)
4           % Method to transmit a message from node to dst node
5           self.packets_transmitted = self.packets_transmitted + 1;
6           dst.packets_received = dst.packets_received + 1;
7           dst.message_to_transmit = true;
8           dst.message_table{dst.id} = true;
```

```
9           dst.message_table{self.id} = true;
10          self.message_table{dst.id} = true;
11       end
12    end
13 end
```

Listing 6: Code snippet showing the `transmit` function for the `DREAMNode` class

5.3.2 Reactive Algorithm

The Reactive routing algorithm is simulated using objects of the `ReactiveNode` type. These objects are instantiated from their class and initialised using the same method as for `DREAMNode` objects, but the object attributes vary slightly. The location and message table initialising lines are removed, as the Reactive routing algorithm is designed to have minimal use of routing tables. This means the loop where location table entries corresponding to stationary nodes are filled for `DREAMNode` objects is also removed.

```
for src=1+number_of_stationary_nodes:number_of_nodes
   if nodes{src}.message_to_transmit
      [nodes{src},~,~] = nodes{src}.broadcast; % Source node broadcasts route request
          packets

      % Check if any nodes are within BLE range
      in_range = getNodesInRange(); % NOTE: Inputs removed for readability

      % Send replies if nodes are within BLE range
      if ~isempty(in_range)
         % Vector will always update since only nodes without the
         % message are returned

         for i=1:min(nodes{src}.transmissions_per_second,length(in_range))
            dest = in_range(i);

            % Send reply to broadcasting node and update tables
            [nodes{dest},nodes{src}] = nodes{dest}.sendReply(nodes{src});

            % Transmit from src to dest
            [nodes{src},nodes{dest}] = nodes{src}.transmit(nodes{dest});

            % Update paths for destination node
            nodes{dest} = updatePaths() % NOTE: Inputs removed for readability
         end
      end
   end
end
```

Listing 7: Code snippet showing the key aspects of the transmission step for simulating the reactive routing algorithm

Listing 7 shows the key aspects of the transmission step for the reactive routing algorithm. This step is quite different to that of the DREAM simulation due to the differences in the design of the two algorithms. This routing algorithm is simulated by iterating through each node in the simulation and evaluating whether a node has already received the message packets. If this is true, the node floods the channel with route request packets in order to determine routes to neighbouring nodes. This is implemented using the `broadcast` method in the `ReactiveNode` class. The `getNodesInRange` function returns a vector of node id's, corresponding to nodes within BLE range of the broadcasting node. If this vector is empty, no nodes are nearby and the simulation moves onto the movement step.

If the `in_range` vector is not empty, the simulation iterates through the vector and attempts transmission between the source and destination nodes. Transmission first requires a reply message to be sent from the destination node to the source (broadcasting) node, so a route can be constructed. Listing 8 shows the `sendReply` method of the `ReactiveNode` class.

```
1  classdef ReactiveNode
2     methods
3        function [self,bc_node] = sendReply(self,bc_node)
4           % Method to reply to broadcast so transmission can begin
5           % Self is the replying node i.e. destination node for the
6           % transmission, bc_node is the broadcasting node i.e. the
7           % source node for the transmission
8           self.replies_sent = self.replies_sent + 1;
9
10          self.location_table{bc_node.id} = bc_node.current_position;
11          self.message_table{bc_node.id} = true;
12          self.location_table{self.id} = self.current_position;
13          self.table_updates = self.table_updates + 1;
14          self.update_packets_transmitted = self.update_packets_transmitted + 1;
15
16          bc_node.location_table{self.id} = self.current_position;
17          bc_node.message_table{self.id} = false;
18          bc_node.table_updates = bc_node.table_updates + 1;
19          bc_node.ready_to_transmit = true;
20       end
21    end
22 end
```

Listing 8: Code snippet showing the `sendReply` method of the `ReactiveNode` class

This function updates a number of node attributes including `replies_sent` and `table_updates`. These attributes are used in the power consumption tracking for each node in the simulation. This function also updates location and message tables of both the broadcasting node and the replying node in order to construct a valid route between the two nodes. Once the route is constructed, the broadcasting node is ready to transmit and transmission can occur. Once transmission has completed, the path for the destination node is updated as in the DREAM algorithm simulation.

Line 13 of listing 7 shows the upper bound of the for loop to be the minimum of the number of transmissions per second the node can undertake and the length of the `in_range` vector. This is required to ensure nodes cannot transmit too many packets within a time slice of one second, to maintain the realism of the simulation. The minimum of the two values is used to prevent indexing errors in the event where the length of the `in_range` vector is smaller than the number of transmissions allowed per second. The default value for `transmissions_per_second` is set to 1 to simulate a maximum of 1 broadcast, reply and transmit cycle per second. This value can be modified in the `ReactiveNode` class for different simulations to observe the effect on performance.

The modular approach used for the design of the simulation system means a lot of sections used in the simulation of the DREAM algorithm can be reused in the simulation of the Reactive algorithm. This saved development time and made debugging simpler, as bugs only had to be fixed in one function or script instead of many.

5.3.3 Reactive Delay Algorithm

The Reactive Delay routing algorithm is implemented in the `ReactiveDelayNode` class. The key difference between the two reactive routing algorithms is the blocking period introduced in this version. The duration of the blocking period is defined in the `broadcast_delay` attribute in the `ReactiveDelayNode` class. This blocking period means a node using this routing algorithm cannot discover new routes until the blocking period is over. This reduces the number of route request flooding stages a node can initiate, therefore reducing the power consumption of this algorithm.

```
1  % Simulation script
2  if nodes{src}.message_to_transmit && ~nodes{src}.checkBlockingPeriod(t)
3     % NOTE: Code removed for readability
4  end
5
6  classdef ReactiveDelayNode
7     methods
8        function blocked = checkBlockingPeriod(self,t)
9           % Method to check whether the node is within the blocking
10          % period for broadcasting route request packets
11          if t-self.last_broadcast_time < self.broadcast_delay
12             blocked = true;
13          else
14             blocked = false;
15          end
16       end
17    end
18 end
```

Listing 9: Code snippet showing the blocking period check in the simulation of this algorithm, along with the `checkBlockingPeriod` method of the `ReactiveDelayNode` class

Line 2 of listing 9 shows the blocking period check in the simulation script. If a value of true is returned from `checkBlockingPeriod`, the route discovery step is prevented from occurring. The `checkBlockingPeriod` function is relatively simple; the time difference between the current time slice of the simulation and the last time the node initiated route discovery is calculated. If the result is less than the duration of the blocking period, a value of true is returned to indicate the node is blocked from discovering new routes.

5.3.4 Lightweight Mobile Routing (LMR)

The version of the LMR algorithm implemented in this project is described in the `LMRNode` class. Some similarities exist with the other reactive routing algorithms implemented, namely the process of determining routes on demand. However, this algorithm uses a route cache which allows invalid routes to be present, unlike the other reactive routing algorithms implemented in this project. The potential inclusion of invalid routes is made possible by the implementation of a blocking period, similar to the Reactive Delay algorithm.

Listing 10 shows the overall algorithm flow for the simulation of the LMR algorithm implemented in this project. Some code has been removed from this snippet for readability.

```
1  % Moving node to moving node transmission step
2     for src=1+number_of_stationary_nodes:number_of_nodes
3        nodes{src}.packets_transmitted_slice = 0;
4        if nodes{src}.message_to_transmit
5           if isempty(nodes{src}.route_cache)
6              if ~nodes{src}.checkRouteRequestBlocked(t)
7                 % Discover routes via flooding
8              end
9           else % Route cache not empty
10             % Transmit to nodes where routes are present in the route cache
11             for dest=nodes{src}.route_cache
12                if checkBTRange(nodes{src},nodes{dest}) &&
                      nodes{src}.packets_transmitted_slice <
                      nodes{src}.max_transmissions_per_sec
13                % If route is still valid and the node has not hit
14                % the transmissions per second limit, transmit
15
16                % Increment packets transmitted in the current time slice
```

```
17                  nodes{src}.packets_transmitted_slice =
                        nodes{src}.packets_transmitted_slice + 1;
18
19              elseif nodes{src}.packets_transmitted_slice >=
                    nodes{src}.max_transmissions_per_sec
20                  % Transmission limit reached, do nothing
21              else
22                  % If the route is no longer valid, transmission fails
23                  nodes{src}.failed_transmissions = nodes{src}.failed_transmissions + 1;
24              end
25          end
26      end
27   end
28 end
```

Listing 10: Code snippet showing the overall algorithm flow for the simulation of the LMR algorithm

This algorithm uses the same method as the other two reactive routing algorithms to discover routes via flooding, but the transmission step differs. This LMR algorithm can either discover routes or transmit message packets in a time slice, but not both unlike the other reactive algorithms. A limit on the maximum packet transmissions a node can initiate within a time slice is imposed, to prevent invalid results being produced. As with the DREAM algorithm, the limit on transmissions per second has a default value of 10, but can be modified easily in the `LMRNode` class.

Since routes are not always discovered on demand, they may become invalid by the time the transmitting node wishes to transmit to a certain destination node. In this scenario, the number of failed transmissions for a LMR node is incremented. At the end of the simulation, the packet transmission error rate can be calculated using equation 5.

Once transmission to a destination node has been attempted, the route to the destination node is removed from the route cache. Listing 11 shows the implementation of this route deletion process. Initially, the `remaining_routes` vector contains the entire route cache (line 4). Line 10 shows how routes to a specific destination node are removed from this vector using the 'NOT' operator. Note that routes are removed regardless of the outcome of the attempted transmission. If the attempted transmission fails, a new route to the destination node may be created in a future time slice. At the end of the time slice, only the unused routes are retained in the route cache and the process repeats until the simulation is over. This route removal process is implemented to prevent redundant transmissions in the simulation. It also allows the route cache to only be updated when no more routes remain. This reduces the power consumption of the algorithm by reducing the number of energy intensive route discovery stages required.

```
1  if isempty(nodes{src}.route_cache)
2  % NOTE: Code removed
3  else
4     remaining_routes = nodes{src}.route_cache;
5     for dest=nodes{src}.route_cache
6        if checkBTRange(nodes{src},nodes{dest}) && nodes{src}.packets_transmitted_slice <
             nodes{src}.max_transmissions_per_sec
7        % NOTE: Code removed
8        end
9        % Remove routes from route cache once transmission has been attempted to that node
10       remaining_routes = remaining_routes(remaining_routes~=dest);
11    end
12    % Only keep the remaining (unused) routes in the cache for transmission in the next
          time slice
13    nodes{src}.route_cache = remaining_routes;
14 end
```

Listing 11: Code snippet showing the method for removing routes from the route cache once transmission has been attempted

5.4 Performance Metrics

Performance metrics of Group Transmission Time (GTT) and power consumption are required to allow an objective comparison between the different routing algorithms simulated in this project. During the simulation of each algorithm, data is collected about the power consumption of each node in the network. At the end of the simulation, this data is processed to give a usable result. Similarly, as the simulation progresses, timings are collected for the first and last packet transmissions for the test group in the simulation. These timings allow the calculation of the GTT performance metric.

5.4.1 Group Transmission Time (GTT)

To recap, in this project, Group Transmission Time (GTT) is the time difference between the first and last relevant nodes in the network receiving the message packets about the local area event. At the end of each time slice within the simulation, the `storeTransmissionTimes` function is called to evaluate whether all or some of the nodes in the test group have received the message packets. This function calls `getNumberOfNodesWithMessage` which returns the number of nodes within a specified group that have received the message being propagated. This is calculated by iterating through each node in the specified group and incrementing a counter if the node has received the message.

Storing the relevant timings to calculate GTT is implemented through the use of two 'if' statements in the `storeTransmissionTimes` function:

```
1  function [first_transmission_time,last_transmission_time] = storeTransmissionTimes()
2  % NOTE: Inputs removed for readability
3     % Two if statements are used rather than an if..elseif statement since
4     % for sparse networks, all the nodes in a group may receive the message
5     % within the same time slice so both conditions can be true
6     % NOTE: Some function inputs have been removed for readability
7     if getNumberOfNodesWithMessage() == nodes_per_group && last_transmission_time == -1
8        last_transmission_time = t;
9     end
10    if getNumberOfNodesWithMessage() > 0 && first_transmission_time == -1
11       first_transmission_time = t;
12    end
13 end
```

Listing 12: Code snippet showing the key aspects of the `storeTransmissionTimes` function used to allow the calculation of GTT for the test group in a simulation

The comment from lines 3 to 5 in listing 12 give a justification for the use of two 'if' statements rather than an 'if..elseif' statement. The `first_transmission_time` and `last_transmission_time` variables are initialised to -1 when the simulation starts. The 'if' statement from lines 10 to 12 in listing 12 set the value for `first_transmission_time` if more than 0 nodes in the test group have received the message and the first transmission time has not yet been set. The 'if' statement from lines 7 to 9 is similar, but it sets the value for `last_transmission_time` if all the nodes within the test group have received the message.

Once the simulation has ended, either by reaching the last time slice or satisfying the early exit requirements, the `getSimulationResults` function is called. This calculates GTT as the difference between `last_transmission_time` and `first_transmission_time`, as long as neither value is still -1. The GTT result is then printed to the console to allow the result to be viewed.

5.4.2 Power Consumption

A similar method is used to collect the data required for the power consumption results, but instead of returning a single value, five different power consumption results are produced. These measures are:

- Initial power consumption - The power consumption of nodes in the test group incurred before propagation of the message has begun. This is a measure of the power consumption incurred only due to the movement of the test group.

- Final power consumption - The total power consumption of the test group by the time all nodes in the test group have received the message packets.
- Transmission power consumption - The power consumption incurred as a result of propagating the message through the network to all the nodes in the test group. This is calculated as the difference between the final power consumption and the initial power consumption.
- Average transmission power consumption (ATPC) - The transmission power consumption divided by the number of nodes in the test group.
- Overall power consumption - The total power consumption for all nodes in the test group once the maximum time of the simulation is reached. If the simulation is set to exit early, this value will not be calculated.

The calculation of the Transmission Power Consumption (TPC) requires the power consumption to be measured when the first and last transmission times are measured. This is done using a similar method to the process described in the GTT section above, using the `calculatePowerConsumption` function at the end of each time slice.

At the end of the simulation when the `getSimulationResults` function is called, the different power consumption measures are calculated. The results are then output to the console if required by the simulation parameters. The main metric used to compare power consumption between routing algorithms is the Average Transmission Power Consumption (ATPC).

The implementation for power consumption calculations are relatively straightforward since each node object tracks its own power consumption. For example, listing 13 shows the power consumption calculation for DREAM nodes. A similar function is used for the other algorithms.

```
1 classdef DREAMNode
2    methods
3       function power_consumption=get.power_consumption(self)
4          % Method to calculate power consumption
5          power_consumption = (self.packets_transmitted +
               self.update_packets_transmitted) * self.transmission_cost +
               self.table_updates * self.table_update_cost;
6       end
7    end
8 end
```

Listing 13: Code snippet showing the power consumption calculation function in the `DREAMNode` class

The power consumption results obtained from this project should only be used to compare performance between the different algorithms implemented. This is because the power consumption values used in the simulations are estimates derived from related works. As mentioned in section 2.9, the values used give an overestimate for the power consumption of the nodes in the network.

6 Testing

For this project, the testing stage was required to test the functional correctness of the different aspects of the system. Testing was carried out alongside the implementation of the different components to avoid issues further down the line. The mobility model was tested to ensure the paths generated for each node were realistic for the area being simulated, while the routing algorithms were tested to ensure packet transmission was correctly simulated. The collection of data to allow the performance metrics to be calculated was also tested, as part of the routing algorithm testing phase.

The Results section (section 7) covers the simulation procedure and the simulation parameters used for each run.

6.1 Mobility Model

The implementation and testing of the Shortest Path Map Based Movement (SPMBM) mobility model was done in stages. First the shortest path calculation function was tested to ensure the paths generated were correct. This was done using the `plotshortestpath_test` function. This function requires the start and end points of each edge in the graph to be entered, along with the edge weights. The start and end points of a required path are entered and a figure similar to figure 9 is generated.

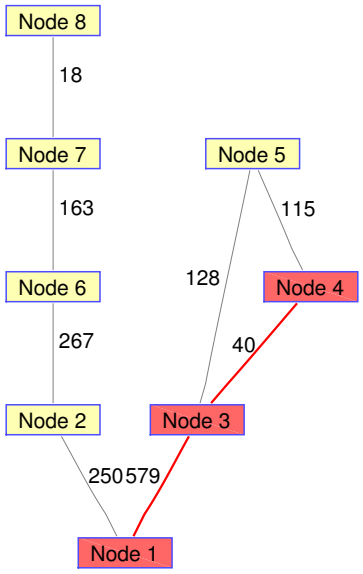


Figure 9: Figure generated from the graph testing procedure. The highlighted path shows the shortest path from node 1 to 4.

Once the path generation function was determined to be correct, a 100 × 100 XY grid was set up as a simulation environment, where start and end points for a moving node could be specified. The mobility model was then used to generate paths between these end points and the output was analysed to verify correctness. Initially, a straightforward path between two points was specified. The mobility model implementation was improved until the path generated was seen to be correct. Figure 10 shows a correct path generated by the mobility model once the first stage of testing was complete.

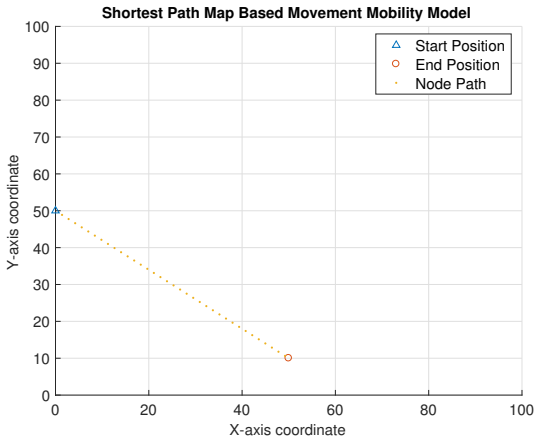
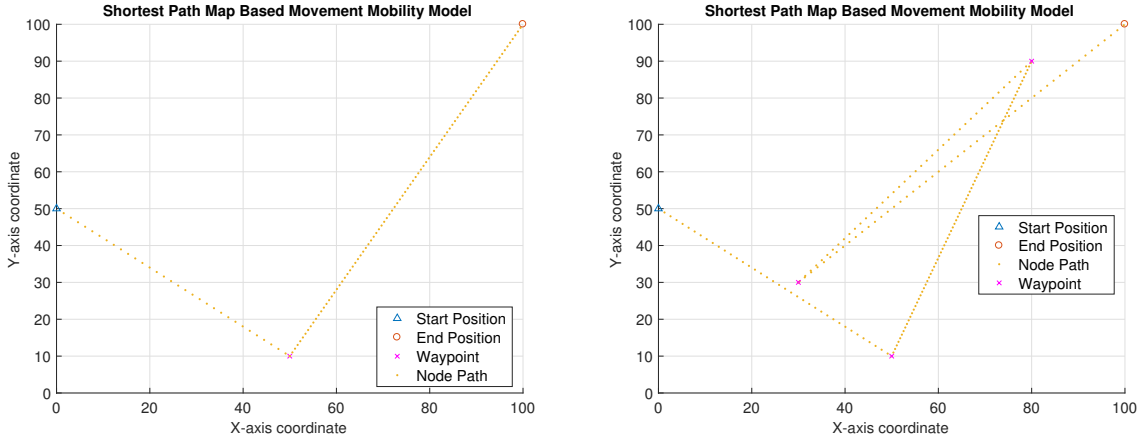


Figure 10: Shortest Path Map Based Movement mobility model testing in a 100×100 XY grid. This first testing stage involved simple paths between two points.

Once simple paths were tested and found to be correct, more complex paths were tested in the same 100 × 100 XY grid. These more complex paths involved multiple stages where nodes would have to travel to waypoints before moving to the end point. For example, a start point of (0,50), an end point of (100,100) and a waypoint at (50,10) was specified. The path generated by the mobility model was analysed and errors in the implementation were corrected. This stage of testing allowed the handling of waypoints to be checked.

The outcome of the situation described is seen in figure 11a. Figure 11b shows a slightly more complex path involving more waypoints.



(a) Mobility model testing using a 2 stage path

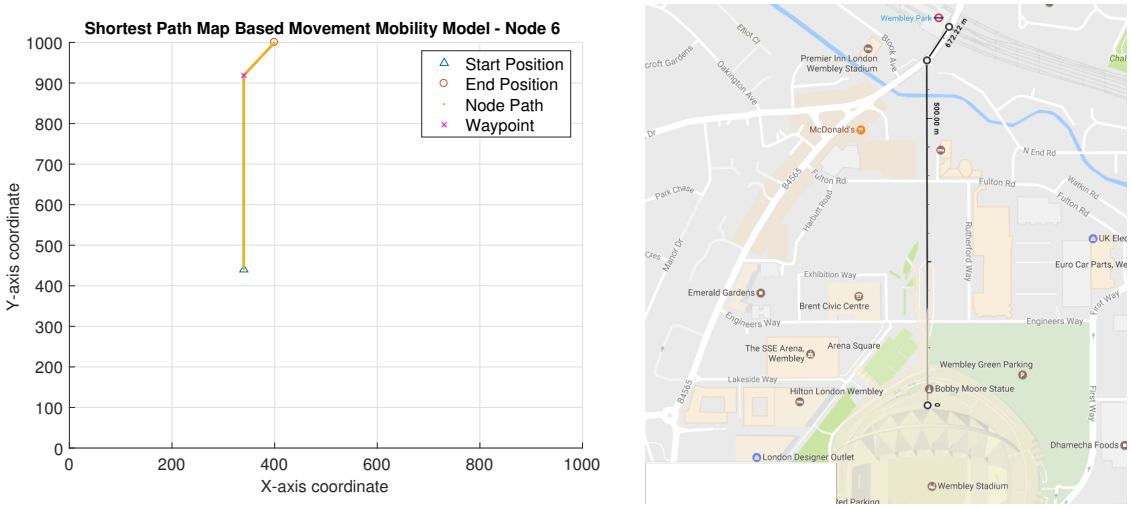
(b) Mobility model testing using a 4 stage path

Figure 11: Shortest Path Map Based Movement mobility model testing in a 100×100 XY grid.

These output plots generated allowed errors in the mobility model to be seen visually, which sped up the testing and development process. When errors were found, the relevant functions in the mobility model implementation were analysed to identify the cause. For example, an issue in the mobility model was discovered where a node would continue to move past the specified end point. This occurred because only one coordinate (X or Y) was being checked to see if the end point had been reached, rather than both coordinates. The visualisation of the mobility model output made it easy to spot and correct this error.

Once bugs had been fixed using the test simulation area, the mobility model was used to generate paths for nodes in the actual simulation area. These paths were then verified as accurate by analysing the paths people can take in real life. This verification process used data from Google Maps, as seen in figure 12. The path from Wembley Stadium to Wembley Park Underground station generated by the mobility

model is displayed in figure 12a, with a map of the area given for comparison in figure 12b. This path corresponds to the highlighted path in figure 9.



(a) Path generated by the mobility model

(b) Real life path determined using Google Maps data

Figure 12: Shortest Path Map Based Movement mobility model correctness testing

An important aspect of the simulations in this project is the route recalculation step. This step is initiated when a node receives message packets containing information about the local area event. The functionality of the `updatePaths` function has been discussed in the Implementation section. An error with the implementation of this function was found when testing packet transmission. The test was conducted using the full 400 m × 1000 m simulation grid, but with only 6 moving nodes. Packet transmission was 'seen' by debugging messages printed to the MATLAB console. These debugging messages output the time stamp of the transmission, the nodes involved in the transmission and position of the two nodes in the simulation grid.

The nodes in the test simulation did not transmit packets as expected. Unfortunately, the plots showing the node paths did not assist in the correction of this error, as the paths generated by the mobility model were correct. Finding the cause of the issue required line by line debugging to find out why transmission was not occurring. The time stamp of each transmission was used to narrow down the scope of the debugging process. This process involved verifying the position of each node in the simulation before and after the first packet transmission occurred. The design of the mobility model made this possible as the position of each node at each time slice in the simulation is stored. The row number of the path matrix matches the time stamp of the simulation and the two columns represent the X and Y coordinates respectively. This verification process led to the discovery that upon receiving a message packet, the path of the node was recalculated correctly, but was stored incorrectly. The old path was being overwritten instead of the new path being appended to the old path. Due to the indexing system used in the simulation, this meant the position of the node would 'jump' to an incorrect position. This issue in the `updatePaths` function was fixed by making sure the new path was correctly appended to the old path.

6.2 Routing Algorithms

The main aspect of the routing algorithms that required testing was the packet transmission step. Once the initial implementation of an algorithm was complete, a small area with static nodes was simulated to verify packet transmission occurs correctly. This was done by setting two nodes to be within BLE range of each other and modifying the attributes of one node so it was in a position to transmit message packets. The expected result was packet transmission to the second node. If this result was not seen, the simulation script was debugged to find the cause of the error. This static test also allowed the data collection process to be tested, as the change in the number of packets transmitted etc. could easily be tracked.

Once small scale correctness testing with static nodes was complete, testing in the full size simulation area was carried out. The number of nodes in the simulation was kept low to make the testing process more straightforward. Setting the `debug` simulation parameter to true causes information about each packet transmission event in the simulation to be output to the MATLAB console. Listing 14 shows the debugging framework implemented for packet transmission. The same framework is used for the testing of each algorithm implemented.

```
1  if debug
2
3     % The debug flag is set in the getSimulationParams function
4     % Debug information includes:
5                           % - timestamp of transmission
6                           % - nodes involved in transmission
7                           % - positions of the nodes involved in the simulation area
8     fprintf('Time = %d \n',t);
9     fprintf('Transmitting from node %d to %d \n',src,dest);
10    fprintf('Node %d position = [%d,%d]
          \n',src,nodes{src}.current_position(1),nodes{src}.current_position(2));
11    fprintf('Node %d position = [%d,%d]
          \n',dest,nodes{dest}.current_position(1),nodes{dest}.current_position(2));
12
13 end
```

Listing 14: Code snippet showing the debugging framework implemented in the packet transmission stage of each algorithm

The output of the position of the two nodes allows for a quick verification of the transmission range of the algorithm. If the two nodes are further than 10 m apart, an error has occurred in the simulation. This is likely to have been caused by an error in the maximum transmission distance definition of the algorithm in its class file.

During the testing process, the time stamp of a transmission event along with the path of each node were used to verify that packet transmission could not occur at an earlier time. The transmission time stamp corresponds to the row number of the position matrix for all nodes in the simulation. This verification process involved inspecting the position matrices for the two nodes involved in the transmission. The rows just before the transmission occurred were analysed to check whether the two nodes were within 10 m of each other at an earlier time.

Along with testing the packet transmission simulation using moving nodes, the testing phase for the routing algorithms involved testing the data collection process. This was a key aspect of the correctness testing as the data collected is what is used as the basis for the performance comparison. This testing was done by adding breakpoints to the section of code where transmission was simulated. When the breakpoint was reached, the attribute values of the transmitting node were stored. The next line of code in the simulation was then run and the new attribute values of the transmitting node were compared with the old values. The differences in the values were then compared to the expected difference in values. If a discrepancy was found, the bug in the code was identified by line by line debugging, and then it was rectified to ensure correctness.

The GTT results obtained from initial simulations were compared to the theoretical minimum values,

calculated using equation 4, to verify the validity of the simulation results. If results below the minimum value were obtained, the simulation scripts were analysed to identify the cause of the error. On occasion this error did occur as the limit checking mechanism was incorrectly implemented. This meant setting a limit of 10 transmissions per second actually allowed 11 to occur. This error was quickly rectified in small scale testing before progressing to the final simulations.

As mentioned in the Analysis and Design and Implementation sections, the modular approach and the common framework used for each routing algorithm greatly sped up the testing process. This common framework meant that once the first algorithm was tested and found to be correct, any modifications made could also be applied to the remaining algorithms.

7 Results

This section discusses the results of the simulations run and performance will be evaluated by comparing Group Transmission Time (GTT) and power consumption. As mentioned previously in the Implementation section, the power consumption results should only be used for a comparison of the algorithms in this project. The results are unlikely to match real life performance as the simulations have been designed to slightly overestimate the power consumption of the network. The design of the mobility model results in no two simulations being identical, so 10 simulations were run with mean and median performance results used where necessary.

Simulations were run using a varying number of nodes in the simulation to assess the suitability of each algorithm for a range of node densities. The constant simulation parameters used are listed in table 1. Where simulation parameters vary, they have been explicitly detailed in this section. Figure 13 shows the graph used for the shortest path calculations. Nodes 4, 5 and 8 are the exit points simulated. The Shortest Path Map Based Movement (SPMBM) mobility model was used for all simulations.

Table 1: Simulation parameters

Parameter	**Value**
Simulation Area (m^2)	400,000
Maximum Simulation Time (s)	600
Moving Node Groups	3
Mobility Model	SPMBM
Minimum Walking Speed (m/s)	0.6
Maximum Walking Speed (m/s)	1.8
Exit Points	3
Exit Point Closures Simulated	1
Maximum Transmission Distance (m)	10
Maximum Transmissions per Second (DREAM, LMR)	10
Maximum Transmissions per Second (Reactive, Reactive Delay)	1

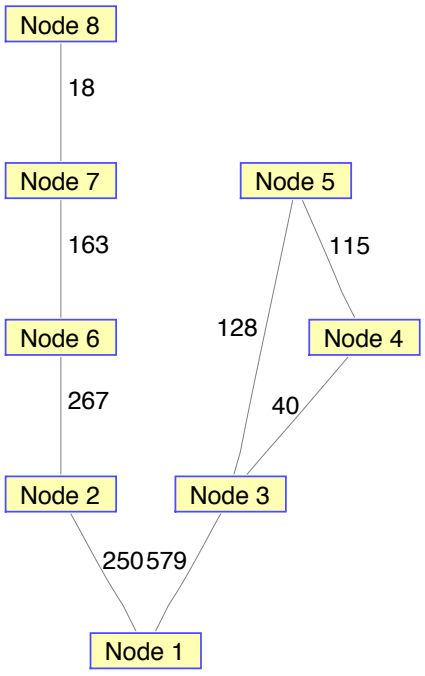


Figure 13: Graph used for the shortest path calculations in each simulation. Nodes 4, 5 and 8 are exit points with the closure of node 8 being simulated. Note: The weight of the edge connecting nodes 1 and 2 is 250 and the weight of the edge connecting nodes 1 and 3 is 579.

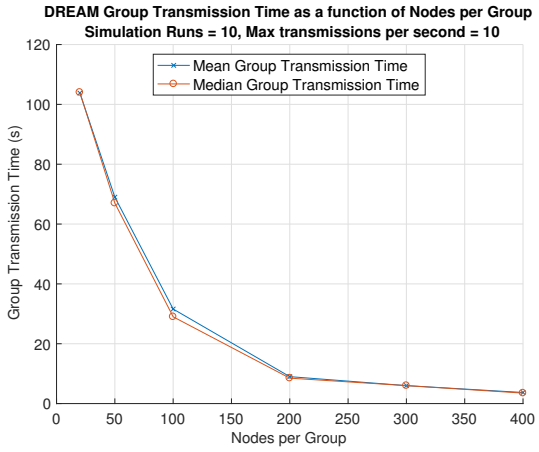
7.1 Distance Routing Effect Algorithm for Mobility (DREAM)

Results for the DREAM algorithm were obtained by running 10 simulations and taking the mean results. The median results for the 10 simulations are also used to evaluate performance if the mean value is affected by outliers. The raw simulation results can be found in tables 9 to 19. Performance metrics of Group Transmission Time (GTT) and Average Transmission Power Consumption (ATPC) are used.

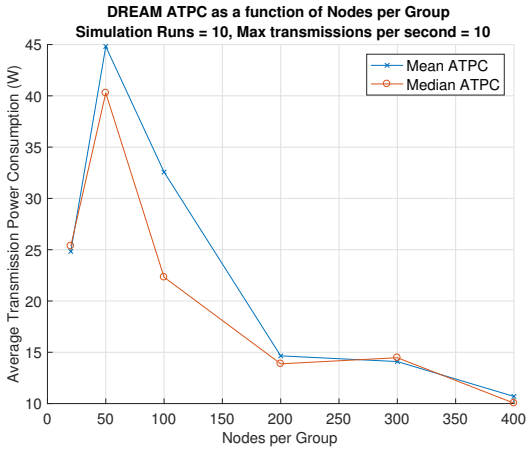
Table 2: DREAM algorithm performance when the number of nodes per group is varied

Nodes per Group	Mean GTT (s)	Median GTT (s)	Mean ATPC (W)	Median ATPC (W)
20	103.8	104.0	24.83	25.36
50	69.0	67.0	44.81	40.28
100	31.6	29.0	32.57	22.32
200	9.0	8.5	14.65	13.88
300	6.0	6.0	14.10	14.47
400	3.7	3.5	10.70	10.04

The decreasing trend for GTT seen in table 2 is expected as a dense network can propagate a message faster than a sparse network. The trend for ATPC is generally decreasing, but appears to peak at a node density of 50 nodes per group. These results are presented in figure 14. The simulations run had 3 groups of moving nodes, so a node density of 400 nodes per group means the movement of 1200 nodes was simulated.



(a) Group Transmission Time as a function of Nodes per Group for the DREAM algorithm



(b) Average Transmission Power Consumption as a function of Nodes per Group for the DREAM algorithm

Figure 14: Results for the DREAM routing algorithm as the number of nodes per group is varied

Figure 14a shows an inverse logarithmic relationship between GTT and node density for the DREAM algorithm. This is expected as increasing the density of nodes will increase the number of available routes, decreasing the overall propagation time. This decrease is physically limited due to the time taken to transmit packets in the network, giving a minimum value of approximately 3.7 seconds. In practice, a GTT of 3.7 seconds is extremely fast, so this algorithm has excellent performance in dense networks with respect to GTT. With a density of 400 nodes per group, the measured mean GTT of 3.7 seconds is close to the theoretical minimum of 2.5 seconds calculated using equation 4.

Figure 14b shows a similar logarithmic decrease in ATPC after an increase for sparse networks. This increase is seen in both the mean and median ATPC values so cannot necessarily be attributed to anomalous data points. While the performance of this algorithm in terms of ATPC increases with node density, the power consumption is still significant. While the simulations have been designed to give an overestimate for ATPC, this algorithm does not seem to be suitable for use in a network between smartphones due to the high power consumption.

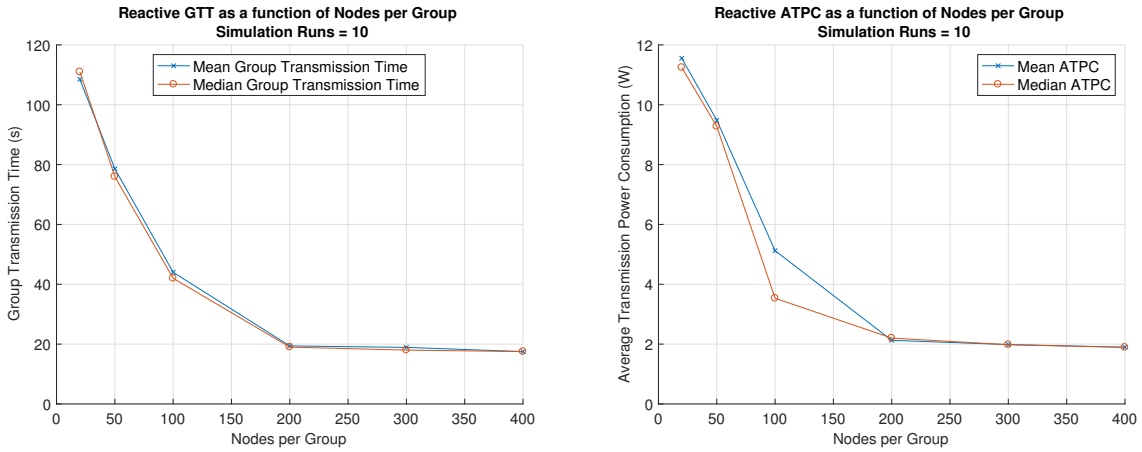
7.2 Reactive Algorithm

Results for the Reactive algorithm were obtained by running 10 simulations and taking the mean results. The median results for the 10 simulations are also used to evaluate performance if the mean value is affected by outliers. The raw simulation results can be found in tables 9 to 19. Performance metrics of Group Transmission Time (GTT) and Average Transmission Power Consumption (ATPC) are used.

Table 3: Reactive algorithm performance when the number of nodes per group is varied

Nodes per Group	Mean GTT (s)	Median GTT (s)	Mean ATPC (W)	Median ATPC (W)
20	108.5	111.0	11.55	11.25
50	78.6	76.0	9.48	9.28
100	44.0	42.0	5.12	3.53
200	19.4	19.0	2.13	2.20
300	18.9	18.0	1.99	1.98
400	17.5	17.5	1.90	1.89

Table 3 shows a clear decreasing trend for both GTT and ATPC as the number of nodes per group is increased. Figure 15 shows these results graphically. The simulations run had 3 groups of moving nodes, so a node density of 400 nodes per group means the movement of 1200 nodes was simulated.



(a) Group Transmission Time as a function of Nodes per Group for the Reactive algorithm

(b) Average Transmission Power Consumption as a function of Nodes per Group for the Reactive algorithm

Figure 15: Results for the Reactive routing algorithm as the number of nodes per group is varied

The logarithmic shape of the GTT curves in figure 15a imply a lower limit of approximately 17.5 seconds exists for the GTT of the Reactive algorithm. In practice, 17.5 seconds is a short amount of time for the propagation of an emergency message amongst a large crowd, so this algorithm has good performance. The mean and median curves are close, indicating the simulation results are consistent for the Reactive algorithm. The performance of the algorithm in terms of GTT when the node density is low is not as good, taking more than a minute to propagate the message in some cases. In terms of GTT, this algorithm appears to be more suited to dense networks, so should be used in scenarios where large crowds of people may be present. However, even with a density of 400 nodes per group, the mean GTT of 17.5 seconds is significantly higher than the theoretical minimum of 8.6 seconds, calculated using equation 4.

Figure 15b also has a logarithmic shape, implying a lower limit of approximately 1.9 W in dense networks for power consumption. This value is an overestimate; in practice the power consumption of a network using the Reactive routing algorithm for packet transmission is likely to be lower. Overall, the mean and median results are close, with a slight discrepancy for a density of 100 nodes per group. As with the GTT results, this algorithm seems more suited for use in large crowds, looking at the ATPC results.

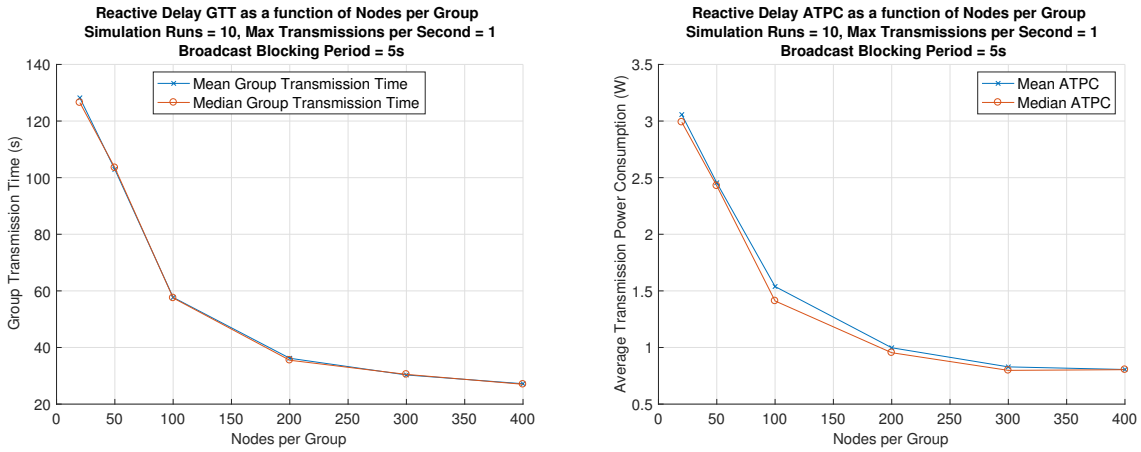
7.3 Reactive Delay Algorithm

Results for the Reactive Delay algorithm were obtained by running 10 simulations using the default blocking period of 5 seconds and taking the mean results. The median results for the 10 simulations are also used to evaluate performance if the mean value is affected by outliers. The raw simulation results can be found in tables 9 to 19. Further simulations were run where the duration of the transmission blocking period was varied to see the effect on the performance of this algorithm. Performance metrics of Group Transmission Time (GTT) and Average Transmission Power Consumption (ATPC) are used.

Table 4: Reactive Delay algorithm performance when the number of nodes per group is varied. The default broadcast blocking period of 5 seconds was used.

Nodes per Group	Mean GTT (s)	Median GTT (s)	Mean ATPC (W)	Median ATPC (W)
20	128.2	126.5	3.06	2.99
50	102.9	103.5	2.45	2.43
100	57.7	57.5	1.54	1.41
200	36.2	35.5	1.00	0.95
300	30.3	30.5	0.83	0.80
400	27.2	27.0	0.81	0.80

Table 4 shows a decreasing trend for GTT as expected. A decreasing trend is also seen for ATPC when the density of nodes in the simulation increases. Figure 16 shows these results graphically. The simulations run had 3 groups of moving nodes, so a node density of 400 nodes per group means the movement of 1200 nodes was simulated.



(a) Group Transmission Time as a function of Nodes per Group for the Reactive Delay algorithm

(b) Average Transmission Power Consumption as a function of Nodes per Group for the Reactive Delay algorithm

Figure 16: Results for the Reactive Delay routing algorithm as the number of nodes per group is varied

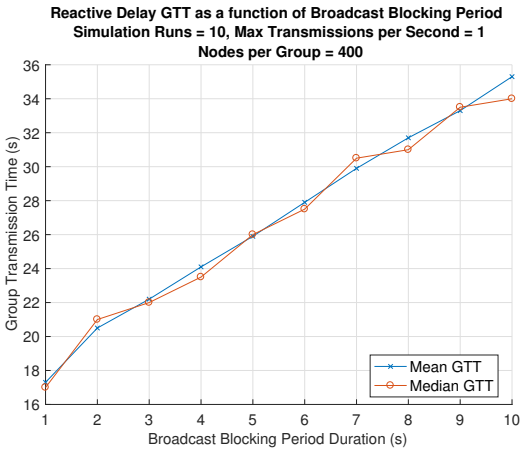
Figure 16a shows a decreasing trend for GTT with a lower limit of approximately 27.0 seconds. This is significantly higher than the theoretical limit of 8.6 seconds, calculated using equation 4. This indicates that the Reactive Delay algorithm would be suitable for alerting a dense crowd of a local area event, but performance is limited in practice by the blocking period. Performance for sparse networks is poor with a GTT of over 120.0 seconds for a node density of 20 nodes per group.

A similar decreasing logarithmic relationship is seen in figure 16b for the ATPC of the Reactive Delay algorithm, as the node density is varied. The minimum value of approximately 0.8 W indicates that this algorithm has excellent performance, with respect to power consumption, in dense networks. This is despite the ATPC results being designed to give an overestimate for the power consumption of the algorithms simulated in this project.

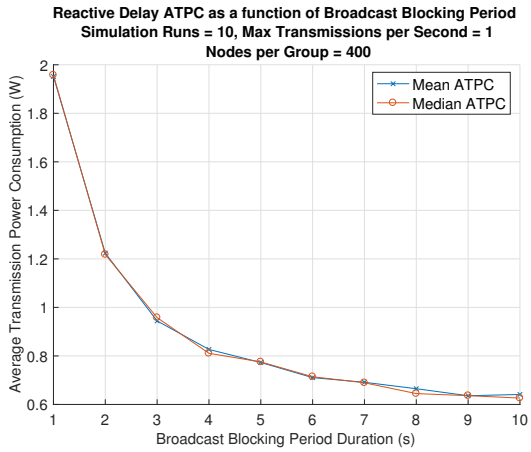
As discussed in the Analysis and Design section, the Reactive Delay algorithm was designed to have a blocking period where the nodes cannot broadcast their location to initiate route discovery or packet transmission. The default value for this blocking period of 5 seconds was used for the results presented in table 4 and figure 16. Table 5 summarises the key results obtained from varying the duration of this blocking period. In each of these simulations, a node density of 400 nodes per group is used to investigate performance for dense networks. 3 moving groups were simulated, as with previous simulations, giving a total of 1200 nodes simulated. The data is presented graphically in figure 17. The raw simulation results are given in tables 21 to 30.

Table 5: Reactive Delay algorithm performance when the number of nodes per group is varied. 10 simulation runs were used and a node density of 400 nodes per group was used.

Blocking Period Duration (s)	**Mean GTT (s)**	**Median GTT (s)**	**Mean ATPC (W)**	**Median ATPC (W)**
1	17.3	17.0	1.95	1.96
2	20.5	21.0	1.22	1.22
3	22.2	22.0	0.94	0.96
4	24.1	23.5	0.83	0.81
5	25.9	26.0	0.77	0.78
6	27.9	27.5	0.71	0.71
7	29.9	30.5	0.69	0.69
8	31.7	31.0	0.66	0.64
9	33.3	33.5	0.64	0.64
10	35.3	34.0	0.64	0.63



(a) Group Transmission Time as a function of blocking period for the Reactive Delay algorithm



(b) Average Transmission Power Consumption as a function of blocking period for the Reactive Delay algorithm

Figure 17: Results for the Reactive Delay routing algorithm as the number of nodes per group is varied

Figure 17a shows an almost linearly increasing trend for GTT as the duration of the blocking period is increased. This result is expected as nodes can transmit fewer packets with a longer blocking period, so message propagation is slower. The duration of the blocking period has a significant effect on GTT as the results range from 17.3 seconds for a 1 second duration to 35.3 seconds for a 10 second duration. These results are not close in absolute terms and they represent a 2.04x increase in relative terms.

Figure 17b shows a logarithmic decreasing trend for the ATPC of the Reactive Delay algorithm when the duration of the blocking period increases. This trend is also expected as a longer blocking period results in fewer energy intensive route request stages being initiated. However, if the blocking period was made too long, the ATPC may increase in sparse networks as potential transmissions will be missed. This could lead to more route request stages being required in the long run. The range of ATPC results is wider than the range of GTT results obtained when the blocking period duration is varied. The maximum of 1.95 W and the minimum of 0.64 W are close in absolute terms, but represent a 3.05x

increase in relative terms. A default blocking period duration of 5 seconds was selected as it gives a good compromise between GTT and ATPC.

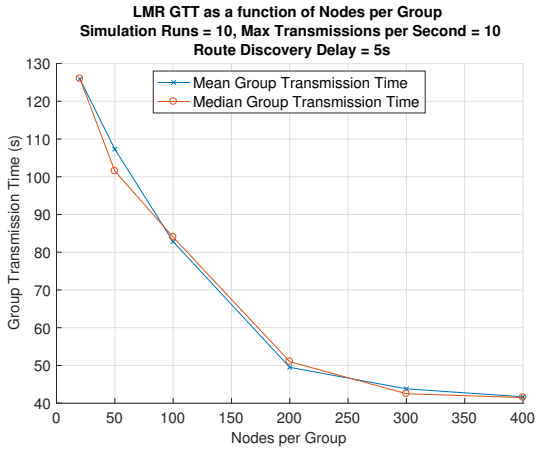
7.4 Lightweight Mobile Routing (LMR)

Results for the LMR algorithm were obtained by running 10 simulations using the default blocking period of 5 seconds and taking the mean results. The median results for the 10 simulations are also used to evaluate performance if the mean value is affected by outliers. The raw simulation results can be found in tables 9 to 19. Further simulations were run where the duration of the route discovery blocking period was varied, to see the effect on the performance of this algorithm. Performance metrics of Group Transmission Time (GTT), Average Transmission Power Consumption (ATPC) and Packet Transmission Error Rate (PTER) are used.

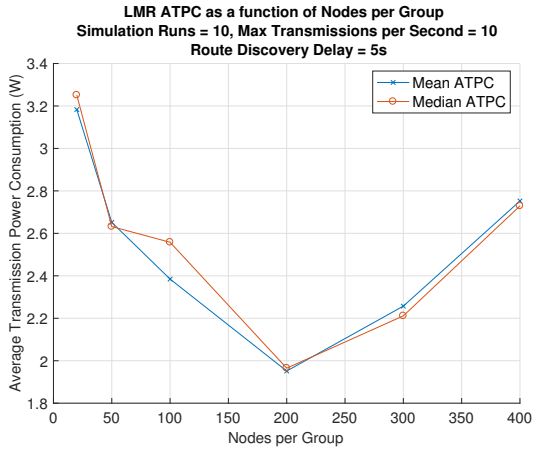
Table 6: LMR algorithm performance when the number of nodes per group is varied. The default route request blocking period of 5 seconds was used.

Nodes per Group	Mean GTT (s)	Median GTT (s)	Mean ATPC (W)	Median ATPC (W)	Mean PTER (%)	Median PTER (%)
20	125.9	126.0	3.18	3.25	2.62	0.00
50	107.3	101.5	2.65	2.63	1.77	1.27
100	82.8	84.0	2.38	2.56	1.10	1.03
200	49.5	51.0	1.95	1.97	0.56	0.53
300	43.8	42.5	2.26	2.21	0.31	0.27
400	41.7	41.5	2.75	2.73	0.31	0.24

Table 6 shows a decreasing trend for the GTT and PTER of the LMR algorithm, but shows a decrease in ATPC followed by an increase. These results are presented graphically in figures 18 and 19. The simulations run had 3 groups of moving nodes, so a node density of 400 nodes per group means the movement of 1200 nodes was simulated.



(a) Group Transmission Time as a function of Nodes per Group for the LMR algorithm



(b) Average Transmission Power Consumption as a function of Nodes per Group for the LMR algorithm

Figure 18: GTT and ATPC results for the LMR algorithm as the number of nodes per group is varied

The trend seen in figure 18a for GTT is expected as dense networks can have a higher rate of packet transmission, resulting in lower message propagation times. The LMR algorithm is slow, with a minimum GTT of approximately 41.7 seconds with a node density of 400 nodes per group. This result indicates that this algorithm may not be suitable for use in practice as the message about the local area event is likely to propagate slowly. This algorithm is not appropriate for use in sparse networks either, due to the high GTT.

Figure 18b shows an almost parabolic relationship between ATPC and node density for the LMR algorithm. This relationship is somewhat unexpected, but is not caused by anomalous data points since the median ATPC curve shows the same trend. The cause of this trend could be investigated in a further work if the LMR algorithm is deemed to be promising for real world use. However, the minimum point of the curve, 1.95 W, is high so the LMR algorithm has poor performance in terms of power consumption. This combined with the GTT results indicate that this algorithm is not suitable for the use case presented in this project.

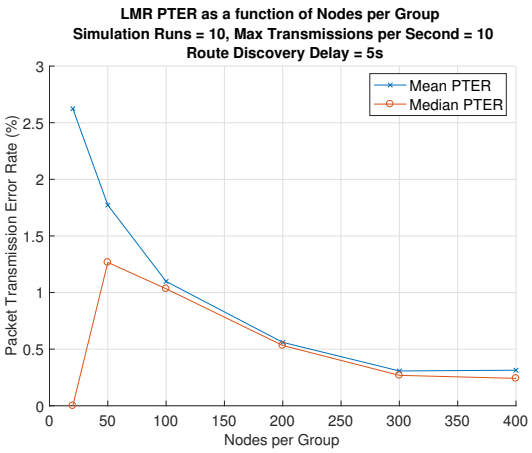


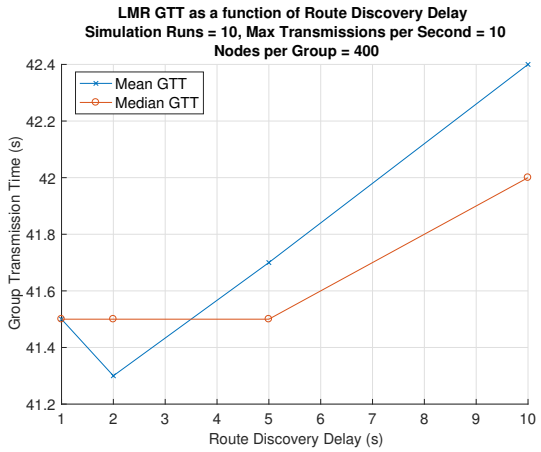
Figure 19: Packet Transmission Error Rate as a function of Nodes per Group for the LMR algorithm

As mentioned in the Analysis and Design section, the LMR algorithm differs from the others implemented as it can allow invalid routes to be present in the route cache. This can lead to transmission failure due to the invalid routes. Figure 19 displays the Packet Transmission Error Rate (PTER) results obtained from the LMR simulations. These results are calculated using equation 5. The mean result shows a decreasing trend which reaches a limit at approximately 0.3 % for a node density of 400 nodes per group. While this value is low, it is greater than the 0 % PTER achieved by the other algorithms. This result, combined with the GTT and ATPC results, indicates the LMR algorithm has poor all-round performance.

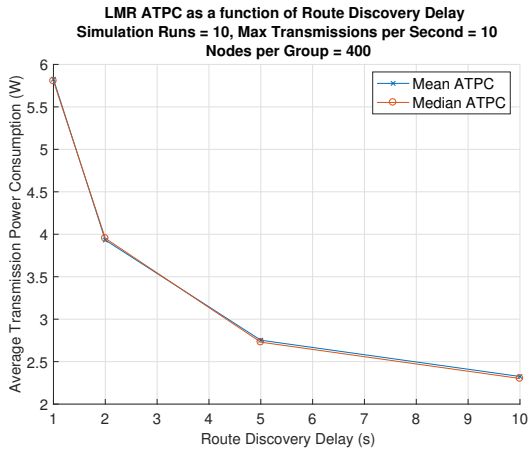
The cause of potentially invalid routes in the route cache is the route discovery delay in the LMR algorithm implemented in this project. This blocking period prevents nodes from refreshing the routes in the routing table so invalid routes persist. The default value for this blocking period of 5 seconds was used for the results presented in table 6 and figures 18 and 19. Table 7 summarises the key results obtained from varying the duration of this blocking period. The raw simulation results are given in tables 31 to 34. In each of these simulations, a node density of 400 nodes per group, with 3 moving groups giving 1200 nodes in total, is used to investigate performance for dense networks. The data is presented graphically in figures 20 and 21.

Table 7: LMR algorithm performance when the duration of the blocking period is varied. 10 simulation runs were used and a node density of 400 nodes per group was used.

Route Discovery Delay (s)	**Mean GTT (s)**	**Median GTT (s)**	**Mean ATPC (W)**	**Median ATPC (W)**	**Mean PTER (%)**	**Median PTER (%)**
1	41.5	41.5	5.83	5.80	1.37	1.50
2	41.3	41.5	3.93	3.95	1.13	1.20
5	41.7	41.5	2.75	2.73	0.31	0.24
10	42.4	42.0	2.32	2.30	0.33	0.23



(a) Group Transmission Time as a function of blocking period for the LMR algorithm



(b) Average Transmission Power Consumption as a function of blocking for the LMR algorithm

Figure 20: GTT and ATPC results for the LMR algorithm as the route discovery delay is varied

The mean GTT curve in figure 20a shows an almost linear relationship between GTT and the duration of the blocking period. This is expected as a longer blocking period results in fewer packet transmissions occurring. The median GTT curve shows a slightly different trend as the curve is flat until a blocking period of 5 seconds. The range of GTT results is small, 41.3 seconds to 42 seconds, so the duration of the blocking period can be seen to have virtually no effect on performance in terms of GTT.

Figure 20b shows an almost identical decreasing trend, for both the mean and median ATPC values, when the duration of the blocking period is varied. This trend is expected as a longer blocking period limits the number of energy intensive route discovery stages, so the ATPC is lower. Since the GTT and ATPC curves show opposite trends, the default blocking period of 5 seconds appears to give the best compromise between the two performance metrics.

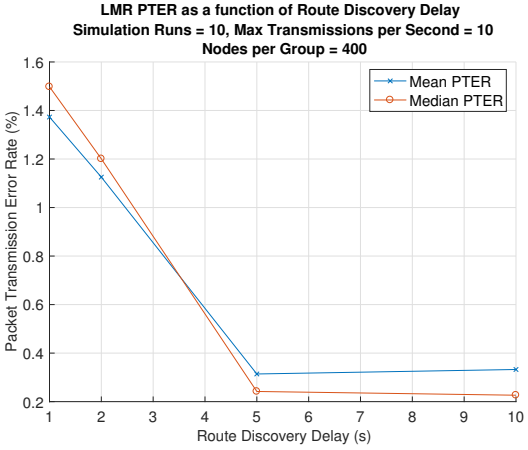


Figure 21: Packet Transmission Error Rate as a function of blocking period for the LMR algorithm

Varying the blocking period also results in the PTER varying. Figure 21 shows a linear decrease in PTER as the blocking period duration increases to 5 seconds. After 5 seconds, the curves remain roughly constant. This trend is unexpected and the cause could be investigated in a future work. The choice of blocking period duration can have a significant effect on PTER as the maximum value of 1.37% is 4.4x greater than the minimum value of 0.31%. As with the GTT and ATPC results, the PTER results indicate an optimal value for the blocking period of 5 seconds.

7.5 Performance Comparison

The results presented in detail earlier in this section have been summarised below to allow for a performance comparison between the four routing algorithms.

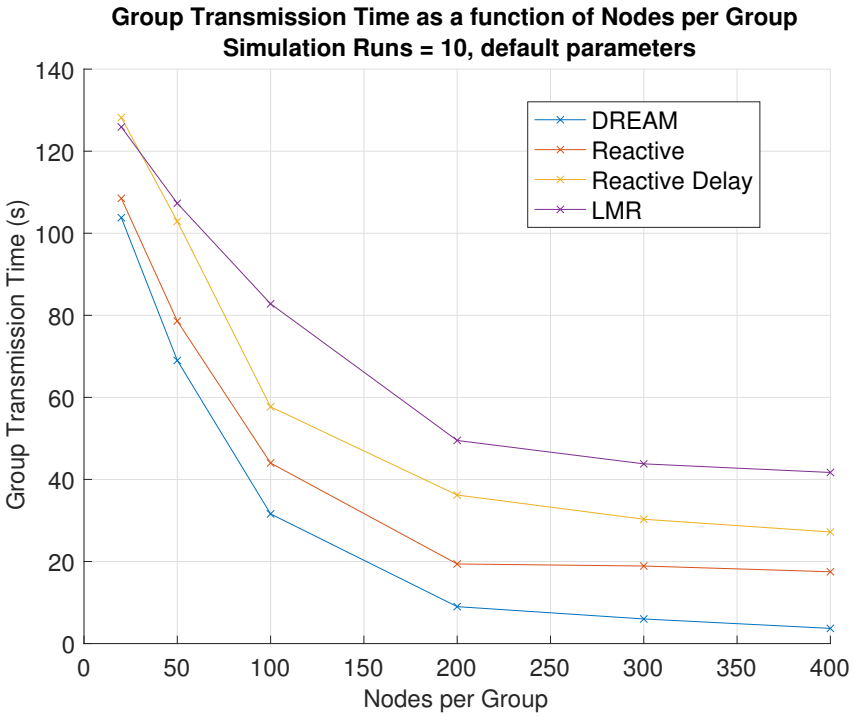


Figure 22: Group Transmission Time as a function of node density for all four algorithms simulated

Figure 22 shows that the four routing algorithms simulated in this project follow the same general trend for GTT with increasing node densities. Dense networks give objectively better performance for each algorithm in terms of GTT. This suggests that a MANET used to propagate messages about local area events is suitable when large crowds are present. In the event of people leaving a football stadium, the density of the crowd will vary over time, but will on average be high due to the capacity of the stadium.

The DREAM algorithm clearly has the best performance in terms of GTT. This is likely to be due to the active nature of the algorithm, meaning valid routes are always available. The range of GTT results for the different algorithms is large in relative terms, with the average GTT for LMR with a density of 400 nodes per group being approximately 7.3× that of the DREAM GTT.

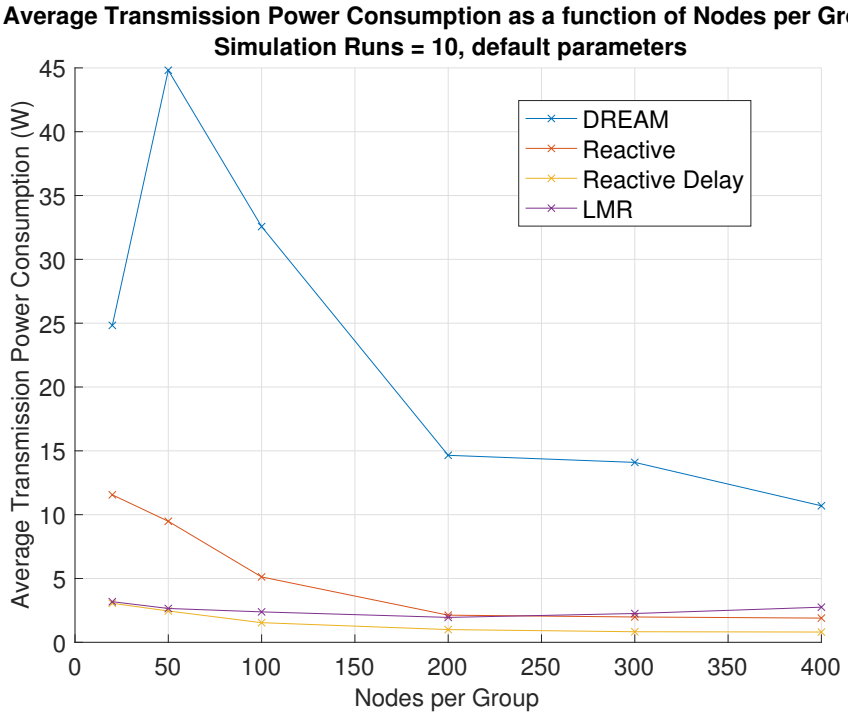


Figure 23: Average Transmission Power Consumption as a function of node density for all four algorithms simulated

Figure 23 shows that while the four routing algorithms simulated have the same general decreasing trend for ATPC, some differences are present. The DREAM algorithm shows a spike in power consumption for a node density of 50 nodes per group and the LMR algorithm shows a slight increase for dense networks. These results can be seen more clearly in the relevant subsections within this results section. The general trend suggests that a MANET used to propagate local area messages may be feasible after football matches due to the high density of the crowd leaving the stadium.

The DREAM algorithm has the worst performance in terms of ATPC, with the Reactive Delay algorithm having the best performance. The poor performance of DREAM is due to the active nature of the algorithm, which incurs a large overhead to maintain valid routes at all times.

These results suggest the most appropriate algorithm for the MANET in question is a choice between the Reactive and Reactive Delay algorithms. The lower power consumption of the Reactive Delay algorithm is offset by the faster message propagation of the Reactive algorithm. As the primary objective of the MANET is to propagate local area messages quickly, with minimising power consumption a secondary objective, the **Reactive** algorithm is the best routing algorithm to use.

8 Evaluation

This section evaluates the success of this project by determining whether the objectives outlined in section 3.1 have been met. The objectives have been reproduced here for clarity.

1. Simulate people leaving a football stadium and heading towards nearby public transport hubs through the use of a realistic mobility model.
2. Simulate a minimum of two routing algorithms in a MANET created between smartphone users in a local area. This should include at least one active and one reactive routing algorithm.
3. Measure the speed of packet transmission in the MANET. Group Transmission Time (GTT) will be defined as the time difference between the first and last relevant nodes in the network receiving the message being propagated.
4. Identify the most suitable routing algorithm for use in the MANET based on the performance metric of GTT. Power consumption should also be used where appropriate to identify the best routing algorithm.

Objective 1 focuses on the mobility model used for the simulations in this project. Overall, the Shortest Path Map Based Movement (SPMBM) mobility model gives a good approximation of the movement of people leaving a football stadium. The key features of the mobility model are implemented in this project, but there are some limitations. The main limitation arises when a node has to recalculate its path to avoid a local area event. The size of the simulation area means the graph used for the shortest path calculations cannot contain every point in the simulation area. This means nodes have to move to the nearest waypoint before the shortest path to the new end point can be calculated. This is a limitation because in practice, this may not be possible if the waypoint has been cordoned off by police for example.

As this is only a small aspect of the simulation, the SPMBM mobility model implemented does simulate the realistic movement of people leaving a football stadium, hence objective 1 has been met.

Objective 2 focuses on the number of routing algorithms implemented in this project. A minimum of two algorithms is required to allow a performance comparison to be made. The requirement to implement at least one active and one reactive routing algorithm is an addition made to objective 2 after the Interim report submission. This change was made to ensure the performance difference between active and reactive routing algorithms was investigated in this project. Without this criteria, the recommendation made regarding the best algorithm for the use case presented could be invalid, if only active or only reactive routing algorithms were considered.

A total of four routing algorithms, one active and three reactive, have been implemented in this project, so objective 2 has been met.

Objective 3 focuses on the main performance metric used to objectively assess performance. This is required to justify the choice made for the best algorithm. Initially, objective 3 defined the speed of packet transmission in the network as the time difference between the first and last nodes in the network receiving the message. This was replaced with Group Transmission Time (GTT) as it is a more relevant performance metric. This is because GTT focuses only on the nodes that will be affected by the local area event, so require the message to be received as quickly as possible. A large proportion of nodes in the simulation will not be affected by the local area event, so measuring the time taken for them to receive the message is not a good performance metric.

As seen in the Results section, GTT was measured for all the simulated algorithms, so objective 3 has been met.

Objective 4 focuses on the outcome of this project, where a recommendation is made regarding the most suitable routing algorithm to use in the MANET in question. This objective was modified slightly to use GTT as the main performance metric instead of overall transmission time. The power consumption criteria was also an addition made after the Interim report submission. This addition was made to make the recommendation of the best algorithm stronger as it would prevent an algorithm with low GTT, but

extremely high power consumption from being selected.

As detailed in the Results section, the Reactive routing algorithm was identified as the best algorithm for use in this network, after considering both GTT and power consumption. The addition of the power consumption criteria prevented the DREAM algorithm from being selected due to the significantly higher power consumption, compared to the Reactive algorithm. Since the best routing algorithm has been identified, this objective has been met.

A number of related works have investigated the use of MANETs in disaster situations to find survivors or to enable an evacuation [24] [25]. The application of the MANET in this project is slightly different. In the use case presented, the focus is not on a disaster event itself, but on alerting people in the nearby area about the incident so they can avoid the affected area. This aspect of disaster management does not appear to have as much focus due to the lower risk posed to people outside the disaster zone. However, a system such as the one proposed in this project can reduce the burden on police and other emergency services when it comes to securing an area after a disaster.

The 2017 Manchester Arena attack prompted police to set up an exclusion zone and the general public was urged to avoid the area. This message was mainly conveyed through social media and news channels. While these methods allow the message to have a wide audience, the message is not targeted. The use of a MANET where local area messages are propagated will allow people in the local area to get information about the event and re-plan their route to avoid the incident area. Large scale incidents such as the Manchester Arena attack are rare, but the system proposed in this project can be used for small scale incidents too, such as road or station closures.

A disadvantage of this project compared to some related works is the lack of real world results. A mobility model cannot model unexpected behaviour, so real world performance is likely to differ from the results obtained in this project. If this project is considered as more of a feasibility study for the use of a MANET to propagate information about a local area, the results obtained do a good job at identifying the best algorithm to use.

The recommendation made in this project regarding the best routing algorithm to use only considered four algorithms. While this is a sufficient number to conduct a comparison, it is likely that some algorithms that were not tested give better performance. However, it is unrealistic to test all existing algorithms, especially given the time constraint of this project. The four algorithms simulated had a some significant differences in design, so a good range of results were obtained.

Some of the algorithms implemented do not model all the key features of the algorithm. For example, the LMR algorithm only maintains a single route to neighbouring nodes instead of maintaining multiple routes. Similarly, some of the location-based behaviour of the algorithms was not implemented. The reason behind this was that message packets were to be transmitted to all nodes within a certain area, rather than between specific nodes. While the design of the simulation system was kept as modular as possible, some aspects limit the use of the system to the specific case of people leaving a stadium. This is sufficient for this project, but some related works produce generic simulation tools which can then be used by others. In that sense, the scope of this project is quite narrow compared to some related works.

The simulations in this project make some assumptions that may not be true in practice. An assumption is made that a system exists where smartphones with BLE connectivity can be used to form a MANET without any additional software requirements. While additional software may not be required to simply transmit packets within the MANET, some sort of user interface may be required to alert the user to the occurrence of a local area event. If the user has no way of knowing a message has been received, they will not change their path as simulated. This is only an issue if the system as proposed in this project is being implemented for real world use in its current state.

9 Conclusions and Further Work

This project has simulated a Mobile Ad Hoc network, between mobile devices with Bluetooth Low Energy connectivity, amongst a crowd of people leaving a football stadium. The Shortest Path Map Based Movement mobility model allows the realistic movement of people to be modelled with a few limitations. A range of routing algorithms, including active and reactive algorithms, have been implemented and simulated in this project. The implementation of these algorithms has captured their main features but some aspects have been ignored due to their complexity or due to their minimal relevance to this project. The design choices made have been discussed in detail in the Analysis and Design and Implementation sections. Performance metrics of Group Transmission Time and power consumption have been used to objectively compare the performance of the different algorithms. The key results from the simulations run have been summarised and discussed in the Results section of this report, where the **'Reactive'** algorithm was identified as the best performing algorithm. The Evaluation section discusses the success of this project, and as all the objectives have been met, to at least some degree, this project can be considered to have been successful.

Implementing the different routing algorithms was the most difficult part of the project. The Background section identified the key features of each algorithm, but these had to be evaluated to determine which features were relevant and possible to implement. Some of these difficulties were overcome by limiting the number of algorithms implemented to four, along with only implementing the main features of each algorithm. The design choices involved are described in detail in the Analysis and Design section, but the main trade-off was between accuracy and implementation cost. A balance was struck between the two, as having the most accurate implementation of a routing protocol isn't beneficial to this project if it takes too long to implement. The implementation was made simpler by the modular design employed throughout this project. This allowed common elements between algorithms to be reused which greatly sped up development. Debugging and testing was also simplified by this approach as the amount of testing required was reduced. This modular approach also sets up the possibility of further works extending this project as new algorithms can be implemented and simulated using the established framework.

The main learning aspect of this project relates to the routing algorithms investigated. A large number of routing algorithms were researched as part of the Background section of this report. Most of these were unfamiliar, so this project broadened my knowledge of routing algorithms. The implementation of these routing algorithms required a good understanding of the features of each one. As a consequence of this, this project also increased the depth of my knowledge about routing algorithms, especially the four implemented and simulated. The results obtained from the different algorithms also allowed me to gain an intuition regarding the expected performance of different algorithms, based on their design and features. The decision to write the simulation scripts from scratch allowed a deeper understanding of the routing algorithms involved to be achieved, than would have been possible using existing tools or simulators.

This project also required a lot of learning about object oriented programming in MATLAB. The fundamentals of object oriented programming were already familiar to me, but the MATLAB specific aspects were not. A significant proportion of time spent on the implementation of the routing algorithms was spent designing the classes required and the interactions between objects. This involved quite a steep learning curve, as some of the object oriented programming aspects in MATLAB were difficult to understand at first. Once the first algorithm was implemented and tested, it was used as a framework around which the remaining algorithms were developed. The use of this framework greatly sped up development and I would consider this the most clever part of the project.

Two key design choices in the implementation stage of this project saved a lot of time in the long run. The first is to do with the mobility model. The design decision to create a path for each node, where the position at every second of the simulation is stored, made testing both the mobility model and the routing algorithms significantly easier. It also simplified the simulations as the route recalculation step allowed for some of the original route to be kept, again making testing far more straightforward. This design choice also opens up the possibility of a further work that increases the realism of the mobility model by building on the existing code.

The second design choice is a combination of using object oriented programming to implement each

algorithm and using a cell array data structure to store all the node objects. The use of different classes for each routing algorithm allowed the different features of each algorithm to be implemented, while appearing as a black box to the rest of the simulation. This allowed for modifications to the algorithms to be made without causing issues in the rest of the simulation. The use of a cell array, along with the group system seen in figure 8, allowed for a consistent method for network monitoring regardless of the node density in the MANET. This allowed a range of simulations to be run with minimal modification required, meaning results were easy to obtain using a wide range of simulation parameters.

Since the scope of this project was limited to investigating the performance of existing routing algorithms, nothing entirely novel was invented. Some of the algorithms implemented have slight differences from their description in the literature, but they would likely be considered derivative works rather than something novel. The use case for the MANET appears to be relatively new, as most related works focus on evacuating people in disaster situations, rather than helping people avoid such situations once they have occurred.

A number of opportunities for further work building on this project exist. In the simulations run in this project, the initial message transmission is between a stationary node and a moving node in the crowd. The stationary node is assumed to be a Bluetooth beacon of some description in the London Underground station that has closed. This can be expanded on to include beacons set up by local police to increase the speed of message propagation. These can be mobile nodes carried by the police while they move towards the oncoming crowd, to propagate the message faster. Another more complex simulation could investigate the processing of conflicting information propagated by the MANET. This situation could arise where the message being propagated by the network has changed as a response to the local area event. This could model a local area event, such as a building fire, where the emergency services may offer different advice as they receive more information.

Due to the large number of routing algorithms available, it is quite likely that an algorithm with better performance than the 'Reactive' routing algorithm exists for this use case. A further work could investigate a range of different routing algorithms to identify one with better performance. A specific routing algorithm could also be developed for this use case to maximise performance. These further works could utilise the simulation framework designed in this project to reduce the amount of development required.

A further work could use modified versions of the routing algorithms implemented in this project to see if performance improves. A straightforward modification would be a power limited version of each of the four algorithms already implemented. These modified algorithms would prevent nodes from transmitting messages if its power consumption was above a specified threshold. The effect on performance can then be seen by comparing results of the modified algorithms to the results obtained in this project.

The use of MATLAB for simulations in this project means a lot of existing functions and toolboxes are available for use if required, but performance is not great in terms of simulation time. The DREAM algorithm in particular takes a significant amount of time to simulate due to the 2D iteration space involved. A further work could consider moving the simulations to Python as this would allow the use of cloud computing services, such as Amazon Web Services (AWS). A number of Python libraries already exist that replicate functionality found in MATLAB. Using AWS could significantly reduce the time taken for simulations to run, which would allow a larger range of parameters to be tested. The use of AWS would also open up the opportunities to use Graphics Processing Units (GPUs) to run the simulations as this can be faster than using a Central Processing Unit (CPU) only. However, the code in this project would have to be modified and written specifically to take advantage of the power of the GPUs available. This may require the use of OpenCL in conjunction with Python which may require significant development time.

10 User Guide

The source code for this project can be cloned from `https://github.com/psachinl/FYP.git` or `git@github.com:psachinl/FYP.git` using SSH. At the time of writing, this repository is private, but will be made public in the near future. Once the source code has been downloaded, the containing folder needs to be added to the MATLAB path. This can be done by right clicking the folder in the 'Current Folder' pane on the left of the MATLAB window and selecting 'Add to Path' then 'Selected Folders and Subfolders'. This step is required as functions in different folders are required to run the simulation scripts.

Table 8: Corresponding simulation scripts and class files for each routing algorithm implemented in this project

Routing Algorithm	Simulation Script	Class File
DREAM	Transmission/active.m	Transmission/DREAMNode.m
Reactive	Transmission/reactive.m	Transmission/ReactiveNode.m
Reactive Delay	Transmission/reactive_delay.m	Transmission/ReactiveDelayNode.m
LMR	Transmission/lmr.m	Transmission/LMRNode.m

Table 8 shows the location of the simulation scripts. The active.m script runs the simulation of the DREAM routing algorithm. The scripts reactive.m, reactive_delay.m and lmr.m run the simulations for the Reactive, Reactive Delay and LMR algorithms respectively. The simulation parameters for each algorithm are defined in the `getSimulationParams` function in the 'Transmission' folder. Simulation parameters should only be modified within this function to allow batch simulations using different algorithms to run using the same parameters. If more parameters are added to the `getSimulationParams` function, the initial step of each simulation script must be modified so the new parameters are correctly assigned for the simulation.

```
function params = getSimulationParams
   % Function that returns simulation parameters

   % Graph parameters
   edge_start_points = [1 3 3 2 6 1 7 4 7 8];
   edge_end_points = [3 4 5 6 7 2 6 5 8 7];
   edge_weights = [579 40 128 267 163 250 0 115 18 0];
   start_node = [1,2,1,2]; % Array of start points for each group
   end_node = [4,8,5,8]; % End points
   min_speed=[0.6,0.6,0.6,0.6]; % Min and max speeds for each group
   max_speed=[1.8,1.8,1.8,1.8];
   map_node_positions = [340,440; 267,181; 340,919; 360,1000; 400,1000; 0,181; 0,18;
       0,0];

   % Debugging flags
   debug = false; % If true, text printed to console
   plot_path = false; % If true, initial node paths are plotted

   % If result flag is true, final result is printed to console
   print_timing_result = true;
   print_power_result = true;

   % If quit flag is true, the simulation will exit once all nodes in the test
   % group have recieved the message
   quit_simulation_early = true;

end
```

Listing 15: Code snippet showing the graph parameters that can be modified for each simulation. Some sections of this function have been removed for readability.

The graph parameters in the `getSimulationParams` function (listing 15) can be modified to use a graph different to figure 13. The start and end points for an edge can be specified and the corresponding edge weight can be added to the `edge_weights` vector. The location of each node in the graph is stored in the `map_node_positions` vector. If more nodes are added to the simulation area, the positions must be added to this vector along with the edge weights if an edge exists between the new and existing nodes. The `start_node` and `end_node` vectors contain the start and end points for each moving group in the simulation. For example, the end point for the third group can be changed by modifying the third entry in the `end_node` vector. The same indexing system is used for the minimum and maximum speeds for each group whereby the index of the `min_speed` and `max_speed` vectors correspond to the group number. The number of entries in each vector specifying group attributes can be greater than the number of moving groups, but cannot be fewer. Having fewer entries will give runtime errors as the simulation scripts will attempt to access a value that does not exist.

The `getSimulationParams` function also contains a number of 'flag' variables that affect the simulations. Setting the `debug` flag to true results in information about each transmission event being printed to the console. This information is useful for correctness testing, but can interfere with the output of simulation results. Therefore, this flag should only be set to true during testing or debugging and not during final simulations. The `plot_path` flag will display plots of the initial paths for each moving node in the simulation. This is useful for testing and debugging and can be used in final simulations too as it does not affect the output of results. During debugging, it may be beneficial to set the result flags to 'false', but these flags must be set to 'true' for final simulations for the results to be displayed on the MATLAB console. If simulations are run individually, the results will still be accessible with the result flags set to 'false', but will not be printed to the MATLAB console. For batch simulation runs, these result flags must be set to 'true' or the results will be cleared between simulation runs.

Individual simulations can be run by simply running the corresponding MATLAB script (table 8). The MATLAB scripts provided also allow for batch simulations to run. Each algorithm simulation can be run a defined number of times using the batch version of each script e.g. `active_batch.m` for the DREAM algorithm. This function has the number of simulations to run as an input parameter and simply runs the original simulation script a number of times. The `simulations_batch.m` script can be used to run batch simulations for each algorithm. This script is useful for collecting large sets of data at a time, but the result flags must be set to 'true' in the `getSimulationParams` function. These batch scripts are found in the 'Transmission' folder along with the individual simulation scripts. As the simulation scripts were designed to run individually, no plots are automatically generated by running the batch scripts. The results collected need to be entered into scripts in the 'Results' folder to generate the output plots, as discussed later in this section.

Blocking period durations can be modified in the corresponding class file for the relevant routing algorithms. The corresponding class files for each algorithm are given in table 8.

As mentioned earlier in this section, the graph used for simulations can be modified. This should first be tested using the `plotshortestpath_test` script in the 'Mobility' folder. The important parts of this script are given in listing 16.

```
1  start_points = [1 3 3 2 6 1 7 4 7 8]; % Edge start points
2  end_points = [3 4 5 6 7 2 6 5 8 7]; % Edge end points
3  Weights = [579 40 128 267 163 250 0 115 18 0]; % Edge weights
4  DG = sparse(start_points,end_points,Weights); % Directed Graph
5  % First vector shows start point for edge, second shows end point e.g. edge
6  % from node 1 to node 3. Weights shows the weight of each edge
7
8  UG = tril(DG + DG') % Generate Undirected Graph
9  h = view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))
10 [dist,path,pred] = graphshortestpath(UG,1,4,'directed',false) % Shortest path from node
       1 to 4
11 set(h.Nodes(path),'Color',[1 0.4 0.4]) % Changes the colour of the nodes on the
       shortest path for clarity
12 fowEdges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
```

```
13 revEdges = getedgesbynodeid(h,get(h.Nodes(fliplr(path)),'ID'));
14 edges = [fowEdges;revEdges];
15 set(edges,'LineColor',[1 0 0])
16 set(edges,'LineWidth',1.5)
```

Listing 16: Code snippet showing how graph parameters should be tested using the `plotshortestpath_test` function before modifying the simulation graph

This test script requires the start and end points of each edge to be defined, along with the edge weights. An undirected graph is then generated from the parameters defined and the shortest path between two nodes can be viewed. The parameters on line 10 in listing 16 can be modified to verify the shortest path between different nodes in the simulation. Lines 11 to 16 change the colour of the shortest path so it can easily be seen in the figure generated by the script. The colour can be modified by changing the coefficients in the vector on line 11. The coefficients are in RGB format, but can be replaced by a standard MATLAB short name or long name for colours. Testing using this script is required to ensure the modifications made to the graph are correct before simulations can occur.

The 'Results' folder contains raw simulation results along with some MATLAB scripts used to plot the results. The `plot_<N>pg` scripts are used to plot the results obtained when a density of N nodes per group are used in the simulations. In these scripts, the raw results are stored in a cell array for each algorithm used for ease of use. This cell array also contains other results obtained from the raw simulation results, as seen in listing 17.

```
1
2  active{1} = [ % DREAM GTT
3     105
4     112
5     137
6     105
7     77
8     96
9     97
10    116
11    103
12    90
13    ];
14
15 active{2} = mean(active{1}); % Mean GTT
16 active{3} = median(active{1}); % Median GTT
17 active{4} = range(active{1}); % Range of GTT values
```

Listing 17: Code snippet showing how raw data is entered into the plotting scripts.

The same process is used for power consumption results and the results for each algorithm use this same format. This script then plots the GTT, ATPC and PTER results for each simulation run.

The `plot_<algorithm>` scripts are used to plot the results for a single algorithm. This includes the mean and median results obtained from the `plot_<N>pg` scripts. These scripts can be used to generate plots similar to figure 14a and figure 14b. These scripts can easily be modified to account for new data from the same simulations. These scripts are also used to plot the results of the blocking period sweeps for the relevant algorithms. The data structures used are straightforward so new data can easily be added for different blocking periods for example.

11 References

[1] D. P. I. I. Ismail and M. H. F. Ja'afar, "Mobile ad hoc network overview," in *Applied Electromagnetics, 2007. APACE 2007. Asia-Pacific Conference on*, pp. 1–8, IEEE, 2007.

[2] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *SIGCOMM Comput.Commun.Rev.*, vol. 24, pp. 234–244, oct 1994.

[3] A. Sachdev, K. Mehta, and L. Malik, "Design of protocol for cluster based routing in vanet using fire fly algorithm," in *Engineering and Technology (ICETECH), 2016 IEEE International Conference on*, pp. 490–495, IEEE, 22-27 May 2016.

[4] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," *Ad Hoc Networks*, vol. 2, pp. 1–22, 1 2004.

[5] S. Murthy and J. J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *Mob.Netw.Appl.*, vol. 1, pp. 183–197, oct 1996.

[6] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE personal communications*, vol. 6, no. 2, pp. 46–55, 1999.

[7] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*, pp. 62–68, IEEE, 2001.

[8] L. Feng, *Break link prediction of location-based routing in the VANET*. PhD thesis, 2011.

[9] S.-C. M. Woo and S. Singh, "Scalable routing protocol for ad hoc networks," *Wireless Networks*, vol. 7, no. 5, pp. 513–529, 2001.

[10] T. Camp and Y. Liu, "An adaptive mesh-based protocol for geocast routing," *Journal of Parallel and Distributed computing*, vol. 63, no. 2, pp. 196–213, 2003.

[11] H.-S. K. Young-Chul Shim, "An efficient geocast algorithm using 2-hop neighbour knowledge in sensor networks," *Int. J Latest Trends Computing*, vol. 2, no. 4, pp. 521–526, 2011.

[12] P. Phoummavong, K. Utsu, C. O. Chow, and H. Ishii, "Location-aided route discovery mechanism based on two-hop neighbor information for ad hoc network," *The Journal of Supercomputing*, vol. 72, no. 3, pp. 1201–1214, 2016.

[13] B. Karp and H.-T. Kung, "Gpsr: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 243–254, ACM, 2000.

[14] E. Hyytiä and J. Virtamo, "Random waypoint mobility model in cellular networks," *Wireless Networks*, vol. 13, no. 2, pp. 177–188, 2007.

[15] A. Keranen, "Opportunistic network environment simulator," *Special Assignment report, Helsinki University of Technology, Department of Communications and Networking*, 2008.

[16] A. Al-Akkad, *Working Around Disruptions of Network Infrastructures: Mobile Ad-Hoc Systems for Resilient Communication in Disasters*. Springer, 2016.

[17] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen, "How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, pp. 232–237, IEEE, 2012.

[18] K. Nair, J. Kulkarni, M. Warde, Z. Dave, V. Rawalgaonkar, G. Gore, and J. Joshi, "Optimizing power consumption in iot based wireless sensor networks using bluetooth low energy," in *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on*, pp. 589–593, IEEE, 2015.

[19] G. Corbellini, S. Schmid, and S. Mangold, "Two-way communication protocol using bluetooth low energy advertisement frames," in *Proceedings of the 1st International Workshop on Experiences with the Design and Implementation of Smart Objects*, (New York, NY, USA), pp. 19–24, ACM, 2015.

[20] R. Schrader, T. Ax, C. Röhrig, and C. Fühner, "Advertising power consumption of bluetooth low energy systems," in *Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), 2016 3rd International Symposium on*, pp. 62–68, IEEE, 2016.

[21] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with wi-fi direct: overview and experimentation," *IEEE wireless communications*, vol. 20, no. 3, pp. 96–104, 2013.

[22] W.-F. Alliance, "How far does a wi-fi direct connection travel?." Available From: `http://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel`.

[23] Apple, "Multipeerconnectivity." Available From: `https://developer.apple.com/reference/multipeerconnectivity`.

[24] C. Okada, R. Miyamoto, A. Yamane, T. Wada, K. Ohtsuki, and H. Okada, "A novel urgent communications technologies for sharing evacuation support information in panic-type disasters," in *Networking and Services (ICNS), 2010 Sixth International Conference on*, pp. 162–167, IEEE, 2010.

[25] Y. Hayakawa, K. Mori, Y. Ishida, K. Tsudaka, T. Wada, H. Okada, and K. Ohtsuki, "Development of emergency rescue evacuation support system in panic-type disasters," in *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pp. 52–53, IEEE, 2012.

[26] R. C. Browning, E. A. Baker, J. A. Herron, and R. Kram, "Effects of obesity and sex on the energetic cost and preferred speed of walking," *Journal of applied physiology*, vol. 100, no. 2, pp. 390–398, 2006. http://jap.physiology.org/content/100/2/390.full.pdf.

[27] M. Samson, A. Crowe, P. D. Vreede, J. Dessens, S. Duursma, and H. Verhaar, "Differences in gait parameters at a preferred walking speed in healthy subjects due to age, height and body weight," *Aging Clinical and Experimental Research*, vol. 13, no. 1, pp. 16–21, 2001.

[28] M. Department for Culture and Sport, *Guide to Safety at Sports Grounds*. United Kingdom: The Stationery Office, 5th ed., 2008. Available from: `http://www.safetyatsportsgrounds.org.uk/sites/default/files/publications/green-guide.pdf`.

[29] R. A. Smith, "Density, velocity and flow relationships for closely packed crowds," February 1995 1995. ID: 271730271730.

[30] Z. Fang, S. M. Lo, and J. A. Lu, "On the relationship between crowd density and movement velocity," *Fire Safety Journal*, vol. 38, pp. 271–283, 4 2003.

[31] K. Karami, "Maximising ble throughput on ios and android," 2016. Available from: `https://punchthrough.com/blog/posts/maximizing-ble-throughput-on-ios-and-android`.

12 Appendices

A Source Code

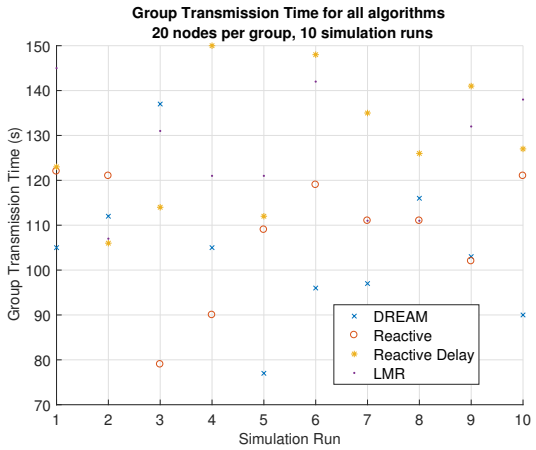
The source code for this project can be cloned from `https://github.com/psachinl/FYP.git` or `git@github.com:psachinl/FYP.git` using SSH. At the time of writing, this repository is private, but will be made public in the near future. An electronic version will also be submitted along with this report. The User Guide section details how the source code can be run or modified for future works.

B Raw Simulation Results

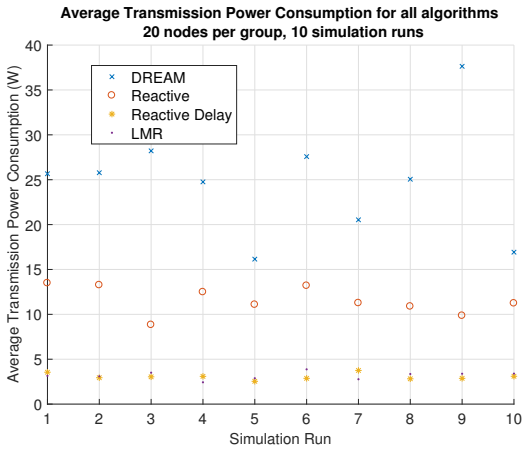
Main Simulation Results

Table 9: Raw results for all algorithms with **20** nodes per group

Routing Algorithm	Simulation Run	Group Transmission Time (s)	Average Transmission Power Consumption (W)
DREAM	1	105.0	25.6650
	2	112.0	25.7787
	3	137.0	28.2150
	4	105.0	24.7575
	5	77.0	16.1513
	6	96.0	27.5750
	7	97.0	20.5313
	8	116.0	25.0525
	9	103.0	37.6337
	10	90.0	16.9262
Reactive	1	122.0	13.4800
	2	121.0	13.2600
	3	79.0	8.8300
	4	90.0	12.4800
	5	109.0	11.0800
	6	119.0	13.1900
	7	111.0	11.2600
	8	111.0	10.8800
	9	102.0	9.8500
	10	121.0	11.2300
Reactive Delay	1	123.0	3.5500
	2	106.0	2.9400
	3	114.0	3.0400
	4	150.0	3.0900
	5	112.0	2.5400
	6	148.0	2.8700
	7	135.0	3.7500
	8	126.0	2.8200
	9	141.0	2.8700
	10	127.0	3.1000
LMR	1	145.0	3.1500
	2	107.0	3.0950
	3	131.0	3.5000
	4	121.0	2.4300
	5	121.0	2.8800
	6	142.0	3.8725
	7	111.0	2.7700
	8	111.0	3.3525
	9	132.0	3.3825
	10	138.0	3.3975



(a) Raw Group Transmission Time results



(b) Raw Average Transmission Power Consumption results

Figure 24: Raw GTT and ATPC results for simulations with a density of 20 nodes per group

Table 10: Raw LMR results with **20** nodes per group

Simulation Run	Packet Transmission Error Rate (%)
1	0.0000
2	0.0000
3	0.0000
4	0.0000
5	3.8462
6	0.0000
7	5.0000
8	0.0000
9	8.6957
10	8.6957

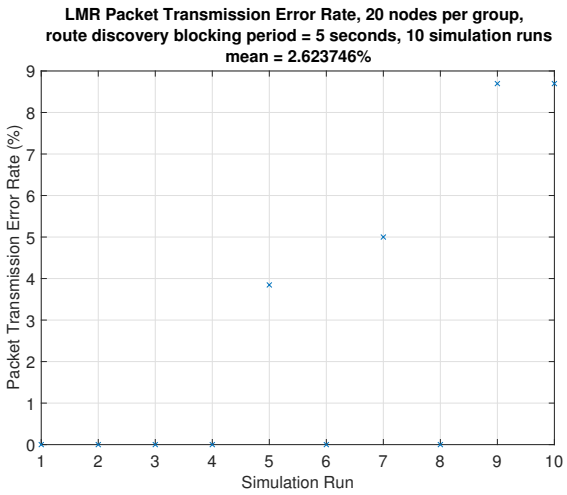
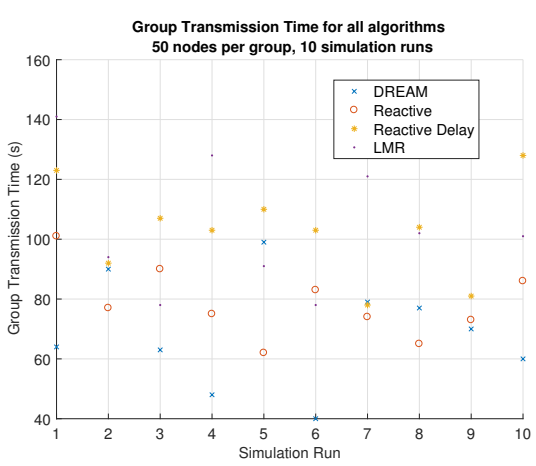


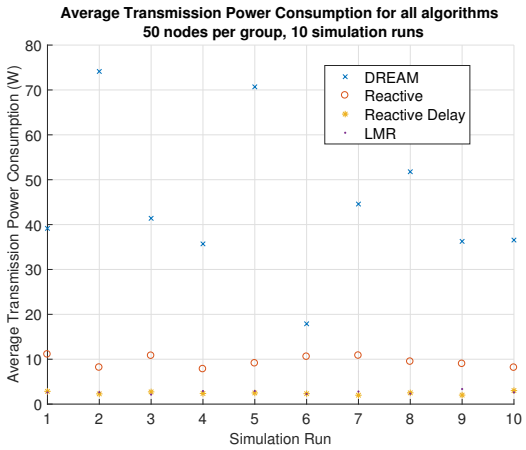
Figure 25: Raw LMR Packet Transmission Error Rate results with a density of 20 nodes per group

Table 11: Raw results for all algorithms with **50** nodes per group

Routing Algorithm	Simulation Run	Group Transmission Time (s)	Average Transmission Power Consumption (W)
DREAM	1	64.0	39.1540
	2	90.0	74.1245
	3	63.0	41.3985
	4	48.0	35.7015
	5	99.0	70.6965
	6	40.0	17.8930
	7	79.0	44.5765
	8	77.0	51.7760
	9	70.0	36.2585
	10	60.0	36.5520
Reactive	1	101.0	11.0760
	2	77.0	8.1440
	3	90.0	10.7800
	4	75.0	7.8000
	5	62.0	9.0920
	6	83.0	10.5600
	7	74.0	10.8120
	8	65.0	9.4640
	9	73.0	8.9720
	10	86.0	8.1200
Reactive Delay	1	123.0	2.8440
	2	92.0	2.2800
	3	107.0	2.7240
	4	103.0	2.3720
	5	110.0	2.4840
	6	103.0	2.3240
	7	78.0	1.9800
	8	104.0	2.5120
	9	81.0	2.0120
	10	128.0	3.0160
LMR	1	141.0	2.5790
	2	94.0	2.6520
	3	78.0	2.1220
	4	128.0	2.9040
	5	91.0	2.9200
	6	78.0	2.2560
	7	121.0	2.7980
	8	102.0	2.3160
	9	139.0	3.3570
	10	101.0	2.6120



(a) Raw Group Transmission Time results



(b) Raw Average Transmission Power Consumption results

Figure 26: Raw GTT and ATPC results for simulations with a density of 50 nodes per group

Table 12: Raw LMR results with **50** nodes per group

Simulation Run	Packet Transmission Error Rate (%)
1	1.2500
2	0.0000
3	1.2821
4	2.3529
5	0.0000
6	6.4935
7	2.5974
8	0.0000
9	3.7500
10	0.0000

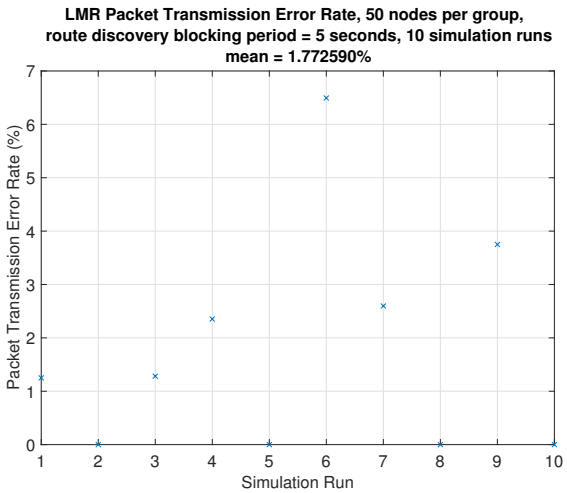
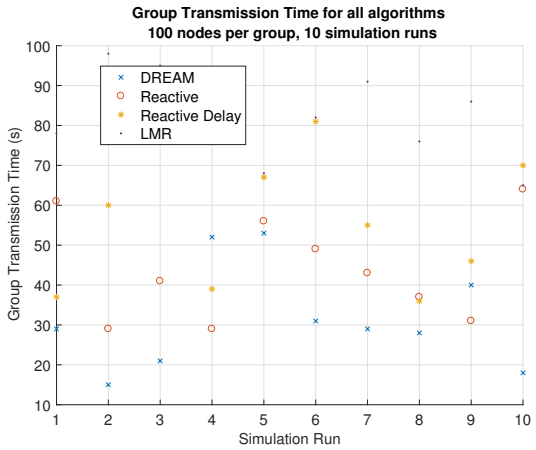


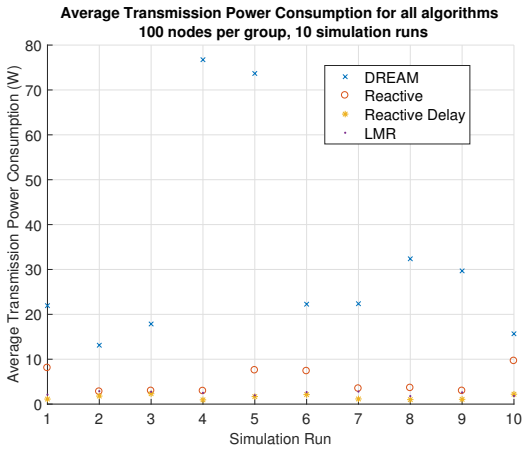
Figure 27: Raw LMR Packet Transmission Error Rate results with a density of 50 nodes per group

Table 13: Raw results for all algorithms with **100** nodes per group

Routing Algorithm	**Simulation Run**	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
DREAM	1	29.0	21.9380
	2	15.0	13.1195
	3	21.0	17.8523
	4	52.0	76.7390
	5	53.0	73.6710
	6	31.0	22.2653
	7	29.0	22.3693
	8	28.0	32.3768
	9	40.0	29.6837
	10	18.0	15.6723
Reactive	1	61.0	8.0580
	2	29.0	2.7740
	3	41.0	2.9480
	4	29.0	2.9160
	5	56.0	7.5400
	6	49.0	7.3600
	7	43.0	3.4600
	8	37.0	3.6040
	9	31.0	2.9500
	10	64.0	9.6240
Reactive Delay	1	37.0	1.1200
	2	60.0	1.7920
	3	86.0	2.3120
	4	39.0	0.9480
	5	67.0	1.6740
	6	81.0	2.1160
	7	55.0	1.1480
	8	36.0	1.0000
	9	46.0	1.0660
	10	70.0	2.2100
LMR	1	78.0	2.1045
	2	98.0	2.8790
	3	95.0	2.7460
	4	89.0	2.5010
	5	68.0	1.9645
	6	82.0	2.6615
	7	91.0	2.7875
	8	76.0	1.7800
	9	86.0	2.6150
	10	65.0	1.8085



(a) Raw Group Transmission Time results



(b) Raw Average Transmission Power Consumption results

Figure 28: Raw GTT and ATPC results for simulations with a density of 100 nodes per group

Table 14: Raw LMR results with **100** nodes per group

Simulation Run	Packet Transmission Error Rate (%)
1	1.6327
2	0.8000
3	0.8097
4	0.9259
5	2.2422
6	1.1407
7	1.1858
8	0.4405
9	0.3333
10	1.4706

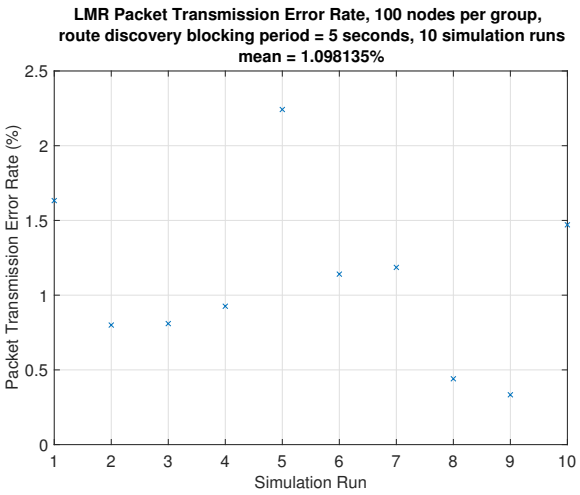
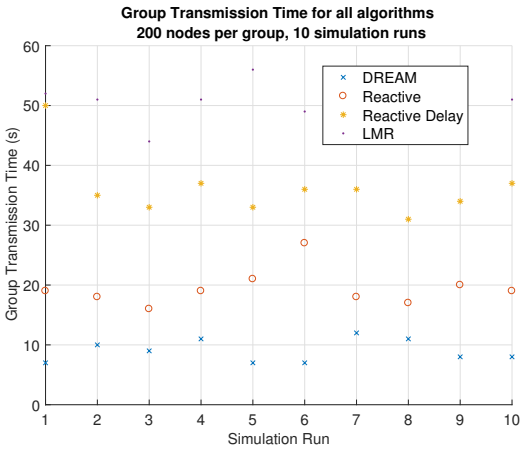


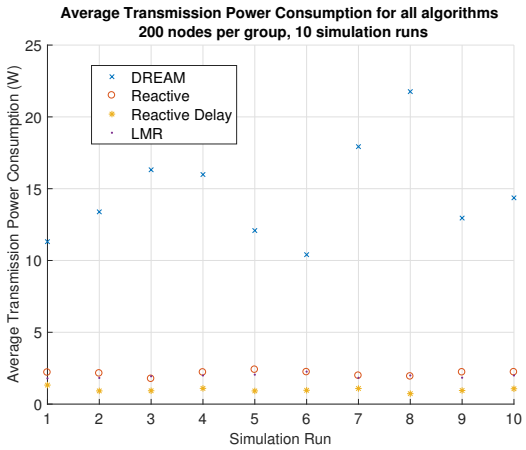
Figure 29: Raw LMR Packet Transmission Error Rate results with a density of 100 nodes per group

Table 15: Raw results for all algorithms with **200** nodes per group

Routing Algorithm	Simulation Run	Group Transmission Time (s)	Average Transmission Power Consumption (W)
DREAM	1	7.0	11.3111
	2	10.0	13.3870
	3	9.0	16.3165
	4	11.0	15.9857
	5	7.0	12.0858
	6	7.0	10.4057
	7	12.0	17.9280
	8	11.0	21.7601
	9	8.0	12.9548
	10	8.0	14.3640
Reactive	1	19.0	2.1980
	2	18.0	2.1380
	3	16.0	1.7610
	4	19.0	2.2060
	5	21.0	2.3960
	6	27.0	2.2270
	7	18.0	1.9790
	8	17.0	1.9270
	9	20.0	2.2150
	10	19.0	2.2210
Reactive Delay	1	50.0	1.3250
	2	35.0	0.9190
	3	33.0	0.9290
	4	37.0	1.0950
	5	33.0	0.9180
	6	36.0	0.9610
	7	36.0	1.0870
	8	31.0	0.7270
	9	34.0	0.9470
	10	37.0	1.0690
LMR	1	52.0	1.7942
	2	51.0	1.8228
	3	44.0	1.9365
	4	51.0	1.9935
	5	56.0	2.0500
	6	49.0	2.2605
	7	51.0	1.8300
	8	46.0	1.9950
	9	44.0	1.8390
	10	51.0	1.9935



(a) Raw Group Transmission Time results



(b) Raw Average Transmission Power Consumption results

Figure 30: Raw GTT and ATPC results for simulations with a density of 200 nodes per group

Table 16: Raw LMR results with **200** nodes per group

Simulation Run	Packet Transmission Error Rate (%)
1	0.4756
2	0.4878
3	0.2299
4	1.0067
5	0.6579
6	0.5769
7	0.3555
8	0.7592
9	0.2484
10	0.8018

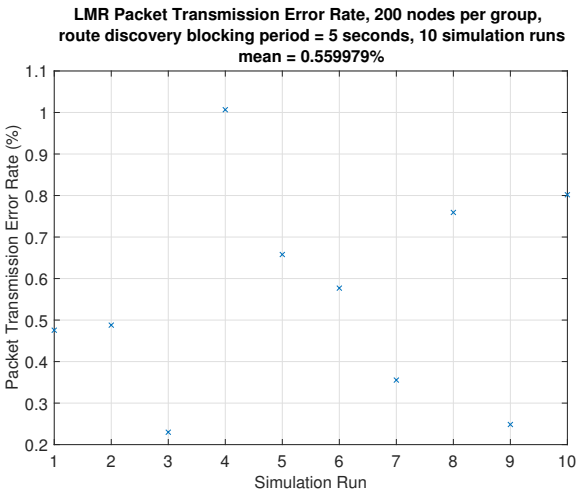
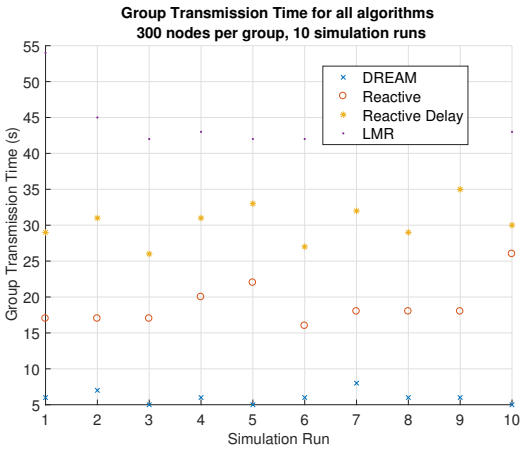


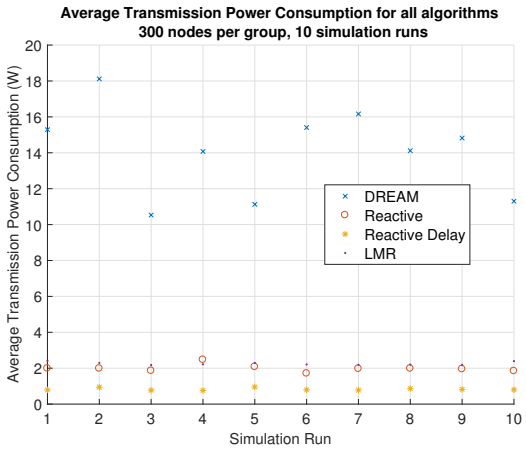
Figure 31: Raw LMR Packet Transmission Error Rate results with a density of 200 nodes per group

Table 17: Raw results for all algorithms with **300** nodes per group

Routing Algorithm	Simulation Run	Group Transmission Time (s)	Average Transmission Power Consumption (W)
DREAM	1	6.0	15.2888
	2	7.0	18.1168
	3	5.0	10.5325
	4	6.0	14.0741
	5	5.0	11.1273
	6	6.0	15.4053
	7	8.0	16.1637
	8	6.0	14.1180
	9	6.0	14.8210
	10	5.0	11.3025
Reactive	1	17.0	1.9900
	2	17.0	1.9853
	3	17.0	1.8593
	4	20.0	2.4733
	5	22.0	2.0773
	6	16.0	1.7100
	7	18.0	1.9727
	8	18.0	1.9860
	9	18.0	1.9507
	10	26.0	1.8460
Reactive Delay	1	29.0	0.7920
	2	31.0	0.9447
	3	26.0	0.7727
	4	31.0	0.7613
	5	33.0	0.9553
	6	27.0	0.7967
	7	32.0	0.7800
	8	29.0	0.8627
	9	35.0	0.8233
	10	30.0	0.7993
LMR	1	54.0	2.4102
	2	45.0	2.2942
	3	42.0	2.1778
	4	43.0	2.2135
	5	42.0	2.2952
	6	42.0	2.2092
	7	43.0	2.1885
	8	42.0	2.2017
	9	42.0	2.1872
	10	43.0	2.3947



(a) Raw Group Transmission Time results



(b) Raw Average Transmission Power Consumption results

Figure 32: Raw GTT and ATPC results for simulations with a density of 300 nodes per group

Table 18: Raw LMR results with **300** nodes per group

Simulation Run	Packet Transmission Error Rate (%)
1	0.9809
2	0.0570
3	0.1668
4	0.2722
5	0.1104
6	0.3359
7	0.3315
8	0.3874
9	0.1684
10	0.2650

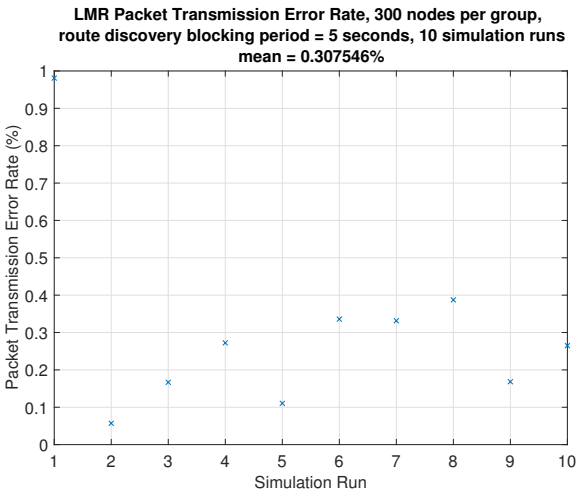
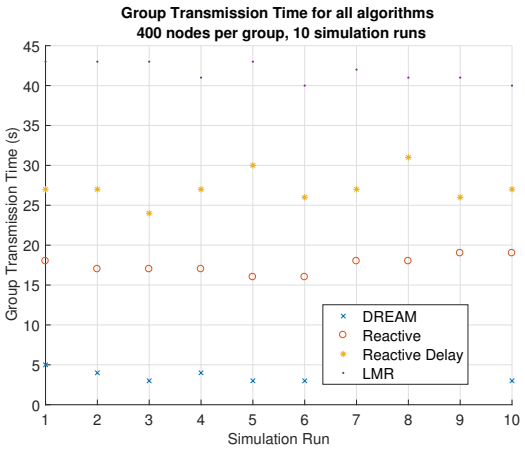


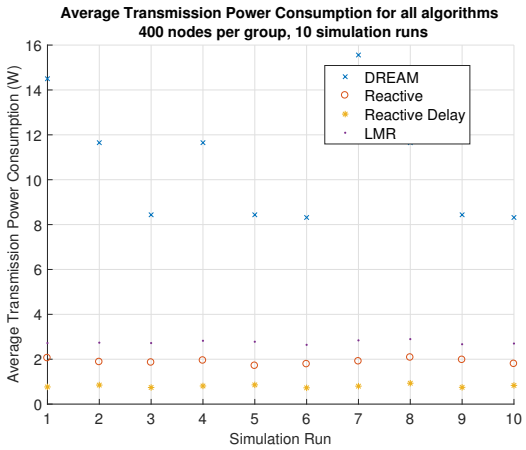
Figure 33: Raw LMR Packet Transmission Error Rate results with a density of 300 nodes per group

Table 19: Raw results for all algorithms with **400** nodes per group

Routing Algorithm	Simulation Run	Group Transmission Time (s)	Average Transmission Power Consumption (W)
DREAM	1	5.0	14.4996
	2	4.0	11.6503
	3	3.0	8.4366
	4	4.0	11.6503
	5	3.0	8.4366
	6	3.0	8.3169
	7	5.0	15.5576
	8	4.0	11.6503
	9	3.0	8.4366
	10	3.0	8.3169
Reactive	1	18.0	2.0510
	2	17.0	1.8765
	3	17.0	1.8535
	4	17.0	1.9430
	5	16.0	1.7090
	6	16.0	1.7820
	7	18.0	1.9075
	8	18.0	2.0765
	9	19.0	1.9725
	10	19.0	1.7940
Reactive Delay	1	27.0	0.7650
	2	27.0	0.8485
	3	24.0	0.7415
	4	27.0	0.8080
	5	30.0	0.8565
	6	26.0	0.7260
	7	27.0	0.8000
	8	31.0	0.9310
	9	26.0	0.7475
	10	27.0	0.8330
LMR	1	43.0	2.7211
	2	43.0	2.7374
	3	43.0	2.7200
	4	41.0	2.8225
	5	43.0	2.7797
	6	40.0	2.6412
	7	42.0	2.8404
	8	41.0	2.8960
	9	41.0	2.6673
	10	40.0	2.6993



(a) Raw Group Transmission Time results



(b) Raw Average Transmission Power Consumption results

Figure 34: Raw GTT and ATPC results for simulations with a density of 400 nodes per group

Table 20: Raw LMR results with **400** nodes per group

Simulation Run	Packet Transmission Error Rate (%)
1	0.3724
2	0.1343
3	0.3106
4	0.0698
5	0.0353
6	0.5344
7	0.1733
8	0.8981
9	0.1709
10	0.4410

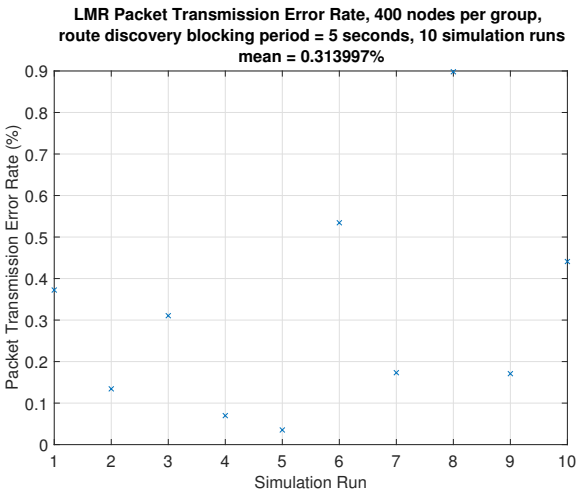


Figure 35: Raw LMR Packet Transmission Error Rate results with a density of 400 nodes per group

Reactive Delay Blocking Period Results

Table 21: Raw Reactive Delay algorithm results with a blocking period of **1** second

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	16.0	1.7925
2	18.0	2.1945
3	18.0	1.9835
4	17.0	1.9080
5	19.0	1.9935
6	16.0	1.7855
7	18.0	1.9995
8	17.0	1.8495
9	17.0	1.9310
10	17.0	2.0810

Table 22: Raw Reactive Delay algorithm results with a blocking period of **2** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	19.0	1.1870
2	21.0	1.1995
3	19.0	1.1540
4	18.0	1.0800
5	19.0	1.1890
6	23.0	1.2955
7	21.0	1.2815
8	23.0	1.3360
9	21.0	1.2365
10	21.0	1.2755

Table 23: Raw Reactive Delay algorithm results with a blocking period of **3** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	21.0	0.8850
2	20.0	0.8690
3	21.0	0.8980
4	23.0	1.0420
5	22.0	0.9455
6	23.0	0.9715
7	23.0	1.0165
8	22.0	0.9700
9	25.0	1.0190
10	22.0	0.8270

Table 24: Raw Reactive Delay algorithm results with a blocking period of **4** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	27.0	0.9650
2	22.0	0.7675
3	23.0	0.7830
4	24.0	0.8265
5	25.0	0.8610
6	22.0	0.7495
7	23.0	0.7950
8	26.0	0.8910
9	23.0	0.7830
10	26.0	0.8455

Table 25: Raw Reactive Delay algorithm results with a blocking period of **5** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	27.0	0.7650
2	27.0	0.8485
3	24.0	0.7415
4	27.0	0.8080
5	30.0	0.8565
6	26.0	0.7260
7	27.0	0.8000
8	31.0	0.9310
9	26.0	0.7475
10	27.0	0.8330

Table 26: Raw Reactive Delay algorithm results with a blocking period of **6** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	25.0	0.6455
2	27.0	0.7110
3	30.0	0.7530
4	29.0	0.7175
5	27.0	0.6865
6	28.0	0.7390
7	29.0	0.7230
8	27.0	0.7025
9	25.0	0.6270
10	32.0	0.8005

Table 27: Raw Reactive Delay algorithm results with a blocking period of **7** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	27.0	0.6815
2	26.0	0.6285
3	34.0	0.7385
4	31.0	0.6990
5	30.0	0.6840
6	25.0	0.5935
7	34.0	0.7480
8	31.0	0.6945
9	30.0	0.6745
10	31.0	0.7715

Table 28: Raw Reactive Delay algorithm results with a blocking period of **8** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	33.0	0.6960
2	35.0	0.7320
3	29.0	0.5955
4	31.0	0.6580
5	27.0	0.6315
6	30.0	0.6090
7	29.0	0.6160
8	37.0	0.7075
9	31.0	0.6270
10	35.0	0.7760

Table 29: Raw Reactive Delay algorithm results with a blocking period of **9** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	28.0	0.5480
2	31.0	0.6340
3	40.0	0.5715
4	36.0	0.7470
5	35.0	0.6785
6	28.0	0.5610
7	32.0	0.6385
8	36.0	0.6615
9	35.0	0.7210
10	32.0	0.5985

Table 30: Raw Reactive Delay algorithm results with a blocking period of **10** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**
1	34.0	0.6015
2	34.0	0.6335
3	37.0	0.6845
4	40.0	0.6910
5	37.0	0.6655
6	33.0	0.6190
7	34.0	0.6090
8	36.0	0.6845
9	34.0	0.6130
10	34.0	0.6095

LMR Blocking Period Results

Table 31: Raw LMR algorithm results with a blocking period of **1** second

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**	**Packet Transmission Error Rate (%)**
1	42.0	5.8022	1.3205
2	40.0	6.0950	2.0609
3	43.0	5.8076	1.6313
4	41.0	5.7302	0.7097
5	42.0	5.6956	0.6821
6	40.0	5.6552	1.0820
7	43.0	5.8279	1.6013
8	41.0	5.9127	1.6361
9	40.0	5.8003	1.6118
10	43.0	5.9331	1.3940

Table 32: Raw LMR algorithm results with a blocking period of **2** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**	**Packet Transmission Error Rate (%)**
1	41.0	3.9099	0.5208
2	42.0	3.9811	1.2426
3	42.0	3.9756	1.0104
4	40.0	3.8056	1.1589
5	41.0	3.9869	1.2907
6	42.0	3.9290	1.4271
7	40.0	3.8555	0.7596
8	43.0	3.9891	1.5532
9	42.0	3.8919	1.5379
10	40.0	3.9915	0.7521

Table 33: Raw LMR algorithm results with a blocking period of **5** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**	**Packet Transmission Error Rate (%)**
1	43.0	2.7211	0.3724
2	43.0	2.7374	0.1343
3	43.0	2.7200	0.3106
4	41.0	2.8225	0.0698
5	43.0	2.7797	0.0353
6	40.0	2.6412	0.5344
7	42.0	2.8404	0.1733
8	41.0	2.8960	0.8981
9	41.0	2.6673	0.1709
10	40.0	2.6993	0.4410

Table 34: Raw LMR algorithm results with a blocking period of **10** seconds

Simulation Run	**Group Transmission Time (s)**	**Average Transmission Power Consumption (W)**	**Packet Transmission Error Rate (%)**
1	40.0	2.2757	0.2077
2	42.0	2.4712	0.6966
3	41.0	2.1845	0.2452
4	41.0	2.2360	0.4492
5	42.0	2.3799	0.5135
6	51.0	2.3272	0.1745
7	41.0	2.2694	0.0699
8	42.0	2.2107	0.1058
9	42.0	2.4285	0.1709
10	42.0	2.4678	0.6916