David Cottingham
Churchill College

Collaborative Power Management in Wireless Mesh Networks

Dissertation Submission
Computer Science Tripos Part II, 2004

March 2004

David Cottingham
Churchill College

Collaborative Power Management in Wireless Mesh Networks

Dissertation Submission
Computer Science Tripos Part II, 2004

Approximate Word Count: 11,670

Project Originator: D. N. Cottingham
Project Supervisor: Dr. J. K. Fawcett

Proforma

Original Aims

Mobile nodes in mesh networks draw on limited energy resources. In most mesh routing protocols the metric used is latency. Hence, some nodes' energy resources become more rapidly exhausted than others', due to their location on low latency (and therefore high traffic) routes. This project aims to create an energy-aware routing protocol, supporting reliable communication, that increases average node lifetime, and lowers the standard deviation in energy reserves throughout the network. These are crucial in applications where the failure of any one node is unacceptable: instead all fail at approximately simultaneously, which then defines the length of a maintenance cycle.

Work Completed

A distance-vector mesh routing protocol was implemented, using ideas from various existing algorithms. The protocol supports reliable transmission using bitmapped ARQ, and supports connection multiplexing. The routing algorithm is able to perform metric calculations using link latency or with two different algorithms depending on energy level. An energy quota system protects against certain DoS attacks. Hashing of routing information is performed to prevent malicious spoofing of identity or incorrect topological information. All core components of the project have been completed and tested, and the more efficient of the two energy-aware algorithms has been found. Three extensions have also been completed.

Special Difficulties Encountered

No special difficulties were encountered.

Declaration of Originality

I David Cottingham of Churchill College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any susbstantial extent for a comparable purpose.

Signed:

Date:

Contents

List of Figures

1 Introduction

In the last two years the prevalence of wireless networking technologies has increased dramatically. Previously infrared was the mainstream technology used to interconnect portable devices, with low bandwidth, and over a limited range. It is now commonplace to find an 802.11b "WiFi" transceiver in laptops and mobile telephones. Such explosive growth in wireless communication has found many applications where nodes are required to be mobile, or where fixed cabling is undesirable.

1.1 Mesh Networks

Traditional wireless deployments are based on a paradigm best illustrated by mobile telephone networks. Client nodes, probably with limited resources, communicate with a fixed access point or base station, which has (comparably) limitless resources. Data flow is asymmetric in that it is generally assumed that content will flow towards the client, with only control data flowing in the reverse direction, and the access point, connected to a wired network in a static topology, performs any routing of packets.

With wireless mesh (also known as *ad hoc*) networks[1] the paradigm is very different: the majority of the nodes making up the network are not in close proximity to wired infrastructure. Information flow may be symmetric or asymmetric, and is likely to be between nodes as well as to/from fixed networks. Perhaps the most important distinction is that the passage of data from one node to another is via other wireless nodes – each device performs routing for its peers. This is especially significant in mobile networks where the topology varies frequently, and where multiple routes to a destination are common. Transmission range and power consumption are lessened as data flows through multiple short hops rather than over a single long distance link to an access point, as can be seen in Figure 1. This also means that the network is more robust as it has no single point of failure (assuming a great enough node density to enable multiple routes from each node to all others), and that rollout is both rapid and cost-effective.

[1]Comprehensive explanations of mesh networks can be found in [7] and [10]

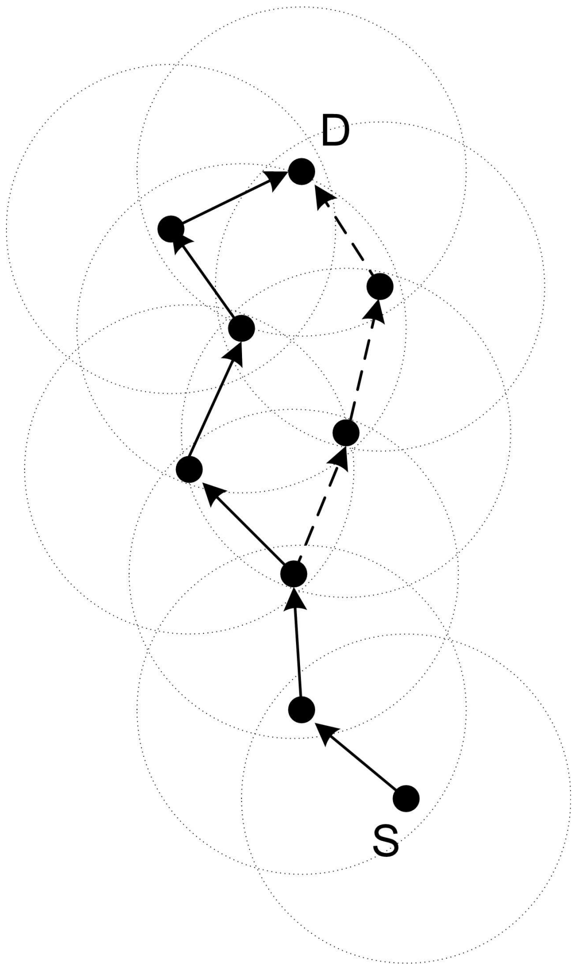


Figure 1: Multiple routes in a mesh network: two routes are illustrated from a source, S, to a destination D.

1.2 Background

Mesh networks are an active field of research: some implementations exist, but there are comparatively few large-scale initiatives. One of the main issues is the design of efficient routing algorithms that can allow for:

- **Dynamic topologies:** traditional routing protocols such as the Routing Information Protocol[11] are not designed for the very high rates of topology change present in mobile mesh networks.
- **Lossy radio links:** on wired networks packet loss is uncommon except that due to congestion. When using a radio link the quality of the channel is highly variable due to

atmospheric effects, multipath fading due to terrestrial obstructions, and interference from other signals. This necessitates any protocol to have a method of ensuring reliable transmission, similar to the Transmission Control Protocol's (TCP) Automatic Repeat Request (ARQ) system. Most wireless technologies such as 802.11b have link-layer automatic retransmission (a Radio Link Protocol, RLP), but running TCP over these can be subject to performance issues due to varying delay caused by a non-uniform distribution of errors, (causing the TCP algorithm to infer congestion), as described in [35].

- **Energy conservation:** a major target in many mobile ad hoc networks (MANETs), is the maintenance of the liveness of the entire network – i.e. to avoid partitioning due to the loss of one or more key nodes caused by power exhaustion. This is traded off with network latency and/or processing power. The protocol implemented in this project deliberately reduces the number of control data packets to conserve energy by piggybacking as much of the control information on normal data packets.

- **Security:** it is the nature of mesh networks that control is decentralized. This means that restricting access is a difficult problem, as is ensuring that routing tables are not maliciously corrupted by bogus information. There is also the issue of privacy that is inherent of using a wireless medium, as well as the possibilities of denial of service (DoS) attacks, and spoofing. Finally, encryption strength is restricted by the processing power it requires on nodes.

Applications of mesh networks are many and varied. Examples include:

- *Mobile networks:* in battlefield or search and rescue operations personnel need to communicate without fixed infrastructure. Such technology must be rapidly and easily deployable. Mesh networks do not require lengthy configuration, and allow the mobility of the network as a whole, compared to only node mobility when using a fixed access point. Moss outlines these uses further in [17].

- *Sensor networks:* most manufacturing facilities require a high degree of instrumentation, much of which must be connected to one central processing point. Mesh networks can be deployed to avoid the cost of wiring by sensors relaying data for each other.

- *Wireless MANs:* the provision of broadband Internet access over wireless mesh has been studied as an alternative to using the last mile of the telephone network. The trial described in [27] is one such example. Such networks take a fraction of the deployment time and cost required for a wired solution.

1.3 The Case for Energy Aware Routing

With any mobile node, energy is stored in batteries, as connections to the main grid are not feasible. This results in mobile nodes requiring both hardware and software which consume as little energy as possible. In a mesh network the communication paths between nodes depend on there being intermediate nodes with sufficient energy remaining to relay data; if these intermediate nodes are overused, they quickly become exhausted, causing the network to partition. Clearly this is unsatisfactory in many situations, such as in search and rescue operations, where the greatest benefit is obtained when the node graph is fully connected.

For example, Figure 2 shows a simple network with six nodes, with arcs representing connections all of equal latencies. A packet travelling from node 6 to node 3 will be routed via node 7, if the metric is based on latency or distance alone. This route choice will be true for any node wishing to communicate with another more than two hops away from it, therefore node 7 will quickly have its energy resources exhausted.

To avoid this *key node exhaustion* effect, routing protocols need to take into account the power remaining in each node in the network, rather than simply the latency, bandwidth, or geographical length of routes. By increasing the metric of a route as its component relay nodes' energy levels decrease, data can be routed in a balanced manner throughout the network. The end result will be that the standard deviation from the mean of

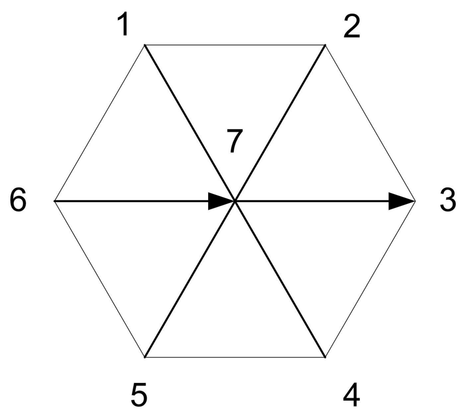


Figure 2: The effect of using "pure" latency routing – the central node in this diagram will route the most traffic, resulting in its energy resources being exhausted rapidly.

energy levels in the network will be lower than with purely latency routing.

Such balanced routing will inevitably mean that routes contain a greater number of hops than they would with optimal latency routing, which will in turn increase the average energy used per packet. There is therefore a trade-off between the aim of keeping the entire network alive and unpartitioned, and that of using as little energy as possible, but risking partition due to key node exhaustion. With the latter the result will be that a subset of nodes will be alive for longer, but will be unable to communicate with each other – in some situations this may be preferable to all nodes failing simultaneously (although it should be noted that this makes for easy maintenance cycles). Hence the usage of an energy aware routing algorithm is very much dependant on application.

1.4 Related Work

A great variety of routing protocols have been proposed for ad hoc networks. These can be divided into two main groups; those that track the state of the entire network and thereby employ "pro-active" routing (table-driven), and those that construct routes on demand by flooding query packets to the network (source-driven). Notable examples include:

Table-Driven:

- DSDV: Destination-Sequenced Distance Vector [22]
- CGSR: Cluster-head Gateway Switch Routing [4]
- WRP: Wireless Routing Protocol [18]

Source-Driven:

- AODV: Ad-hoc On-demand Distance Vector [23]
- DSR: Dynamic Source Routing [13]
- TORA: Temporally Ordered Routing Algorithm [21]

Each of these protocols is summarised in Appendix A. The design of the protocol implemented in this project combines several of the ideas from the above, therefore the summaries are provided for reference.

Work has also been carried out on specifically conserving energy in mesh networks. The Piconet (later PEN) project [1] is perhaps the most relevant example, which concentrated on building devices that consumed very little power and employed energy saving features at the MAC level. Later work by Stefanova *et al.* on the PEN protocol included a proactive routing scheme described in [33], but which did not specifically make route choices on the basis of energy.

Younis *et al.* have created a system [36] that attempts to take energy management into consideration when routing, but this scheme appears to be based on CGSR [4], where the network is not truly *ad hoc*, and instead requires access points that are powered and connected to wired networks. The protocol is link state and does not appear to scale well given the need for fixed gateway nodes. It does however achieve (perhaps predictably, given fixed base stations), an order of magnitude improvement in time to network partition.

Maleki, Dantu, and Pedram propose a Power-aware Source Routing (PSR) algorithm in [15]. This is based on DSR, and therefore suffers from the same problems of scalability due to large routing protocol data units. Additionally, it causes even greater latency in route discovery due to waiting for multiple possible routes to be reported to the destination, to enable it to forward the lowest energy-cost route back to the source. Finally, although each individual node continually recalculates the path costs to its neighbours based on its remaining energy, this metric is not used to update existing routes until the energy has dropped below a certain level. This has the potential to cause sudden losses of routes, rather than increasingly discourage the sender from using the route as the energy level decreases. This is important if any kind of quality of service is to be achieved – such "on or off" behaviour results in significant delays whilst a new route is found to replace the one lost.

Singh *et al.* list several different ways of calculating metrics based on energy remaining in individual nodes in [30]. Whilst they also list the potential benefits of using each algorithm, several of the algorithms are at best very difficult to implement successfully, and one is impossible. Toh proposes many similar algorithms in [34], but also gives an excellent summary of the features required of a protocol that would make use of such battery powered metrics.

2 Preparation

The initial proposal outlined the goals of the project, i.e.

> To design and implement (possibly entirely within a simulator) a protocol for mobile *ad hoc* networks that will conserve node resources by making routing choices that are dependent on the remaining energy of the nodes making up the alternative routes. This will provide resilience and high availability for communication between all nodes in the network.

It is crucial to realise that the project's aim is to spread energy usage throughout the network such that the variation in energy reserves is significantly lower than with conventional protocols. This may well mean that data does not travel by the lowest latency route (although the metric does factor in latency), but instead the time to network partition should be significantly increased.

2.1 Design Goals

The following were identified as the goals that would be needed to be achieved to satisfy the above aim:

- **Reliable transport protocol:** given TCP's performance over wireless links, (see [35]), a form of ARQ is required for guaranteed delivery of packets to their destinations.
- **Mesh routing algorithm:** a mechanism for acquiring, maintaining, propagating, and invalidating routes is required, that can allow for potentially high rates of topology change.
- **Minimal control packets:** this will conserve bandwidth, and will reduce the number of transmissions (although it may lengthen a few by a small amount).
- **Energy management:** the protocol must ensure that key nodes in the network are not quickly exhausted despite being on shortest path routes, and it must aid in the prevention of DoS attacks without preventing normal communication. The routing metric should take into account both the latency of the route, and the energy remaining on the intermediate nodes along that route. It should also allow for other parameters to be introduced should the need arise.

2.2 Assumptions

The project has been conducted on the basis of the following assumptions, to ensure that it is a manageable workload, and that the main focus remains on energy-aware *routing*, rather than other energy saving concepts:

- Node radio receiver circuits remain continuously powered up. Whilst this is likely to be the main energy consumer in a node, it is assumed that the application cannot tolerate the latencies that would be inherent in using MAC-level co-ordinated hibernation periods. For this reason the protocol aims to reduce the number of packets that are transmitted (and therefore received) to reduce energy consumption.
- Hardware energy saving concepts have not been investigated – see [1] for details on these.
- Nodes have omnidirectional antennae, and use a non-adaptive transmission power, regardless of target. This is a reasonable assumption given that:
  - Nodes have low powers of transmission in any case
  - In a mobile mesh network antennae are necessarily omnidirectional unless they are movable
  - To ensure good route convergence times and a reasonable degree of resilience in the network, routing information needs to be broadcast as widely as possible.
- Only rudimentary collision avoidance mechanisms are implemented for the wireless channel (i.e. there are no specific Ready To Send [RTS] or Clear To Send [CTS] packets), although if the project were running over standard WLAN equipment, these would be sent by the hardware itself.
- Traffic flow is expected to be approximately balanced between nodes: this is not

the case in some applications such as sensor networks where data flows towards a common *sink*.

2.3 Requirements

The implementation of a network protocol that required more than two or three nodes to adequately test as standalone code was deemed to be impractical, given that testing of such code would have necessarily involved either a large number of physical devices with potentially tens of mobile users, or a large quantity of extra interface code with a simulation package. The decision was therefore taken to implement the entire project inside the OPNET simulation package. This decision was not taken lightly, given that OPNET is the industry standard simulation package, and therefore has a very steep learning curve. A significant quantity of time was spent reading documentation on the package, and experimenting with it.

Deciding to use OPNET dictated that I learn the C programming language, which I had not had any development experience with. A significant amount of time was therefore also spent becoming proficient in it, both for simple standalone network programming, and more complex work within the OPNET simulator.

The scope of the project was further clarified, as outlined in the previous section.

Related work in the field was examined. The topic of mesh networking is not specifically taught in the Tripos, and therefore, in order to gain sufficient grounding in the subject, various references in addition to those listed in the Introduction above were consulted.

It was decided that the protocol would be a form of distributed Bellman-Ford algorithm (i.e. distance-vector), but that ensured loops were prevented. The mechanisms for preventing routing loops and distributing routing information with as little overhead as possible were considered. Pathological test cases were analysed by hand, resulting in various refinements to the protocol.

The project was divided up into specific "modules" in the original proposal. This transpired to be more of a beneficial theoretical separation than physical modules, due to the form in which the simulator accepts code. However, the protocol is implemented in the form of a (very high-level) state machine, where each state has a significant body of code executed on entry to it; Appendix D contains a diagram of the finite state machine as used in the simulator. The project was therefore further subdivided from the initial modules into more granular states that in turn made work packets of shorter lengths – increasing ease of planning and management. A layering approach was used, as some states span multiple layers. The protocol's structure is therefore such that it can be overlaid on any physical interface, without adjustment.

2.4 Component Outline

The final core components of the project, and possible extensions, are outlined below. Figure 3 provides an overview diagram of how the different layers fit together.

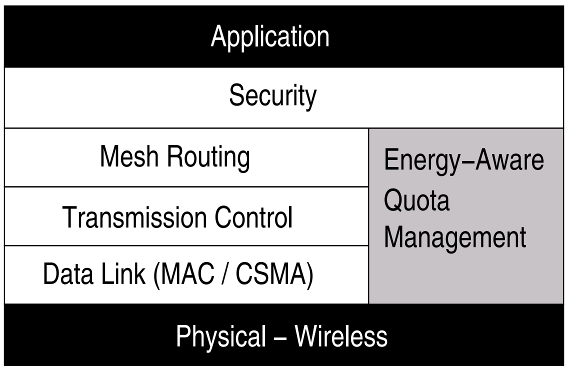


Figure 3: An overview of the different layers of the protocol.

Core

- *Data Link layer:* an interface between the physical layer provided by the simulator, and the overall protocol. Provides addressing (akin to a MAC identifier), and transmit/receive queue management.
- *Transmission Control layer:* incorporates connection tracking, multiplexing, and automatic retransmission.
- *Mesh Routing layer:* discovers routes by passive listening and active querying, maintains them using timeouts and passive listening, and propagates them by piggy-backing

the data on normal packets. Quickly adjusts to topology changes by direct propagation of changes in routes to key choice points.

- *Energy Aware Routing layer:* transcends layers one, two and three by tracking energy levels in network nodes. This is used in calculating route metrics. The module also maintains a quota system to guard against Denial of Service (DoS) attacks, and ensure that bandwidth is shared out equally. Nodes are able to predict their quotas on others, lessening the need for frequent updates.
- *Application layer:* provides console output from the simulation software, multiple debug levels, utility functions, and processes global interrupts.

Extensions

- *Bitmapped ARQ:* to reduce the quantity of control data, several packets would be acknowledged with one ACK, using an array of bits, each of which would be set if the relevant packet had been correctly received.
- *Trusted Routing Information:* by using a shared secret key, the routing information contained in the packets could be securely hashed, and the result appended to the packet. This would prevent external attack on the mesh routing protocol.
- *Link Contention Detection (CSMA):* by detecting whether the radio channel was in use, nodes would be able to wait to transmit. If a collision occurred then nodes would retransmit.
- *Encryption:* data carried in the packets is likely to require encryption, given that it is broadcast over a wireless link.
- *Multicast transmission:* a multicast protocol would allow group communication.

By carefully planning the implementation phase, it was possible to leave “hooks” for extensions to be easily integrated into the protocol as they were implemented, whilst existing code did not rely on them being in place. This added flexibility to the project whilst avoiding possible confusion as features were added.

2.5 Evaluation Methods & Milestones

To ensure that the protocol functioned correctly, various methods of evaluation were considered for the different stages of the project. In this way the necessary statistic collection routines and related code could be incorporated whilst implementation was taking place, and the code could be tested at each stage. The individual component testing methods are outlined in the Evaluation chapter, but the following are high level tests and comparisons that were part of the aims of the project:

- Compare the average time to node exhaustion using energy aware routing with that using purely latency routing.
- Evaluate the spread of energy levels in the network after a fixed period of time in the same situations.
- Compare different algorithms for energy aware routing.
- Compare the performance of these algorithms for stationary and mobile nodes.

As specified in the original proposal, the project would be deemed a success if all core components functioned as specified. The success of the project is not dependent on the algorithms being used yielding significant energy savings (although it is expected that they will), given that the project is an investigation rather than an implementation of a well known result.

Each two week packet in the project had an associated milestone. The implementation of each layer was to be tested in isolation prior to the implementation of the layer above. This incremental model [31] style of software engineering meant that system integration testing was simplified. Such testing was aided by the use of the OPNET debugger, which allows packet tracing and and debugging of radio channel characteristics.

3 Implementation

3.1 Transport Protocol

Unlike some wired networks, wireless networks are inherently lossy due to interference and multipath effects. This results in the need for transport protocols that adapt to loss, i.e. do not exhibit the poor performance of TCP when it encounters temporary packet loss, (as detailed in [35]). Such a protocol should be focussed on ensuring reliable transmission of data, whilst attempting to ensure as little bandwidth as possible is used for control information.

The protocol supports multiple reliable connections to each host, similar to using different TCP port numbers. Connection tracking code multiplexes the various connections a node has onto the wireless link, whilst at a higher level on the protocol stack packets are tracked to ensure that each has been received at the destination. These aspects are described in more detail below.

3.1.1 Packet Format

The protocol packet format is shown in Figure 4. There is only one type of packet, the objective being to include as much control information in each data packet as possible, rather than expend the extra overhead of sending distinct control data packets.

Note: A detailed specification of the packet format and the function of each field is given in Appendix B.

Error checking is assumed to take place at hardware level, i.e. CRCs are performed at the wireless interface. Hence no checksumming fields (other than the Routes Message Authentication Code, for security) are included in the packet format.

3.1.2 Connection Tracking

Communication is supported in a similar manner to the Transmission Control Protocol's ability to multiplex several concurrent connections from a particular node to other destinations, and allow multiple connections to each of those destinations using port numbers, here termed *connection identifiers.*

The protocol is based upon the idea of *connectionless* communication, similar to the Universal Datagram Protocol (UDP), which does not require explicit connection set up and tear down packets, reducing overhead. Connections are initialised when the first data packet arrives at the destination, and are torn down if no data is received for a particular period of time: unlike TCP there are no explicit `SYN` or `FIN` packets. However, unlike UDP, the protocol guarantees delivery if a route is available: Automatic Repeat Request (ARQ) detects packet loss and resends the lost data.

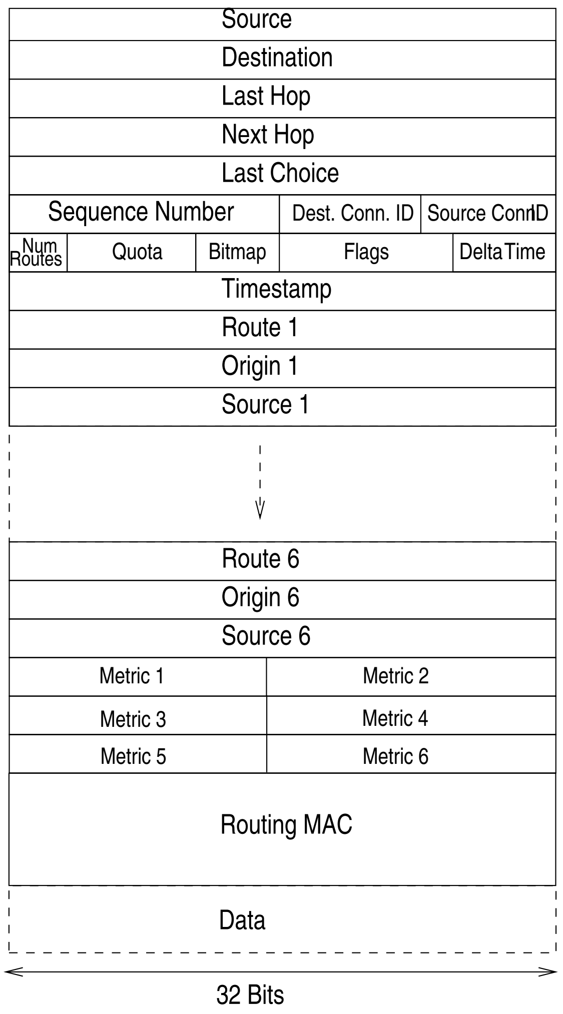


Figure 4: The format of the packets used in the protocol. each row represents 32 bits. There are 6 sets of three routing data fields, only numbers 1 and 6 are shown. The data field is of variable length.

Connection Establishment

Note: the complete process described below is illustrated in Figure 5, which the reader may find helpful to refer to.

If a node x wishes to communicate with a node y, which is more than a single hop away, x must first examine its Routing Table (RT) to ascertain the next hop for the route to y. Assuming a route is known, x creates a packet with the destination field set to y, and with the appropriate next hop address. This is known as the *initialisation packet*.

A connection identifier must then be assigned: these range from 1 to $2^8 - 1$, and are allocated in a sequential order (there is currently no provision for "well known" port numbers such as those used in TCP). Two buffer queues are also allocated to the connection, for storage of packets already transmitted (in case retransmission is required), and to buffer incoming data on that connection. The packet's source connection identifier is set to the ID allocated to it. Node x adds the connection to its established connections table, with the `established` flag unset, indicating that the destination node has not yet replied to the initialisation packet. Note that, currently, without the implementation of a scheme similar to SYN cookies [29] or a SYN gateway, the protocol is susceptible to a "SYN flood" style attack, as resources are allocated on receiving the initialisation packet.

The packet is then transmitted containing a destination connection ID of zero. This is a reserved identifier, indicating a new connection is being requested. It is assumed that nodes will only require a single new connection with each other node to be initialised at any one instant (but are able to request another once the connection is established).

On receiving the initialisation packet, y ascertains whether it has the resources to support a new connection (e.g. possesses sufficient buffer space), and inserts the entry into its connection table. Initially, it adds the connection with an identifier (of its own) of zero, given that this is the "port" on which it is receiving packets from the source.

Once the first five packets, (the first *chunk*), have been received, y is required to acknowledge their reception (see section 3.1.3 for details). At this point x is not permitted to send any further data until it has received the acknowledgement, and therefore y may update its identifier for the connection to a number other than zero, and correct its Connections Table accordingly. The acknowledgment it transmits contains as the source connection ID this new identifier, but maintains the destination connection identifier as x began to use on initialising the connection (as would be expected). In the case where there is not enough data to make up a final chunk, x will transmit the remainder as empty packets.

On receiving the acknowledgement, x updates its connection table accordingly, modifies the destination connection identifiers of any packets queued for transmission on the connection, sets the `established` flag in the connection entry, and continues transmission.

For simplicity, reliable delivery has not been implemented using a sliding window scheme. It would be possible to construct such a mechanism round the protocol, by allowing transmission of a chunk, whilst waiting for the previous one to be acknowledged. This would increase the speed at which communication could be carried out.

Sending & Receiving

Once a connection has been established between two nodes, they may communicate with each other by simply transmitting packets to the correct destination address and connection identifier. New connections may be initialised as needed.

The Connections Table keeps a record of the identifier of the route that is in use for the connection. In the event that significant packet loss is encountered, this identifier is used for blackmarking the route in the RT (see section 3.2.5).

Broadcast communication, i.e. the sending of datagrams addressed to all local neighbours, is to the reserved destination address 0.

Connection Termination

There is no formal connection termination sequence in the protocol, unlike the `FIN` packet in TCP. It is instead expected that the data stream at application level will signal termination.

An entry is removed from the Connections Table when no further traffic is received on that connection for the period `MAX_IDLE_TIME`. In addition, after initialisation, until there has been any communication from the remote node, the `established` flag in the Connections Table remains unset: if this is the case after a certain period of time, the connection entry is removed. Such removal of

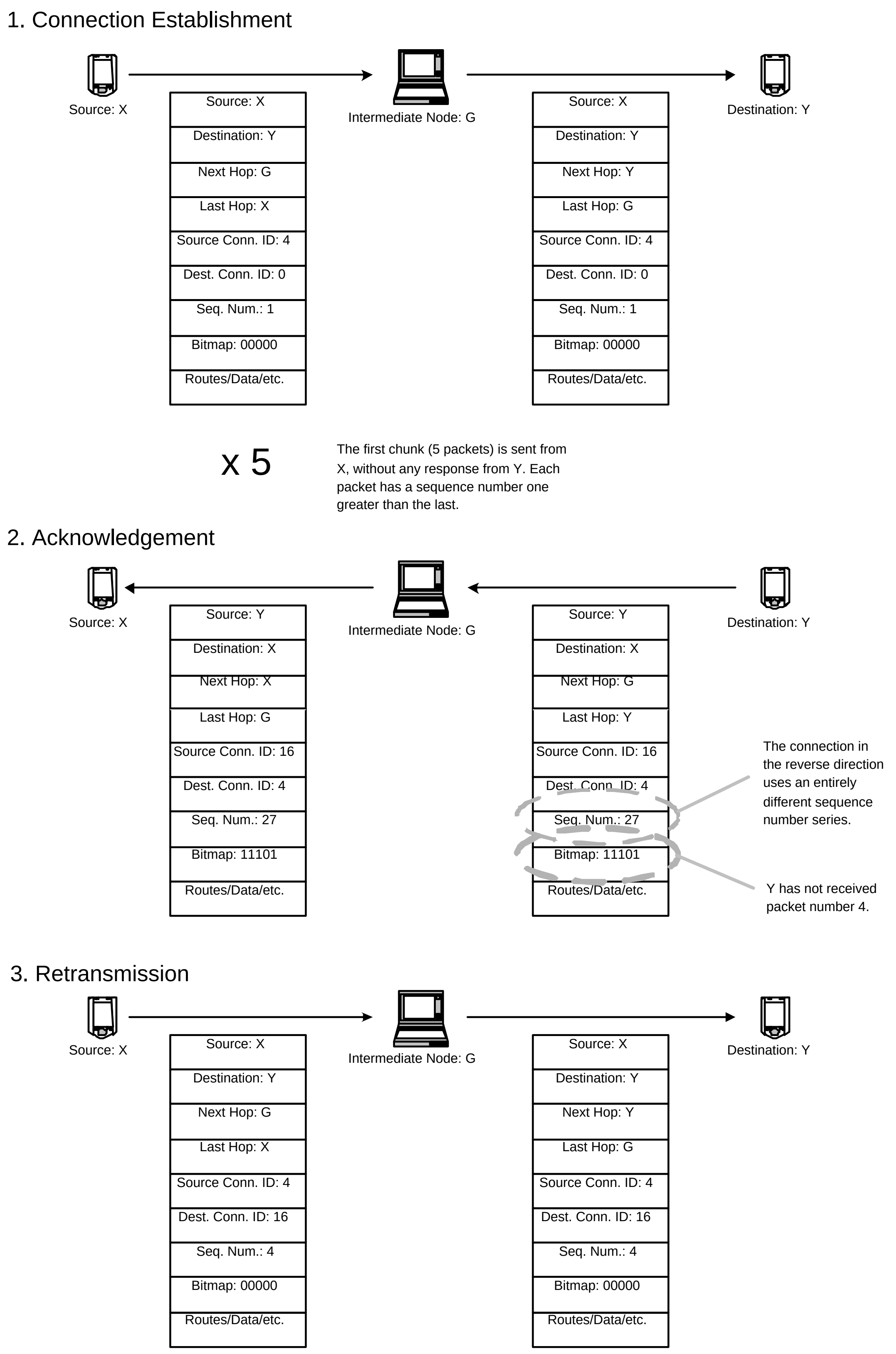


Figure 5: Connection Establishment by the Transport Protocol: a source x initiates a connection to a destination y, via a next hop g. After receiving the first chunk, y responds with an ACK, indicating that the fourth packet has been lost. x therefore resends the required packet.

connections is necessary to free connection identifiers and queue space.

3.1.3 Reliable Delivery

To ensure reliable transmission of data, the protocol implements an Automatic Repeat Request (ARQ)[9] feature. These schemes allow a receiver node to notify the transmitter that packets have been lost *en route*, and request their re-transmission.

Acknowledging each packet sent (as in TCP) is costly in terms of the number of control data packets that must be sent from the receiver. Therefore, two enhancements are made:

1. Piggybacked acknowledgements: instead of transmitting dedicated acknowledgement packets, ACKs are included in any datagrams travelling in the opposite direction to the flow being listened to.
2. Bitmapped acknowledgements: the ACK field consists of a series of five bits, each of which is set to indicate a particular packet has arrived, or unset if it has not. An ACK is therefore only sent per chunk.

Sequence Numbers

Each packet carries a 16 bit sequence number, beginning at 1, that is unique to that packet for the connection it belongs to. Packets on a connection from x to y have sequence numbers that are unrelated to those from y to x on the same connection. Sequence numbers are sufficiently large to ensure that if and when they overflow to zero there is no confusion possible between packets prior and subsequent to, the overflow.

Bitmapped ARQ

The protocol employs a Stop-and-Wait strategy. Although this causes transmission to be interrupted whilst the sender waits for an acknowledgement once each chunk has been sent, there are benefits to such a scheme. A Stop-and-Wait protocol provides selective retransmission, rather than the *Go-Back-n* solution used by TCP, reducing the number of retransmissions.

In this scheme, a source node x initialises a connection to a destination y, and proceeds to send the first chunk of data. It then ceases transmission, queueing any packets from the application level for the connection, until it receives an ACK from y, indicating that all packets in the transmitted chunk have been received. If this is not the case, x retransmits only those that y indicates did not arrive.

For its part, y receives the initial packet, p_0 from x, and notes its sequence number, s_0. For subsequent packets, p_i, it compares their sequence numbers s_i to s_0. If no re-ordering occurs, the s_i will simply be the series obtained by incrementing s_0 once each time a packet is received. Node y sets bit $s_i - s_0$ in its bitmap for the connection when packet p_i is received. A complete (i.e. with all bits set) bitmap indicates all the s_i in the chunk have been received at y.

Once the bitmap is complete, or `TIMEOUT_RTT_MULTIPLE` round trip times (RTTs) have elapsed since the initial packet was received, y sends an ACK to x. This is a normal data packet, but with the `ack` flag set, and the bitmap included in the `bitmap` field. In the case where not all packets have been received, the bitmap will have unset bits, e.g. `11101` would indicate that the fourth packet in the chunk had not been received by y. The reason that no sequence number related to the connection being acknowledged is needed in the ACK is due to the stop-and-wait scheme: the bitmap must refer to the last five packets transmitted.

Those packets indicated by the bitmap are resent, and x again waits until it receives an acknowledgement from y that the full chunk has been received. It may be that some of the missing packets have then been received, whilst others have been lost a second time, therefore the bitmap returned will indicate whether further retransmission is required.

In the event that flow is highly asymmetric between y and x, there may not be enough data packets to piggyback ACKs on. In this case, dedicated ACKs are transmitted, that do not contain data, but do have the `ack` bit set. To avoid confusion with data packets, dedicated ACKs have a sequence number of zero. In order to maximise piggybacking, acknowledgements are queued for a maximum of `TIME_TO_WAIT_FOR_CARRIER`, in case a "host" packet is sent in that time. Only if this is unsuccessful is a dedicated ACK packet sent.

If an entire chunk is lost, no acknowledgement will be received by node x, as node y will be waiting for the start of that chunk. In this case, a timeout occurs on x after

TIMEOUT_RESEND_RTT_MULTIPLE RTTs, causing it to retransmit the entire chunk.

Packet Re-ordering

Another aspect that must be handled is the possible re-ordering of packets during their traversal of the wireless link, due to some packets will incurring more delay than others.

With bitmapped ARQ this is not a problem, given that the relevant bit is set for each packet that arrives. Provided that all the packets in a chunk arrive within the relevant time-out period, buffering at the receiver ensures that packets can be sorted into transmission order once more.

Re-ordering of packets becomes a significant issue when the initialisation packet for a connection is not the first packet to arrive. In this case, any packets that subsequently arrive are deleted, as the node does not have a connection entry to which those packets match. Whilst this could be prevented, it is considered that the effort required to do so significantly outweighs the small cost of the transmitting node resending the entire chunk once more.

Round Trip Time Measurement

When a connection is initialised, nodes record a fixed RTT value RTT_ESTIMATE for the connection. On receiving a packet, the timestamp field is subtracted from the time the packet was received, to obtain a value for the actual RTT. Subsequent values provide an average RTT which is then used to set the time-outs mentioned above.

3.2 The Energy Aware Routing Algorithm

The basic premise on which the routing protocol functions is that of pro-active route discovery and maintenance, and draws on ideas from the DSDV [22] and WRP [18] protocols. to this algorithm ensures that routes are available as soon as they are needed. Routing is distance vector based, i.e. nodes are only aware of which the next hop node is for any destination they wish to communicate with.

3.2.1 Route Discovery

Beacons

On start-up, a node transmits three identical beacon packets, advertising the node's availability. Nodes within the broadcast range, termed *neighbours*, receive the beacon packets and add the new node to their Neighbours Table (NT). The new node remains in a *listen* state for T_BEACON seconds, and processes any packets it overhears from any neighbour nodes, to populate its own NT.

Data Packets

When a packet is received, a node will examine the routing information contained in it. The num_routes field in the packet indicates the quantity of routing information that the packet contains. Each route carried by the packet is made up of a triple of the form

$< \texttt{destination\_addr}, \texttt{source\_addr}, \texttt{origin\_addr}, \texttt{metric} >$

The destination_addr field is the target destination of the route, and origin_addr is the node that is a neighbour of the target destination, which propagated this route. The node that this packet was received from is known by the last_hop field of the packet. This *last hop node* in turn obtained the route from a particular *source node*, the address of which it then includes in the source_addr field of any routes it propagates. This enables routing loop prevention (see later).

Pleas

If a node wishes to communicate with a destination it does not have in its RT, it broadcasts a routing plea packet. This has its destination field set to the address of the node to which a route is required, but also has its route_query bit set.

Any node hearing a plea packet searches through its RT for a route to the destination requested. If it has a route, it places it on an urgent queue of routes to be propagated on the next routing data transmission (see section 3.2.3).

In the event that the node that emitted the plea packet does not receive any routes from its neighbours for the destination within the timeout period T_ROUTE_PLEA_WAIT, the packet is dropped. Prior to this timeout, any packets for the destination that is the subject of the plea are re-queued in the transmission FIFO queue.

Plea records are held in a dedicated hash table, which enables the routing algorithm to determine whether a plea has recently been transmitted for a particular destination. Each plea has an additional *hold* time-out, `T_HOLD_PLEA_FAILURE`, until which the plea's "result" remains in force; i.e. if no routes were received from any neighbours after `T_ROUTE_PLEA_WAIT`, a new plea is not sent until the hold timeout has expired – during this time packets are dropped. In this way plea transmissions of are not wasted. Note that were a new route to be received by the node whilst a plea were in force the route would be used for the next transmission: the pleas table is only queried when a route cannot be found in the RT.

3.2.2 Route Propagation

Nodes transmit routes by writing the data for each entry in the `route_`*i*, `origin_`*i*, `source_`*i*, and `metric_`*i* fields of packets, where *i* ranges from 1 to 6. A node routing a packet on behalf of another rewrites the `last_hop`, `next_hop`, and all of the routing data fields with its own information. A node's RT contents is transmitted on a rolling basis, by maintaining a record of which destination in the RT had its routes transmitted last. Only the lowest latency route for each destination in a node's RT is propagated.

On receiving a packet containing routing information, a node processes all the triples in the packet, and compares the entries to its current RT. The decision process is illustrated in Figure 6

To prevent routing loops, the `source_addr` and `origin_addr` are used. Figure 7 shows the process governing whether a route is passed to the processing routine above, or is discarded.

If all of the above are false, the route is added to the RT, with a `next_hop` address of the neighbour node from which the routing information came. Note that this is made possible by the fact that each packet has, a `last_hop` field, that is updated by each intermediate node that the packet is routed through.

A short example now follows. In Figure 8 the transmission ranges of the different nodes are indicated by the dashed circles: this implies that nodes may only communicate with others that are adjacent to them, as shown by the solid lines.

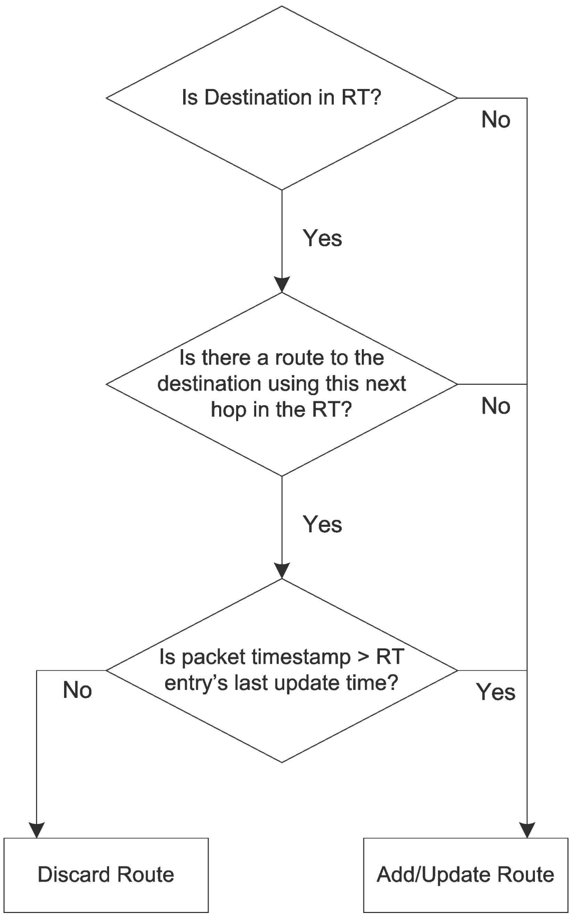

Is Destination in RT?
No
Yes
Is there a route to the destination using this next hop in the RT?
No
Yes
Is packet timestamp > RT entry's last update time?
No
Yes
Discard Route
Add/Update Route

Figure 6: Decision flowchart for route propagation.

- All nodes begin by transmitting beacon packets. It is assumed for ease of illustration that they do so almost simultaneously, although in practice this is not true. After having heard each other's beacons, node routing tables are as shown in Table 1 at time 0.
- Each node propagates the lowest metric entries for each destination in its routing table. Adjacent nodes receive these routes, and process them according to the rules above. For example, node C receives the following updates from nodes A and D (notation as defined in section 3.2.1, metrics not shown):

  1. $<$ B, –, A $>$ (from A)
  2. $<$ C, –, A $>$
  3. $<$ B, –, D $>$ (from D)
  4. $<$ C, –, D $>$
  5. $<$ E, –, D $>$

  Node C adds route 1 to its table, as $<$ B, A, A $>$ (as the next hop is now node

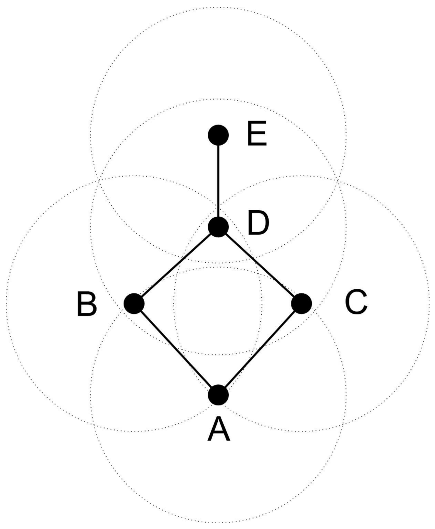


Figure 8: Maintaining a loop free routing table in a topology containing loops.

Time	Node				
	A	B	C	D	E
0	< B, –, A > < C, –, A >	< A, –, B > < D, –, B >	< A, –, C > < D, –, C >	< B, –, D > < C, –, D > < E, –, D >	< D, –, E >
1	< D, B, B >† < D, C, C >	< C, A, A > < C, D, D > < E, D, D >	< B, A, A > < B, D, D > < E, D, D >	< A, B, B > < A, C, C >	< B, D, D > < C, D, D >
2	< C, B, D > < E, B, D > < B, C, D > < E, C, D >	< D, A, C > < A, D, C >	< D, A, B > < A, D, B >	< C, B, A > < B, C, A >	< A, D, B > < A, D, C >
3	–	< E, A, D >	< E, A, D >	–	–‡

Table 1: Route propagation in a cyclic topology (Figure 8): Each node's routing table is shown as it grows over time (only those *new* entries gained at each iteration are shown, previous entries persist). The – at time 0 indicates that the node received a beacon packet and added the target destination as a neighbour. The `origin_addr` field is set to `current_addr` as this node will initiate and propagate the route. The – at time 3 indicates that no new routes are learnt, and that the routing tables have now converged.

† In normal use, the protocol only propagates the lowest metric route it is aware of. In this case as metrics are not known, all routes a node is aware of propagate, to show that loops will not occur in either case.

‡ Node E discards < C, D, A > and < B, D, A > as it already has routes to these destinations via D, and these routes are guaranteed to have a greater metric than those already gained in its routing table at time = 1.

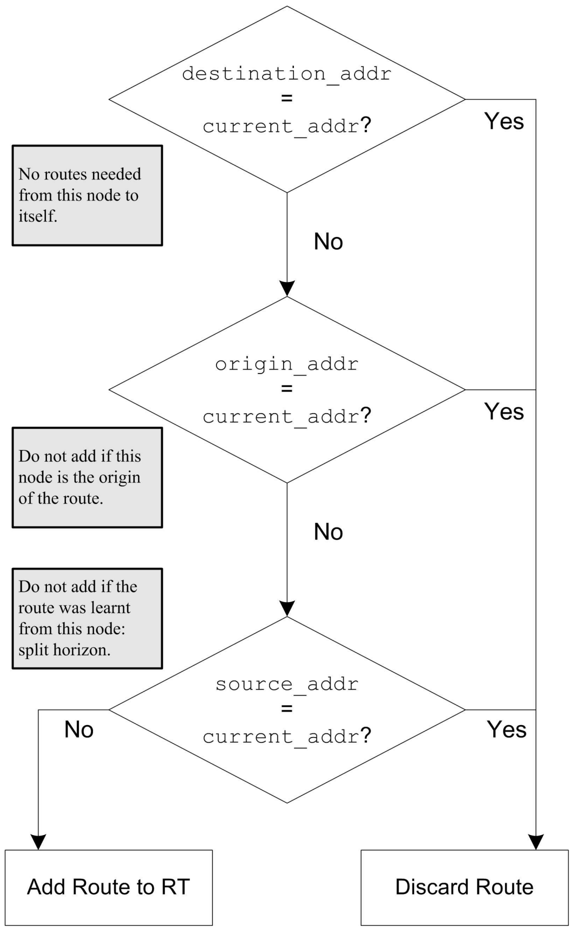

destination_addr = current_addr?
Yes
No routes needed from this node to itself.
No
origin_addr = current_addr?
Yes
Do not add if this node is the origin of the route.
No
Do not add if the route was learnt from this node: split horizon.
source_addr = current_addr?
No
Yes
Add Route to RT
Discard Route

Figure 7: Decision flowchart for preventing routing loops. The final rule is the equivalent of the split horizon algorithm [19], as used in mainstream wired protocols such as RIP [11]

A), and does the same with route 2. Route 3 is an analogous case, except that the next hop is now node D. Route 4 is discarded due to having a `destination_addr` equal to node C's own address. Route 5 is added.

- Propagation happens a second time (time = 2). Taking the example of node D, it receives all new routes that were learnt by nodes B, C and E in the last iteration, i.e.:

  1. < C, A, A > (from B)
  2. < C, D, D >
  3. < E, D, D >
  4. < B, A, A > (from C)
  5. < B, D, D >
  6. < E, D, D >
  7. < B, D, D > (from E)
  8. < C, D, D >

  Routes 1 and 4 are added as they have not been seen before. Routes 2 – 8 are discarded due to having their `source_addr` equal to node D's address.

- On the 3rd iteration only nodes B and C add any new routes to their tables – all other nodes' tables have converged. Any subsequent iterations will result in no new routes being learnt, and only metrics will be updated.

3.2.3 Route Maintenance

Ensuring Route Freshness

All nodes maintain a timer of when a packet with routing information was last transmitted. Under normal operation, this is reset each time a data packet is transmitted, but with low data traffic, regular, routing updates are transmitted. This ensures that routes are maintained whilst the topology changes, remaining prepared for a burst of traffic.

Urgent Propagation List

In addition to propagating their RT on a rolling basis, nodes maintain a linked list of routes that should be urgently propagated. The contents of this list is used to fill the routing information fields of packets in preference to the contents of RT. In this way routes that have been requested by pleas are transmitted speedily, as well as those for which there have been significant changes in metric, or which have been recently added to the RT.

Metric Propagation

For a node x to a destination y, a route passing through a graph (which may be cyclic) whose edges represent wireless links having weights $w_{i,j}$ (where $w_{i,j}$ is the non-energy aware metric for the link from node i to node j), has a route metric m of

$$m = \sum_{i=x}^{y-1} w_{i,i+1} \qquad (1)$$

(Where the intermediate nodes are assumed to be numbered consecutively $x + 1 \ldots y - 1$).

A node n calculates a route metric for any route it receives from a neighbour m by adding $w_{n,m}$ to the metric propagated to it. The measurement of $w_{n,m}$ is described in section 3.2.4.

When a route is updated, if the proportion its metric changes by is greater than `THRESHOLD_FRACTION_METRIC_UPDATE`, the route is added to the urgent propagation list.

Routing Table Structure

The RT is stored as an open hash table, as shown in Figure 9, being indexed by a hash of the destination address for the route. The entry in the hash table corresponding to this destination points to an ordered (by metric) linked list of pointers to route entries, each of which makes use of a different next hop node.

Regular pruning of the table is performed to delete routes that are no longer valid.

3.2.4 Neighbours Table

If an edge exists between two nodes in the routing graph, they are termed neighbours. Nodes' Neighbour Tables (NT) maintain quota[2] information about each of their neighbours, which is used in metric calculations.

Each packet carries a `timestamp` field, which is set at the moment when the transmission code *at the original sender* (not the last hop), outputs the packet to the wireless interface.

The `delta_time` field is updated by each hop along the route the packet passes through. Whilst the timestamp remains at the value the original sender set it to, the `delta_time` value gives an indication of the latency of the link up to the last hop. Taking equation 1, and using w_Δ to represent the value of `delta_time`:

$$w_\Delta = \sum_{i=x}^{y-2} w_{i,i+1} \qquad (2)$$

$$m = w_\Delta + w_{y-1,y} \qquad (3)$$

Hence by subtracting the time the packet is received from the timestamp we obtain m, and therefore can calculate $w_{y-1,y}$, the latency of the link to the neighbour, as required.

Nodes are added to the NT if a packet is received with a `last_hop` address of a node which is not present in the table. Explicit "join" or "leave" messages are not employed, as node mobility would cause the overhead of such a scheme to be severe. Instead, in a relatively dense mesh, nodes are likely to still be making use of a subset of the neighbours from one area as they move into another, as shown in Figure 10, and therefore they will emit packets that a subset of potential neighbours in the new area will overhear.

Entries in the NT are removed as described in section 3.2.5.

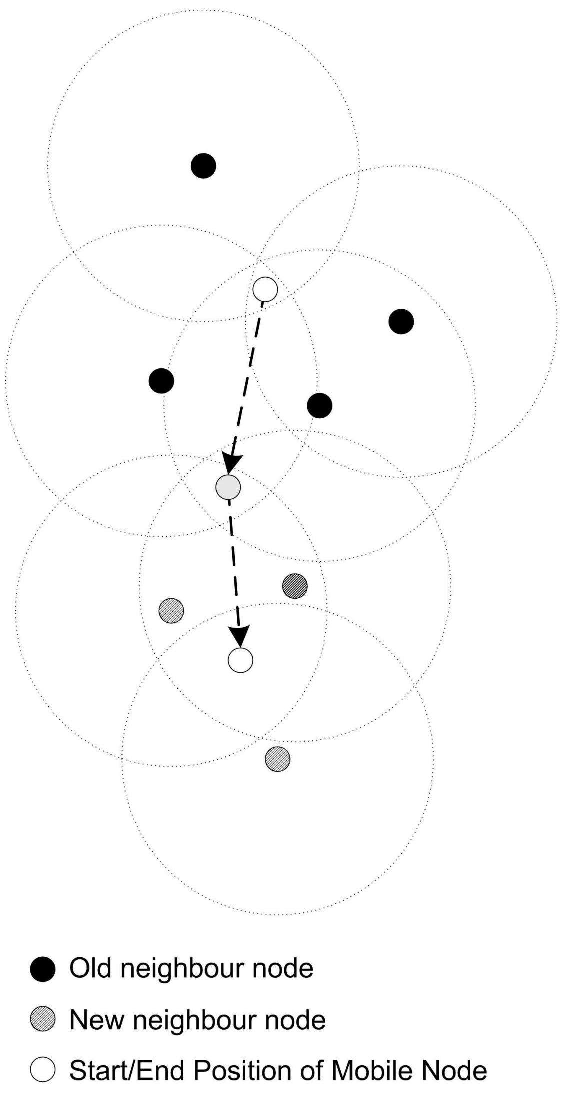


Figure 10: Sharing Neighbours with Mobile Nodes: As the mobile node moves from the start position to the target position (marked by ○s), it maintains contact with a subset of its old neighbours. This ensures that its new neighbours "overhear" its approach and add its address to their RTs.

[2]For an explanation of quotas please see section 3.2.8

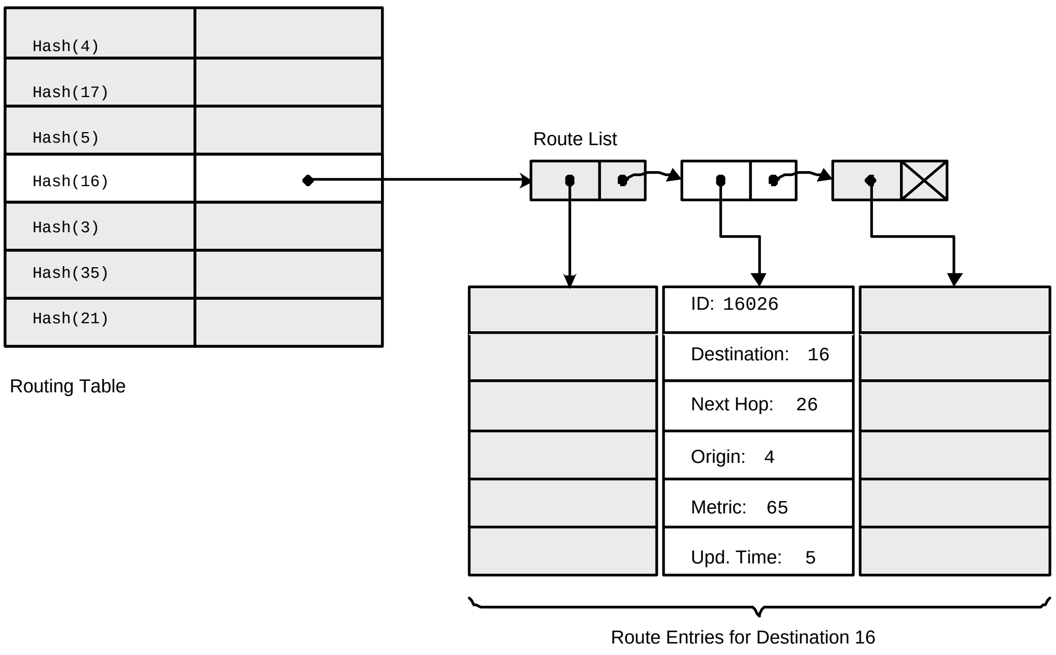


Figure 9: Structure of a Node's Routing Table: routes are stored in an open hash table, indexed by the hash of the destination address. The entry in the table contains a pointer to a linked list (ordered from least to greatest metric), each element of which has a pointer to a route structure, containing the route entry. This figure shows the entry for destination node `16`.

3.2.5 Blackmarking

If a link in the network fails, the change in topology must be propagated as soon as possible to avoid nodes attempting to use routes that include the failed link. Node failure is detected by two methods:

- Nodes are required to send at least one packet every interval of length `T_BEACON`. If no packets are received from a particular neighbour node in twice this interval, the link to that neighbour is assumed to have failed.
- When packet loss occurs on a connection to a particular destination, if more than `MAX_RETRIES_PER_CHUNK`, packets are lost for a chunk of five, (e.g. all the packets require one re-transmission, and more than one requires a second re-transmission), the route is assumed to have failed.

When a link failure is detected, an infinite metric is assigned to it; this process is known as *blackmarking* the route. For the first case above it results in all routes associated with the neighbour concerned being blackmarked, whereas for the second case only the route that was in use by the connection, (known from the entry in the connections table), is blackmarked. Figure 11 illustrates how destination unreachable messages cause blackmarking.

When a route's metric is updated from any value to infinity, or vice versa, any routing information arriving from other nodes about the entry concerned is ignored for a time `T_HOLD_TIME`. This allows the blackmarking information to propagate throughout the network.

Figure 12 illustrates one issue with this scheme: blackmarking on packet loss may result in a node temporarily being unaware of any route to a destination, despite there being a viable route via one of its neighbours. This problem is partially solved by the LCPN optimisation.

3.2.6 Last-Choice Point Notification Optimisation

As described in the previous section, ensuring that link failures are propagated throughout the network rapidly is a key factor in ensuring packet delivery with minimal retransmissions. To this end, an optimisation I term *last-choice point notification* (LCPN) has been implemented in the protocol. It is based on the fact that when a packet with destination y passes through a choice point for y (i.e. a node having at least two different routes to y), the choice point node inserts its own address into the packet's `last_choice` field, and then routes it in the usual fashion.

The LCPN optimisation is best explained by considering its effect on the blackmarking problem outlined in the previous section. Figure 13 depicts the alternative outcome.

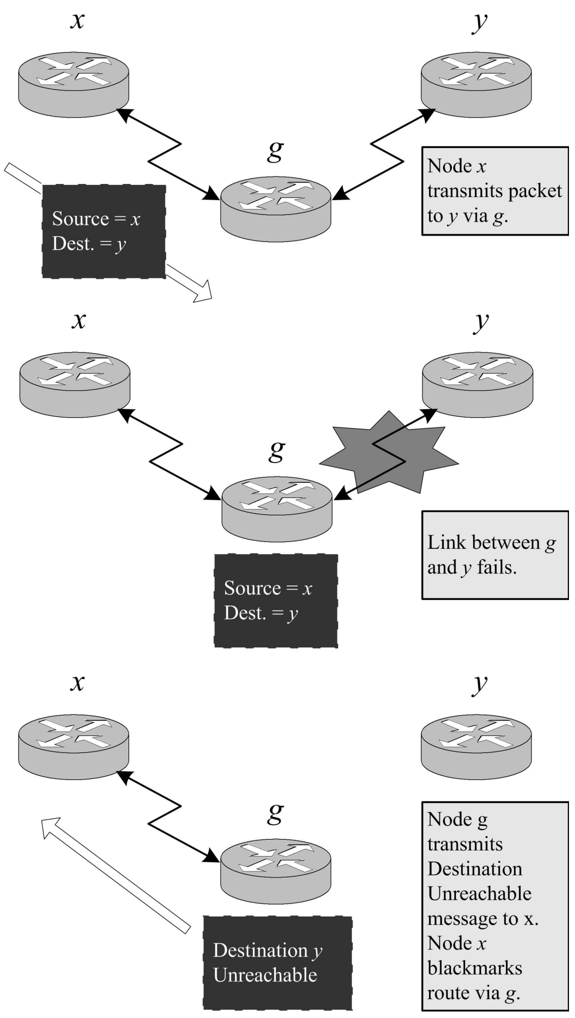
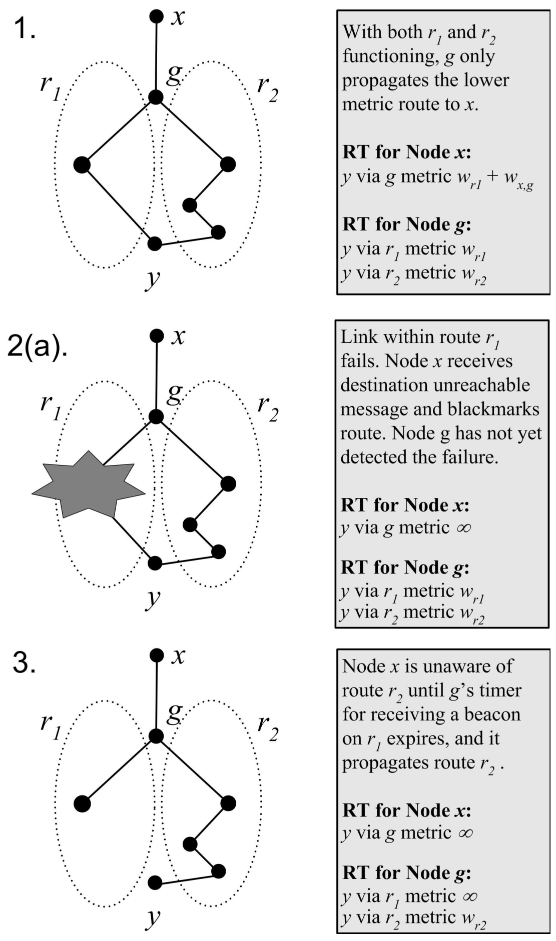

x
y
g
Node x transmits packet to y via g.
Source = x
Dest. = y
x
y
g
Link between g and y fails.
Source = x
Dest. = y
x
y
g
Node g transmits Destination Unreachable message to x. Node x blackmarks route via g.
Destination y Unreachable

Figure 11: Destination Unreachable Messages: when a route to a destination cannot be found for a packet to be routed, nodes send a destination unreachable message to the original sender.



With both r_1 and r_2 functioning, g only propagates the lower metric route to x.

RT for Node x:
y via g metric $w_{r1} + w_{x,g}$

RT for Node g:
y via r_1 metric w_{r1}
y via r_2 metric w_{r2}

Link within route r_1 fails. Node x receives destination unreachable message and blackmarks route. Node g has not yet detected the failure.

RT for Node x:
y via g metric ∞

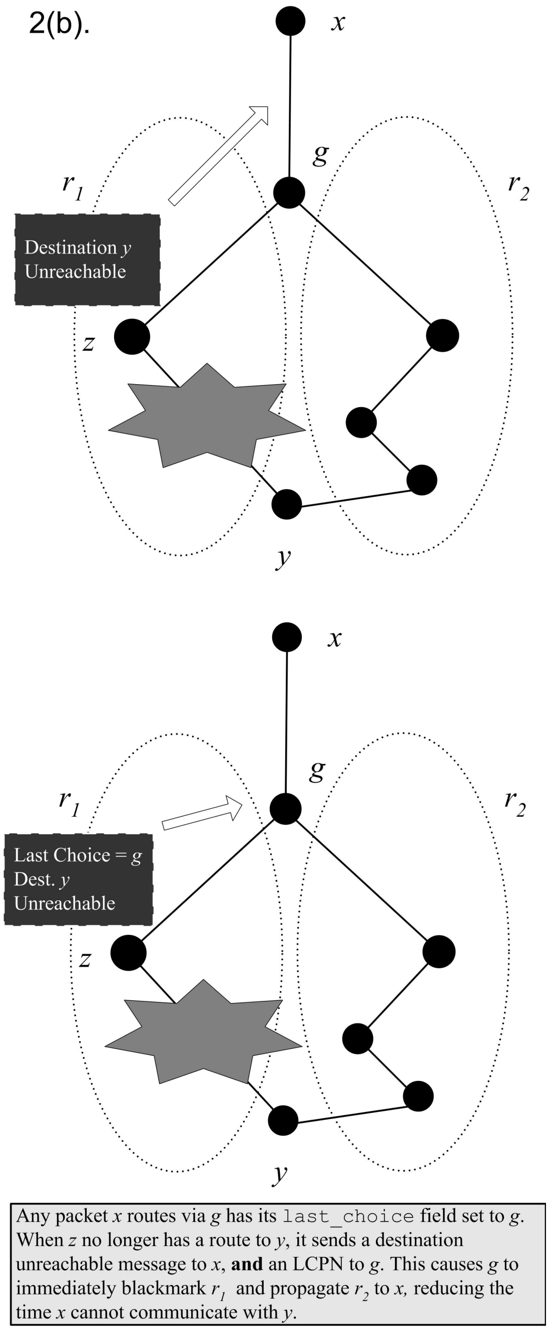
RT for Node g:
y via r_1 metric w_{r1}
y via r_2 metric w_{r2}

Node x is unaware of route r_2 until g's timer for receiving a beacon on r_1 expires, and it propagates route r_2 .

RT for Node x:
y via g metric ∞

RT for Node g:
y via r_1 metric ∞
y via r_2 metric w_{r2}

Figure 12: Illustration of a Possible Problem with Blackmarking: if two separate routes exist from x to y, with a common "gateway" g, there exists the possibility that if route r_1 fails, x will not be aware of r_2. This situation is made less probable by the LCPN optimisation.



Any packet x routes via g has its `last_choice` field set to g. When z no longer has a route to y, it sends a destination unreachable message to x, **and** an LCPN to g. This causes g to immediately blackmark r_1 and propagate r_2 to x, reducing the time x cannot communicate with y.

Figure 13: The effect of the LCPN optimisation. The diagram shows an alternative stage 2 from Figure 12. With LCPN the source node x has a viable route propagated to it rapidly, instead of waiting for the beacon time-out for z on g to expire.

3.2.7 Energy-Awareness

The principle goal of the project is to implement the protocol such that it is sensitive to the energy remaining in each of the nodes in the network when making its routing decisions. The metric of each link should therefore be influenced by the energy levels of the nodes that are at either end of that link.

The protocol initially made use of a metric adjusted as follows:

$$w_{i,j} = l_{i,j} * (1 + \frac{1}{e_i}) \tag{4}$$

Where $l_{i,j}$ is the latency of the link from i to j, and e_i is the energy level of node i. Note that this metric, $w_{i,j}$, is only applicable for data flow from i to j, as the reverse ($w_{j,i}$), is proportional to e_j (as node j would be the transmitter). This metric assumes that the cost that should be accounted for by the metric is the transmission, rather than the reception, energy usage, given that nodes are assumed to receive all packets that they can “hear”.

Equation 4 has the disadvantage that a metric will not be substantially changed until e_i decreases to a relatively low number. This results in energy awareness only being evident at the point where a node has a very low energy level. Other metrics were therefore developed to attempt to cause the effect to be seen earlier in the lifetime of the network.

A variation on Equation 4 is to cause the metric to be dependent on the proportion used of the original quota assigned to the destination, i.e.:

$$w_{i,j} = l_{i,j} * (1 + \frac{\frac{p}{n}}{e_i}) \tag{5}$$

Where p is the total energy level a node is initialised with, and n is an approximation to the number of neighbours a node is likely to have on average, (experiment indicates that $n = 2$ functions well). In this way as the energy level decreases, $w_{i,j}$ increases at a greater rate than in Equation 4.

Subsequently, a third metric was also examined. This places the bound of the maximum metric at twice the latency of the link:

$$w_{i,j} = l_{i,j} * (2 - \frac{e_i}{p}) \tag{6}$$

Such a bound produces a more realistic metric for certain applications, given that its relationsip to e_i is linear, rather than inverse as in Equation 5.

Another feature of energy aware routing is that nodes themselves can refuse to route packets once their energy level drops below a certain threshold, `RESIDUAL_ENERGY_LEVEL`. This enables nodes that are near exhaustion to be kept alive for reception and transmission of data associated with them, rather than expending resources routing data for others.

3.2.8 Quota-Based Routing

To ensure fair sharing of energy reserves and attempt to prevent DoS attacks that seek to exhaust nodes, the protocol implements a quota system. Each of a node’s neighbours are allocated a share of the remaining energy resources on the node for forwarding their packets. This quota is decreased each time the node routes a packet for the relevant neighbour.

Quotas are allocated by dividing the total energy reserves of a node equally between its neighbours. The metrics to the neighbours are then calculated on the basis of those quotas, using the same formulae as given in the previous section. Every interval of length `T_QUOTA_RESET_INTERVAL` the quotas are recalculated to ensure that the distribution of energy is not skewed (e.g. one neighbour may have a significant amount of traffic to send, whilst others use very little of their quota). Quota-based routing therefore causes nodes with high request frequencies for their traffic to be routed to be served less and less, a concept similar to that used in the Secure Routing Protocol (SRP) [20].

Nodes are able to keep track of their actual quotas on their neighbours by reading the `quota` field of any packets that have them as the next hop. Additionally, nodes estimate their quotas on their neighbours by decrementing their current estimates of their quotas each time they route a packet through the relevant neighbours. In this way they are able to maintain their routing tables closer to reality, without large numbers of dedicated updates.

On each quota reset, the node performing the reset calculates the fractional change in the quota that is assigned to each neighbour. If the difference from the previous quota and the quota to be assigned is greater than a factor of

`QUOTA_DELTA_FRACTION_FOR_UPDATE`, the node sends a dedicated quota update packet to the neighbour concerned.

When a node's quota reaches zero on a neighbour, that neighbour will silently drop any packets sent to it for routing by the node concerned. This will continue until the next quota reset, whereupon it will begin to route packets for that node once more.

3.2.9 Node Mobility

To simulate node mobility, the simulation makes use of a "billiard ball" model, obtained from the US NIST [16], where nodes move randomly within a defined bounding box. On reaching an edge of the area, the node's trajectory is reflected in a specular fashion. The model allows for the speed of movement and the step length to be varied as desired, enabling the simulation of different types of mobility (e.g. pedestrian and vehicular).

3.3 Security

Wireless mesh networks present various key security issues[3]:

- The need for data encryption due to the broadcast nature of wireless propagation.
- Introducing the concept of trust into the network to ensure that deliberately incorrect routing information is not propagated.
- Preventing DoS attacks on nodes that may cause their energy reserves to be exhausted quickly (the sleep depravation problem [32]). This has been covered in section 3.2.8.
- The problem of selfishness in so called "open" ad hoc networks, where membership is not restricted.

3.3.1 Encryption

The first of the above issues has not been considered in the design of the protocol, as it is assumed that encryption will be performed at the application layer.

[3]An excellent overview of the security issues (among many other architectural points discussed) can be found in the MobileMAN project's deliverable 5 report [6].

3.3.2 Trusted Routing Information

In a MANET, the distribution of correct routing data is of great significance in ensuring communication, due to the constantly changing characteristics of links and overall structure of the network.

In solving this problem the protocol assumes that it is to be used for a *closed ad hoc* network, i.e. where the membership is restricted. This is true in many scenarios, e.g. military applications or rescue teams, but in others such as peer to peer file sharing in a traffic network it does not apply.

All nodes in the network share a single secret key. This is assumed to stored in a physically tamper-proof form[4]. A Message Authentication Code (MAC) of the routing information that is to be transmitted can then be produced using a one way hash function. Such a scheme is implmented in the SEAD protocol [12], which uses one way hash chains.

By including a MAC in each packet, nodes are able to verify that routing informaiton received has been transmitted by a trusted node by recalculating the MAC of the data concerned.
Provided the hash function has the weak collision resistance property, it is deemed computationally infeasible (for a suitable key length; according to [26] the length of the output must now be at least 80 bits) for an attacker to generate an alternative set of data that will give an identical MAC. By the pre-image resistance property it is equally unlikely that the secret key could be obtained from a MAC and its corresponding input text.

In this way nodes are able to discard any routing data that they do not regard as trusted, and consequently prevent incorrect data being propagated by an attacker. It does not, however, guard against a trusted node becoming in some way compromised and performing an attack "from the inside".

The algorithm implemented makes use of a 128 bit shared secret key, supplied to any library encryption algorithm, with a 128 bit block of plain text. The resulting ciphertext is then used as the key for the next block of plaintext (Cipher Block Chaining). Once all the blocks (i.e. the last_hop, timestamp, delta_time, num_routes, and all routing information) have been enciphered, the most

[4]Although note that Anderson and Kuhn [14] caution against using this where possible.

significant 96 bits of the resulting 128 are taken and included in the MAC field of the packet.

3.3.3 Selfishness in Open Networks

This issue is not directly relevant to the project, (given that a closed network is assumed), but is considered in the additional implementation details in Appendix B.

4 Evaluation

To aid evaluation, testing was performed at each stage of the project, ensuring that individual components functioned correctly prior to the implementation of further layers based upon them. All evaluation was conducted inside the OPNET simulator.

4.1 Data Link Layer

A simple network consisting of two nodes, each composed of a packet source and a radio transceiver. The radio channel characteristics were initially configured to be lossless to confirm that nodes were able to send and receive. Interference was then introduced into the channel to cause a proportion of packets to be lost. Testing was achieved by outputting messages to the console when a packet was received. The implementation of CSMA was tested by detecting when a node entered its waiting to transmit state for a particular packet, simultaneously with another node in the process of broadcasting on the wireless channel. This was repeated with larger networks.

4.2 Transmission Control Layer

This was divided into two sub-goals:

- Connection tracking
- Automatic repeat request

A network of 3 nodes was used to test connection tracking, with each node generating and receiving traffic, and direct connections from each node to all others. Packet destinations were assigned according to a Uniform(1,3) distribution, which resulted in each node on average having three simultaneous connections in progress. Packets could be tracked through the network using sequence numbers. As shown in Figure 14, messages were output to the console when a packet was received or transmitted, indicating the connection identifier it was associated with. This provided confirmation that packets were reaching their correct destinations. In addition, graphs of numbers of packets transmitted and received over time provided confirmation that an appropriate number of packets were being received at each destination (and that therefore connections were being correctly identified).

For testing ARQ, the bit error rate of the wireless channel was increased to make the proportion of packet loss significant. For each packet retransmitted, a console message was output detailing the packet sequence number, and the connection identifier. Retransmissions were observed for a

```
Module (146), (top.Office Network.mobile_node_1.node_proc)
From procedure: AA_wbatt_procBattRouting [tx_pkt enter execs]
Packet transmitted to:
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Sent to router for (proc_source_pkts):
3
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Connection already in table (proc_source_pkts):
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Incrementing seq. number as we are sending
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [tx_pkt enter execs]
Packet transmitted to:
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Sent to router for (proc_source_pkts):
3
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Connection already in table (proc_source_pkts):
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Incrementing seq. number as we are sending
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [tx_pkt enter execs]
Packet transmitted to:
31
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Sent to router for (proc_source_pkts):
1
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Connection already in table (proc_source_pkts):
11
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Incrementing seq. number as we are sending
11
-----
Module (266), (top.Office Network.mobile_node_2.node_proc)
From procedure: AA_wbatt_procBattRouting [tx_pkt enter execs]
Packet transmitted to:
11
-----
Module (386), (top.Office Network.mobile_node_3.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Sent to router for (proc_source_pkts):
1
-----
Module (386), (top.Office Network.mobile_node_3.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Connection already in table (proc_source_pkts):
11
-----
Module (386), (top.Office Network.mobile_node_3.node_proc)
From procedure: AA_wbatt_procBattRouting [proc_source_pkts enter execs]
Incrementing seq. number as we are sending
11
-----
Module (386), (top.Office Network.mobile_node_3.node_proc)
From procedure: AA_wbatt_procBattRouting [tx_pkt enter execs]
Packet transmitted to:
11
-----
Module (146), (top.Office Network.mobile_node_1.node_proc)
From procedure: AA_wbatt_procBattRouting [process enter execs]
Packet Received from:
2
-----
```

Figure 14: Console Output for Connection Tracking: Node IDs are at the end of the first line of each message. The identifiers on the last line of each message indicate the connection identifier, where the first digit is the destination address, and the second the identifier for that address.

random selection of packets, as would be expected. In addition, graphs were obtained of the number of packets retransmitted by ARQ over time. Figure 15 shows such a plot. Varying error rates resulted in a greater number of retransmissions. By combining this information with the knowledge of which packets were dropped (given by the simulator), it could be seen that the correct packets were being retransmitted.

4.3 Mesh Routing Layer

The requirements for this layer were:

- Discover and maintain routes
- Ensure that routes remain loop free

Route discovery was tested by examining the routing tables of (stationary) nodes in the network periodically throughout the simulation. Figure 16 shows a sample routing table outputted to the console during a simulation with 20 nodes in the network. By comparing the topology of the network to the routing tables, the routes and metrics learnt were ascertained to be correct. To examine the convergence time of the network, (under the traffic patterns being used), a count of distinct destinations known to each node was obtained – an example is shown in Figure 17. This shows all nodes becoming aware of all other destinations shortly after the network was initialised, as required.

Routing was confirmed loop free by carefully examining pathological cases by hand, and by analysing node routing tables as simulations progressed. Were routing loops to have been present metrics for particular routes would have rapidly become infinite. This was not observed, and instead the expected sums of latencies making up

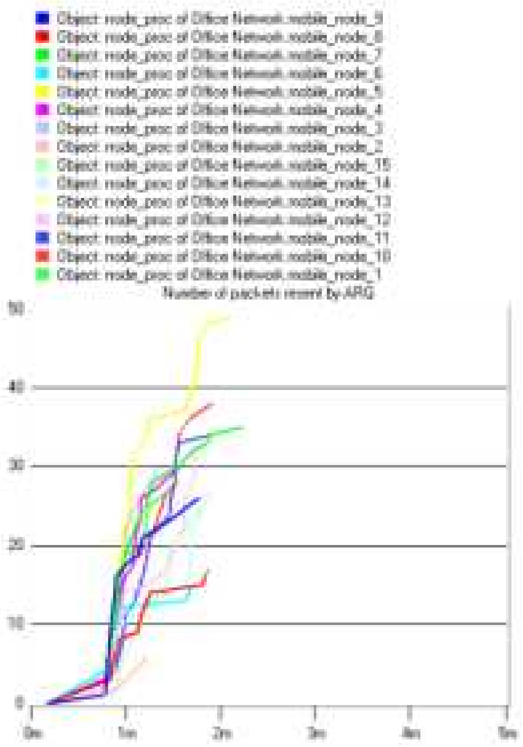


Figure 15: The number of packets retransmitted by the Automatic Repeat Request component of the Transport Layer by each node in the network over time.

```
Routing Table for node 4
Dest.    Via     Originator       Metric
12       9       9                100
12       5       9                133
12       8       9                151
12       1       9                174
13       5       8                105
13       9       8                128
13       8       8                135
13       1       2                156
1        1       4                35
1        5       5                68
1        8       8                96
2        5       8                100
2        8       8                101
2        9       8                127
2        1       8                155
3        5       5                92
3        8       5                121
3        1       2                150
3        9       5                153
3        3       4                9999
5        5       4                27
5        9       9                77
5        8       8                99
5        1       1                155
7        5       5                99
7        8       8                103
7        9       9                125
7        1       5                161
8        8       4                16
8        5       5                55
8        1       2                73
8        9       9                77
9        9       4                30
9        5       5                73
9        8       8                141
9        1       5                145
```

Figure 16: the contents of node 4's routing table at an arbitrary instant.

the routes were obtained as the metrics.

4.4 Energy Aware Routing & Quotas

The main goal of this project was to create a mesh networking protocol that conserved energy throughout the network, and moreover reduced the standard deviation of node energy levels.

The two energy aware algorithms (below referred to as *Energy 1* and *Energy 2*), detailed in Section 3.2.7 are compared to Latency based routing (referred to as *Normal*).

4.4.1 Simulation Environment

The simulations carried out made use of a network of 20 nodes, distributed uniformly over a space of 250 by 625 metres. The topology can be seen in Figure 18. Packet generation followed an exponential(1.0) distribution, (i.e. one packet generated per second), with a destination given by a Uniform(1,20) distribution.

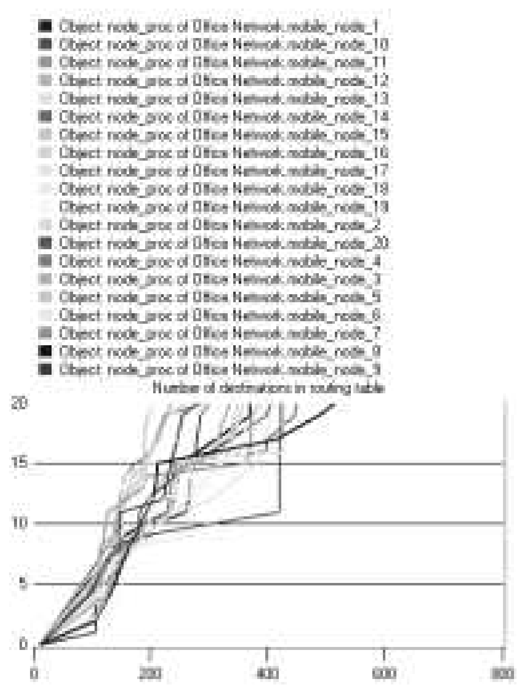


Figure 17: Number of distinct destinations in node routing tables over time. Convergence time in the simulation topology of 20 nodes is approximately 300 s for the majority of nodes.

The radio channel used was at a data rate of 1,024 bps, the background noise expected due to temperature fluctuations at 290 K, and an acceptable bit error rate of 0.01 (below which it is assumed the CRC used by the physical interface is of high enough degree to correct the error).

Simulations were performed using both stationary and mobile nodes, each combination (of the three algorithms and stationary or mobile) being simulated 5 times with differing seeds supplied to the random number generator, to obtain averages. Each simulation was conducted with an initial energy level of 1000 (packets-worth) per node. A total of 2,000 data points were collected, therefore only summary graphs are provided below.

To provide a representative estimate of the standard deviation (S.D.) of node energy levels after a certain period, an arbitrary time of 300 s was chosen. At this point, in all simulation runs, no nodes had yet been exhausted, and therefore a fair estimate of the S.D. could be obtained. The S.D. provides an indication to the degree to which nodes are being equally used for routing – greater spreads in values imply key nodes will be exhausted earlier.

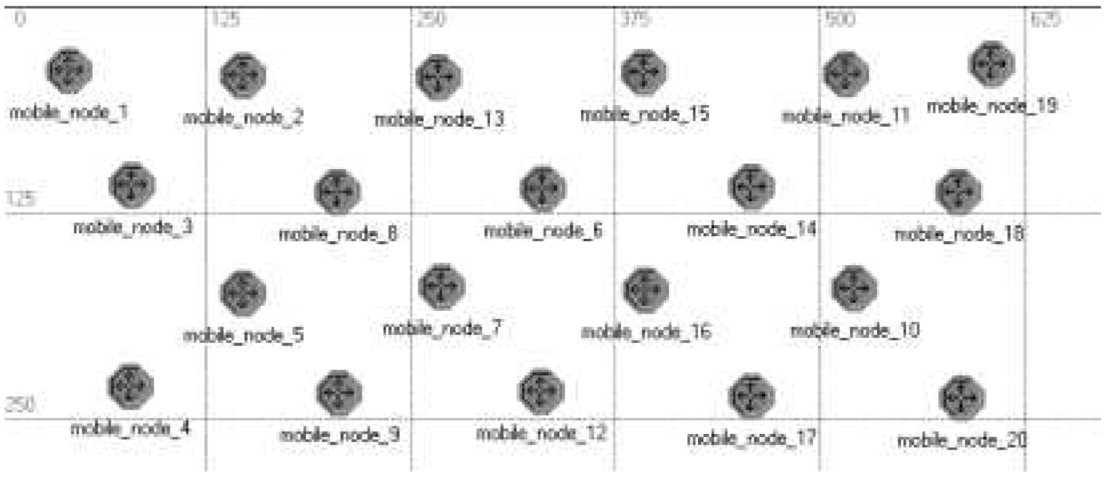


Figure 18: Topology used for simulations testing energy-aware routing: 20 nodes are distributed uniformly over an area of 250 m by 625 m.

4.4.2 Stationary Topology

Exhaustion Time

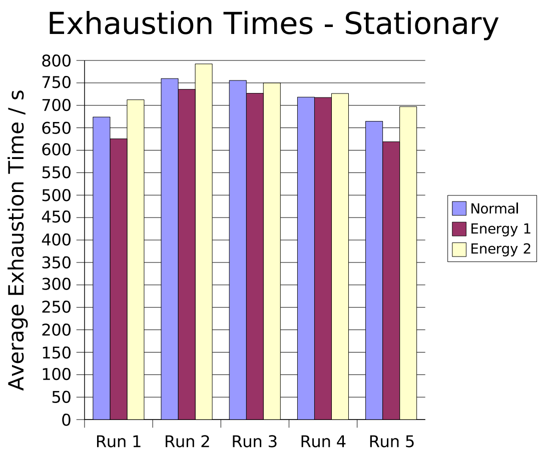


Figure 19: Average exhaustion time for stationary nodes: the Energy 2 algorithm yields a 3.1% average improvement.

The average node exhaustion times are shown in Figure 19 for five runs. From this it can clearly be seen that the Energy 2 algorithm increases or has no adverse effect on node lifetime. On average it yields a 3.1% improvement (maximum 6.0%) over Normal routing. The Energy 1 algorithm consistently underperforms in this case.

Level at 300s

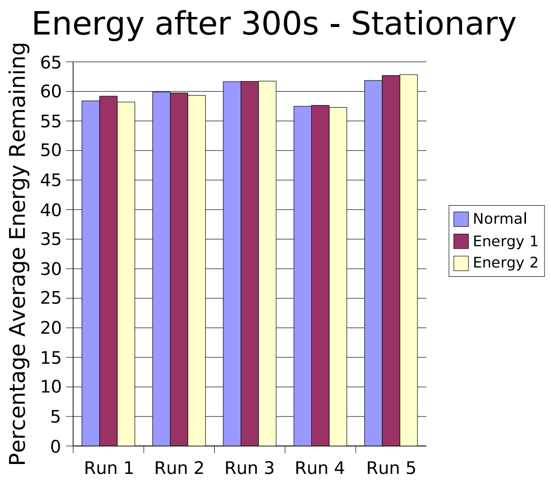


Figure 20: Average energy level after 300 s for stationary nodes: on average the Energy 2 algorithm marginally outperforms the Normal algorithm.

After 300 s, the average energy level in the network was found to be very marginally greater when using the Energy 1 algorithm, as shown in Figure 20. On average, the improvement is so small (0.02%) as to be attributable to the natural variation in the results. The Energy 2 algorithm again underperformed compared to Normal routing.

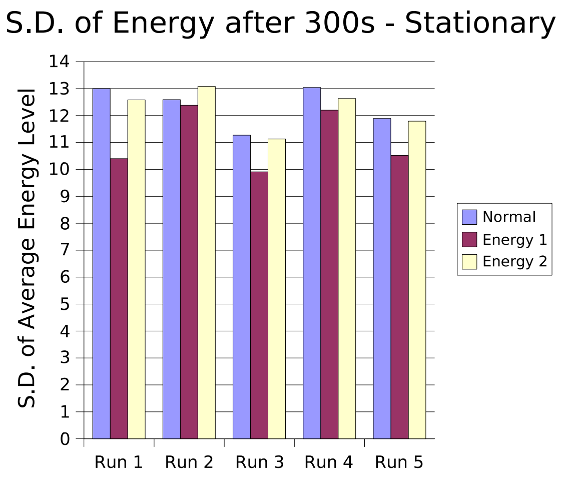


Figure 21: S.D. of energy levels after 300 s for stationary nodes: the Energy 1 algorithm gives (on average) a 10% lower S.D. than Normal routing.

The S.D. of node energies after the same time period is shown in Figure 21. The Energy 1 algorithm exhibits an average 10% lower S.D. as compared with Normal routing, with a maximum decrease of 20%. The Energy 2 algorithm gives only an average 4.1% improvement.

4.4.3 Mobile Topology

Mobility was simulated by moving nodes at a rate of 1 m per second, typical of a person's walking speed. The angle of direction of movement was random, according to a Uniform$(-1.0, 1.0)*\pi$ distribution. On reaching the boundary of the area, a node's direction of movement would be reflected.

Exhaustion Time

With mobile nodes, topology changes are frequent and affect average node lifetime considerably. There is no algorithm that in all runs gave an improvement over Normal routing, however, the Energy 1 algorithm yields a 3.3% improvement on average. The Energy 2 algorithm does not improve node lifetime in this case.

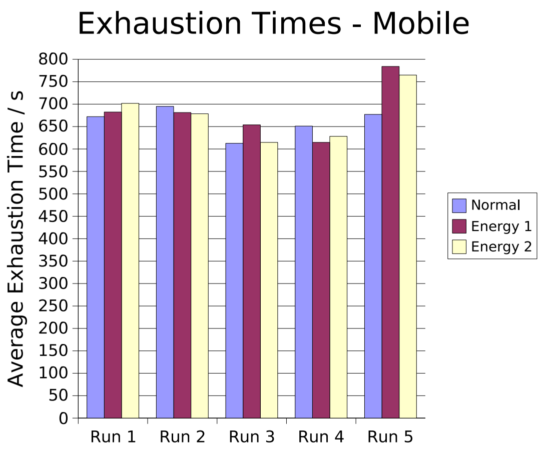


Figure 22: Average exhaustion time for mobile nodes: the Energy 1 algorithm yields a 3.3% average improvement.

Level at 300s

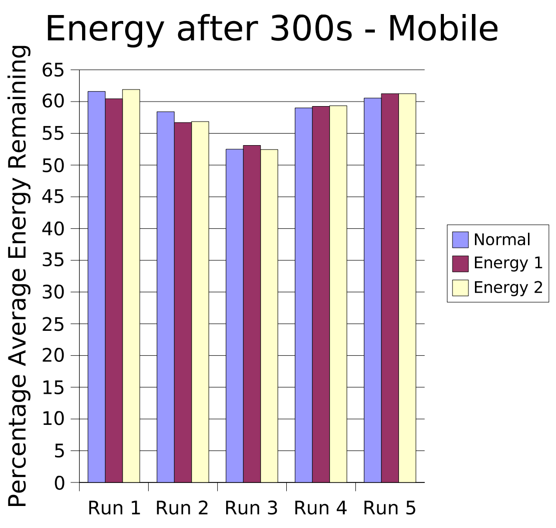


Figure 23: Average energy level after 300 s for mobile nodes: on average no benefit is gained from either of the Energy algorithms.

The average energy level after 300 s (Figure 23) is not on average changed by either of the Energy algorithms from the values obtained using Normal routing. This is not of particular concern, given that with all nodes being mobile, the key node exhaustion effect is not as prevalent compared to a static topology, resulting in less gain being obtained by using energy routing.

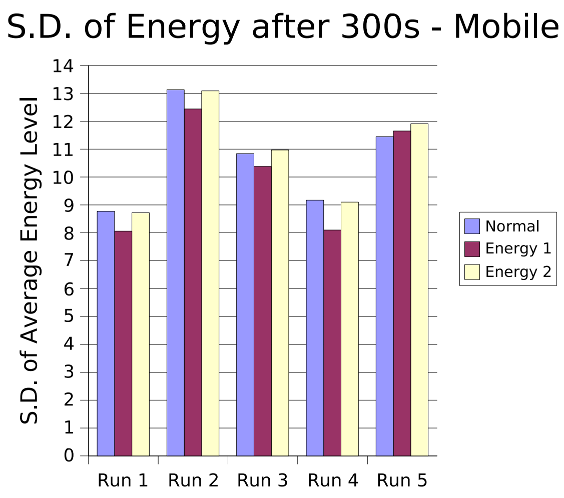


Figure 24: S.D. of energy levels after 300 s for mobile nodes: the Energy 1 algorithm gives (on average) a 5.5% lower S.D. than Normal routing.

In terms of the S.D. in energy levels after 300 s (Figure 24), the Energy 1 algorithm is again superior, decreasing the S.D. by an average of 5.5%, up to a maximum of 8.1%.

4.4.4 Discussion of Algorithms

The results above indicate, as expected, that in all respects an energy-aware routing protocol outperforms a purely latency-based scheme. For node exhaustion time, it is of note that the better algorithm is dependent on whether nodes are stationary or mobile. In the former case the Energy 2 algorithm outperforms the Energy 1 algorithm, whilst in the latter it is the opposite.

For all other tests the Energy 1 algorithm outperforms the Energy 2 scheme. The data show that energy-aware routing does provide real improvements in the standard deviation of node power levels at an arbitrary point during the network's lifetime, as was hoped for in the original proposal, in addition to the increase in average node exhaustion time described above.

4.4.5 Reliability of Simulation

It should be noted that the comparison of different ad hoc network routing algorithms has been questioned (see [3]), due to their significantly different methods of operation (and the implementation of these). The comparisons presented here are for one (custom) protocol, with an energy-aware layer constructed above it, using a common simulator. This therefore allows a fair analysis to be performed as to whether energy-aware routing is worthwhile.

4.4.6 Goals Achieved

The aims of the project have been achieved:

- All core components have been completed.
- Several extensions, namely, bitmapped ARQ, one way hashes of routing data, and rudimentary collision avoidance, have been implemented.
- Two different algorithms for energy-aware routing have been compared with purely latency based routing, using the same underlying protocol.

The only work that was not completed was the remainder of the extensions, as was expected (these were proposed for implementation if time allowed).

5 Conclusions

5.1 Outcome

The project has been a success: all acceptance criteria given in the proposal have been achieved, and several extensions implemented.

As evidenced by the evaluation, using an energy-aware routing protocol can result in worthwhile improvements in mesh network lifetime, up to 6% in certain cases. Depending on whether the mesh network is composed of mobile or stationary nodes, a different energy-aware algorithm should be used: the Energy 1 algorithm performs better with mobile nodes.

It is highly likely that the energy-aware metric calculation algorithms presented could be refined to have a much greater impact: with greater amounts of time and better automated results collection it would be interesting to investigate this.

5.2 Further Work

Mobile *ad hoc* networks are still very much an active research area, with security, energy management, and scalability work ongoing. For the protocol implemented in this project, the following are possible extensions:

- **Investigating trust systems for open MANETs:** various protocols have been suggested using hash chains (e.g. TESLA [24]) to prove the authenticity of a datagram's source in an open network. Adding support for such networks would mean the protocol was suitable for *ad hoc* peer to peer (P2P) systems.

- **Co-ordinate systems for mesh:** traditional peer to peer systems on wired networks are subject to a high degree of churn, similar to the constantly changing topology of a MANET. Research has been conducted on co-ordinate systems for P2P [25], with a view to enabling nodes to access content on the nearest host to them. Applying such co-ordinate systems to mesh networks could assist in decreasing the energy resources used, by relating the distances in the co-ordinate space to the energy quotas system.

- **Hardware level power saving:** the PEN system [33] made use of layer 2 techniques to reduce the power consumption of nodes by having defined hibernation periods. This does, however, increase the latency of the links in the network. Implementing such power saving features in the protocol from this project would certainly be worthwhile, although their use would depend on the latency that could be tolerated.

- **Zone routing and wired networks:** modifying the routing protocol to have a hierarchical address space, to allow routing to different zones [36], is possible, but would tend to imply the existence of gateways to each zone. In the same way some sort of gateway paradigm is needed for interfacing with wired networks. Reliability and energy management at the gateway would make worthwhile research.

6 Appendix A: Survey of Ad Hoc Routing Protocols

A more in depth analysis can be found in the survey by Royer and Toh [28], but specific details are given in the references for each protocol:

Table-Driven:

- *DSDV:* Destination-Sequenced Distance Vector [22]. Each node maintains a table of hop counts to all other destinations in the network. Routing information is broadcast periodically in dedicated packets. This treats all nodes equally, but has a relatively high routing data overhead.

- *CGSR:* Cluster-Head Gateway Switch Routing [4]. Based on DSDV, but groups of nodes (clusters) elect a control node that allocates channel access and bandwidth. Routing is via inter-cluster gateway nodes. This scheme has a very high overhead in calculating gateway/head nodes if the topology is changing very frequently. It also places more of a processing workload on the cluster-heads and gateways, and has them as key single points of failure.

- *WRP:* Wireless Routing Protocol [18]. Similar again to DSDV, but to avoid the "count to infinity" problem, uses the second-to-last hop for a destination as the attribute of a route rather than a sequence number, to avoid loops. This provides faster route convergence on network partition.

Source-Driven:

- *AODV:* Ad-hoc On-demand Distance Vector [23]. Nodes only store routes that are currently in use. When a node wishes to communicate with another, if it does not have a valid route in its table it broadcasts a route request (RREQ) packet, which is then propagated throughout the network. When the destination receives the RREQ, it unicasts a route reply (RREP) packet back towards the source. Intermediate nodes maintain state as to which neighbour they received the corresponding RREQ from first, and unicast the RREP back to that neighbour. The source therefore obtains a route that is set up with the destination as the root. This method has the problem of high route-setup latency.

- *DSR:* Dynamic Source Routing [13]. When a route is required, the source node broadcasts a request, containing its own address, and that of the destination. This is rebroadcast by all its neighbours, each adding their own address to the list of hops composing the route in the request packet. When the packet reaches the destination, it will contain the full route as a list of hops; the destination simply unicasts this data back to the source. This implies that for a large number of hops routing packet sizes will be significant. On network partition a large number of request messages are also likely, as new routes will need to be set up, rather than neighbours having multiple entries for a destination in their tables as in table-driven algorithms. Given that for each new destination, regardless of similarity to a previously learnt route, a node will cause network-wide broadcasts of requests, power usage will be inefficient. In a network where the topology rarely changes, the routing overhead for DSR can be as low as zero, once the network has stabilised. However, the same could be said for source-routed wired network protocols.

- *TORA:* Temporally Ordered Routing Algorithm [21]. A distributed link-reversal algorithm that relies on a synchronised clock (such as that provided by the Global Positioning System), to order the protocol's reaction to topology changes. During route creation or maintenance phases, each node is assigned a *height* metric which determines the direction of flow in a directed acyclic graph (DAG) rooted at the destination of the route. The algorithm's strength is that in very large networks topology changes are communicated only to a small set of nodes local to the change. However, this requires the mesh to be dense. An additional problem is that the protocol has the potential for "link flapping" as nodes erase routes and discover new ones based on each other's tables, but convergence will eventually be achieved.

7 Appendix B: Additional Implementation Details

7.1 Detailed Packet Format

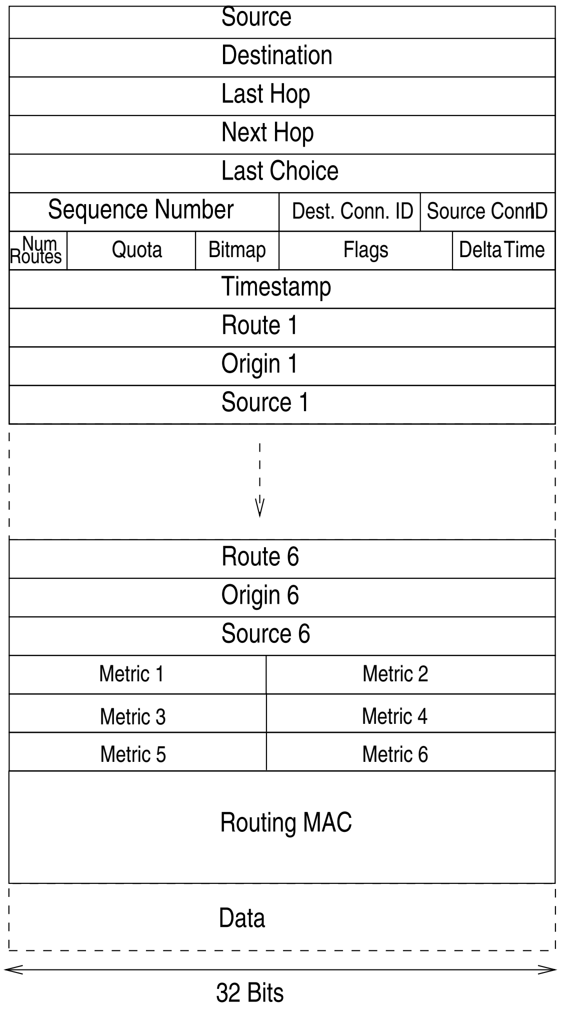


Figure 25: The format of the packets used in the protocol. each row represents 32 bits. There are 6 sets of three routing data fields, only numbers 1 and 6 are shown. The data field is of variable length.

Source, Destination, Last Hop, Next Hop, Last Choice: contain the 32 bit addresses of the appropriate nodes.

Sequence Number: a 16 bit number identifying the position of the packet in the overall stream being transmitted.

Destination Connection ID: an 8 bit identifier, equivalent to a TCP destination port number.

Source Connection ID: the source "port" number.

Number of Routes: the number of routing entries carried in the packet (ranging from 0 to 6).

Quota: 8 bits specifying the transmission quota the next hop node has on the last hop node.

Bitmap: 5 bits, set to indicate which packets in the last *chunk* (series of five) have been received, and which have not (used for bitmapped ARQ).

Flags: 9 bits. Please see below for a complete listing.

Delta Time: 6 bits storing the time elapsed from the packet's timestamp and it being re-transmitted by the last hop node.

Timestamp: 32 bits, set by the original sender and unchanged by
subsequent hops.

Route x, Origin x, Source x: three 32 bit fields, holding the addresses needed for a routing entry. There are 6 sets of these fields, although not all are required to be filled. See section 3.2.2 for details.

Metric x: six 16 bit fields, each specifying the metric from the last hop node to the target destination in the corresponding Route field (e.g. the route entry to destination Route 1 has a metric of Metric 1).

Routes MAC: a 96 bit Message Authentication Code hash of the routing data in the packet, calculated using a shared secret key, (see section 3.3.2).

Data: a variable length field, used to carry application level data.

Flags

The following 1 bit fields are located in the area marked `Flags` in Figure 25:

Priority: set if the packet should be given priority in the transmission or routing queues (provides a crude method of rating the importance of traffic).

Ack: set if the packet contains an acknowledgement, i.e. if the bitmap field should be read by the destination.

Multicast: set if the packet is for multicast (unimplemented – for future use).

Encrypted: set if the packet data is encryped (unimplemented – encryption is currently assumed to take place at application level, but could easily be implemented in the protocol).

Routing Data: set if the packet contains any type of routing data (route entries, or destination unreachable messages), or if the packet is a plea.

Route Query: set if the packet is a plea for a route to be supplied.

Destination Unreachable: set if the source field of the packet is not reachable from the last hop node (see section 3.2.5).

Enough Resources: if unset in a destination unreachable message, indicates that there was an insufficient quota on one of the nodes on the route to forward the packet.

First Packet: set if this is the first packet to be sent on this connection (see section 3.1.2).

7.2 Selfishness in Open Networks

Open networks, i.e. those with unrestricted membership, differ from closed networks in that there is no *a priori* trust available. Nodes may be trusted, or they may mount active or passive attacks on the mesh. The protocol designed in this project has not attempted to solve the security problems of open networks, but this section is included to justify this simplification.

Passive attacks are those in which a node does not act as a router in the network. This can result in “black holes” in the network, causing network partition. Such attacks can be serious if there are collaborating nodes that carry out this attack.

Active attacks include spoofing, and reporting incorrect information, such as energy levels or quotas, to avoid the perpetrator being called upon to route data.

Attempting to ensure that routing is somehow “secure” in such an open network is complex, and an active area of research. Credit based systems [8] appear to be a possible solution, providing incentives for principles to be honest, but others involving decentralised public key management [2] have been proposed.

7.3 Event List & Time-outs

A significant part of any network protocol is its usage of time-outs, and therefore its concept of events. In simulating the protocol under OPNET, the discrete event list management was handled by the simulation software itself. However, were the protocol to be implemented outside the simulator, event/timeout management could be performed using a *delta list* data structure as described by Comer and Stevens in [5]. Briefly summarised, this system maintains a linked list of events that are scheduled to take place, ordered by their *relative* due times. This is efficient as the periodic timer need only decrement the time of the event at the head of the list. When the time in the first item reaches zero, the event is executed, and the next event moves to the head of the list.

8 Appendix C: Values of Simulation Constants

The following details the values of the constants used in the simulation that have been mentioned throughout the text. The below is taken directly from the C source file – any constants that relate to time are in units of milliseconds.

```c
/* Estimate of average RTT to initially give a connection */
#define RTT_ESTIMATE 40

/* Number of RTTs the node waits for a packet from the destination
before a time out occurs. */
#define TIMEOUT_RTT_MULTIPLE 2

/* Number of RTTs the node should wait for an ACK before starting resend.*/
#define TIMEOUT_RESEND_RTT_MULTIPLE 2

/* Maximum amount of simulation time that a connection can go without
receiving a packet before it is deemed unused and can be freed by the
queue allocator. */
#define MAX_IDLE_TIME 80

/* Maximum amount of time before ACK gets sent: it may be that before
this expires the ACK can be piggybacked on a data bearing packet, but in
the case it cannot, this time-out will expire and a non-data bearing ACK
will be sent. */
#define TIME_TO_WAIT_FOR_CARRIER 10

/* The maximum number of repeat transmissions allowed for one chunk
before the route the connection has been using is blackmarked. Value is
one more than one entire chunk. */
#define MAX_RETRIES_PER_CHUNK 6

/* The maximum time period between forced "beacon" signals from nodes,
if they have not sent any other packets in the meantime (the timeout
should execute and test a flag to see if a packet has been sent. If so
then no action need be taken. If not, a beacon packet is transmitted. */
#define T_BEACON 50

/* The time for which a node should not allow a blackmarked route to be
updated by routes propagated for others. This is twice T_BEACON, to
allow network convergence. */
#define T_HOLD_TIME 70

/* Above what fractional threshold of change in a route's metric should
a routing update be transmitted urgently. */
#define THRESHOLD_FRACTION_METRIC_UPDATE 0.5

/* The amount of time a route that has its metric set to 999 can be kept
in a routing table entry list, before it is deleted. This time is
dependent on how long it will take to update all the neighbours that
this route is no longer valid. */
#define T_KEEP_DEFUNCT_ROUTES 45
```

```
/* Total number of pleas likely to be needing to be kept track of at one
time in the pleas hash table. */
#define PLEAS_TABLE_SIZE 20

/* The maximum time for which a packet for a destination should be
continuously re-queued into the routing queue whilst we wait for our
neighbours to respond to our plea. */
#define T_ROUTE_PLEA_WAIT 80

/* The maximum amount of time for which, after a plea has expired (i.e.
no route could be found), packets to the same destination will simply be
dropped, without a new plea being sent. This is dependent on rate of
topology change. */
#define T_HOLD_PLEA_FAILURE 100

/* The fractional change in estimated node quota above which route
metrics are recalculated. */
#define QUOTA_METRIC_UPDATE_THRESHOLD_FRACTION 0.1

/* The fraction by which a quota may change on quota reset below which
no quota update packet is sent. */
#define QUOTA_DELTA_FRACTION_FOR_UPDATE 0.1

/* The interval between quota resets. Should be the same as T_BEACON. */
#define T_QUOTA_RESET_INTERVAL 80

/* The initial estimate of this node's quota on another node. Should be
QUOTA_ZERO_THRESHOLD * 4 */
#define START_QUOTA_ESTIMATE 20
```

9 Appendix D: Simulator Finite State Machine

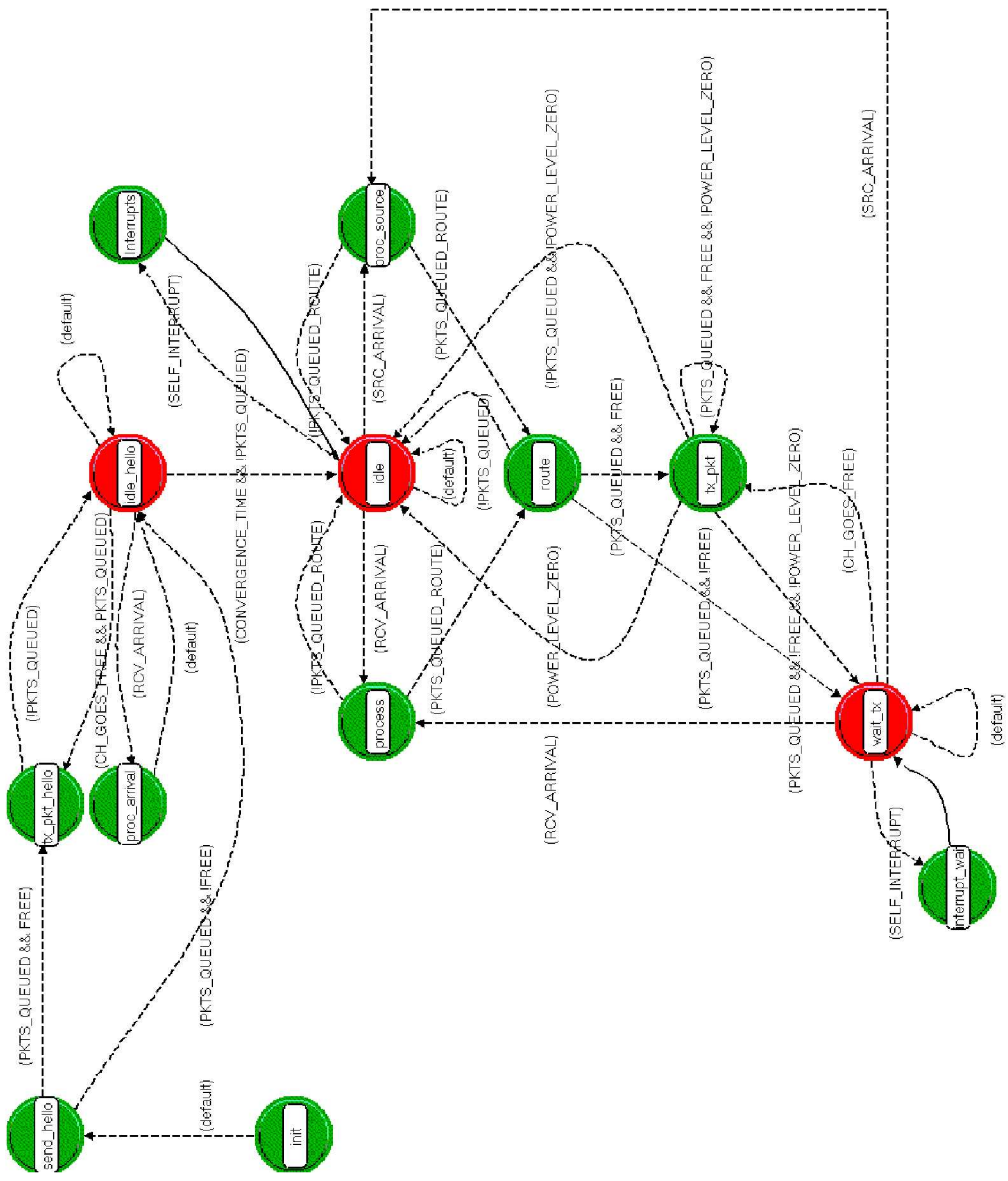


Figure 26: The finite state machine representing the protocol implemented in the OPNET simulator. Each state's name is indicative of its function, with the conditions causing the transition above their arcs.

10 Appendix E: Sample Code

The below is the code used for quota management and metric calculation based on quotas on each node. Each function should be self explanatory.

```
void update_quota_estimate(Packet* packet_to_processx) {
    /* Updates our estimate of our quota on the last_hop node of the packet. */

    int srcQuotaAddress;
    int quotaq;
    struct neighbour* neighbourQuotaToUpdatey;
    char hashStringQuotay[4];

    op_pk_nfd_get(packet_to_processx, "last_hop", &srcQuotaAddress);
    op_pk_nfd_get(packet_to_processx, "quota", &quotaq);

    //itoa(srcQuotaAddress, hashStringQuotay, 3);
    sprintf(hashStringQuotay, "%d", srcQuotaAddress);

    neighbourQuotaToUpdatey = prg_string_hash_table_item_get(neighbours_table,
                              hashStringQuotay);
    if (neighbourQuotaToUpdatey != PRGC_NIL) {
        neighbourQuotaToUpdatey->ourQuotaEstimate = quotaq;
    }
    else {
        printf("srcQuotaAddress = %d", srcQuotaAddress);
        op_sim_end("SIMULATION TERMINATED:", "Entry not found in neighbours
            table", "in update_quota_estimate()", "(PROBABLY called from
            route state).");
    }
    if (debug_level >= 4) printf("\nAbout to call reCalcMetricsBasedOnQuota() in
                          update_quota_estimate().\n");
    reCalcMetricsBasedOnQuota(hashStringQuotay);
}

void decrease_quota_estimate(Packet* packet_to_process) {
    /* Decreases our esimate of our quota on a neighbour node as we are
    sending a packet. */
    int destQuotaAddress;
    int nxHopQuotaAddress;
    struct neighbour* neighbourQuotaToUpdatex;
    char hashStringQuotax[4];

    op_pk_nfd_get(packet_to_process, "next_hop", &nxHopQuotaAddress);
    op_pk_nfd_get(packet_to_process, "destination", &destQuotaAddress);

    sprintf(hashStringQuotax, "%d", nxHopQuotaAddress);
    /* Only decrease the quota if this node is expected to transmit the
    packet on - if not they will incurr no transmission energy usage. */
    if (nxHopQuotaAddress != destQuotaAddress) {
        if ((neighbourQuotaToUpdatex = prg_string_hash_table_item_get(neighbours_table,
                                              hashStringQuotax)) != PRGC_NIL) {
            if (neighbourQuotaToUpdatex->ourQuotaEstimate > 0) neighbourQuotaToUpdatex->
                    ourQuotaEstimate = neighbourQuotaToUpdatex->ourQuotaEstimate
```

```
                                        − QUOTA_PER_PACKET_COST;
            reCalcMetricsBasedOnQuota(hashStringQuotax);
        }
    }
}

void update_quota_on_this_node(Packet* packet_to_process_upd) {
    /* Decreases the quota of the last_hop node of the packet on us (this node). */

    int lastHopQuotaAddress;
    struct neighbour* neighbourQuotaToUpdate_upd;
    char hashStringQuota_upd[4];

    op_pk_nfd_get(packet_to_process_upd, "last_hop", &lastHopQuotaAddress);
    sprintf(hashStringQuota_upd, "%d", lastHopQuotaAddress);

    neighbourQuotaToUpdate_upd = prg_string_hash_table_item_get(neighbours_table,
                                hashStringQuota_upd);
    neighbourQuotaToUpdate_upd−>theirQuota = neighbourQuotaToUpdate_upd−>
                        theirQuota − QUOTA_PER_PACKET_COST;
}

int insert_quota_for_next_hop(Packet* pk_to_insert_quota) {
    /* Inserts the next hop node's quota on this node into the packet. */

    int nxHopAddr;
    char hashString[4];
    struct neighbour* nxHopQuota;
    int quota_to_ins;

    op_pk_nfd_get(pk_to_insert_quota, "next_hop", &nxHopAddr);
    sprintf(hashString, "%d", nxHopAddr);

    nxHopQuota = prg_string_hash_table_item_get(neighbours_table, hashString);
    quota_to_ins = nxHopQuota−>theirQuota;

    return quota_to_ins;
}

void reCalcMetricsBasedOnQuota(char* hashStringQuotaUpdatedz) {
    /* Decides whether the updated quota has undergone a significant
    enough change to warrant updating all the route metrics associated
    with the node the quota is concerned with. */

    struct neighbour* neighbourQuotaUpdatedz;
    int delta_quota;
    int boolUpdateMetrics = 0;
    PrgT_List* keysList;
    int i = 0, j = 0;
    char* keyRemoved;
    List* routeList;
    int lengthList;
    struct route* route_examining;
    int neighbourAddress;
```

```
neighbourQuotaUpdatedz = prg_string_hash_table_item_get(neighbours_table,
                        hashStringQuotaUpdatedz);
/* Ensures that the difference in quota is positive for the
fractional test below. */
delta_quota = neighbourQuotaUpdatedz->ourQuotaEstimate
            - neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate;
if (delta_quota < 0) delta_quota = 1 - delta_quota;

if ((neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate > 0) &&
    ((delta_quota/neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate) >
            QUOTA_METRIC_UPDATE_THRESHOLD_FRACTION)) {
    /* Update all routes associated with this neighbour,
    as the quota has changed substantially since the last
    metrics update. */
    boolUpdateMetrics = 1;
}

if ((neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate == 0) &&
    (neighbourQuotaUpdatedz->ourQuotaEstimate > QUOTA_ZERO_THRESHOLD)) {
    /* The quota for this neighbour was previously 0, but has now been updated
    above a threshold value, so update routes. If it were 0 and the update is
    below the threshold it is not worth updating the routes to make them valid
    again. */
    boolUpdateMetrics = 1;
}

if (boolUpdateMetrics == 1 && use_battery_routing == 1) {

    neighbourAddress = neighbourQuotaUpdatedz->address;

    /* Update all routes associated with this neighbour. */
    keysList = prg_string_hash_table_keys_get(routing_table);

    while (i < op_prg_list_size(keysList)) {
        if (debug_level == -1) printf("\nrecalcMetricsBasedonQuota().");
        keyRemoved = (char*)op_prg_list_remove(keysList,i);
        routeList = (List*)prg_string_hash_table_item_get(routing_table, keyRemoved);
        lengthList = op_prg_list_size(routeList);
        for (j=0; j<lengthList; j++) {
            route_examining = op_prg_list_access(routeList, j);
            if (route_examining->nextHop == neighbourAddress) {
                /* This route uses the neighbour that has had its
                quota updated, so update the entry and resort the list. */
                route_examining->metric = metric_calculator(route_examining->
                    metric, neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate,
                    neighbourQuotaUpdatedz->ourQuotaEstimate);
                if (route_examining->metric == 0) {
                    printf("\nneighbourQuotaUpdatedz->ourQuotaLastMetricUpdate = %d,
                        neighbourQuotaUpdatedz->ourQuotaEstimate = %d",
                        neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate,
                        neighbourQuotaUpdatedz->ourQuotaEstimate);
                    op_sim_end("SIMULATION TERMINATED", "Metric is
                                0.", "See console", "x");
                    }
                    route_examining->updateTime = (int)op_sim_time();
                    route_examining->listInserter = 1818;
                    op_prg_list_insert(urgent_route_list, route_examining,
                                    OPC_LISTPOS_TAIL);
                }
            }
            op_prg_list_sort(routeList, compRouteMetrics);
        }

        /* Deallocate the elements on the list */
        op_prg_list_free (keysList);
        /* Deallocate the list itself */
        op_prg_mem_free (keysList);
        /* The quota used for the last metrics update for this neighbour must be
        changed to the current quota. */
        neighbourQuotaUpdatedz->ourQuotaLastMetricUpdate =
            neighbourQuotaUpdatedz->ourQuotaEstimate;
    }
}

int quota_remaining(Packet* packet_to_test_rem) {
    /* Tests to see whether the last_hop of the packet has sufficient quota on this node to
    have the packet relayed. Returns 1 if so, otherwise returns 0. */

    int last_hop_v;
    char hashStringq[4];
    struct neighbour* neighbour_to_test_rem;

    /* If not using battery routing quota values do not matter. */
    if (use_battery_routing == 0) return 1;

    neighbour_to_test_rem = prg_string_hash_table_item_get(neighbours_table, hashStringq);
    if (neighbour_to_test_rem->theirQuota < QUOTA_PER_PACKET_COST) return 0;
    else return 1;
}

int metric_calculator(int old_metric, int old_quota, int new_quota) {
    /* Calculates a new route metric based on the old metric and the change in neighbour
    quota.  Old Metric, m. New metric, n. Latency, l. Old quota, q. New quota, r.
    m = l * (1/q + 1)
    n = l * (1/r + 1)
    l = m/(1 + 1/q)
    n = (m/(1 + 1/q))(1/r + 1)
    */
    int new_metric = old_metric;
    int latency;
    double quotaFrac = old_quota/INITIAL_POWER_LEVEL;

    /* Without battery routing we only distribute latencies, therefore the old metric will
    be a time, not a composite that must be re-calculated, as with battery routing enabled. */
    if ((use_battery_routing == 1) && (old_quota != 0) && (old_metric !=
        INFINITE_METRIC) && (battCalcMethod == 1)) latency = (int)
        (old_metric / (1.0 + (INITIAL_POWER_LEVEL / 2.0)/old_quota));
    if ((use_battery_routing == 1) && (old_quota != 0) && (old_metric !=
        INFINITE_METRIC) && (battCalcMethod == 2)) latency = (int)
        (old_metric / (2.0 − quotaFrac));
    if ((use_battery_routing == 1) && (old_quota <= QUOTA_ZERO_THRESHOLD)
                && (old_metric == INFINITE_METRIC)) {
        /* Cannot calculate the latency from the old_metric if this was set to
        INFINITE_METRIC, so use initial estimate. This would be due to a quota
        below the threshold. */
        latency = RTT_ESTIMATE;
    }
    if ((use_battery_routing == 1) && (old_quota > QUOTA_ZERO_THRESHOLD)
                  && (old_metric == INFINITE_METRIC)) {
        /* Cannot calculate the latency from the old_metric if this was set to
        INFINITE_METRIC, so use initial estimate. This would be due to a latency
        that was infinite. */
        latency = INFINITE_METRIC; /* This is for clarity - it will have no effect. */
        /* This causes metric_from_latency_and_quota() to return a metric
        of INFINITE_METRIC, as desired, as this route is still infinite latency. */
        new_quota = 0;
    }
    if ((use_battery_routing == 1) && (old_quota == 0) && (old_metric !=
                          INFINITE_METRIC)) {
        /* Should not get here: any neighbour with zero quota should have a metric of
        INFINITE_METRIC already. */
        if (debug_level > 0) op_sim_message("WARNING: In metric_calculator(),
        old_quota == 0 with old_metric != INFINITE_METRIC", "Strange effects
                may result! (setting latency = old_metric)");
        latency = old_metric;
    }
    if (use_battery_routing == 0) { latency = old_metric; }

    new_metric = metric_from_latency_and_quota(latency, new_quota);

    return new_metric;
}

int metric_from_latency_and_quota(int latency, int quota_to_use) {
    /* Calculates the metric to use from the latency and the quota to use. (Simple, but
    provides one function to update). */

    int metric_to_return;
    if ((use_battery_routing == 1) && (quota_to_use != 0) && (battCalcMethod == 1))
        metric_to_return = latency * (int)(1.0 + (INITIAL_POWER_LEVEL/2.0)/quota_to_use);
    if ((use_battery_routing == 1) && (quota_to_use != 0) && (battCalcMethod == 2))
        metric_to_return = latency * (int)(2.0 − quota_to_use/INITIAL_POWER_LEVEL);
    if ((use_battery_routing == 1) && (quota_to_use == 0)) {
        /* If the quota_to_use is 0 then the metric must be INFINITE_METRIC. */
        metric_to_return = INFINITE_METRIC;
    }

    if (use_battery_routing == 0) metric_to_return = latency;
    if (metric_to_return > INFINITE_METRIC) {
        metric_to_return = INFINITE_METRIC;
    }
    return metric_to_return;
}

void quotas_reset() {
    /* Executed on a self_interrupt (code 4). Resets the quotas for all neigbours of this
    node to a proportional share of the total power remaining on this node. */

    int neighbours_table_size = 0;
    int quota_to_assign = 0;
    List* neighbours_table_list;
    struct neighbour* neighbour_updating;
    int i = 0;
    int delta_quota = 0;

    neighbours_table_list = prg_string_hash_table_values_get(neighbours_table);
    neighbours_table_size = op_prg_list_size(neighbours_table_list);

    /* Prevent division by 0 when no neighbours are present, or have yet been found. */
    if (neighbours_table_size <= 0) neighbours_table_size = 1;
    quota_to_assign = (power_level − RESIDUAL_POWER_LEVEL) /
                    neighbours_table_size;
    op_stat_write(quotaResetValue_stathandle, quota_to_assign);
    neighbours_table_size = op_prg_list_size(neighbours_table_list);

    /* Assign this to all neighbours. */
    for (i=0; i < neighbours_table_size; i++) {

        neighbour_updating = op_prg_list_access(neighbours_table_list, i);
        delta_quota = neighbour_updating−>theirQuota − quota_to_assign;
        if (delta_quota < 0) delta_quota = −delta_quota;
        if (((neighbour_updating−>theirQuota == 0) && (delta_quota >
            QUOTA_ZERO_THRESHOLD)) || ((neighbour_updating−>
            theirQuota != 0) && ((delta_quota/neighbour_updating−>
                theirQuota) > QUOTA_DELTA_FRACTION_FOR_UPDATE))) {
            sendQuotaUpdate(neighbour_updating−>address, quota_to_assign);
        }
        neighbour_updating−>theirQuota = quota_to_assign;
        if (debug_level >= 4) printf("Quota assigned to %d was %d",
            neighbour_updating−>address, neighbour_updating−>theirQuota);
    }

    /* Destroy the old quota reset timeout, if it exists, and create a new one. The
    interrupt code understood by process_interrupts() is 4. */
    if ((op_ev_pending(quota_reset_timeout)) && (op_ev_time(quota_reset_timeout) <=
                    op_sim_time())) op_ev_cancel(quota_reset_timeout);
    quota_reset_timeout = op_intrpt_schedule_self((op_sim_time() +
                    T_QUOTA_RESET_INTERVAL), 4);
}

void sendQuotaUpdate(int neighbourAddress, int new_quota) {
    /* Transmits a routing update packet to a specific node to cause its estimate of its
    quota on this node to be updated. These packets are sent due to greater than
    QUOTA_DELTA_FRACTION_FOR_UPDATE fractional changes in quotas controlled
    by this node. */

    Packet* pk_to_send;

    pk_to_send = op_pk_create_fmt("AA-wbatt_pkt_format");
    op_pk_nfd_set(pk_to_send, "source", thisNodeAddress);
    op_pk_nfd_set(pk_to_send, "destination", neighbourAddress);
    op_pk_nfd_set(pk_to_send, "last_hop", thisNodeAddress);
    op_pk_nfd_set(pk_to_send, "last_choice", thisNodeAddress);
    op_pk_nfd_set(pk_to_send, "seq_number", 0);
    op_pk_nfd_set(pk_to_send, "ack", 0);
    op_pk_nfd_set(pk_to_send, "bitmap_ack", 0);
    op_pk_nfd_set(pk_to_send, "dest_unreach", 0);
    op_pk_nfd_set(pk_to_send, "route_query", 0);
    op_pk_nfd_set(pk_to_send, "routing_data", 1);
    op_pk_nfd_set(pk_to_send, "dest_unreach", 0);
    op_pk_nfd_set(pk_to_send, "num_routes", 0);
    /* Note that this is for clarity only: in the routing algorithm the next_hop is assigned
    then the quota for that hop is inserted. Therefore this relies on the lowest metric
    and route to the neighbour being transmitting directly to it, which may not always be the
    case... but for simplicity I assume that it is. The pathological case is when this
    node has 0 quota on the neighbour node, causing the packet to take a different route.
    This is, however, unlikely. */
    op_pk_nfd_set(pk_to_send, "quota", new_quota);

    op_subq_pk_insert(ROUTING_QUEUE, pk_to_send, OPC_QPOS_TAIL);
}
```

Bibliography

[1] F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones, and D. Leask. Piconet – embedded mobile networking. *IEEE Personal Communications*, 4(5):8–15, October 1997.

[2] S. Capkun, L. Buttyan, and J. Hubaux. Self-organized public-key management for mobile ad hoc networks. In *Proc. ACM International Workshop on Wireless Security, WiSe 2002.*, 2002.

[3] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of manet simulators. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43. ACM Press, 2002.

[4] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks. In *Proc. IEEE SICON'97*, pages 197–211, April 1997.

[5] D. Comer and D. Stevens. *Internetworking with TCP/IP Volume 2.* Addison-Wesley, 1995.

[6] M. Conti, G. Turi, G. Maselli, J. Crowcroft, P. Michiardi S. Ostring, R. Molva, J. Costa Requena, I. Defilippis, S. Giordan, and A. Puiatti. Mobileman: Architecture, protocols and services, deliverable d5. IST 2001-38113, CNR, Cambridge University, Eurecom, HUT, SUPSI, September 2003. `http://cnd.iit.cnr.it/mobileMAN/pub-deliv.html`.

[7] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evauluation considerations. RFC 2501, IETF, January 1999.

[8] J. Crowcroft, R. Gibbens, F. Kelly, and S. Ostring. Modelling incentives for collaboration in mobile ad hoc networks. In *Proc. of WiOpt'03*, 2003.

[9] G. Fairhurst and L. Wood. Advice to link designers on link automatic repeat request (ARQ). RFC 3366, IETF, August 2002.

[10] Laura Marie Feeney. Ad hoc networking & IEEE 802.11. `http://www.sics.se/~lmfeeney/snus_802.11.pdf`.

[11] Hedrick. Routing information protocol. RFC 1058, IETF, January 2001.

[12] Y. Hu, D. Johnson, and A. Perrig. Sead: Secure efficient distance vector routing in mobile wireless ad hoc networks. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA '02), pages 3–13, June 2002.

[13] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[14] M. Kuhn and R. Anderson. Tamper resistance – a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1–11, Nov 1996.

[15] K. Dantu M. Maleki and M. Pedram. Power-aware source routing protocol for mobile ad hoc networks. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 72–75. ACM Press, 2002.

[16] L. Miller. Simulation model for the AODV MANET routing protocol. `http://w3.antd.nist.gov/wctg/manet/prd_aodvfiles.html`.

[17] C. Moss. The nodes revolution. *IEE Communications Engineer*, 2(1):18–21, 2004.

[18] S. Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.

[19] W. Odom. *Cisco CCNA, Exam #640-507 Certification Guide.* Cisco Press, 2001.

[20] P. Papadimitratos and Z.J. Haas. Secure routing for mobile ad hoc networks. In *Proc. of SCS Communication Networks and Distributed Systems Modeling and Simulation Conference 2002*, pages 27–31, 2002.

[21] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM (3)*, pages 1405–1413, 1997.

[22] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94*, pages 234–244, 1994.

[23] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2nd IEEE Wksp. Mobile Comp. Sys. and Apps.*, pages 90–100, February 1999.

[24] A. Perrig, R. Canetti, J. Tygar, and D. Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5(Summer), 2002.

[25] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, feb 2003.

[26] B. Preneel. Preimage resistance. `http://www.win.tue.nl/~henkvt/preimagev3.pdf`.

[27] K. Rayner. Mesh wireless networking. *IEE Communications Engineer*, 1(5):44–47, 2003.

[28] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, pages 46–55, April 1999.

[29] E. Schenk and D. Bernstein. SYN cookies – mailing list archive. `http://cr.yp.to/syncookies/archive`.

[30] S. Singh, M. Woo, and C.S. Raghavendra. Power aware routing in mobile ad hoc networks. In *Proc. 4th annual ACM/IEEE conference on Mobile computing and networking*, pages 181–190. ACM Press, 1998.

[31] I. Sommerville. *Software Engineering*. Addison-Wesley, 1995.

[32] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, 2002.

[33] R. Stefanova, G. Girling, J.L.K. Wa, and P. Osborn. The design and implementation of a low power ad hoc protocol stack. In *Proc. IEEE Wireless Communications and Networking Conference*, volume 3, pages 1521–1529, September 2000.

[34] C.K. Toh. Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks. *IEEE Communications Magazine*, 39(6):138–147, June 2001.

[35] G. Xylomenos, G. C. Polyzos, P. Mähonen, and M. Saaranen. TCP performance issues over wireless links. *IEEE Communications Magazine*, 39(4):52–58, 2001.

[36] M. Younis, M. Youssef, and K. Arisha. Energy-aware management for cluster-based sensor networks. *Computer Networks*, 43(5):649–668, 2003.

Proposer:	David Cottingham
User ID:	dnc25
College:	Churchill

CST Part II Project Proposal

Collaborative Power Management in Wireless Mesh Networks

October 2003

Project Originator:	D. N. Cottingham
Resources Required:	Please see the accompanying Project Resource Form
Project Supervisor:	Dr. J. K. Fawcett
Signature:	
Director of Studies:	Ms. C. H. Northeast
Signature:	
Overseers:	Prof. J. Crowcroft Dr. K. Moody

Introduction

In the last two years the prevalence of wireless networking technologies has increased dramatically. Previously infrared was the mainstream technology used to connect portable devices, with low bandwidths, and over a limited range: it is now commonplace to find an 802.11b “WiFi” transceiver in laptops and mobile telephones. Such explosive growth in wireless communication has found many applications where nodes are required to be mobile, or where fixed cabling is undesirable.

Many wireless devices use a paradigm best illustrated by the mobile telephone network: a mobile node connects to a fixed (i.e. wired) base station, which then routes all data to and from that node. There are several consequences of this:

1. The transmission range of mobile nodes is limited, and therefore base station deployment is required to be relatively dense (or, consequently, coverage for mobile entities is not guaranteed in many areas). This is both costly, and makes such devices useless in areas where the appropriate wired infrastructure is not in place.

2. Mobile node battery life is reduced due to transmission ranges being large (and therefore requiring more power).

3. For any mobile node in a particular area, there is likely to be a single point of failure in its connection to other nodes: the wired base station.

4. As a consequence of the relatively high power of transmission, the likelihood that a signal will be detected is increased – this is unfortunate in situations where a node’s location is to be kept secret.

Mesh networks aim to lessen the above by dynamically routing data between nodes, i.e. packets are sent via other nodes to eventually reach their destination, (be this a wired node or another wireless entity). The RFC for mobile ad hoc networks[5] provides an excellent description of the needs and issues surrounding their implementation, and mentions specifically energy resources as a major concern.

Several applications of wireless mesh networks are:

- Sensor networks – large manufacturing plants have hundreds of sensors that require a connection back to a central control room. Instead of wiring them in, they are implemented as a mesh network, providing far greater flexibility and lowered cost.

- Traffic networks – data can be routed between cars in a city to and from wired base stations.

- Wireless MANs – broadband access is made possible in remote areas by routing packets via other subscriber nodes. Whilst these are not mobile nodes, the system must deal with nodes being unavailable, and interference.

This project aims to design and implement (possibly entirely within a simulator) a protocol for mobile ad hoc networks that will conserve node resources by making routing choices that are dependent on the remaining energy of the nodes making up the alternative routes. This will provide resilience and high availability for communication between all nodes in the network.

[5]RFC 2501 – Corson, Macker, Batsell. IETF, January 2001

Work to be Undertaken

The implementation of collaborative power management in an ad hoc wireless network necessitates several core components, which will be required for the project to be deemed as a success. Other components that improve the security of the system or that enable it to run more efficiently will be extensions. These are outlined below.

Core Components

Each component listed has the OSI layer to which it corresponds approximately at the end of its description. Note that physical connectivity, framing, CRCs (Layer 1), will be provided by the C/C++ libraries being used. There is therefore no specific module to be written for this. This project will *not* attempt to investigate hardware solutions for increasing battery life.

1. *Transport Module:* defines packet format, sending and receiving methods, ARQ (Automated Repeat Request) system (Layer 2). This will allow two nodes to connect and send/receive data on a direct connection with some packet loss.
2. *Routing Module:* discovers, maintains, and propagates routes. Chooses optimum route for sending (Layer 3). This will allow multiple nodes to relay packets between each other on the optimum routes with diverse paths in existence.
3. *Energy-Aware Routing Module:* varies link “goodness” (equivalent to latency) with energy reserves remaining. Interacts with Routing Module to predict remaining energy on remote nodes, and provides a credit-based energy system, whereby each relayed transmission decreases the sender’s credit on the relaying node (Layer 3).
4. *Application Module:* This will take input from the console, output results, and interface with simulation software (Layer $\sim$5).

Extensions

The following are possible extensions to the project, should time allow.

- *Trusted Routing Information:* routing using the core modules will result in the security risk that bogus routes might be introduced, or genuine ones deleted by malicious nodes. To overcome this problem, a shared secret would be used by all trusted nodes in the network to obtain a one way hash of any routing information they transmit. On reception, if another node cannot obtain the same hash value, the routes received are ignored. Note that this does not prevent against internal attacks.
- *Bitmapped ARQ:* to conserve energy, a transmission control protocol is envisaged that would only send an ACK every, say, 5 packets of data received. A bitmapped ARQ system allows the receiver to tell the sender which of 5 packets have been received successfully.
- *Encryption:* wireless networks are a great deal more open to eavesdropping than wired links. Therefore it is necessary that data be encrypted between the source and destination, but using as little CPU time (i.e. energy) as possible.
- *Multicast Traffic:* the project is designed only for unicast traffic. It is likely that in some applications there would be several nodes that data should be sent to. To avoid sending out identical data multiple times, multicast groups could be established, and the data only transmitted once. The beauty of this is that no extra transmissions are needed: because transmissions are broadcast, all nearest neighbours (i.e. next hops) receive them, and therefore all potential subscribers to the group can be (indirectly) reached.

- *Link Contention/Collision Detection:* for the purposes of the core project it will be assumed that collisions will be detected indirectly by the ARQ protocol, and on retransmission collisions will not continuously occur. In a real application it would be necessary to implement randomised retransmission timers to prevent a second collision on retransmission.

Resources Required

Please see the associated project resource form for full details of the resource requirements for this project.

Starting Point

There has a substantial amount of work carried out on wireless mesh networks in many different contexts. Perhaps the most relevant work has been carried out at the Cambridge University Laboratory for Communications Engineering (LCE), and AT&T Labs: their Piconet (later PEN) project[6], concentrated on building a low power mesh network using a combination of hardware and low level network protocol features. Later work on a proactive routing protocol for PEN was also carried out[7], and some of the ideas from this work are to be re-implemented here. Other mesh technologies and routing algorithms such as the Temporally-Ordered Routing Algorithm[8], Destination-Sequenced Distance Vector protocol[9], and wired protocols such as the Routing Information Protocol[10] have also been considered and various ideas will be used from them.

To the best of my knowledge there has been no published work on routing algorithms that attempt to take into account the resources (and in this case, specifically energy levels remaining) of nodes in the network.

Security in wireless mesh networks is also a well researched topic. Stajano's text on the implications of ubiquitous computing[11] points out various security issues, some of which could be solved by the project and its extensions (in particular the problem of DoS attacks resulting in "sleep deprivation" and thereby battery exhaustion). Another useful overview by Wang, Lu, and Bhargava[12] will be the basis for several design decisions for the protocol.

My experience with C/C++ programming is minimal, and therefore initially time will need to be spent sharpening my network programming skills in this area. In addition, I do not have any experience with network simulation software (in this case I hope to use OPNET), which will again warrant a somewhat steep learning curve. I have a fairly wide experience of networking technologies.

Measures of Success

The project will be deemed to have been a success when all items described above as core components have been implemented and tested successfully, along with any extensions that time allows.

[6]Piconet - Embedded Mobile Networking – Bennett, Clarke, Evans, Hopper, Jones, Leask. IEEE Personal Communications, Vol. 4, No. 5, October 1997, pp 8-15.

[7]The Design and Implementation of a Low Power Ad Hoc Protocol Stack – Li Kam Wa, Osborn, Stefanova. Presented at IEEE Wireless Communications and Networking Conference, September 2000.

[8]A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks – Park, Corson. Published in the proceedings of INFOCOM'97.

[9]Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers – Perkins, Bhagwatt. ACM SIGCOMM October 1994.

[10]RFC 1058 – Hedrick. IETF, January 2001

[11]Security for Ubiquitous Computing – Stajano. Wiley 2002.

[12]On Security Study of Two Distance Vector Routing Protocols for Mobile Ad Hoc Networks – Wang, Lu, Bhargava. CERIAS, 2003.

Network simulations will be performed to compare the spread of energy levels in an ad hoc network after a specified running time with “standard” routing, and with energy aware routing. Other measurements may be performed as time allows.

Project Plan

The work for this project will be broken down into 2 week work “packets”, each of which will have an identifiable deliverable associated with it.

Noteworthy dates:

- Progress Report due Friday 30^{th} of January.
- Final Dissertation due Friday 14^{th} of May.

#	Start	End	Description	Deliverable(s)
1	18/10/2003	31/10/2003	**Investigation (80%) & Design (20%)** *Investigation:* Research C/C++ network programming, understand possibilities of network simulation software and how best to code to make use of it. *Design:* Specify protocol packet format, begin specification of transport protocol.	Packet format specification, draft transport protocol specification.
2	01/11/2003	14/11/2003	**Design (80%) & Test Plan (20%)** *Design:* Complete specification of routing prot. specify energy aware routing prot. & Application module. *Test Plan:* Devise high level module test plan.	Specifications for Routing and Energy Routing complete. High level test plan.
3	15/11/2003	28/11/2003	**Implementation I** Implement enough of the system to enable direct communication between two hosts running the Transport and Application modules. ARQ.	Hosts to be able to carry out bi-directional communication using the basic Transport protocol, with ARQ.
4	29/11/2003	12/12/2003	**Implementation II** Implement Routing module, initially with static routes (i.e. no updates), then with dynamic distance vector routing. Begin Energy Routing module.	Multiple hosts able to be on network. Dynamic routing with updates to work. Energy Routing module begun.
5	13/12/2003	23/12/2003	**Implementation III** Finish Energy Aware Routing. Interface with simulation software. Fix problems. *(Work over Christmas)*	All routing should now work. Hosts should be able to communicate despite network partitions and packet loss.
6	10/01/2004	23/01/2004	**Progress Report, Testing** Write progress report. Construct simulation models and carry out simulations to obtain data.	Progress report deadline: 30/01/2004. Initial simulations complete.
7	24/01/2004	06/02/2004	**Testing** Continue simulations.	None.
8	07/02/2004	20/02/2004	**Testing/Dissertation Writing I** Complete simulations, interpret data. Begin dissertation.	Simulation conclusions.
9	21/02/2004	05/03/2004	**Dissertation Writing II** Continue dissertation.	None.
10	06/03/2004	19/03/2004	**Dissertation Writing III** Complete first draft.	First draft complete.
11	20/03/2004	02/04/2004	**Dissertation Writing IV** Correct draft, finalise report.	Dissertation complete.
12	03/04/2004	16/04/2004	**Contingency I** Just in case...	N/A.
13	17/04/2004	30/04/2004	**Contingency II**	N/A.

Contingency Plan

If, once simulation software has been investigated, it is decided that none of the third party packages is a viable option, a simple custom simulator will be developed, with test cases developed by hand involving a limited number of nodes in the network. This should be sufficient to show that the main body of code functions correctly.