Routing in Mobile ad hoc networks

Srinath Perur

20th February 2001

Abstract

Mobile ad hoc networks are infrastructure-less, wireless networks of mobile nodes that are formed dynamically without much set up time or cost. They are used where communications infrastructure is unavailable or infeasible to use, as in battlefield or disaster relief operations. Routing in such ad-hoc networks is an issue since node mobility makes the network topology highly dynamic. This thesis surveys some of the work done in the area of ad hoc routing and proposes Kelpi, a new routing algorithm. Kelpi is based on the observation that with most routing algorithms, an intermediate node's movement could cause routes between distant nodes to be affected. Kelpi is designed to maintain stable routes that form a dynamic virtual backbone in an ad hoc network. The principle behind this is to keep routing information confined to a geographical area despite nodes being mobile. This is done by imposing a cellular structure on the ad hoc network using an accurate positioning service at every node. Kelpi also aims to increase link-layer throughput by using multiple levels of transmission power to reduce radio interference.

An implementation of Kelpi for the Network Simulator ns-2 is described.

Contents

Chapter 1

Introduction

1.1 Context

Since the advent of computing, advances in technology have constantly been making computers smaller, cheaper and more powerful than before. This has had an impact on the way we use computers. Mobile computing has gained increasing popularity in the last decade or so due to the availaibility of small, economical and portable computers. Minituarization has allowed devices to have computers embedded within them to automate their control. These trends seem to indicate that we are moving towards a scenario where we will have a large number of mobile, wirelessly networked computers in use around us.

The traditional way of enabling wireless networking has been to use a cellular infrastructure. The mobile user registers with a service provider who maintains a number of base stations over a fixed area of operation. Each of these base stations handles communications with mobile devices in its 'cell', that is, the region over which it can send or receive a radio transmission. These base stations in turn are connected with a fixed network, enabling communication between devices in different cells. While this constitutes a fairly reliable means af providing networking to mobile hosts, it has its drawbacks in requiring large expenditure on infrastructure. Further, this kind of service is restricted to areas where the supporting infrastructure exists: where it is viable, physically and economically, to establish the required infrastructure.

In the emerging scenario mentioned above, it may not always be desirable to depend on such a cellular infrastructure. Situations can arise where it would be useful to network mobile computers, but there is no existing infrastructure. Typical examples include battlefield or disaster relief operations where such support either does not exist or has been destroyed. Other examples include a meeting of people with mobile hosts, in which case, the people involved may not want to use a cellular system because the cost involved in setting up and using such a facility might outweigh the benefits of using the network for their specific task.

An alternate approach to network mobile computers, without establishing a communications infrastructure, is the mobile ad hoc network.

1.2 Mobile ad hoc networks

An ad hoc network is the co-operative engagement of a collection of mobile hosts without the required intervention of any centralized access point [1].

Mobile Ad hoc NETworks (MANETs) are infrastructure-less wireless networks where nodes are capable of moving. They are formed arbitrarily and dynamically without much set up time or cost. Generally, some or all nodes of a MANET function as routers and communication between two hosts is done by multi-hop routing through the nodes of the network. This is required because nodes which want to communicate may not be within direct radio range of each other. The nodes in such a network work together to discover and maintain routes between hosts in the network. Some characteristics of MANETs are:

- Dynamic topology

  Direct links between nodes can be broken and reformed rapidly due to mobility. As a result, the topology of the network changes very frequently in comparision with wired networks, where topology change is generally due to occasional link failure or link re-establishment.

- Constrained power

  Often, the nodes in a MANET are battery operated devices and need to conserve power in order to remain operational for as long as possible.

- Uni-directional links

  Due to a difference in radio transmission powers, a node may be able to receive from a particular node, but not send to it.

1.2.1 Routing

Routing in a network deals with the task of finding a path through the network between a given pair of nodes. A source wanting to send a packet to a particular destination sends the packet to a neighbouring node with a route to the destination. This node in turn sends the packet to the next-hop on the route, and so on, till the destination is reached.

Routing in wired networks

Several routing protocols exist for wired networks. Almost all can be classified as using either the *distance vector* or the *link-state* algorithms. In distance vector routing [2], each router periodically sends its view of its distance from every other node in the network to its neighbours. Based on this information, each router calculates its next-hop neighbours along the shortest path to every node. In link-state routing [2], each router periodically sends its view of the status of its adjacent network links to *all* the routers in the network. Each router can then take forwarding decisions based on a complete picture of the network obtained by combining the latest updates from all the routers.

Routing in MANETs

The algorithms described above were designed for use in static wired networks where topology changes are infrequent and all links are bi-directional. They are also computation intensive, making them difficult to use with constrained resources. Due to these problems, new routing algorithms are required that take into account the characteristics of MANETs. The issue of routing in MANETs deals with finding paths between nodes that are part of a rapidly changing topology with possibly uni-directional links, while using minimum resources.

1.3 Scope of this work

The work presented in this thesis deals with the issue of routing in MANETs. A survey of existing routing algorithms for MANETs showed that most of them had the drawback of allowing a node's mobility to affect routes between other nodes using that node in the route between them. This called for an algorithm that provided relatively stable routes. A new routing algorithm for MANETs, Kelpi, is presented and its implementation using the ns-2 network simulator described. Kelpi aims at maintaining long-lived, stable routes in a MANET.

1.4 Thesis organization

Chapter 2 describes a survey of existing work relating to routing in MANETs. Chapter 3 outlines the algorithm Kelpi; chapter 4 offers a detailed description. Chapter 5 introduces the Network Simulator ns-2 and describes the implementation of Kelpi for ns. Chapter 6 contains conclusions and directions for future work.

Chapter 2

Related Work

A number of algorithms have been devised for routing in MANETs. Most of these can be classified as using one of two approaches: *reactive* or *pro-active*. The following sections classify several existing algorithms according to the approach used in them and outline the basic mechanism used in these algorithms. Ad hoc routing algorithms that use location services are of special interest in light of the work presented in this thesis; the final section presents some of the work published in this area.

2.1 Reactive algorithms

The reactive approach (or on-demand approach) to routing in MANETs is characterized by routes being created only when required by some node. When a node requires a route to a destination, it initiates a route discovery process within the network. Once a route is established, it is maintained by some form of route maintenance till the route is either invalid or unnecessary [3].

2.1.1 Dynamic Source Routing (DSR)

Dynamic Source Routing (DSR) [4] is designed to allow nodes to dynamically discover a source route across multiple network hops to any destination in the ad hoc network. When using source routing, each packet to be routed carries in its header the complete, ordered list of nodes through which the packet must pass. An advantage of source routing is that intermediate hops do not need to maintain routing information in order to route the packets they receive, since the packets already contain the necessary routing information. An example of a packet moving through an ad hoc network with source routing is illustrated in Figure 2.1.

DSR does not require the periodic transmission of router advertisements or link status packets, reducing messaging overhead when mobility is low. DSR has also been designed to compute correct routes in the presence of uni-directional links.

Protocol Overview

DSR consists of three functional components: *routing*, *route discovery* and *route maintenance*. Routing uses source routing as described above. Route discovery is the mechanism by which a node

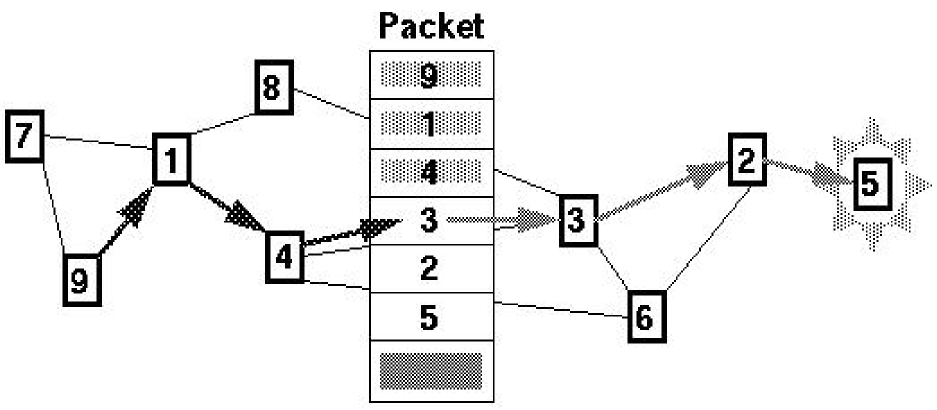


Figure 2.1: A packet being source routed from node 9 to node 5.

wishing to send a packet to a destination obtains a path to the destination. Route maintenance is the mechanism by which a node detects a break in its source route and obtains a corrected route.

Route Discovery

The source node broadcasts a route request packet with a recorded source route listing only itself. Each node that hears the route request (RREQ) forwards the request to its neighbours, adding its own address to the recorded source route in the packet. The RREQ propagates hop-by-hop outward from the source node until either the destination node is found or until another node is found that can supply a route to the destination. Nodes forward RREQs if they are not the destination node and they are not already listed as a hop in the route. In addition, each node maintains a cache of recently received RREQs and does not propagate any copies. All source routes learned by a node are kept (memory permitting) in a route cache, which is used to further reduce the cost of route discovery. A node may learn of routes from virtually any packet the node forwards or overhears. When a node wishes to send a packet, it examines its own route cache and performs route discovery only if no suitable source route is found. Further, when a node receives a route request for which it has a route in its cache, it does not propagate the route request, but instead returns a route reply (RREP) to the source node. The RREP contains the full concatenation of the recorded route from the source, and the cached route leading to the destination. If a RREQ packet reaches the destination node, it returns a RREP packet to the source node with the full source to destination path listed.

Route Maintenance

If a node along the path of a packet detects an error, the node returns a route error packet to the sender. The route error packet contains the addresses of the nodes at both ends of the hop in error. When a route error packet is received or overheard, the hop in error is removed from any route caches and all routes which contain this hop must be truncated at that point. There are many methods of returning a route error packet to the sender. The easiest of these, applicable in networks that use only bidirectional links, is to simply reverse the route contained in the packet from the original host.

The above description of DSR is a modified extract from [5].

2.1.2 Ad-hoc On-Demand Distance Vector Routing

This builds on the DSDV algorithm described in section 2.2.1. It tries to improve DSDV by minimizing the number of broadcasts by creating routes on an on-demand basis. Route discovery is initiated by the sending of a Route Request (RREQ) packet to its neighbours, which in turn forward the packet to their neighbours till a fresh enough route is found. Destination sequence numbers are used to keep routes loop free and fresh. On reaching a node with a fresh route, the node sends a route reply (RREP) packet. Route maintenance is similar to the policy followed in DSR with link failure notification packets. This protocol is described in [6].

2.1.3 Temporally Ordered Routing Algorithm (TORA)

This algorithm is based on the concept of link reversal. TORA is designed to work in a highly dynamic MANET. It also provides multiple routes to any desired destination. This protocol has three phases. In the first and second, route creation and maintenance, nodes use a height metric to establish a Directed Acyclic Graph (DAG). Thereafter, links are assigned a direction (upstream or downstream) based on the relative height of neighbouring nodes. When a link is broken, a node generates a new height which propagates and causes the direction of links to change. The third phase is a erase phase where invalid routes are erased. The TORA protocol is described in [7].

2.1.4 Associativity Based Routing (ABR)

This defines an interesting metric for routes in MANETS, *degree of association stability.* In ABR, a route is selected based on the degree of association stability of nodes. Each node generates regular beacons on which neighbouring nodes update a count. The count increases proportional to the association stability between nodes. A high value of this metric generally indicates low mobility. The fundamental objective of ABR is to derive longer-lived routes in a MANET. This protocol is described in [8].

2.1.5 Signal Stability Routing (SSR)

This is similar to ABR, but uses the metric of *signal stability* between nodes. This algorithm, therefore, tends to generate routes which have a strong signal strength. More about this protocol can be found in [9].

2.2 Pro-active algorithms

Pro-active algorithms try to maintain consistent, up-to-date routing information for all nodes in the network. They require each node to maintain tables of routing information, and respond to changes in network topology by propagating updates throughout the network in order to maintain a consistent network view [3].

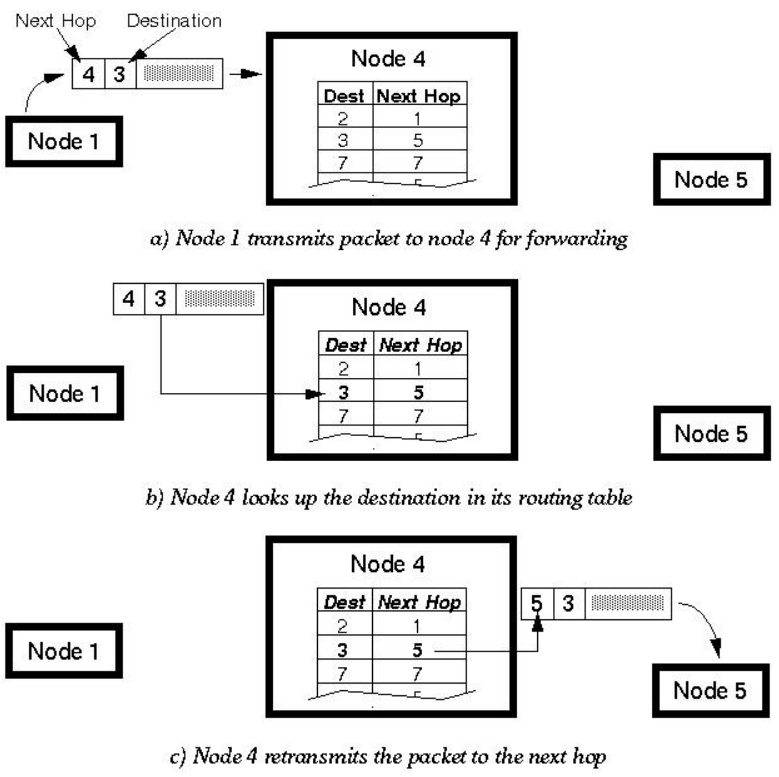

Next Hop
Destination
4 3
Node 1
Node 4
Dest | Next Hop
2 | 1
3 | 5
7 | 7
Node 5
a) Node 1 transmits packet to node 4 for forwarding
b) Node 4 looks up the destination in its routing table
5 3
c) Node 4 retransmits the packet to the next hop

Figure 2.2: Routing in DSDV

2.2.1 Destination Sequenced Distance Vector Routing (DSDV)

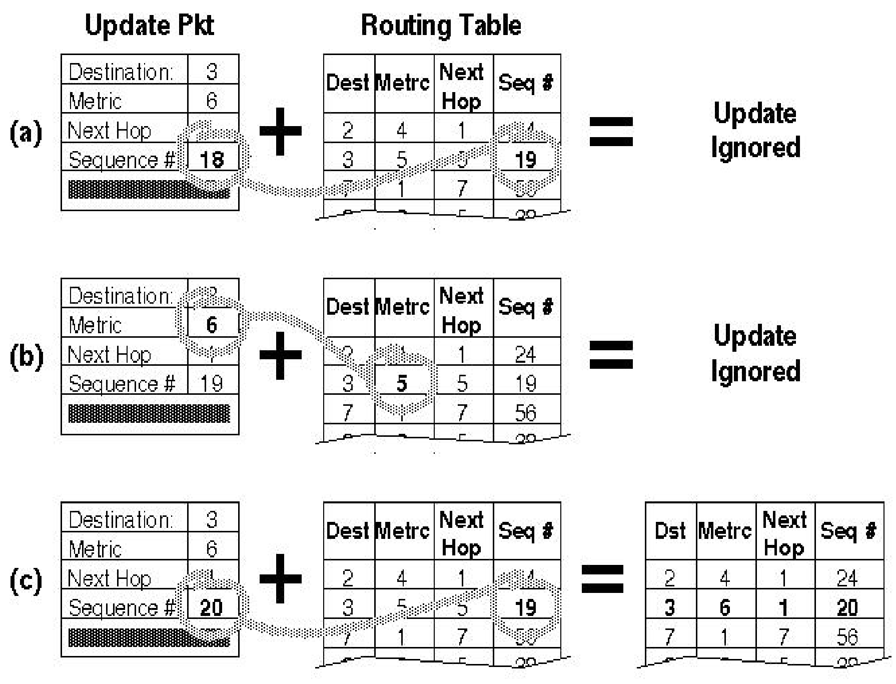
DSDV is based on the distance vector protocol commonly used in wired networks. It makes some modifications to adapt it for routing in MANETs.

Protocol Overview

In DSDV, packets are routed between nodes of an ad hoc network using routing tables stored at each node. Each node contains a routing table with an entry for every node in the network, containing the next hop node to reach that node. Figure 2.2 illustrates the routing procedure in DSDV. In this example, a packet is being sent from node 1 to node 3 (node 3 is not shown). From node 1, the next hop for the packet is node 4 (Figure 2.2 a). When node 4 receives the packet, it looks up the destination address (node 3) in its routing table (Figure 2.2 b). Node 4 then transmits the packet to the next hop as specified in the table, in this case node 5 (Figure 2.2 c). This procedure is repeated as required until the packet finally reaches its destination.

Routing Table Management

While routing is a relatively trivial task, the maintainance of up-to-date routing tables is not. Every time the network topology changes, the routing table in every node needs to be updated. Also, when routing tables are out of sync (i.e. the routing protocol has not converged), routing loops may form. To facilitate routing table maintenance, several additional pieces of information



Update Pkt

Routing Table

(a)

Destination:	3
Metric	6
Next Hop	1
Sequence #	**18**

+

Dest	Metrc	Next Hop	Seq #
2	4	1	24
3	5	5	**19**
7	1	7	56

= Update Ignored

(b)

Destination:	3
Metric	**6**
Next Hop	1
Sequence #	19

+

Dest	Metrc	Next Hop	Seq #
2	4	1	24
3	**5**	5	19
7	1	7	56

= Update Ignored

(c)

Destination:	3
Metric	6
Next Hop	1
Sequence #	**20**

+

Dest	Metrc	Next Hop	Seq #
2	4	1	24
3	5	5	**19**
7	1	7	56

=

Dst	Metrc	Next Hop	Seq #
2	4	1	24
3	**6**	**1**	**20**
7	1	7	56

Table 2.1: Updation of routing tables in DSDV

are stored in the routing tables. In addition to the destination address and next hop address, routing tables maintain the route metric and the route sequence number.

Periodically, or immediately when network topology changes are detected, each node will broadcast a routing table update packet. The update packet starts out with a metric of one. This signifies to each receiving neighbor they are one hop away from the node. The neighbors will increment this metric (in this case, to two) and then retransmit the update packet. This process repeats itself until every node in the network has received a copy of the update packet with a corresponding metric. If a node receives duplicate update packets, the node will only pay attention to the update packet with the smallest metric and ignore the rest. To distinguish stale update packets from valid ones, each update packet is tagged by the original node with a sequence number. The sequence number is a monotonically increasing number which uniquely identifies each update packet from a given node. Consequently, if a node receives an update packet from another node, the sequence number must be equal to or greater than the sequence number already in the routing table; otherwise the update packet is stale and ignored. If the sequence number matches the sequence number in the routing table, then the metric is compared and updated as previously discussed. Each time an update packet is forwarded, the packet not only contains the address of the eventual destination, but it also contains the address of the transmitting node. The address of the transmitting node is entered into the routing table as the next hop. Table 2.1 illustrates how a node processes an update packet under varying conditions. Note that update packets with higher sequence numbers are always entered into the routing table, regardless of whether they have a higher metric or not.

Responding to Topology Changes

Mobile nodes cause broken links as they move. The broken link may be detected by the communication hardware, or it may be inferred if no broadcasts have been received for a while from a former neighbor. A broken link is described as having a metric of infinity. Since this qualifies as a substantial route change, the detecting node will broadcast an update message for the lost destination. This update message will have a new sequence number and a metric of infinity. This will essentially cause the routing table entries for the lost node to be flushed from the network. Routes to the lost node will be established again when the lost node sends out its next broadcast. To avoid nodes and their neighbors generating conflicting sequence numbers when the topology changes, nodes only generate even sequence numbers for themselves, and neighbors responding to link changes only generate odd sequence numbers.

The above description of DSDV is a modified extract from [5].

2.2.2 Clusterhead Gateway Switch Routing (CGSR)

This is based on the DSDV algorithm. It is hierarchical in nature in an attempt to improve scalability. It is different in that it divides the nodes in the network into clusters and each cluster elects a node as a clusterhead. A node that is in range of two clusterheads is called a gateway. Routing takes place from clusterhead to gateway to clusterhead till the message reaches the cluster of the destination. Each node maintains a table of all nodes in the network and their cluster heads. These clusterhead tables are propagated regularly through the network as in DSDV. This is described in [10].

2.3 Hybrid Routing

2.3.1 Pro-active vs. Reactive routing

Pro-active schemes provide valid routes instantly. But, they result in more messaging overhead and computation for route maintenance. They are useful in real-time applications. In the case of reactive protocols, power and bandwidth are saved by preventing constant maintenance of routes, but there is a latency involved in finding a route. Pro-active protocols can prove even more expensive when the degree of mobility involved is high. There also exists a hybrid combination of the two in the Zone Routing Protocol.

2.3.2 Zone Routing Protocol

This routing algorithm tries to combine the pro-active and reactive approaches. For every node, a zone is maintained that contains all nodes within *zone radius* hops from the node. Two routing protocols are employed, the IntrAzone Routing Protocol (IARP) and the IntErzone Routing Protocol (IERP). The IARP is pro- active and the IERP is reactive. The zone radius can be varied according to the needs of the application. More details about this protocol can be found in [11].

2.4 Location based algorithms

2.4.1 Location Aided Routing (LAR)

LAR makes the use of geographic location of nodes in a MANET by using a Global Positioning System (GPS) receiver with each node. Based on this every node defines a request zone and an expected zone where the destination is to be found. When a route request packet is sent, only those nodes which are in the expected zone forward the packet thus minimizing the area in which flooding is to be done. This brings about a saving in the number of messages required for finding a route. If a route is not found within an expectation zone, the zone can be widened. LAR uses GPS for optimizing routing, but is not dependent on it. LAR is presented in [12].

2.5 Observations

The work presented in the following chapters deals with a routing algorithm designed to provide long-lived routes. In this respect, most of the algorithms above tend to have some shortcomings. In the case of reactive algorithms like DSR and AODV, the movement of a node can have the following effect:

1. routes to and from the node in question can be broken; and
2. routes that use this node as an intermediate hop can be broken.

In the case of pro-active algorithms like DSDV, the following undesirable effects can be observed:

1. large overheads are incurred in broadcasting periodic routing advertisements; this occurs when nodes are stationary too;
2. the size of these advertisements increase with the size of the network, leading to poor scalability.

The problem of scalability is generally handled by using a hierarchical scheme as in CGSR. Since CGSR is based on DSDV, the problem of large overheads in periodic advertisements is present here too. Further, there is more overhead involved in electing a clusterhead and managing the hierarchy.

The next chapter presents an overview of a routing algorithm, Kelpi, that aims to address these problems.

Chapter 3

Kelpi: An overview

Kelpi is a routing algorithm designed to provide long lived stable routes when possible and to reduce routing overheads; it is also designed with the aim of improving throughput at the link layer by reducing interference between transmissions of different nodes.

3.1 Preliminaries

The choice of a routing algorithm to deploy in a MANET depends on the characteristics of the nodes and their application: the routing needs of a MANET that is being used in a conference room are quite different from that being used in a battlefield. Some of these characteristics are area of operation, rate of mobility, density of nodes, transmission range of nodes, type of traffic, whether there exists some fixed infrastructure, power constraints of nodes, etc. Loosely, Kelpi is intended for operation in a MANET spread over a fairly large area, say a few kilometres, with nodes moving at or less than vehicular speeds, with power not being a major constraint and with a node density of around tens of nodes per square kilometre. These seem to characterize an application in the battlefield or in a disaster relief operation. The exact characteristics of the application that Kelpi is best suited for will emerge only after detailed experimentation.

3.1.1 Principle of operation

The dynamic virtual backbone

Kelpi tries to ensure stable routes by imposing a cellular structure on an ad hoc network. The area of operation is geographically divided into cells. Each node is capable of computing a 'map' of these cells using its initialization parameters set before deployment. Each node is also capable of detecting which cell it is currently within by using a positioning system such as the *Global Positioning System* (*GPS*). At any given time, an occupied cell has a router through which all other nodes in the cell route their communication; the routers communicate among themselves. The existance of a router in a cell is independent of any particular node; that is, any node can function as a router. A router that is moving out of a cell can appoint a node in that cell as a router after transferring the necessary information to it. This ensures that even if a set of nodes are steadily moving across a cell, the routes going through that cell are unaffected. The routers of cells form a '*virtual backbone*' which is expected not to change very frequently. Routing information for this virtual backbone is maintained by pro-active periodic broadcasting of route update packets.

3.1.2 Assumptions

The following assumptions are made about the nodes to be used in the network:

- each node is equipped with an accurate positioning device such as a GPS receiver; and
- each node is capable of *multi-level transmission range.*

Further, it is assumed that the bounding rectangle of the area of operation is known before the network is deployed. While multi-level transmission is preferred, the algorithm is capable of functioning with single power transmitters. The assumptions are not unreasonable given the typical usage scenarios.

3.2 Operation

All nodes first undergo initialization; then nodes communicate with their routers, and routers communicate among themselves. When a node communicates with another node or its cell router, it operates at a lower transmission power which is enough for its transmission to be received at any point within the cell. When a router transmits to another router, it changes its power of transmission to enable it to be received at any point in any of its neighbouring cells.

3.2.1 Initialization

Before the MANET is deployed, it is initialized with the following details:

- the bounding rectangle of the area of operation; and
- a value of transmission power at which all nodes can transmit, typically the maximum value if the nodes are similar.

In figure 3.1, the area of operation is denoted by a shaded oval and the co-ordinates $< x_0, y_0 >$and $< x_l, y_l >$denote the bounding rectangle. These co-ordinates would depend on the kind of positioning system being used. If GPS is being used they are assumed to be the absolute two-dimensional geographical co-ordinates of the points. Using these initialization parameters, each cell can construct the grid shown in figure 3.1; this process is described in detail in Chapter 4. Further, each node can determine, using its current GPS co-ordinates, its current cell.

3.2.2 Node-Router communication

Each cell has a node that performs the role of a router. This node responds to a *Cell Router Address (CRA)* that is a function of the cell for which that node is a router, in addition to its own network address. This is to keep the changing of cell routers transparent to other nodes in the network.

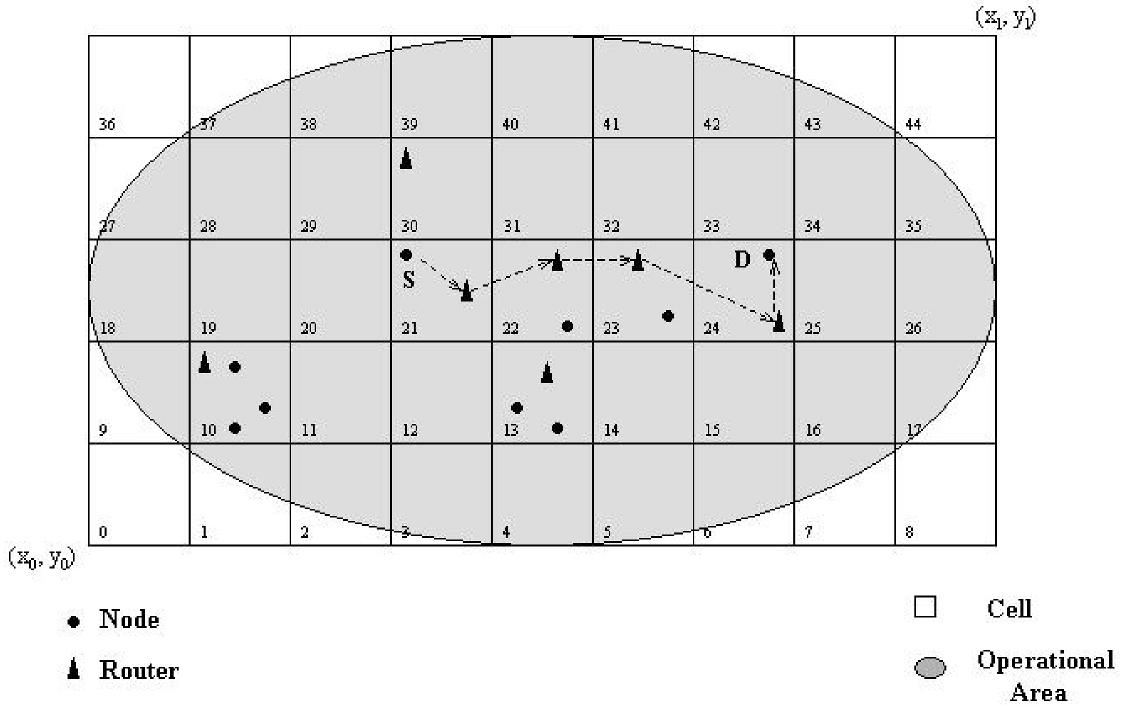


Figure 3.1: Routing in Kelpi

A node on being switched on, or on entering a new cell, sends a HI (Here I am) message to the router using the CRA for that cell. If a router does not acknowlege the HI, the node declares itself the router of that cell and begins responding to the CRA. When a node S wants to send a packet to a node D, S sends its router a FIND_CELL message containing D's address. The router finds out which cell D is currently in and sends a reply to S with D's cell. S can then send a packet addressed by D's address and cell to the router which knows the next hop cell to reach D's cell.

A node also sends a BYE message to its cell's router while leaving a cell which contains the new cell of this node. A router, before leaving a cell, broadcasts a RTR_MOVE message to all the nodes in the cell. This initiates a sequence of messages that leads to a particular node being choosen as the new router after existing routing information has been transferred to this node.

3.2.3 Router-Router communication

Routers, among themselves, exchange periodic routing updates, forward packets to other routers and propagate FIND_CELL messages. Each of these is explained below in some detail.

Periodic Routing Updates

Each router needs to know the next hop cell for a packet's destination cell. This information would be static if all the cells were occupied, guaranteeing a router in every cell and reducing the network to a static cellular network. However, it is quite likely that cell occupancy changes; therefore, periodic distance vector updates are essential to maintain up-to-date forwarding information. This also ensures that partitioned sub-networks in a MANET can combine smoothly.

Periodic updates are essential only when a cell becomes empty or when a router comes up in an empty cell. Due to the router handoff mechanism mentioned earlier, this is not expected to be very frequent. A node broadcasts a destination sequenced update (see section 2.2.1) containing its distance in hops from all other occupied cells, to all its neighbouring cells.

Forwarding packets

Each router maintains a routing table that stores the next hop cell for a given destination cell. This table is updated when any changes in the network are noticed through the periodic distance vector updates from the neighbouring cells.

When a packet arrives at a router, the router checks if the packet is destined for its current cell. If not, the router forwards it to a neighbouring cell determined by the routing table.

Flooding FIND_CELL messages

When a node sends a FIND_CELL message to a router, the router floods the neighbouring routers with this packet. A router that has the destination node in its cell, replies directly to the node that initiated the message. Each FIND_CELL has a sequence number; a router will not forward any such message with a sequence number lower than one it has already seen for that source-destination pair.

3.2.4 Node-Node communication

If a node learns from a reply to its FIND_CELL message that the destination node is in the same cell as itself, it initiates communication directly with the node. Also, when a node in active communication with some other nodes leaves its current cell, it sends these nodes a CHANGED_CELL message containing its new cell to enable communication to continue uninterrupted.

3.2.5 An example of routing using Kelpi

Figure 3.1 shows an example of routing using Kelpi. A node S in cell 21 wants to communicate with node D. The sequence of events that occur are as follows:

1. S sends a FIND_CELL message containing D to the router in its cell. Although S does not know the address of the node acting as the router, it knows that the router is listening on a cell router address (CRA) that is fixed for the cell and sends the message to that address.

2. The router for cell 21 receives the FIND_CELL message. It first looks for D in a list of nodes that are present in the cell, and not finding it there, broadcasts the message.

3. Routers of cells neighbouring 21, that is, 30, 22 and 13, receive the message, check if D is present in their cell, and broadcast the message. A router never repeats the broadcast of a FIND_CELL since it discards the message if it has seen the sequence number of the message before.

4. The router in cell 24, on receiving the message, finds that D is present in its cell. It sends a FIND_CELL_REPLY to node S containing D's cell, 24.
5. On receiving D's cell, S sends its data packet addressed to D, to the router of cell 21. The packet also contains the destination cell number, 24.
6. The router of cell 21 on receiving this packet, looks into its routing table and finds that the next hop cell for cell 24 is cell 22. The packet is therefore forwarded to the CRA of cell 22. The packet is similarly forwarded till it reaches the router of cell 24.
7. The router of cell 24 on finding that the packet is addressed to its current cell, transmits the packet to node D. The path followed is indicated in figure 3.1 with dotted arrows.
8. Assume that subsequently, the router in cell 22 approaches the cell boundary adjoining 31 and sends a RTR_MOVE broadcast.
9. The only node in cell 22 on receiving this message replies with a RTR_MOVE_ACK containing its current co-ordinates.
10. The router waits for some time for responses and times out. It appoints the node closest to the centre as the new router; in this case, by default, it is the solitary node.
11. The router sends the node a RTR_HANDOFF message containing all the router data structures.
12. The node on receiving the RTR_HANDOFF becomes the new router and starts responding to the CRA for cell 22. The old router now functions as a node.
13. This change of router has no impact on the route already passing through cell 22. The router of cell 21 is still sending packets to cell 24, using the CRA of cell 22 as a next hop address. Several nodes could come and go in any of the cells, but as long as there is at least one node in a cell at any given time, routes are unaffected. This approaches the concept of a backbone in cellular networks and is therefore called a *virtual backbone*.
14. Now let the node in cell 22, the ex-router, move to cell 31. It sends a BYE message to the router in cell 22 and a HI message to the router of cell 31.
15. Since there is no router in cell 31, no HI_ACK is received by this node. It times out and declares itself the router.
16. It is possible that another node could have similarly entered cell 31 and sent a HI. One node would set its timer as the other one is waiting for its own timer to expire. This situation would result in two routers in the same cell. To prevent a node on timing out sends a RH (Router Here) message. Any other nodes with their timers set would cancel their timers and send a HI to the new router, the sender of the RH.

3.3 Comments

The approach taken by Kelpi has some trade-offs; some of the advantages and disadvantages are listed below.

3.3.1 Advantages

- Results in formation of more stable routes since using the cell router address (CRA) and the router hand-off mechanism shield the rest of the network from the effects of node mobility.
- Reduces link layer interference since all nodes do not transmit at their maximum power. This is expected to result in increased network throughput.
- Since direct communication between two nodes is always within the same cell, multiple broadcast channels, as in a cellular system, can be used.

3.3.2 Disadvantages

- The hierarchical nature of the algorithm results in routers being loaded much more heavily than the other nodes.
- Kelpi requires an accurate positioning system for its functioning. Although this is not an unreasonable assumption to make given the target applications, it would be preferable for an algorithm to function correctly, if not optimally, in the case of unavailability or failure of the positioning system.

This chapter presented an overview of the functioning of Kelpi. Several details have not been mentioned here; the next chapter explains the algorithm in greater detail, including descriptions of the data structures involved, contents of routing messages and exact behaviour at various events that occur during the lifetime of a node or router.

Chapter 4

Kelpi: Details

This chapter provides a detailed description of the algorithm Kelpi. The organization of this chapter is as follows: the data structures maintained at every node are described; then, events occurring in the lifetime of a node are listed; messages that occur with different events are mentioned along with the events. Design decisions are commented upon where required.

4.1 Data Structures

Since every node is a potential router, no distinction is made between the data structures maintained at a node and router. Some of them, however, are used only when a node functions as a router.

Each node maintains the following values: *current-cell*, *previous-cell*, a *is_router* flag that is set if the node is a router, the bounding co-ordinates of the area of operation, present co-ordinates and last known co-ordinates. Since the nodes can transmit at two different powers, they store these powers in *maxTxPower* and *nodeTxPower*. Other data structures required for operation are:

node-list: Each router maintains a *node-list* which contains the addresses of nodes that are in its cell.

node-cache: Each node maintains a cache mapping nodes it has communicated with to their cells. This is to prevent repeated FIND_CELL messages for the same destination.

forwarding-pointers: Each router keeps track of which cell a node has moved to after leaving. This is used to forward packets that arrive for the node just after it has left the cell. Entries for nodes are removed after a timer times out.

routing-queue: Each node maintains a *routing-queue* where packets are buffered while the destination cell is being found. Although this function is typically performed by routers in wired networks, nodes perform the function here to reduce router load.

routing-table: Contains the next-hop forwarding cell for every possible destination cell. Each entry contains a destination node address, its cell and the destination sequence number.

In addition to the above mentioned, each router also keeps track of the last sequence number seen in the FIND_CELL for every source-destination pair.

4.2 Messages

The routing messages used in Kelpi, their contents, and the events corresponding to their generation are as follows:

- HI - $<$*msg_type, current_cell, previous_cell*$>$: sent by a node when it enters a new cell.
- RH - $<$*msg_type, cell*$>$: sent by a node that declares itself a router.
- HIACK - $<$*msg_type, cell*$>$: sent by a router in acknowledgement of a HI message.
- BYE - $<$*msg_type, current_cell, previous_cell, sender_node_address*$>$: sent by a node after it leaves a cell.
- FIND_CELL - $<$*msg_type, dst_address, src_address, src_cell, sequence_number*$>$: sent by a node that wants to send a packet, but does not have a valid destination cell in its node-cache.
- RTR_MOVE - $<$*msg_type, cell*$>$: broadcast by a router as it approaches a cell boundary.
- RTR_HANDOFF - $<$*msg_type, cell, node-list, routing-table, seq_numbers*$>$: sent by a router after having appointed a new router for that cell.
- RTR_MOVE_ACK - $<$*msg_type, cell, sender_node_address, current_x, current_y*$>$: sent by a node in response to a RTR_MOVE.
- FIND_CELL_REPLY - $<$*msg_type, dst_address, dst_cell, src_address*$>$: sent by a router who finds the target node of a FIND_CELL message in its cell.
- CHANGED_CELL - $<$*msg_type, sender_node_address, new_cell*$>$: sent by a node to other nodes with which it is in active communication after it moves to a new cell.
- RT_UPDATE - $<$*msg_type, routing_table*$>$: sent periodically by routers.

These messages are sent in the context of certain events. These events are described in the next section.

4.3 Events

The algorithm Kelpi is described by enumerating the events that occur in the lifetime of a node and defining the behaviour of a node on occurrence of a certain event. Events are listed separately for nodes and routers, but since all routers are also nodes, several events are common to both. An event listed as occurring for nodes is repeated as occurring for a router only if there is a difference in the resulting behaviour.

4.3.1 Events at a node

1. Initialization

This occurs before the deployment of the network. Nodes are initialized with:

- (x_0, y_0) and (x_l, y_l) - the co-ordinates of the bounding rectangle of the area of operation; and
- *maxTxPower* - the maximum transmission power that a node can use. This value is common for all nodes in the MANET.

2. Node comes on

When a node comes on, it already knows the values of the initialization parameters. From these parameters, it needs to compute *nodeTxPower;* s, the side of each cell in the grid; $cells_x$ and $cells_y$, the number of cells on the X and Y axis respectively; and r, the maximum transmission range. The calculations proceed as follows:

1. r is obtained from the value of maxTxPower by using a relation dependent on terrain, frequency of transmission, etc., which varies from system to system.
2. r is now the range that can be covered by a router. This must be twice the diagonal of a cell, since the routers of two neighbouring cells must be within range. We can therefore derive s as the side of a square, twice of whose diagonal gives r.

   $s = \frac{r}{2\sqrt{2}}$
3. Numbering of cells is as shown in Figure 3.1. We compute $cells_x$ and $cells_y$ next as follows:

   $cells_x = ceil(\frac{x_l - x_0}{s})$

   $cells_y = ceil(\frac{y_l - y_0}{s})$

Given the current co-ordinates of a cell as (x,y) we can calculate its cell as:

$$cell = floor(\frac{x - x_0}{s}) + cells_y * floor(\frac{y_l - y_0}{s})$$

3. Node in new cell

1. The node sends a HI (Here I am) message addressed to the cell router address.
2. If the node does not receive a HI_ACK within a time of HI_TIMEOUT, the node declares itself a router (event 4).

4. Node declares itself router

1. The node broadcasts a RH (Router Here) message. This is a signal to any other nodes who have set the HI timer to cancel the timer and learn that a router is present in the cell.
2. The node starts listening to packets addressed to the cell router address (CRA).
3. Responses to events henceforth are as described in 4.3.2.

5. Node receives RH message

The node performs the following actions if the cell field in the RH message is its current cell:

1. Cancels HI timer if it is set.
2. Sends HI message to the CRA of its *current_ cell.*

6. Node receives RTR_MOVE message

If the cell field in the message is the same as the *current_ cell*, the node replies to the CRA with a RTR_MOVE_ACK message which contains the node's address and its current geographical co-ordinates. This is to enable the router to choose as a new router a node which is closest to the centre of the cell.

7. Node moves cells

Nodes check their current cell every POSITION_POLL_INTERVAL. If the calculated *current_ cell* is different, the node does the following:

1. Sends a BYE message containing its new cell to the CRA of the cell it has just left.
2. Behaves as in event 3.

8. Node wants to switch off

The node sends a BYE message with a special *current_ cell* field indicating that it is switching off.

9. Node wants to send packet

1. The node checks to see if there is a valid entry to the destination in its *node_ cache.*
2. If so, the node sends the packet with the *dst_ cell* field set to the cache entry and sends it to the router of its cell.
3. If there is no valid entry in the *node_ cache*, the node buffers the packet in its routing queue and sends a FIND_CELL message to its router.
4. If a transmission using a valid cache entry fails, after a certain number of retries, the node sends a FIND_CELL message to its router.

10. Node receives FIND_CELL_REPLY

1. The node enters the cell for the destination in its *node-cache.*
2. Queued packets for the destination are dequeued and sent.

11. Node receives CHANGED_CELL

This message is received from a node with which this node has communicated in the recent past (less than CACHE_STALE time). On receiving this, the node updates its *node_ cache* entry for the sender of the message to the cell indicated in the message.

12. Node receives RTR_HANDOFF

1. The node initializes router data structures to those received in the RTR_HANDOFF.

2. Goes to Event 4.

13. Node receives a packet

The packet is forwarded to a higher layer.

4.3.2 Events at the router

1. Router starts operation

When a router starts operation, it receives updates from its neighbours and builds its *routing-table.* It also periodically broadcasts its own routing-table through a RT_UPDATE message. Other data structures are built in responses to received messages.

2. Router receives HI

1. If the node sending the HI is present in the *forwarding-pointers* of the router, the entry is removed.

2. The sending node's address is entered into the *node-list.*

3. The router sends a HI_ACK to the sending node.

3. Router receives FIND_CELL

1. The router checks its list of already seen FIND_CELL sequence numbers and if the message's sequence number is lower than the entry it has for that source-destination pair, it ignores the message.

2. Otherwise, it checks to see if it has the node with the required *dst_ address* in its *node-list.* If so, it sends a FIND_CELL_REPLY to the node that originated the message.

3. Otherwise, it broadcasts the packet to propagate flooding of the message.

4. Router receives packet

1. The router checks if the *dst_ cell* field in the packet is its current cell.

2. If so, the router transmits the packet to the node with *dst_ address.* If the node is not present, it checks if there is a valid forwarding pointer for that node, and if so forwards the packet to its new cell. Otherwise, the packet is dropped.

3. If the *dst_ cell* is not its current cell, the router forwards the packet to the router of the cell indicated in the forwarding hop for that *dst_ cell* in its *routing-table.*

5. Router approaches cell boundary

A router constantly checks to see if it is within a certain threshold distance from a cell boundary. This distance is calculated such that a router moving at the maximum possible speed should be able to complete hand-off of routing information before leaving the cell.

1. Router broadcasts a RTR_MOVE message and starts a timer.
2. It receives a RTR_MOVE_ACK message from all nodes in the same cell containing their geographical co-ordinates. These messages are received until the timer expires.
3. The router chooses the node closest to the centre of the cell and sends it a RTR_HANDOFF containing routing information for that cell.
4. The router then starts behaving as a node.

Note that a router starts behaving as a node even before it crosses the cell boundary, if at all it does. Also, if a router is alone in a cell and receives no RTR_MOVE_ACK messages, it continues to function as a router till it changes cells except for replying to HI messages. If another node enters the cell, times out on its HI message and sends a RH message, the router turns to node and sends a HI to the new router. The principle is that of a node near a cell boundary taking minimum responsibility of being a router.

6. Router becomes node

The router stops responding to the CRA. This at once implies that the router also stops receiving messages addressed to the CRA and routing updates from other routers.

7. Router receives BYE message

1. The router removes the source node of the message from its *node-list.*
2. The new cell of the source node is entered into its *forwarding-pointers.*

8. Router receives RH message

A router receives a RH message in its own cell when it is alone and has approached a cell boundary. The router, on receiving an RH message stops functioning as a router and sends a HI to the CRA.

9. Router receives a RTR_MOVE_ACK

1. If the RTR_MOVE_TIMER is not running, the message is ignored.
2. Otherwise, the source node address and its co-ordinates are stored.

After the timer expires, the node closest in position to the centre is chosen as the router and sent a RTR_HANDOFF message containing routing information.

10.Router wants to switch off

The router's behaviour in this case is the same as that in event 5.

4.4 Exception Handling

The most common source of unreliability is expected to be loss of routing messages. This requires careful handling with re-transmissions since many events are timer driven. Timeouts should be fixed with due consideration to other events that may be affected.

Another source of unreliability in the functioning of the algorithm is expected to be the capricious nature of the wireless medium itself. Calculated transmission ranges may not be very consistent or accurate; this problem can be alleviated to some extent by assuming large margins of error in calculations.

Unexpected router failure is another undesirable case. This can be solved by electing a new router in the cell, with the election being called for by the first node that detected the error. A simple criterion for electing the new router could be the distance of a node from the centre of the cell, with the closest being elected the router.

4.5 Summary

This chapter presented details of the algorithm Kelpi. The next chapter deals with the implementation of Kelpi for the network simulator ns-2.

Chapter 5

Simulation

The last chapter contained a detailed description of the routing algorithm Kelpi. In order to examine its working, it has been implemented for the Network Simulator ns-2. This chapter provides an introduction to ns-2, mentions the various changes that had to be made to ns-2 to incorporate Kelpi into it, and discusses the implementation details of some features of Kelpi.

5.1 Introduction to ns-2

ns-2 [13] is an event driven network simulator for simulating computer networks and network protocols. It was chosen for simulating Kelpi since it is free, open source and is used commonly by the research community for evaluating work in ad-hoc networks. The version of ns used for simulating Kelpi is ns-2.1b6.

5.1.1 ns framework

ns uses a split language approach. OTcl, an object-oriented version of Tcl is used as a control language for simulation. Setting up nodes and links, and controlling their behaviour is done in OTcl. Per packet processing, routing and other computation intensive activities are implemented in C++ for purposes of speed. Even the OTcl objects are represented during simulation run-time by C++ shadow classes. The results of a simulation are dumped in a trace file which can be analysed to extract information of interest. Network Animator (NAM) provides a graphical visualization of simulations.

5.1.2 Wireless support in ns-2

ns was first extended to support mobile and wireless networks by the CMU Monarch extensions [14]; these extensions were later incorporated into ns. For wireless support, ns defines a *MobileNode* inherited from the basic *Node* object with the additions that the *MobileNode* is able to move, keep track of its location and use radio transmission and reception with several propagation models to communicate instead of using a fixed link. To be received or transmitted by a *MobileNode*, packets have to move through several layers. A schematic is shown in Fig. 5.1.

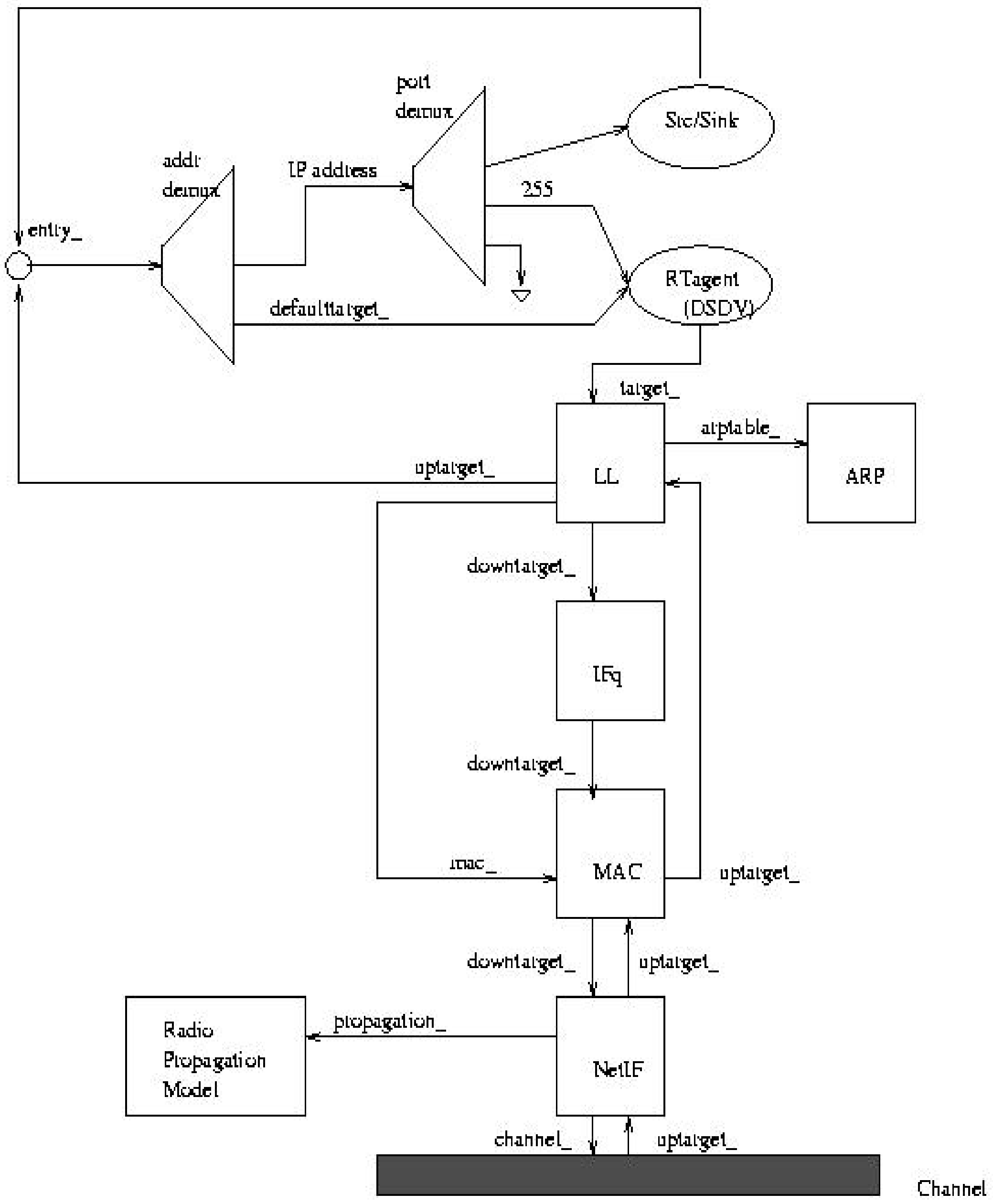


Figure 5.1: Schematic of the wireless model in ns-2 [13]

Ad hoc routing is implemented for ns-2 as a routing agent on port 255. Currently ns comes with four ad hoc routing protocols: DSR, DSDV, AODV and TORA.

5.1.3 Packet Format

A packet in ns-2 consists of the header of various layers of the network stack. In addition, there is a common header that is used by the simulator itself. This header accompanies the packet through the network and can be used to carry information required for simulation such as *packet_size*, *next_hop* for routing, etc.

5.2 Kelpi: Implementation

5.2.1 Preliminary modifications to ns-2

As mentioned in the previous section, implementing an ad hoc routing protocol involves creating a routing agent on a *MobileNode*. This section describes some of the changes that were made to ns to include the Kelpi routing agent.

The Kelpi routing agent is a C++ class called *KelpiAgent* which inherits from the class *Agent*. The class *KelpiAgent* contains the implementation of the algorithm described in Chapter 4.

The routing agent is set in the user's OTcl script which is interpreted on execution to initialize various simulation parameters. The file ns/tcl/lib/ns-lib.tcl contains code that instantiates the right routing agent. This was modified to attach an instance of *KelpiAgent* to each node that had Kelpi listed as its routing algorithm. Since this occurs at the beginning of the simulation, the Kelpi start procedure also contains code to pass essential simulation information contained in the user script to class *KelpiAgent*. This is done by using a 'command' method of the agent class which allows values to be passed from OTcl to C++. Information passed in this manner includes the node's address, transmission power set in the script and a handle to a topography object which contains location information. The file ns/tcl/mobility/kelpi.tcl contains initialization of the *KelpiAgent* variables. This had to be sourced into ns-lib.tcl.

In addition, make files for ns had to be changed to ensure that the newly created objects are compiled into ns-2.

5.2.2 Modifications to the ns-2 packet header

The common packet header was modified to carry certain extra information required for implementing Kelpi. In order to attach the destination cell field with each packet and the source cell field with some packets, *source_cell* and *destination_cell* fields were introduced in the common header. Another field *txPower* was introduced to help implement a multi-level wireless network interface. These modifications can be found in ns/packet.h.

5.2.3 Implementation of data structures

In the implementation of Kelpi, it is assumed that the maximum number of nodes, MAX_NODES is 999.

Node-list

The *node-list* is implemented as a single dimensional array of characters indexed by node address. If a node is present in the cell, the value for that particular element is set to 'y'; otherwise, it is 'n'.

Router-queue

Other routing algorithms like TORA and AODV use this data structure for buffering. Therefore, it is already implemented in ns-2. It supports the operations enque, deque, and deque all packets for a particular node address.

Forwarding pointers

This consists of a MAX_NODE sized array of records which contain:
$<$*new cell, timestamp*$>$

Node-cache

This is implemented as a MAX_NODE sized array of records which contain:
$<$*node_ address, cell, time_ last_ accessed, request_ sent, request_ time*$>$

The *time_ last_ accessed* field is used to recognize a stale cache entry. The *request_ sent* and *request_ time* fields are associated with the FIND_CELL messages sent by a the node.

5.2.4 Implementation of Cell Router Address

In Kelpi, each cell has a unique address to which the node acting as router of that cell will respond to. This has been simulated in the implementation by maintaining a mapping of cells to their current router's address. A node wanting to send a packet to the CRA sends it to CRA(cell) which is transformed into the actual address of the current router of the cell.

5.2.5 Implementation of multi-powered network interface

This is simulated by modifying the physical layer ns-2 code in the file ns/wireless-phy.cc. The required power of transmission for the calculated cell size is calculated using the Two Ray Ground propagation model used by ns-2. This value is passed along with each packet through the *txPower* field introduced in the common header. Just before transmission, this value is accessed and copied into the variable that defines the transmission power at the physical level.

5.2.6 Current implementation status

The algorithm Kelpi has been implemented as described in Chapter 4, with the exception of two features. The periodic routing updates are not used currently; instead, correct routing information learnt from a global picture is given to the routers. This is to examine whether the rest of the algorithm is working as expected. Also, the CHANGED_CELL message, sent by a node to inform other nodes in active communication with it about a change in its cell, has not been implemented.

5.3 Experimentation

Simulations have been run for small test cases involving 4 to 5 nodes spread across 2 to 3 cells. The trace files for these simulations show that cell discovery, destination cell caching, buffering, packet forwarding and router hand-offs occur successfully.

Chapter 6

Conclusion

The work presented in this thesis describes a routing algorithm for mobile ad hoc networks, Kelpi, designed to provide stable routes between nodes in a network. It is based on the principle that routing information can be retained in a particular geographical area despite the nodes of the network being transient. This is achieved by imposing a cellular structure on the MANET, which allows node mobility in a region to be hidden from other parts of the network. The routes formed by the routers in Kelpi can be thought of as a *dynamic virtual backbone* enabling stable routes to be formed. Kelpi is also designed to increase link-layer throughput by using multiple levels of transmission power to reduce radio interference.

Preliminary simulations of Kelpi have been carried out using the ns-2 simulator.

6.1 Future directions

Although Kelpi is designed to provide stable routes, its requirements are quite restrictive: it uses an accurate positioning service and a network interface that operates at multiple levels of transmission power. Some directions for future work are:

- Implementation of the complete algorithm.
- Eliminating the need for a positioning system. An alternative could be a compass and an inertial motion detector attached to the mobile device; work on augmenting hardware to aid MANET routing could be an interesting direction.
- The principle behind Kelpi may be applied to other routing algorithms to provide a more general solution.

Bibliography

[1] C.E. Perkins and P.Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," Computer Communications Review, pp. 234-244, October 1994.

[2] A.S. Tanenbaum, "Computer Networks," 3 ed., Prentice-Hall India, 1994.

[3] E.M. Royer and C.-K. Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks, IEEE Personal Communications Magazine," April 1999, pp. 46-55.

[4] D.B. Johnson and D.A. Maltz, "Dynamic Source Routing in Ad-Hoc Wireless Networks," Mobile Computing, ed. T.Imielinski and H.Korth, Kluwer Academic Publishers, pp. 153-181, 1996.

[5] B. Cameron Lesiuk, "Routing in Ad hoc networks of mobile hosts," Technical Report, Department of Mechanical Engineering, University of Victoria, Victoria, Canada, http://phantom.me.uvic.ca/clesiuk/thesis/reports/adhoc/adhoc.html

[6] C.E. Perkins and E.M. Royer, "Ad-Hoc On-Demand Distance Vector Routing," Proceedings of 2nd IEEE Workshop on Mobile Computing Sysems and Applications, February 1999.

[7] V.D. Park and M.S. Corson, "A Highly Adaptive Distributed Routing Protocol for Mobile Wireless Networks," Proceedings of INFOCOM '97, April 1997.

[8] C.-K. Toh, "A Novel Distributed Routing Protocol To Support Ad-Hoc Mobile Computing," Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communication, pp. 480-486, March 1996.

[9] R. Dube, C.D. Rais, K.-Y. Wang, and S.K. Tripathi, "Signal Stability based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks," IEEE Personal Communications, pp. 36-45, February 1997.

[10] C.-C. Chiang, G.Pei, M.Gerla, T.-W. Chen, "Scalable Routing strategies for ad hoc wireless networks," IEEE Journal on Selected Areas in Communication, pp. 1367-79, Aug. 1999.

[11] Z.J. Haas and M.R. Pearlman, "The performance of a new routing protocol for the reconfigurable wireless networks," IEEE International Conference on Communications, pp. 156-60, June 1998.

[12] Y.B. Ko and N.H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," Proceedings of ACM/IEEE MOBICOM '98, October 1998.

[13] "The ns-2 Manual," available at http://www.isi.edu/nsnam/ns/doc/

[14] "CMU Monarch Wireless and Mobility extensions to ns," available at ftp://ftp.monarch.cs.cmu.edu/pub/monarch/wireless-sim/ns-cmu.ps