Wireless Mobile Ad-hoc Network Protocols and Evaluation with Model Checking

Jiang Wu

February 28, 2007

Contents

1 Overview

A Mobile Ad-hoc Network (MANET) is a wireless network set up temporarily without a wired infrastructure (routers, switches, servers, cables, access points, etc.). The wireless nodes in a MANET may move around and each one of them may need to forward packets for other components in the network. Because they can be deployed quickly, MANETs could be used for disaster rescue, battle field communication, sensor networks, etc.

The most important constraint of a MANET is power, since all wireless nodes are powered by batteries. While technologies in other areas of the IT industry follow Moore's Law, battery technologies advance slowly compared to the rapidly increasing need for power on laptops and PDAs. Wireless communication can take one third of a laptop's and half of a PDA's power consumption. A MANET protocol must take power saving into consideration since the lifetime of a network is often determined by the time the first node runs out of power.

The second major limitation of MANETs is the interference between wireless data transmissions. For a specific channel, the data packets sent out by a specific component can be heard by every other component within the wireless range of the sender. On the same receiving node, no two transmissions can happen at the same time on the same frequency. This reduces the average per node throughput. Packet conflicts can happen due to issues like the "hidden terminal problem". The result is that packets get dropped. MANET protocols must be carefully designed to use the wireless channels efficiently while avoiding conflicts.

The third problem of MANET is the short transmission range of wireless nodes. The radio range of small wireless devices are typically on the order of a few hundred meters. This results in large hop counts in large MANETs. Large hop counts in turn result in long latency, high packet drop rates and slow routing convergence.

The fourth problem of MANET is that the network topology may change rapidly due to the movement of wireless nodes. This may make the network slow to converge.

Another big issue for MANET, which has been ignored for years, is the address allocation problem. A MANET usually doesn't have a central address allocation authority or a hierarchical structure. Address allocation could be an issue since a node may keep joining and leaving a MANET. Also, a MANET may partition sometimes and recover later.

Numerous protocols in the area of MANET routing, power management and address allocation have been designed. One important problem is how to determine a protocol is correct. A protocol can be checked by testing after it has been implemented. It can also be tested by simulation or deductive reasoning (theorem proving). These methods usually involve a lot of manual intervention and are slow. Another way to check the correctness of a protocol is model checking, which tests the properties that a protocol is supposed to satisfy using a model constructed for the protocol. The benefit of model checking is that it can be completely automatic and fast. Using counter-examples returned from model checkers, fundamental problems of a protocol can be detected before real implementation.

My research will focus on investigating the specific properties that a wireless routing protocol needs to satisfy, evaluating existing wireless routing protocols with model checking, and designing better protocols.

In this report, existing work in the areas of MANET routing (unicast) and address allocation are reviewed. Three major model checking methods are also investigated. The report is organized as follows. The requirements for and classification of MANET routing protocols are given in the next section. Section 3 reviews works in the area of topology-based routing protocols. Section 4 discusses some geographical-based routing protocols. Three important solutions to address allocation are discussed in Section 5. Model checking and some model checking methods are introduced in Section 6. The last section gives a conclusion and outlines some outstanding issues for MANETs.

2 MANET Routing Protocols

Popular routing protocols (OSPF, RIP2, BGP) used for wired networks cannot be applied directly to MANET for the following reasons.

First, due to the structureless address allocation, for each destination, a routing entry is needed. Since popular routing protocols are mostly subnet based, they are not scalable here anymore. Second, interference between flooding routing messages can cause heavy packet loss. Third, the low channel bandwidth of MANET makes the routing overhead problem even worse. Last, wireless nodes might move quickly. This can cause the network to converge very slowly.

Therefore, new protocols that suit the special needs of MANETs must be designed. This report focuses on unicast routing.

The major measures that we use to evaluate routing protocols are: network size and structure supported, the routing accuracy, and routing overhead. A good routing protocol can find a good tradeoff between the last two measures. Using Zhou's classification in [29], routing protocols for MANET can be classified into the following categories.

- Topology-based routing protocols
  - Proactive routing protocols
  - Reactive routing protocols
  - Hybrid routing protocols
- Geographical-based routing protocols

Topology-based routing protocols are routing protocols calculating the best route to a destination based on the topology information collected from the network.

Within this category, *proactive routing protocols* are routing protocols that calculate the routes to all the destinations before a transmission actually happens. *Reactive routing protocols* are routing protocols that calculate the route to a destination only when it's necessary for a transmission. *Hybrid routing protocols* are routing protocols that combine proactive routing and reactive routing.

Geographical-based routing protocols are routing protocols that calculate routes based on the geographical locations of the destination node and neighboring nodes.

In the following sections, we explain each category of protocols with examples.

3 Topology-based routing protocols

Topology-based routing protocols are widely used in today's Internet. By exchanging topology information (local routing tables or neighbor lists) among the nodes of a network, this set of routing protocols creates routing entries for individual addresses or address blocks. Therefore, the more destinations, the more memory and processing power are needed.

There exist different classifications for topology-based routing protocols in the Internet. Here, we divide them by whether a route is created proactively or on demand, since this introduces the largest distinctions in the behavior and performances of MANET routing protocols.

3.1 Proactive Routing Protocols

Proactive routing protocols calculate the routing table even when there is no packet to send. The benefit of calculating routes beforehand is the short latency in finding a route. The drawback is that to maintain routes for each destination, the nodes has to keep exchanging routing messages even when there is no traffic at all.

3.1.1 Destination Sequence Distance Vector (DSDV)

One of the oldest routing protocols for MANET is DSDV [21]. DSDV is a modified version of the classical Bellman Ford Routing protocol.

For a destination, DSDV's routing table keeps the next hop, the metric (the hop count), and a sequence number, which is generated by the destination to mark the freshness of the route.

A node periodically broadcasts its whole routing table or modifications to its routing table to its immediate neighbors. For each route, the routing update carries a new sequence number which is originally given by the destination node. Upon receipt of a new route for a destination, if there is no route for the destination yet, a node adds the route together with the sequence number to its routing table. If a route already exists, the node picks the route with a greater sequence number. If a route with the same sequence number already exists, the node picks the route with a better metric.

The protocol is not very different from the standard distance vector routing in the wired network. The use of sequence number further reduces the chance of loop forming. One node only needs to update the changed part of it's routing table to its neighbors. The distance vector feature prevents it from being used in large wireless networks.

3.1.2 Fisheye State Routing (FSR)

G. Pei, M. Gerla et al. designed a proactive link state routing protocol called "Fisheye State Routing" [19]. FSR is a hierarchical routing scheme since instead of flooding link state information, it divides the nodes in a MANET into multiple levels in terms of hop distance. FSR nodes exchange link state information with neighbors periodically. A link state message is marked with the number of hops it has travelled. Packets with fewer hop counts are forwarded with a higher probability. A node receives link state messages originated from nearby nodes with higher frequency than those from further nodes. By doing this, a node gets detailed routing information for the nodes in the local area while less detailed information for other nodes. Although less optimal routes might be chosen for some destinations, as a packet nears its destination, it has increasingly correct routing information.

This protocol can improve the scalability while keeping the route accuracy and delivery rate high. The drawback is that many duplicate link state messages are still sent. This is especially harmful in MANET since the bandwidth is limited and frequent duplicate transmissions can cause many conflicts. The following two protocols focus on solving this problem.

3.1.3 Optimized Link State Routing (OLSR)

In [13], P. Jacquet et al. proposed a link state routing algorithm that can eliminate many unnecessary link state message broadcasts using a method called Multi Point Relaying (MPR). In addition, the amount of link state transmitted can also be reduced by only advertising the MPR selectors of a node.

The idea of OLSR is as follows. Every wireless node maintains a list of its immediate neighbors through periodic beacon messages. Neighboring wireless nodes exchange their neighbor lists through HELLO messages. These HELLO messages work like link state routing messages. Every node thus knows the two hop topology around itself.

Every node picks a set of one hop neighbors to cover all of its two hop neighbors. This set of immediate neighbors are called MPR nodes. Every node tells its immediate neighbors whether they are chosen as MPR nodes for it. This is also implemented using HELLO messages. Upon receipt of a link state routing message, a node checks if it has been chosen by the sender as its

MPR node. If true, the node re-broadcasts the link state message.

Only the nodes that are chosen by some nodes as their MPR nodes generate link state messages. The link state messages only contain the nodes that choose them as MPR nodes. This set of nodes are called MPR selectors. Using the Dijkstra algorithm, the route to every single destination can be calculated.

The OLSR routing protocol is very popular and has become IETF RFC 3626 [2]. The greatest strength of the protocol is that the flooding overhead can be greatly reduced.

3.1.4 Joint Architecture Vision for Low Energy Networking (JAVeLEN)

Jason Redi et al. proposed a complete architecture, JAVeLEN[24], for low power consumption MANET. The architecture mainly targets two problems, power management in the link layer and efficient power-aware routing. It is especially suitable for large scale sensor networks.

For power management, JAVeLEN uses two radio channels. One high power, high data rate channel for data packets and another low power, low data rate channel for basic signaling.

Time is divided into equal-sized slots. The clocks on all nodes in a MANET are synchronized. The low power channel is turned on periodically to receive HAIL messages, which indicate that someone have data ready to send. Upon receiving a HAIL message, a node turns on the high power channel. The sender of the HAIL message sends the data.

The time slots in which a node turns on the low power channel are determined by a Pseudo-Random Number sequence and a receiving threshold. When the number generated is less than the threshold, the node turns on its low power channel. A node's schedule, the current position in the PRN sequence and the threshold, are distributed to its immediate neighbors. Therefore, a sending node can find out whether the recipient of a packet is ready to listen. If so, it sends out the HAIL message. Otherwise, it tries to send packets destined for another node.

In [5], L. Dai et al. further improve the power efficiency and delivery rate by turning off low power channels when a node knows it can't do any sending or receiving in a time slot. The knowledge is acquired through collection of 1-hop and 2-hop neighbors' on/off schedules.

JAVeLEN routing uses an optimized version of OLSR that considers the combined transmission power when constructing the MPR tree for a node's 2 hop neighbors [27].

JAVeLEN is a complete solution for a flat MANET. Its main strength comes from the pseudo random number on/off scheduling. It not only saves power but also prevents most conflicts from happening. It can avoid the hidden terminal conflict without using RTS/CTS.

3.1.5 Landmark Routing in Ad Hoc Networks with Mobile Backbones (H-LANMAR)

Most of the existing routing protocols can't support a flat MANET that has over 100 nodes. Kaixin Xu et al. designed a hierarchical routing architecture [28] for large MANETs (on the order of a few thousand nodes) based on the structure of the Internet.

The assumption is that there are a set of wireless nodes that are equipped with multiple radio, more energy and are capable of doing long range communications. For example, the helicopters and armored vehicles on battle fields. These nodes can be used to build a backbone network for a large MANET. In addition, groups of wireless nodes tend to move together, just as soldiers in the same company usually move together, with a few armored vehicles.

Therefore, the network can have two layers. One layer is the backbone layer, the other is the subnet layer. In such a network, local traffic is forwarded as before within the border of a subnet and inter-subnet traffic is relayed by the backbone nodes. The long path problem in a flat MANET no long exists.

The first problem in constructing such a network is how many backbone nodes (BNs) are needed. Since the average per node throughput decreases on the order of $\Theta(\frac{W}{\sqrt{n}})$ (W: average per node bandwidth, n: total number of nodes)[9], we want the size of a subnet to be small. At the same time, we need to keep the number of backbone nodes low in order to increase the throughput of the backbone network. [28] determines that the optimum number of backbone node is $\frac{W_2}{W_1}\sqrt{N}$, where W_1 is the average per node bandwidth of normal wireless nodes, W_2 is the bandwidth of the backbone channel and N is the total number of nodes in a network.

The next problem is how to deploy the BN's. Using a simple *Distributed Clustering* algorithm called *Random Competition based Clustering* (RCC), the node that first declares itself as a BN (by broadcasting a packet) becomes the head of a cluster/subnet. A cluster head claiming message travels an adjustable number of hops in order to achieve ideal subnet sizes. Each subnet gets a unique network id, which is used for backbone routing.

The routing for the hierarchical network uses a modified version of LANMAR [20], a routing protocol for flat MANETs. LANMAR is a routing protocol used in situations where groups of wireless nodes move together (logical groups). Each logical group elects one node as the landmark. LANMAR routing uses landmarks to track the movement of the logical groups. In LANMAR, there are two routing protocols running at the same time. Landmarks use distance vector routing to connect to each other. Local nodes use a link state routing to connect to each other. Every local node has a link state table for its local network and a distance vector for landmark nodes. Local packets are forwarded to the destination using the local routing table. Packets sent to a destination in another logical subnet is sent *toward* the landmark of that subnet using the distance vector route. The forwarding of inter-subnet packets are carried out by local nodes too.

In H-LANMAR, a hierarchical version of LANMAR, inter-subnet packets are forwarded to the nearest BN first. Then the BN forwards the packet to the BN of the destination subnet using the backbone network. Once the packet reaches the destination subnet, local routing is used to forward the packet to the final destination. The length of the forwarding path can thus be reduced greatly.

H-LANMAR nodes runs LANMAR at the same time in case the local BN fails. In that case, the flat LANMAR route is used to forward packet to remote destinations.

3.1.6 Summary

Name	Network Size	Network Structure	Route Update	Route Accuracy	Routing Overhead	Power Awareness
DSDV	Small	Flat	1 hop	High	Medium	No
FSR	Medium	Hierarchical	different relay prob.	Maybe not	Medium	No
OLSR	Large	Flat	Multi-Point Relay	High	Low	No
JAVeLEN	Large	Flat	Multi-Point Relay	High	Low	Yes
H-LANMAR	Large	Hierarchical	Subnet Routing	High	Low	2 Radios

The table above compares the protocols introduced in this section. Proactive routing protocols range from basic DSDV routing to complete routing/power management architectures like JAVeLEN and hierarchical solutions like H-LANMAR. JAVeLEN is for flat MANETs, which don't scale as well as hierarchical MANETs. A solution combining the idea of JAVeLEN and H-LANMAR should be able to support very large MANETs.

3.2 Reactive Routing Protocols

The basic idea of reactive routing protocols is to find the route to a destination only when necessary. By eliminating the periodic routing updates, these routing protocols are aiming at reducing the routing overhead. These routing protocols assume that the network is not very big and the nodes' rate of motion is moderate.

3.2.1 Dynamic Source Routing (DSR)

David Johnson et al. proposed DSR, a reactive source routing protocol for MANET in [14]. DSR is basically a source routing protocol. When a node tries to send a packet to a destination, it checks to see if there is a source route available in its route cache. If so, it attaches the route to the packet and sends it out. The packet is forwarded by the nodes specified in the route. Otherwise, a route discovery process starts.

To find a route for a destination, a requestor node floods a RREQ message to the network. The RREQ message carries a vector that records the nodes that it traverses. At every hop, the forwarding node records the route that the RREQ message traverses into its route cache. The route can be used to send packets to the requestor node. Then the forwarding node checks to see if the RREQ message is destined for it, or it is not the destination but it has a route for it. If so, it returns a RREP message to the requestor node, with the full route to the destination. If not, it appends its id into the vector carried in the message and re-broadcasts the RREQ message. The RREQ message finally reaches the destination or gets discarded when it reaches a node that have already seen it.

When the destination receives the RREQ message, it constructs an RREP message carrying the vector of nodes recorded in the RREQ message. When a node that having a route to the destination receives the RREQ message, it constructs an RREP message carrying the vector of nodes recorded in the RREQ message appended with the route discovered.

For a node to return the RREP message back to the requestor node, if the node has a route to the requestor node, it attaches the route to the RREP message as the source route. Otherwise, it can reverse the route just found and attach it to the RREP message. If network links are directed, the node can start another route discovery in which the RREQ packets are piggybacked with the RREP messages.

The nodes in a network maintain the routes cached locally for every destination. If a node decides a link is broken because of un-acknowledged data messages at a node, it sends a ROUTE_ERROR message back to the sender of the data message. The ROUTE_ERROR message contains the hosts on both sides of the broken link. It can be sent back along the route the data message coming from or some local route in the cache. On the route back, any route found in the forwarding node's route cache that contains the broken link is removed. When receiving a ROUTE_ERROR message, the sender of the data message does a new route discovery for the destination.

The benefits of DSR are its simplicity and its support on directed networks. The problems of it are, flooding is costly, a whole route has to be rebuilt even when a single link is broken and the use of route cache can put a limit on the size of the network supported by DSR.

3.2.2 Temporally-Ordered Routing Algorithm (TORA)

Vincent Park et al. proposed another reactive routing algorithm, TORA [18], to minimize the reaction to topological changes. The protocol can provide multi-path routing and reduce routing overhead by localizing link recovery when link failure happens. Also, the protocol supports networks with directed links.

In TORA, a separate instance of the routing algorithm is run for each destination. The algorithm has three basic functions, creating routes, maintaining routes and erasing routes. Like DSR, a route is created when a packet needs to be sent to a destination and there is no route for it. Route creation begins with a QRY message. The process of creating a route is like creating a directed acyclic graph (DAG) in an undirected MANET topology. Maintaining routes is the process of re-establishing routes during topology changes using UPD messages. Erasing routes is the process of making all the links towards partitioned destinations undirected. Route erasing is done by a CLR packet.

The creation of the DAG for a destination is marking every node in the network with "heights". A node having a higher height than one of its neighbor is said to have a directed link to the neighbor. Initially, for a destination, every node has a height "NULL", treated as infinity, except that the destination has a height 0, the lowest point. A node knows the heights of all its neighbors. If a node has a neighbor with a lower height than its own, the node is defined as having a route to the destination.

For route creation, a node that has a height "NULL" and needs a route to a destination broadcasts a QRY message. Nodes receiving the QRY but don't have a route for the destination re-broadcast the QRY. A node, say i, that has a route to the destination sets its height to H_j+1 where H_j is the next hop node's (node j) height. After this, the node broadcasts a UPD packet that carries the node's height. Every node that receives the UPD packet changes its height to the minimum of its own height and the height carried in the packet plus 1. Eventually, the UPD packet returns to the node that sent the QRY message, giving it at least one route.

After the route discovery, packets are forwarded in the network like water flowing down a slope. They follow the route on which heights descend the fastest.

When link failure happens, if a node has multiple next hops, nothing needs to be done. What's complicated is when link failure happens and no other neighbors have lower height that the node. In this case, route maintenance is needed.

The basic idea of route maintenance is to raise the node's height using an additional reference height so that the node's height becomes a local maximum. This update might make neighboring nodes local minima too. Repeating the process, the new height propagates "downstream". If a node that has a downstream route to the destination can be reached, a route is recovered. A new height is going to propagate back to where the link failure happens. If such a node can't be found, a CLR packet is broadcast throughout the partition to set the heights for the destination on every node to "NULL".

TORA can find a route in finite time for a destination. It supports directed networks and is resilient to network failures in that it doesn't always require an end to end re-route. However, it still involves high routing overhead due to broadcasting QRY, UPD and CLS messages.

3.2.3 Ad-hoc On-Demand Distance Vector routing (AODV)

The routing protocol, AODV [22], proposed by Charles E. Perkins et al. is very similar to DSR. The most important difference is, in AODV, instead of storing the routes a RREQ message traversed, a node stores the last hop that the message comes from. The last hop node is used as the next hop toward the originating node. When a RREQ message reaches the destination or a node having a route for the destination, the RREP message doesn't carry the route. Instead, the RREP message is forwarded back using the local routing tables along the route back. When the RREP message returns, the last hop before it reaches the requestor node is stored in the cache as the next hop node to the destination. AODV scales better than DSR since it doesn't use source routing.

As in DSDV [21], sequence numbers are used in the RREQ and RREP messages to ensure the freshness of the message and to prevent loop from happening. AODV also uses a "Ring Search Algorithm" to reduce the flooding overhead. That is, the algorithm sets a small ttl value for the RREQ first and increases it in case the destination is not found.

In addition to using acknowledgements to data messages to detect link failure, AODV uses explicit HELLO message to detect neighbors leaving. When a neighbor of it is found left, a special REPP message is sent back to all possible source nodes using the link. Upon receiving the special REPP message, the source nodes restart a route discovery.

3.2.4 Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures (CEDAR)

In [25], Sinha et al. proposed a solution solving the high cost of flooding query messages in reactive ad hoc routing protocols. DSR and AODV are two of such protocols.

The paper uses the result of an earlier paper [26] on constructing a minimal set of nodes who can communicate with all other nodes in a MANET, a minimal dominating set. The nodes elected to the dominating set are called "core nodes". The core nodes are at most three hops away from each other. A communication tree can be constructed among the core nodes by exchanging beacon messages in the network. A beacon message is like a link state routing message carrying the list of core nodes connected to the source node. A beacon message travels at most two hops. Using the beacon messages, a core node can find a route to any other core node in its 3 hop neighborhood.

Now, the QUERY messages in DSR and AODV are not broadcast any more. Instead, they are sent to their neighboring core nodes using unicast. By using unicast, 802.11 ACK and

RTS-CTS mechanisms can be used to alleviate conflicts. Only the core node that is directly connected to the QUERY destination needs to forward the QUERY message to it. Therefore, the message overhead can be reduced greatly.

This paper presents a better solution than MPR in OLSR [2] for reducing the broadcast cost of the QUERY based routing protocols. The key to this solution is CEDAR, the distributed dominating set election mechanism designed in [26]. This mechanism would actually be useful for any protocol that requires flooding messages throughout the whole network.

3.2.5 Summary

[1] compares DSR, AODV and TORA. Simulation results show that DSR and AODV scale better than TORA. Also, DSR responds better to topology changes due the use of route caches. However, AODV has a better memory usage thus scales better. It's suitable for a large but more static network.

Reactive routing protocols are more suitable for small MANETs that don't change very fast. Hence the cached routes can be reused and the overlay network of core nodes can last longer. The reduced routing overhead comes at the cost of long delay caused by the query/response process. The following table summarizes the protocols introduced in this section. None of the protocols takes power management into consideration.

Name	Network Size	Network Structure	Route Discovery	Route Stored	Routing Overhead
DSR	Small	Flat	Flooding	Route Cache	Low
TORA	Small	Flat	Flooding	Heights	High
AODV	Small	Flat	Flooding	Next Hop	Low
CEDAR	Large	Hierarchical	Core Forwarding	n/a	Low

3.3 Hybrid Routing Protocols

Proactive protocols response quicker but have a higher routing overhead. They are more suitable for fast changing, larger sized MANETs. Reactive protocols are more suitable for small sized, less dynamic MANETs. Hybrid rooting protocols try to combine the benefits of both of them. ZRP [10], proposed by Z. J. Haas et al., divides a network into zones from the point of view of each single node.

In ZRP, a node propagates its proactive routing message (distance vector or link state) to nearby nodes within a fixed number of hops (a routing zone). The limit on the hop count is called *zone radius*, a critical parameter of ZRP. Hence, each node has complete routing information about every single node within its routing zone.

Each node has its own routing zone. The zones of different nodes may have large overlap. The nodes on the border of a node n_i's routing zone (distance to n_i equals to the *zone radius*) are called peripheral nodes.

When a node has a packet to send, it checks its proactive routing table. If a route can be found, the packet is delivered to the destination within the zone. Otherwise, it checks its inter-zone reactive route cache. If a route can be found, the packet is delivered along the inter-zone route. Otherwise, the node sends a QUERY message to all of its zone peripheral nodes. Here, the QUERY message is delivered by unicast.

Upon receipt of a QUERY message, a peripheral node checks its zone routing table and inter-zone routing cache. If a route can be found, the node returns the route found (inter-zone route found plus the route carried in the QUERY message) back to the requestor node. Otherwise, it appends its id to the QUERY message and sent it to its peripheral nodes. This way, the QUERY message finally reaches the destination in some zone and a route is returned.

The QUERY message is not delivered to the network by broadcast. The way ZRP delivers QUERY messages is called "bordercast". That is, a QUERY message is sent to the sender's peripheral nodes. The peripheral is going to relay the message to its own peripheral nodes. This goes on until the destination can be reached. The route returned to a querying node is not a complete end to end route either. Besides the source and the destination, it contains the set of peripheral nodes that can forward a packet through a string of routing zones to a destination, which is located at the last routing zone.

Therefore, the number of QUERY messages needed and the route discovery delay are both reduced. This requires local proactive routing, which could be costly for large zones. If the sizes of the zones tend to be small, the local routing can be quite efficient.

When the "zone radius" is 1, the protocol becomes a pure reactive routing protocol. When the "zone radius" is the radius of the network, the protocol becomes a pure proactive routing protocol. An optimal radius need to be found to get the best trade-off.

4 Geographical-based routing protocols

Topology-based routing protocols, although widely used in the wired networks, are not scalable in MANET. One reason is that MANET doesn't have a hierarchical structure like the Internet, which can route packets using IP address prefixes. In most of the protocols we have seen above, for each destination, a separate routing entry needs to be created. Another reason is that the topology of a MANET tends to change frequently. A topology changes leads to network wide routing fluctuation. The routing updates or query/reply messages themselves can easily use up the bandwidth of the MANET.

Geographical-based routing protocols use location information for routing, instead, based on the intuition that packet forwarding is to get a packet geographically closer and closer to the destination. These routing protocols usually require the knowledge of the destination node's location through some sort of location service. From its neighbors, a sender forwards a packet to the one that is closer to the destination.

4.1 Greedy Perimeter Stateless Routing (GPSR)

Brad Karp and H. T. Kung designed GPSR [15], a geographical-based routing protocol that uses only node positions and local topologies (adjacent nodes) to make routing decisions. The method assumes that all wireless nodes are roughly on the same plane and there are no obstacles between the nodes. In addition, the wireless channels are all bidirectional. All node positions are available to everyone through a location mapping service like the one proposed in [16].

The algorithm operates in two modes, greedy routing and perimeter routing. First, from its immediate neighbors, each node picks the one that is closest to the destination. If such a node cannot be found, there must be a void area between the current node and the destination in which there does not exist a node that is closer to the destination and is within the radio range of the current node. In this case, a right-hand rule is used to forward a packet around the void area. The rule is, suppose x receives a packet destined for d from y and cannot find a node that is closer to d, it forwards the packet to the node that can be found counter-clockwise around x from the edge (x, d).

Perimeter routing requires that the graph that the nodes use to represent their knowledge of the topology be non-crossing. Using one of two rules, each node removes some of its neighbors from its neighbor list. The process planarizes the local graph into a Gabriel Graph (GG) or Relative Neighborhood Graph (RNG). For example, for any neighboring nodes u and v, if in u's neighbor set, there exists a node w such that $d(u, v) \leq max[d(u, w), d(v, w)]$, eliminate the edge (u, v). This rule turns the graph into an RNG.

GPSR is nearly stateless and uses minimal memory storage. It incurs very low control message overhead. Beacon messages are the only overhead and they can be piggybacked onto data packets.

GPSR has the following possible problems. First, geographic routing may not be optimal. A node closer to the destination may not be a better routing choice. Second, GPSR doesn't consider the influences of obstacles on the protocol. When there are obstacles between wireless nodes, GPSR maybe not be able to find a good route or even any route at all. Third, the location service is not free. The location distribution protocol can also limit the size of the network.

4.2 "Direction" Forwarding (DFR)

In [7], Mario Gerla et al. presents another geographical routing scheme, DFR. The protocol is designed as a backup routing method to work with distance vector routing algorithms. When a routing entry for a destination is lost due to node movement, DFR finds a node in the direction that matches the destination direction the best and forward packets to it.

DFR assumes that every node knows its own coordinates. Thus, by neighbor discovery, every node knows its neighbors' locations. During the normal operation of table based routing, when a new route to a destination is learnt from a neighbor, the position of the neighbor is recorded together with the route. When a node sends a route update for a destination, this information is also carried in the message. Therefore, upon receiving a route for a destination, a node knows the coordinates of the node two hops downstream on the route towards the destination. In the algorithm, such a node is called a virtual destination.

For each destination, knowing the virtual destination's coordinates and its own coordinates, the direction to the virtual destination can be calculated. Suppose that the coordinate of the node calculating the direction is (x_1, y_1) and one of its virtual destination's coordinate is (x_2,y_2), the direction to the destination is represented by (r, θ), where

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

$$\theta = \tan^{-1}(\frac{y_2 - y_1}{x_2 - x_1}) \tag{2}$$

If there are multiple virtual destinations, the θ to the destination node is the average of the θ's of all the virtual destinations.

When a route can't be found using the normal distance vector routing, the algorithm finds a neighbor whose direction matches the direction to the virtual destination the best. If there are multiple virtual destinations, the one that matches the average direction to the destination node best is chosen.

DFR is simple and doesn't require a node to know the position of all the nodes in the network like GPSR. Therefore, it causes very little overhead. However, DFR is not a pure geographical based routing protocol since it need to work together with an existing distance vector routing protocol.

5 Address Allocation

Address allocation in a MANET is difficult since there is no centralized entity that can provide DHCP service. Wireless nodes may move around. This problem was initially ignored since addresses can be manually configured. However, when the number of addresses is limited and many different devices may join and leave the MANET, a reliable address allocation scheme is needed. Due to the ad hoc properties of the MANET, an address allocation mechanism must be completely distributed.

The measures that we use to evaluate an address allocation scheme are: network supported, chance of duplicate address allocation, chance of address leakage, address allocation delay and partition merging capability.

5.1 IP Address Auto-Configuration for Ad Hoc Networks (IAAC)

In [23], Charles Perkins presents a simple solution to address allocation in ad hoc networks. The idea is, in a MANET running DSR or AODV as its routing protocol, a node joining the network picks a random address from the range 2048-65534 of a known address block 169.254/16 and sends a DSR or AODV RREQ message to the network for the address picked. For the RREP message to be returned, the new node picks a random address from the range 1-2047 of the same address block and uses it as a temporary address. If a route can be found toward the picked address, the address must have already been used. The new node picks a new address and tries again. If no route can be found for the address, the new node tries the same query for a few rounds to deal with message losses. If no reply can be received, the new node uses the address.

The solution is simple and efficient. However, it relies on two specific routing protocols. In addition, the solution doesn't consider the situation when the network partitions. When that happens, an address belong to a node in one partition may be allocated to a new node. When the partition recovers, there could be address duplications.

Although the temporary addresses is only used by a node for a very short period of time, there is chance that two new nodes pick the same temporary address. Also, two new nodes can accidentally pick the same address in the range 2048-65534 at the same time while not knowing the existence of each other. The algorithm assumes that the chance that this happens is very low. Hence, there is a finite probability that this algorithm might fail.

5.2 MANETconf: Configuration of hosts in a mobile ad hoc network

In [17], another method, MANETconf, is introduced. Every node maintains two sets, one for "allocated addresses", the other for "allocation pending addresses". When a node wants to join a MANET, it picks one of its neighbors who is already in the MANET as its address allocation proxy. The proxy then picks an address which is not in either of its two address sets. Then, the proxy floods query messages on the MANET to see if any node already uses the address. If a proxy can't be found after a number of retries, the node treats itself as the first node in the network and gives itself a random IP address.

Upon receiving a query message, every node adds the address to its “pending” set. If the address already exists in either of the two sets, the node responds to the query message with a NO message Otherwise, it sends a YES message. After the proxy has collected YES messages from all the nodes in its “allocated” set, it allocates the address to the new node. Otherwise, the proxy picks another address and tries it again. After the address has been given to a new node, the proxy floods another message to the network telling every node that the address has been allocated. On receipt of this message, every node moves the address from the “pending” set to the “allocated” set. Otherwise, an address in the “pending” set is deleted when a timer expires so that other nodes can use it.

If the new node moves away from the proxy before the address allocation finishes, the new node picks another proxy and tells it about the old proxy. The new proxy asks the old proxy for the address allocated for the new node.

When a node leaves a network, it broadcasts an AddressCleanup message, which tells every node in the network to remove its address from their “allocated address” list.

When a node leaves a network before it can broadcast the AddressCleanup message, the departure of the node is detected when another node tries to join the network. The proxy is supposed to collect YES/NO answers from all the nodes in the “allocated address” set. If no response from an address can be heard after a number of retries, the proxy broadcasts an AddressCleanup message for the address.

The method also considers how to deal with network partitioning and merging. Partitioning is easy to handle. Node in one partition just treat the nodes in the other as having left.

To deal with partition mergers, the algorithm defines a partition ID, UUID. Each partition has a unique id determined by the node with the lowest IP address. This can be determined quickly by using the allocated set. When the network partitions, each partition gets a different UUID. Whenever two nodes meet, if they both have addresses allocated, they compare their UUID’s. If the UUID’s are the same, nothing needs to be done. If the UUID’s are different, the two nodes start the merging process. A node of one partition adds the “allocated address” set of the other partition to its own “allocated address” set. If there is any conflict (two nodes share the same address), the node that has fewer TCP connections gives up its address and requests a new one.

The method is more sophisticated than the one above. However, it involves too much message flooding. The solution only works for small MANETs.

5.3 Zero-Maintenance Address Allocation (ZAL)

The essential problem in address allocation is duplicate address detection (DAD). MANETconf [17] is inefficient since each address allocation needs to be confirmed by the whole network.

Zal [12], designed by Z. Hu et al., is a mechanism to *distribute* IP addresses to wireless nodes in MANET's. The method requires very few DAD operations thus the control overhead can be very low. The basic idea is starting from a initial node in the network, which owns the complete address block used by the network, an address block is split and given to new nodes. Each new node picks one address from its address block for itself.

When a node joins a network, it asks its neighbors for addresses. Upon receipt of such a request, a node splits off a piece of its address block to the new node. If a new node gets multiple address block offers, it picks the largest one. Since this is like a binary splitting, an address block of 2^n address can be used for nodes at most n hops away from the initial node. If a node is too far away to get a share of the initial address block, an address from a global temporary address pool is used. As soon as the node meets a neighbor that still has unallocated addresses, it gives up the temporary address and ask the neighbor for addresses.

When multiple nodes pick addresses from the global temporary address pool, address duplication can happen. This problem can be solved by methods detecting address duplication. Since address duplication can only happen on this global temporary address pool, the chance this happens is low. When node moves, a node using a temporary address may meet a node still having addresses to share and get a permanent address.

To deal with the unfair address allocation among the nodes in a network which leads to fast address depletion, a distribution equalization algorithm is designed to distribute excess address space from a node to neighboring nodes. A node measures the total size of the address blocks owned by its immediate neighbors. If the total size is less than a threshold, the node gives a fraction of its address block to the neighbor with the least number of addresses in such a way that the total size can reach the threshold. This process requires only one hop communication.

To deal with network partition merger, the protocol also gives each partition an id, the partition id of a network is determined by the initial node of a network. When two nodes meet and they have the same partition id, nothing needs to be done. If they have different partition id's, the protocol converts node addresses of the smaller network to addresses belonging to the larger network. Starting from the border nodes, the nodes belongs to the smaller network return their address to neighbors still in the smaller network and request address blocks from the bigger network. Eventually, all the nodes in the smaller network get new addresses.

The strength of this protocol is its simplicity, but it has a few problems. First, the address usage of this protocol is not efficient. The address spaces required by a network increases exponentially with the increase of the diameter of the network. Second, an initial node is required. A leader election method might be necessary when there are multiple nodes starting the network at the same time. The paper didn't mention this. Third, a problem in MANET Address Allocation is address leaking. It means some allocated addresses never get recovered when nodes owning them crash or suddenly leave so that they have no chance to return the addresses to their neighbors. The protocol only assumes that the chance that this happens is low.

5.4 Conclusion

The following table compares the protocols introduced in this section. The current solutions are either not reliable or too expensive. None of them can support large networks. Optimization methods that have been used by MANET routing protocols to reduce overheads can also be applied to address allocation since it also need network wide information flooding.

Name	Overhead	Address Leakage	Address duplication	Partition Merging
IAAC	Low	Low prob.	Medium prob.	Not supported
MANETconf	High	Low prob.	Low prob.	Supported
ZAL	Low	High prob.	Low prob.	Supported

6 Model Checking

There are a number of methods that can be used to test whether a system satisfies a set of properties or not: testing, simulation, theorem proving and model checking. Testing requires test cases and implementations of a system. Thus it needs a lot of manual works. Simulation can be done for a system using an abstract model. It is closely related to model checking. Theorem proving is time-consuming and hard to automate.

The idea of model checking is to create a high level model for a system. A model checker traverses every single possible execution path for the model. While traversing, the model checker checks the states against a set of logic formulas. If a formula evaluates to false, a counter-example, consisting of the current execution path, is returned. Otherwise, an answer *true* is returned to show that, for the executions checked, the model satisfies the formulas.

The benefit of model checking is that, except for preparation of the model, it is completely automatic and usually very fast. Also, a complete implementation of a system is not necessary. An abstract model of the system may suffice.

The main disadvantage of model checking is the problem of "state explosion". That is, for large systems using complicated data structures and having many components, the size of the model can be too large when the transition relations are expressed using adjacency lists. Also, traversing every single path in a model to verify a set of properties can take exponential time.

In this section, we talk about three solutions, the first is basic model checking using temporal logic. The second uses a novel representation to reduce the size of models. The third uses a randomized algorithm to check models in order to save time.

6.1 Temporal Logic Model Checking

Chapters 3, 4 in [4] introduce the traditional way of model checking using temporal logic. A model M is defined as a Kripke structure $M = (S_0, S, R, L)$, where S_0 is the set of initial states, S is the set of states in the model, $R = S \times S$ is the set of transition relations, L is the set of properties satisfied by each state.

A temporal logic formula is a formula composed of temporal logic operators and propositional logic formulas. In temporal logic, A and E are called path quantifiers, which specify how paths from the current state satisfy a property. A means for all paths. E means there exists a path. F(eventually), U(until), G(globally) and X(next), etc. are called temporal operators, which specify how a single path starting from the current state satisfies a property. For example, $s \models AG(x = 1)$ means for all paths starting from a state s, the property $x = 1$ is satisfied on every following state. Another example, $s \models EX(f_1 \vee f_2)$ means there exists a path starting for a state s, on which the next state that satisfies either formula f_1 or f_2.

Computation Tree Logic (CTL) are those temporal logic formulas in which each temporal operator must be preceded by a path quantifier (A or E). For example, EX, AX(AU), etc. CTL evaluates the properties for the execution paths starting from a state. There are also LTL formulas which only evaluate the properties of a single execution path.

Temporal logic model checking is a labeling algorithm for the states in a model. First, for all the atomic propositional formulas that are in the temporal logic formulas to be tested, mark each state with those formulas that the state satisfies. Then, based on the result, using a combination of routines like CheckEU, CheckEX, CheckEG, mark each state with the properties it satisfies. The three routines can handle all CTL formulas. For example, CheckEX tests whether there exists a path starting from a state such that the next state on the path satisfies a formula. The method suffers from state explosion.

6.2 Symbolic Model Checking

In [3], Clarke reviews the model checking methods and focuses on the implementation of model checking using OBDDs. In 1987, McMillan found that logic formulas and transition relations in large systems can be expressed symbolically using *ordered binary decision diagrams* (OBDDs). Using this representation method, he was able to check systems with over 10^{20} states. Since then, the number has increased to over 10^{120}.

A formula's Binary Decision Tree (BDT) is a tree representation of the formula's truth table. On the BDT, each path from the root to a leaf represents a truth setting for the ordered list of all the boolean variables and the leaf node is labeled with the truth output. OBDDs is a compact way to represent boolean formulas. An OBDD is a Directed Acyclic Graph (DAG) derived from a BDT using an algorithm that removes all redundant vertices and edges from the BDT. The order of the variables appearing in the BDT is important to the final size of the OBDD. Variables appearing close to each other in the formula should also be close each other in the BDT.

Multiple formulas can be represented by a single multi-root OBDD if there are shared subgraphs in the OBDDs for the formulas. Using this method, more space can be saved. In addition, if two formulas share the same set of paths in such an OBDD, they are equivalent.

A model (S_0, S, R, L) can be encoded by using three types of OBDDs. S_0 and S are two sets of assignments of the state variables. The assignments can be encoded using a truth table. Therefore, S_0 and S can be represented by an OBDD of the truth table. The propositional logic part of L can also be represented by truth tables. Each state can thus be mapped with a set of OBDDs of the properties satisfied by it. R is a set of the state variable assignments of the state pairs that have transition relations. The assignments can still be encoded into truth tables and OBDDs. For example, state $(a,\neg b)$ have a relation to state(a,b) and the state variables are a and b. We need to construct an OBDD for formula $a \wedge \neg b \wedge a' \wedge b'$. a' and b' are used to express a and b in the successor state.

Model checking on a model represented by OBDDs operates on the sets of states and transitions instead of on individual states and transitions. To do this, we use fixpoint characterization of temporal logic operators.

Definition: A set $S' \subseteq S$ is a fixpoint of a function $\tau : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ if $\tau(S') = S'$.

Here τ is called a *predicate transformer*. τ is monotonic if $P \subseteq Q \Rightarrow \tau(P) \subseteq \tau(Q)$. When τ is monotonic, there is a least fixpoint ($\mu Z.\tau(Z)$) and a greatest fixpoint ($\nu Z.\tau(Z)$) for set Z.

$\mu Z.\tau(Z)$ can be found by recursively calculating $\tau(Z)$ from $Z = \phi$, where ϕ is the empty set. $\nu Z.\tau(Z)$ can be found by recursively calculating $\tau(Z)$ from $Z = S$, where S is the complete set of states.

For every Computational Tree Logic (CTL) operator combination, we have a fixpoint characterization equation that converts the problem of looking for states satisfying a formula into a problem of looking for least fixpoints and greatest fixpoints on the model with a *predicate transformer* τ. For example, we have $EGf = \nu Z.f \wedge EXZ$, where Z is a set of states.

Using the equation, the problem of finding all the states from which there exists a path on which every state satisfies f becomes the problem of looking for a greatest fixpoint starting from set S using $\tau(Z) = f \wedge EXZ$. EXZ can be solved trivially by checking the OBDD of the transition relations. For each state in Z if there exists a next state such that the OBDD of Z leads to true, the formula EXZ save the state in the output set.

This method does every computation using OBDD operations, reducing both the time and space complexity.

6.3 Monte Carlo Model Checking

In [8], Radu Grosu and Scott Smolka introduce the Monte Carlo Model Checking method, MC^2. Monte Carlo methods are the set of randomized algorithms that can find an answer quickly with a certain confidence ratio, the probability that the answer is correct.

Büchi automata is a set of finite automata accepting infinite length traces. To accept infinite traces, every path in a *Büchi* automaton ends with a *lasso* (a loop). If the *lasso* contains at least one accepting state, it's called an accepting *lasso*. Otherwise, a rejecting *lasso*. A trace ending up in an accepting *lasso* is accepted by the automaton. A trace ending up in a rejecting *lasso* is rejected by the automaton.

With the specification of a system S, a *Büchi* automaton, namely B_S, with every state marked as accepting can be created. With the temporal logic formulas specifying the properties the system needs to satisfy, namely φ, another *Büchi* automation $B_{\neg\varphi}$ can be constructed. This automaton only accepts traces that don't satisfy the formula φ.

Now, the model checking on the automaton can be reduced to checking if the intersection of the two automata above, $B_S \times B_{\neg\varphi}$ doesn't contain any accepting *lasso*. If there exists such an accepting *lasso*, it means there exists an executing trace for the system that can be accepted by the *Büchi* automaton $B_{\neg\varphi}$. Such a trace is reported as a counterexample.

A modified Depth First Search (DFS) algorithm can be used to traverse every single possible execution trace to look for accepting *lasso*s. If no such *lasso* can be found, the algorithm gives answer *true*. Otherwise, the accepting *lasso* is returned as a counterexample.

Using randomized algorithms, we can just retry the search for a number of traces and make a conclusion with a certain confidence ratio. Grosu et al. find out that using two parameters ϵ and δ , MC^2 takes $N = ln(\delta)/ln(1-\epsilon)$ random samples (random walks ending in a cycle, i.e *lasso*s) to get a result with error probability less than δ. ϵ is the estimated maximum *Bernoulli* probability that a random walk results in an accepting *lasso*.

The paper also includes optimizations methods that can reduce the space complexity of the algorithm by doing on the fly automaton generation. Hence, the time complexity of the algorithm is $O(N*D)$ and space complexity is $O(D)$, where D is B's recurrence diameter, N is the number of retries. Experimental results demonstrate that MC^2 is fast, memory-efficient, and scales well.

6.4 TIOA language

The Timed Input/Output Automaton (TIOA) [6] is a formal language designed by Dilsun K. Kaynar, Nancy Lynch et al. from Massachusetts Institute of Technology, Boston. It can be used to model distributed systems with timing constraints as groups of interacting state machines which can be composed together. Systems can be modeled with different levels of details for the specific purpose of model checking. Properties a system needs to satisfy are defined as invariants that can be checked during simulations. A simulation toolset for the TIOA language, “Tempo”, has been developed. Using the TIOA language, we have developed a model for a MANET channel and a model for the JAVeLEN protocol. In the appendix, the TIOA model of the wireless channel is given.

6.5 AGATE Tool Suite

Another tool that will be used in my research is Agate, a tool suite developed by our group to generate state machines from network traces. Currently, the tool suite accepts libpcap traces. It reads protocol definitions, creates protocol stacks, reads traces and stores the packet content into a database. Using the database, graphic state machines and TIOA models relating to a specific property can be generated from the trace.

7 Summary

In this report, routing protocols in different categories have been described. MANET routing is a very popular research topic nowadays. Many routing protocols have been designed. Due to the limited use of real MANET's, the protocols are largely untested in the real world.

For small networks, existing protocols like DSR, AODV, OLSR are popular although optimization can still be done to improve the power consumption of the protocols. For large sized MANETs, the solution may be to divide the network into hierarchies and use a combination of proactive and reactive routing protocols. Otherwise, the routing overhead itself can congest the network. Geographic routing can reduce routing overhead. However, a real world network is not as ideal as assumed by the existing research. Location services incur additional overhead. GPS service may not be available everywhere. Obstacles may lie between wireless nodes. More work need to be done in this area. Due to the limited deployment of MANETs, most of the MANET routing protocols are only evaluated by simulation. How these routing protocols would behave in real implementations is unknown.

The second reviewed area is address allocation. This is a important problem when a MANET is big and dynamic. Current solutions all have the problem of address duplication and address leaking. They are still either too costly or too unreliable.

The third area of the report is model checking. The knowledge of modeling checking methods facilitates the checking of the protocols in the first two areas. In my research, properties of MANET routing and address allocation protocol will be investigated. Some protocols will be designed or modified, modeled by TIOA and finally evaluated using Monte Carlo model checking.

References

[1] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu and J. Jetcheva, *A performance comparison of multi-hop wireless ad hoc network routing protocols*, Proc. Mobicom, 1998, p85-97.

[2] T. Clausen, P. Jacquet, *Optimized Link State Routing Protocol (OLSR)*, RFC 3626, IETF Network Working Group, October 2003.

[3] E.Clarke, O. Grumberg and D. Long, Verification Tools for Finite State Concurrent Systems, A Decade of Concurrency-Reflections and Perspectives, volum Springer-Verlag, Noordwijkerhout, Netherlands. p124–175, 1993.

[4] Edmund M. Clarke, Orna Grumberg and Doron A. Peled, *Model Checking*, MIT Press, January 2000, ISBN-10: 0-262-03270-8

[5] Lillian Dai and Prithwish Basu, *Energy and Delivery Capacity of Wireless Sensor Networks with Random Duty-Cycles* Proceedings of IEEE International Conference on Communications (ICC 2006), Istanbul, Turkey, June 2006.

[6] Stephen J. Garland, *TIOA User Guide and Reference Manual*, Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology September 30, 2005

[7] Mario Gerla, Yeng-Zhong Lee, Biao Zhou, Jason Chen and Antonio Caruso, *"Direction forwarding for highly mobile, large scale ad hoc networks*, MedHocNet 2005 Porquerolles, FR June 21-24, 2005

[8] Radu Grosu and Scott A. Smolka, *Monte Carlo Model Checking*, Proceedings of TACAS 2005: Eleventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems, April 2005.

[9] P. Gupta and P. Kumar, *Capacity of wireless networks*, Technical report, University of Illinois, Urbana-Champaign, 1999.

[10] Zygmunt J. Hass, *A new routing protocol for the reconfigurable wireless networks.* In Proc. of the IEEE Int. Conf. on Universal Personal Communications. October, 1997.

[11] Zygmunt J. Haas, Marc R. Pearlman, Prince Samar, *The Zone Routing Protocol (ZRP) for Ad Hoc Networks*, IETF MANET Working Group, INTERNET-DRAFT, July, 2002

[12] Zhihua Hu and Baochun Li, *ZAL: Zero-Maintenance Address Allocation in Mobile Wireless Ad Hoc Networks*, in the Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005), pp. 103-112, 2005.

[13] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, L. Viennot, *Optimized Link State Routing Protocol for Ad Hoc Networks* Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings.

[14] David B. Johnson, David A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, Kluwer Academic Publishers, Mobile Computing, volume 353, 1996

[15] Brad Karp, H. T, Kung, *GPSR: greedy perimeter stateless routing for wireless networks* Proceedings of the 6th annual international conference on Mobile computing and networking, Pages: 243-254, 2000

[16] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger and Robert Morris, *A scalable location service for geographic ad-hoc routing*, Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom 2000)

[17] S. Nesargi and R. Prakash, *"MANETconf: Configuration of hosts in a mobile ad hoc network"*, Infocom 2002

[18] Vincent D. Park and M. Scott Corson", *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*, Infocom, 1997, p1405-1413.

[19] Guangyu Pei, Mario Gerla, Tsu-Wei Chen *Fisheye State Routing in Mobile Ad Hoc Networks*, ICDCS Workshop on Wireless Networks and Mobile Computing, D71-D78, 2000

[20] G. Pei and M. Gerla and X. Hong, *LANMAR: Landmark Routing for Large Scale Wireless Ad Hoc Networks with Group Mobility*, in Proceedings of IEEE/ACM MobiHOC 2000, Boston, MA, Aug. 2000, pp. 11-18.

[21] Charles E. Perkins, Pravin Bhagwat, *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers* "ACM SIGCOMM 1994 Conference on Communications Architectures, Protocols and Applications

[22] Charles E. Perkins, Elizabeth M. Royer, *Ad hoc On-Demand Distance Vector Routing.* Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pp. 90-100.

[23] Charles E. Perkins, J. T. Malinen, R. Wakikawa, E. M. Belding-Royer, and Y. Sun, *IP Address Autoconfiguration for Ad Hoc Networks*, draft-ietfmanet-autoconf-01.txt, Internet Engineering Task Force, MANET Working Group, July 2000.

[24] Jason Redi, S. Kolek, K. Manning, C. Partridge, R. Rosales-Hain, R. Ramanathan and I. Castineyra *JAVeLEN C An Ultra-Low Energy Ad hoc Wireless Network* BBN Technologies, Cambridge, MA 2006

[25] Prasun Sinha, Raghupathy Sivakumar, Vaduvur Bharghavan, *Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures* p1763-1772 Infocom 2001.

[26] Raghupathy Sivakumar, Prasun Sinha, Vaduvur Bharghavan" *CEDAR: a Core-Extraction Distributed Ad Hoc Routing Algorithm* Infocom 1999, p202-209.

[27] Vladimir Shurbanov, Jason Redi, *Energy-Efficient Flooding In Mobile Ad-Hoc Networks* BBN Technologies, Cambridge, MA 2002

[28] Kaixin Xu, Xiaoyan Hong, Mario Gerla *Landmark routing in ad hoc networks with mobile backbones* Journal of Parallel and Distributed Computing archive Volume 63 , Issue 2, Special issue on Routing in mobile and wireless ad hoc networks, Pages: 110 - 122, 2003

[29] Hongbo Zhou. *A survey on routing protocols in MANETs.* Technical report, Department of Computer Sciences, Michigan State University, MSUCSE-03-8, 2003

A TIOA model of the wireless channel

```
vocabulary Packets
    types Packet tuple[source, dest: Nat, pos_x, pos_y: Real]
end
automaton Channel(ttl: Real, thresholdR: Int) imports Packets
    signature
        % node_id: the sender
        input  send(p:Packet)
        % node_id: the reader
        output read(p:Packet, node_id: Nat)
        % node_id: the node's id
        input  register(node_id: Nat, pos_x, pos_y: Real)
        % discards packets
        internal discard
    states
        ttl: Real := 1,
        thresholdR: Int:=1,
        inQueue: Seq[Packet] := {},
        timeQueue: Seq[Real] := {},
        flags: Array[Nat, Bool] := constant(true),
        positions_x: Array[Nat, Real],
        positions_y: Array[Nat, Real],
        now: Real := 0,
        % the set of all the nodes in the network
        dests:Set[Nat] := insert(4,insert(3,insert(2,insert(1,{})))),
        % temporary holder for a packet
        tempP: Packet
    transitions
        % receive a packet from a node
        input send(p)
        eff
            inQueue := inQueue |- p;
            timeQueue := timeQueue |- now;
            % allow some nodes to receive depending on the packet's destination
            % broadcast or unicast
            if(len(inQueue) = 1) then
                if p.dest = 0 then
                    for j:Nat in dests do flags[j] := false od
                else
                    flags[p.dest] := false
                fi
            fi;
```

```
% remove a packet from the buffer
internal discard
pre now = head(timeQueue)+ttl
eff
    inQueue := tail(inQueue);
    timeQueue := tail(timeQueue);
    tempP := head(inQueue);
    % allow some nodes to receive depending on the packet's destination
    % broadcast or unicast
    if len(inQueue) ~= 0 then
        if tempP.dest = 0 then
            for j:Nat in dests do flags[j] := false od
        else
            flags[tempP.dest] := false
        fi
    fi;
% send a packet to a specific node when there is no
% conflict, otherwise ignore all packets
output read(p, node_id)
pre
    flags[node_id] = false /\ now < head(timeQueue) + ttl
      /\ p = head(inQueue) /\  ((p.pos_x - positions_x[node_id])
        **2) + ((p.pos_y - positions_y[node_id])**2)  < thresholdR
eff
    flags[node_id] := true;
input register(node_id, pos_x, pos_y)
eff
    positions_x[node_id] := pos_x;
    positions_y[node_id] := pos_y;
trajectories
    trajdef clock
    stop when head(timeQueue) + now = ttl
    evolve d(now)=1
```