Survey of Ad Hoc Network Routing Protocols
Adhoc Networking

May 20, 2007
Professor Kaminsky
Team Adhocracy:

- Jason Winnebeck
- Benjamin Willis
- Travis Thomas

Table of Contents

Description....3
DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks....4
Caching and MultiPath Routing (Champ)....6
Greedy Perimeter Stateless Routing (GPSR)....6
Design....7
- Core Simulation Design....8
- Network Data Design....8
- Node Design....9
- Routing Algorithm Design....10
  - Dynamic Source Routing....11
  - Greedy Perimeter Stateless Routing....11
  - Caching and Multi-path Routing Protocol....11

User Manual....11
- Run....12
  - Compile....12
  - Input File....12
    - Sample....14
  - Output....15
  - Adding Additional Algorithms....16
  - Sample Tests....16

Results....16
Lessons Learned....20
Future Work....21
References....22

Description

Team Adhocracy simulated routing algorithms that claim to be good if used in an Ad hoc network. The ad hoc networking environment presents many challenges that are not present in a normal network. The first such challenge is that ad hoc networks may have quickly changing topologies. Nodes could join and leave the network as they please. Also nodes may be moving around within the network so a path through them to a destination one minute may no longer work the next.

We decided to make our own simulation that we could run and test our algorithms on. The simulator we decided to build was a discrete event simulator. Each routing algorithm implements an algorithm that will be called from the simulator when an event is ready to be processed by a node that is running that routing algorithm. This interface includes fours functions. The first function is arrived this function is called whenever a packet has been routed to the node that is being called. Terminate is called when a packet has reached its destination node. Overheard is called when a packet is in range and is able to overhear a packet it will then be able to gather information that was not directly intended for it but that it may want to hear as well. NewPacket is called when the packet is "being" started at a node in the network.

We decided to implement three routing algorithms. The first was DSR which is an algorithm that has been around for some time. It was originally developed for use of more traditional networks. Many ad-hoc routing protocols use it as a baseline for their results. Many researchers in the field also seem to believe that source routing is the best bet for Ad-hoc routing protocols.

GPSR Greedy Perimeter Stateless Routing is an algorithm that uses the locations of the nodes that it know from some outside means to allow it to route packets. The basic principle behind this algorithm is that it calculates the distance to the destination and then chooses a node that it can reach that is is the closest to it. When this simple strategy fails it tries to get around the void by using perimeter routing. We felt this was interesting since it is radically different then most routing algorithms that exist for more traditional networks.

Champ is an algorithm that talks a lot about how much better it is then other algorithms. It is similar to DSR in that when it gets a new packet it must send out a route request. The difference is that this path is stored a hop at a time along the shortest paths that the route requests used to get to the destination. Each node along the path will store successors on the way to the destination. When a route error occurs Champ is supposed to be able to try and correct this at the intermediate nodes.

Each algorithm was run in the same tests with the same packets being generated as well as the same node movement properties. The algorithms will then be compared against each other using

various metrics.

DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks

by David B. Johnson, David A. Maltz, and Josh Broch

The Dynamic Source Routing (DSR) protocol defines a reactive routing scheme for multi-hop wireless ad hoc networks. A reactive scheme differs from a proactive scheme in that the routing protocol only needs to do routing tasks when packets need to be sent, therefore when there is no network traffic, the wireless network is silent.

DSR is meant to work in a wireless ad-hoc environment, which has some attributes (some explicitly mentioned and some implied by the authors):

- Packet loss is not uncommon, so packet acknowledgments at a per-node level is appropriate.
- Nodes can move, but not so fast that the only algorithm that can work well is a pure flooding algorithm.
- The network size is reasonable. Specifically the important aspect is the network's diameter, which is defined by the paper as the “minimum number of hops necessary for a packet to reach from any node located at one extreme edge of the ad hoc network to another node located at the opposite extreme.” The paper states the assumption of a small diameter of 5 to 10 hops.
- The network links may be unidirectional. Not all other routing algorithms can work with unidirectional links, so this is a benefit of DSR.
- Each node has a single, unique address (therefore nodes with multiple interfaces need to pick a single address)

Some aspects of the network are not required, but DSR can operate better under a few circumstances:

- Wireless adapters can run in “promiscuous” mode so that they receive all packets, including those not addressed specifically for a particular node.
- Assumed bidirectional links allows for better caching, route reversal, and acknowledgments.

DSR is comprised of two main mechanisms:

1. Route Discovery: attempts to find a route or routes from a source node to the destination node (based on the addressing information in the packet)

2. Route Maintenance: handles problems when a neighbor become unreachable or unresponsive due to noise, movement, or power down.

In route discovery, DSR floods a route request packet through the network. As the packet passes through nodes, they append their address on the route taken by the packet so far. When the flooded packet reaches the destination node, it responds with a route reply containing the route taken by the request. The source node receives the replies and can pick the best route and optionally cache any alternative routes found.

When a data packet is sent, the source node places the entire route for the packet in the header. Each node forwards the packet to the next hop based on the information in the packet and *not* their own information.

In route maintenance, each node when sending a packet through the network, checks for positive arrival at the next hop through implicit or explicit acknowledgment. Implicit acknowledgments are possible in networks with bidirectional link-layer protocols (such as 802.11). The nodes will resend a packet a certain number of times, and if the send is not acknowledged within a certain period, a route error is sent back to the source node, which invalidates all routes with the broken link. After a route is invalidated, the source node can attempt a new route discovery.

Nodes can obtain a large amount of information by the route requests, replies and errors that travel through them or are overheard in "promiscuous" networking mode, and can cache and generate routes even if they did not initiate route discovery, to have routes to use when they do send packets, or to respond to route requests before having to reach a destination by responding from a cache.

The authors conclude based on the simulated and real-world experiments that DSR is appropriate for wireless ad hoc networks with high movement and packet loss and that the reactive nature of the protocol allows it to scale to need only overhead for tracking routes currently in use.

DSR is a very flexible protocol, but it does have a few weaknesses:

- DSR does not support multicast routing (although it can approximate this through the same flooded broadcast used in route discovery).
- The network diameter is limited because of the nature of route discovery and source routing requiring full routes to be stored in packet headers.
- The ability for DSR to recover a packet that could be lost in a route error is limited. There is a packet salvaging mechanism, but it can only attempt to salvage a packet once. Therefore,

resends at the application layer is important.

Caching and MultiPath Routing (Champ)

Caching and Multi-Path(Champ) routing protocol. Champ was designed to be a good algorithm to use in a mobile ad-hoc network where the topology of the network is changing often. This problem is caused by the very nature of ad-hoc networks. Nodes in the network may be running on batteries so they may run out of juice and leave the network a user may also simply decide to shut a device off. The device could also be attached to something that is moving so they would be moving through the network and changing the topology. Since ad-hoc networks rely on devices that are using the network to also route the messages this can be difficult for many traditional networking algorithms to handle. The authors believe that reactive protocols are showing promise in this challenging area of research. A reactive protocol is one that reacts to correct route errors as they are detected. Champ tries to take advantage of this idea.

Many algorithms including DSR are considered reactive and are able to repair their routes quickly and on the fly. The authors main contribution in this paper is the idea of caching data packets along the route. This allows the Champ algorithm to not only repair the path but also resend the packet instead of just dropping it. Champ relies on the following protocol messages and information in order to accomplish being able to do this. Nodes in the Champ protocol exchange three types of protocol messages to be able to maintain the network . Route Request messages are flooded to the network. In order to try and find a route to the destination. These route request have a lot of data that travels along the way with them including source of packet, destination of packet,

We plan to use this paper to implement the Champ algorithm in our simulation. Then compare it against GPSR and DSR and determine if this algorithm would be a good fit for an ad-hoc network like that authors claim.

Greedy Perimeter Stateless Routing (GPSR)

The approach take by the algorithm GPSR is based upon the known locations of the nodes in the network. With this advantage it utilizes two separate routing techniques to achieve an optimal path through a moving network. Also, another concern in the development of the algorithm was the ability to scale in a large network, in which it attempts to benefit again from it's location awareness.
One major assumption is made in order for this algorithm to work. That assumption is that the destination location is known. The algorithm does not provide means to identify the location of a given address, but it does suggest ways in which to get the location. These separate techniques are not

considered in the development and scalable design consideration. It is suggested that several location databases are stored throughout the network and they are connected, for replication purposes, via backbone.

In order to allow this algorithm to scale appropriately, many different considerations where evaluated. First and foremost, the size of the of all packets were kept at a minimum. Because of the stateless nature of the algorithm, all packets have, essentially, a fixed size, less the size of the payload. Next, the number of protocol only messages are kept to a minimum. Again, because of the stateless nature of the algorithm, the only protocol messages that are sent throughout the network are beacons, which identify the locations of the neighbors. Another consideration is the footprint of the algorithm on each node. Considering the variability of nodes, which act as some sort of sensor as well as router, the complexity and size is kept to a minimum with a simpler algorithm and limited storage of network data.

Packets can have modes, and the initial and main mode that is used for routing is the first part of the name, greedy. At each node, the immediate 1 hop neighbors and their locations are known. With the destination location known, a simple distance formula can be computed to find the 1 hop neighbor that is in the closer in direction to the destination than the source. If one if found, then this packet is forwarded.

There are other instances in which a greedy path cannot be found. Consider when two nodes are on the opposite side of a lake and there are no nodes in the middle, making a void. Nodes exist on the perimeter, which have a path, but the initial hop may not be closer, hence not satisfying the greedy algorithm. In such a case, the second part of GPSR's name comes into play, perimeter. A packet is changed into perimeter mode only when a greedy path cannot be found. When a packet is in perimeter mode, it evaluates it's neighbors in counter clockwise order from the line between the source and destination. This implements a right hand rule that keeps the next forwarding node selected on the inside of the voids perimeter. In a path from destination to source, continuing with the perimeter routing results in the opposite perimeter being routed. This continues until the greedy algorithm can take over once again. When a packet is continually forwarded around a perimeter, and returns again to the originator, which is stored, it will be dropped, being that the destination is unreachable.

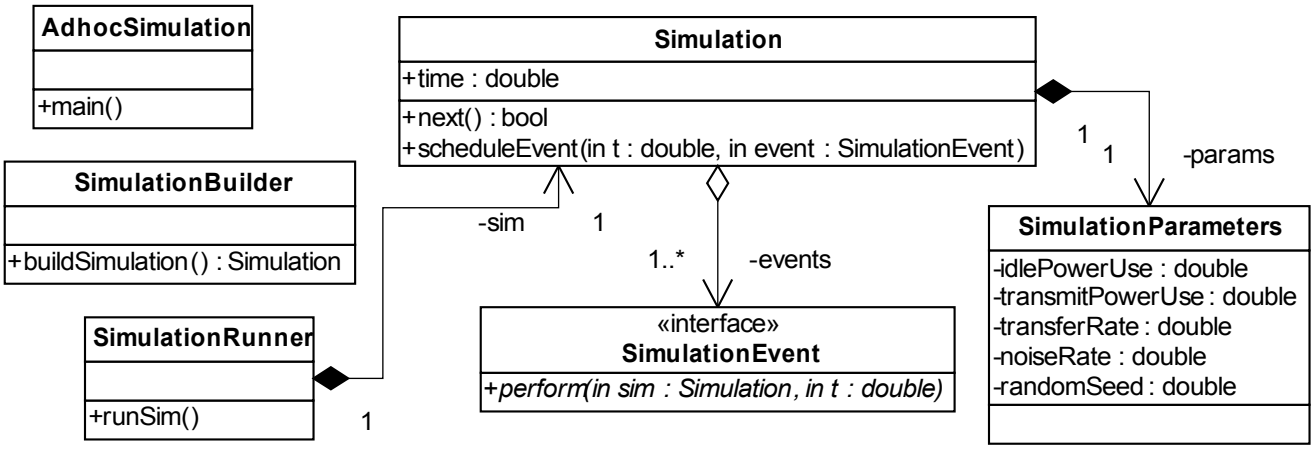
The results achieved in the research paper were compared to DSR. The results showed a higher packet delivery rate and a significantly lower protocol message overhead.

Design

The simulation is implemented as a discrete event simulation, so that it can be run in faster than real-time as well as eliminating any errors related to time discretization that might occur in a time step style

simulation.

Core Simulation Design

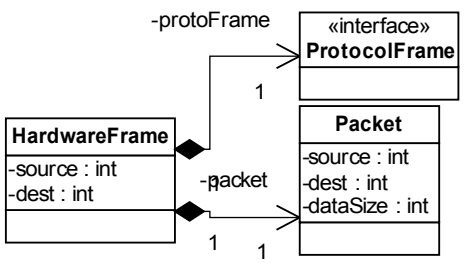


The AdhocSimulation class is the entry point for the program, which takes input and output information (as described later in the user guide). The SimulationBuilder then constructs the simulation and all objects that run in the simulation world (nodes, algorithms, location managers, etc). The SimulationRunner runs the Simulation to completion and collects the results for output to an XML file, whose format is described in the user guide.

The Simulation class itself simply implements an event loop that checks for the next occurring event, advances the simulation time to that event's time, then processes the event by calling its perform method. Any action taken in the simulation that takes time is scheduled as an event. For the most part, it is considered that any simulated computers are “infinitely” fast in that processing code does not advance simulation time. Any network interaction does take time.

Some other core classes not shown are used by the simulation builder to help set up the environment. The TrafficGenerator generates packets to be sent in the network, and a few location generators exist to place nodes in the network.

Network Data Design

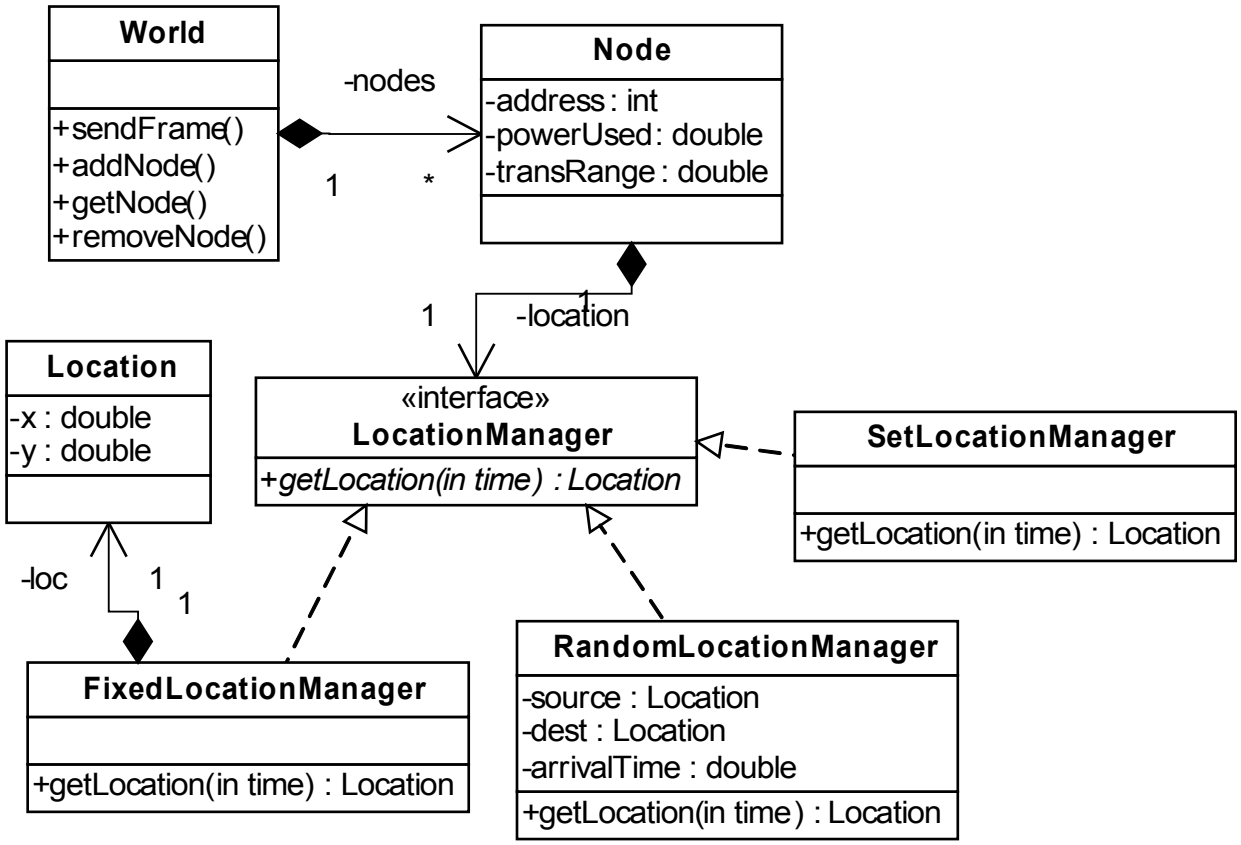


Network data is separated into three components:

1. Data link layer (HardwareFrame)
2. Network layer (ProtocolFrame)
3. Transport and above layers (Packet) “User Data”

The physical layer is simulated by the World class, which is discussed layer. The HardwareFrame and Packet behaviors are fixed and are provided by the simulation engine. The specific routing protocol algorithm implementations can create any number of ProtocolFrame implementations to carry their information. In a strict layering design, the Packet would be contained within the ProtocolFrame, but we put this object in the HardwareFrame because we felt that it would be easier for the simulation framework to monitor and handle data this way, but in a real network if we were to serialize packets, a formal layer distinction would likely be made.

Node Design



The World implements the physical layer in the simulation and keeps track of all of the nodes. When a node transmits a frame, it is sent to the World, which decides which Nodes receive that frame (a Node also receives its own frames as a loopback, after transmission is complete).

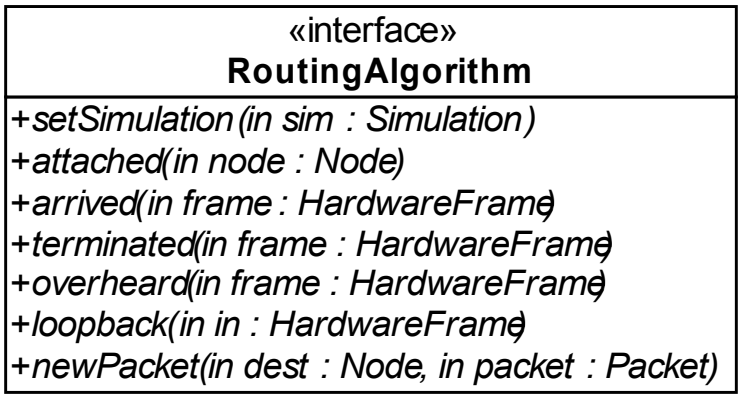
The Node serves as the central class for the simulation, keeping track of a LocationManager to manage its location and a RoutingAlgorithm (design discussed later). The Node class also collects statistics to

be sent to a NodeMonitor (not shown) which can be later harvested by the SimulationRunner to construct the outputs.

The LocationManager manages the Node's location as a function of time, and is usually implemented as a constant or a function of interpolation between points.

1. FixedLocationManager handles stationary nodes
2. RandomLocationManager handles nodes that move to a random location, pause, then start moving again
3. SetLocationManager handles a node that moves between two explicitly specified points

Routing Algorithm Design



The RoutingAlgorithm interface is implemented by all four implemented algorithms in our simulation:

1. FloodingAlgorithm floods every data packet, and is meant as a simple baseline algorithm
2. DSRRoutingAlgorithm implements DSR
3. ChampAlgorithm implements Champ
4. Algorithm (in Sim.Routing.GPRS package) implements GPSR

Each algorithm may implement some or all of the RoutingAlgorithm methods depending on what actions they wish to take.

1. **setSimulation**: Sets the simulation for this RoutingAlgorithm. This method must be called immediately after construction and before any other methods.
2. **attached**: Event called when a Node is given a RemoteAlgorithm to route packets for it.
3. **arrived**: Event when a HardwareFrame arrives destined for this Node that does not have a user Packet, or has a Packet not for this Node. The data in the frame may need to be routed.
4. **terminated**: Event when a HardwareFrame arrives destined for this Node that has a user Packet that is destined for this Node. The data has arrived at its final destination, so it does not need to be routed.
5. **overheard**: Event when a HardwareFrame is overheard by a Node (the HardwareFrame's destination is not the Node).
6. **loopback**: Event when a Node hears its own HardwareFrame (which is guaranteed to happen

after a send completes).

7. **newPacket**: Event when a Node wants to send a new Packet. The RoutingAlgorithm should bundle the Packet in a HardwareFrame and send it through the Node.

Dynamic Source Routing

The DSR paper has a large number of features. Only a very small portion are needed to for DSR to work at all, but many of them help the protocol to work better. We were unable to implement all features. The following features were implemented:

- Route Request with linear backoff resending
- Route caching on all nodes where a request or reply travels through or terminates
- Frame routing based on source path in header
- Frame retransmission with implicit ACK where possible, explicit ACK when needed
- Route maintenance and route error
- Handling of route errors in intermediate nodes and nodes who can overhear

The following features are not implemented:

- Route error propagation in discovery
- Caching of overheard route replies
- Route shortening
- Responding to route requests from route cache in intermediate nodes (mostly because without route shortening this feature added alone could worsen DSR)
- Caching of route errors

Greedy Perimeter Stateless Routing

All features from the GPSR paper were implemented. The main features included in this simple routing algorithm are the following

- Beaconing for neighbor identification
- Overheard message location caching
- Greedy algorithm shortest distance to destination
- Perimeter algorithm using counter clockwise node selection and RHR

Caching and Multi-path Routing Protocol

User Manual

The following section identifies how to compile and run the simulation, the format of the input file, the format of the output file, how to add additional routing algorithms and the sample tests that are available.

Run

A script is provided to run individual algorithms through each of 6 tests. The scripts are

- runGPSR.sh
- runDSR.sh
- runFlooding.sh
- runChamp.sh

Note: Some tests may take several minutes to run.

Compile

To compile the java project another script is provided called compile.sh. This will compile the source code for this project.

Input File

An input file is required to specify the parameters of the simulation. There are several parameters that are identified in the following tables, each table specifies a new line of the text input file.

Input File Line 1, General Parameters			
Name	**Description**	**Type**	**Sample Values**
Routing Algorithm	The algorithm that is to be loaded for simulation, available algorithms are GPSR, DSR and Champ (see sample for actual values)	String	Sim.Routing.GPSR.Algorithm Sim.Routing.DSR.DSRRoutingAlgorithm Sim.Routing.Champ.ChampAlgorithm
Nodes	The number of nodes that will exist in the simulation	Int	10 100 1000
Transmission Range	The area in meters around the nodes that they will be able to communicate	Int	1 5 10
Idle Power	The amount of power for a node to consume when it is not transmitting	Double	0.0 0.5 0.9
Transmit Power	The amount of power for a node to consume when it is transmitting	Double	1.0 5.0 10.0
Transfer Rate	The number of bits per second to transfer.	Double	1000 10000 100000

Input File Line 1, General Parameters			
Name	**Description**	**Type**	**Sample Values**
Noise	The percentage of packets that can be dropped during transmission.	Double	0.0 0.1 0.2
Duration	The total amount of time in seconds for the simulation to run	Int	100 1000 10000
Seed	A random seed to ensure that simulations with different algorithms have the same setup	Int	3 23 2342
Area	The square area (meters) of the simulation in which nodes can be contained.	Double	100.0 1000.0 10000.0
Initial Locations	Specifies whether the nodes start in a uniform grid separation (0) or are randomly (1) placed within the area.	Bool	1 0
Packet Size	The size of the packets in bits.	Int	1 8 16
Movement Speed	Speed in meters per second for a node to move if movement is specified.	Double	1.2 2.3 5.1

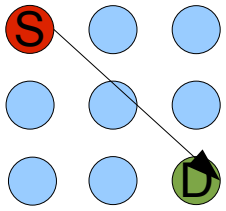
Input Table Line 2, Movement Parameters			
Name	**Description**	**Type**	**Sample Values**
Type of Movement	0 for no movement 1 for random movement 2 for fixed movement	Int	0 1 2
Node	Only for fixed movement, specifies the node address to move. Node's are addressed sequentially from 0 to total nodes – 1.	Int	32 3 21
Source X	Number coordinate that must be within the area specified in the general parameters.	Double	12 3.2 4.1 7.3
Source Y			
Destination X			
Destination Y			

Input Table Line 3, Communication Parameters			
Name	**Description**	**Type**	**Sample Values**
Type of Communication	0 for random 1 for fixed	Int	0 1
Source	Only for fixed communication, address of node to start communication from. Node's are addressed sequentially from 0 to total nodes – 1.	Int	132 3 32
Destination	Only for fixed communication, address of node to complete communication. Node's are addressed sequentially from 0 to total nodes – 1.	Int	21 3 4

Input Table Line 4, Title			
Name	**Description**	**Type**	**Sample Values**
Title	Name of the simulation to put on the output files.	String	My Great Adhoc Simulations Simulation of Death

Sample

The following is a sample of a simulation. The main features of this simulation are a uniform static grid positions with fixed communication from node 0 to node 8 (the corners).




```
Sim.Routing.Champ.ChampAlgorithm 9 2 0 1 100000 0 1000 51 400 0 8 0
0
0 0 8
Champ Test 1
```

Output

At the end of a simulation several outputs are compiled. These outputs are in xml format.

Output Table	
Xpath	**Description**
simulation/parameters/name	From input, simulation namee
simulation/parameters/noiseRate	From input, % noise rate
simulation/parameters/randomSeed	From input, seed
simulation/parameters/duration	From input, simulation time
simulation/parameters/transferRate	From input, transfer rate
simulation/parameters/transmitPowerUse	From input, transmit power
simulation/parameters/idlePowerUse	From input, idle power
simulation/parameters/totalNodes	From input, total nodes
results/packets/totalSent	Total packets send
results/packets/totalReceived	Total packets received
results/packets/loss	% packet loss
results/packets/hopsPerPacket	Average hops per packet.
results/packets/totalHops	Total number of hops for all packets sent.
results/packets/minPacketHops	Shortest hops from source to destination.
results/packets/distPerPacket	Average distance per packet sent.
results/packets/totalDist	Total distance of all packets sent.
results/packets/minPacketDist	Shortest packet distance.
results/packets/timePerPacket	Average latency for a packet.
results/packets/totalTime	Total latency for all packets.
results/packets/minPacketTime	Shortest latency.
results/frames/totalSent	Total number of frames sent
results/frames/totalReceived	Total number of frames received, not unique to a frame, multiple nodes can hear a single frame
results/frames/size	Total size of frame messages
results/power/total	Total power used
results/power/perNode	Average power used
results/transmitting/total	Total time transmitting
results/transmitting/timePerNode	Average time transmitting
results/transmitting/percent	% of time transmitting

Adding Additional Algorithms

Additional algorithms can be created for this simulation. If the algorithms appropriately extend the abstract routing algorithm, then they can be used instead of the three created for this project. Simply just add the path to the class in the input file as described above.

Sample Tests

There were six main tests implemented, 3 stationary node positions and three more adhoc situations. The following are the descriptions of the tests implemented.

1. Small network, 9 nodes, communication from one corner to another, no movement
2. Medium network, 25 nodes, communication from one corner to another, no movement
3. Medium network, 25 nodes, random communication, no movement
4. Small network, 9 nodes, random communication, random movement
5. Medium network 25 nodes, random communication, random movement
6. Large network, 600 nodes, source to destination communication, fixed movement of source node from corner to corner

Results

Our original plan was to compare three Algorithms DSR, GPSR, and Champ. We also developed a simple flooding algorithm we though would make a good baseline.

Champ was never completed and will not be compared instead we will compare DSR, GPSR and Champ. Champ was never able to complete any of the important tests with good results. It seems to be having flooding problems that have been hard to track down and correct. It does ok on static tests. Changes in the Simulation or changes to its source code seem to have made these simple tests fail as well. It is disappointing that Champ will not be able to compared to the other two algorithms we have chosen. Its route caching seemed like it might give it an advantage in this type of ad hoc network.

We ran each of the algorithms through six test. The first test was small network where there was no movement and only one node wished to send to one other node. Test 2 was a bigger network that had the same behaviors as test 1. Test 3 was a large network in which random nodes wanted to contact random other nodes. Test 4 was a large network where the nodes moved and communicated randomly. Test 5 is pretty much the same as 4 with a different random seed. Test 6 represents a vehicle tests basically it represents a vehicle traveling through a field of nodes that wants to send information to one receiver.

Illustration 1:

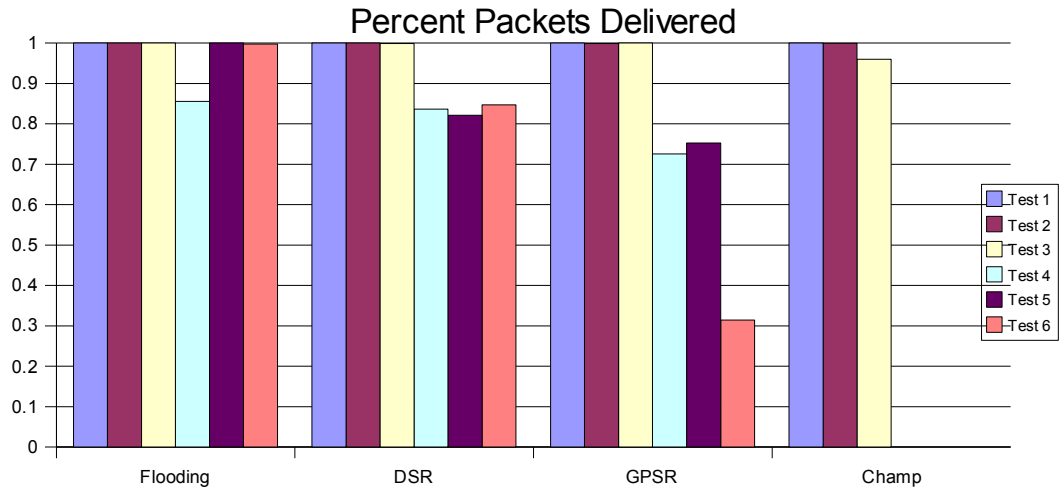


Illustration 1: shows the percentage of the packets that were delivered by the working algorithms. This is maybe the most important thing for a routing protocol. Maybe in some applications the most important would be latency. None of the three tests really sets itself above the rest here. If you really just want to have a no frills algorithm that will deliver nearly all the packets Flooding would be a good choice if you can handle the overhead.

It is not a big surprise that all the working algorithms were able to handle the first three test relatively well since none of the nodes are moving around and the routes aren't changing. All the working algorithms also seems to pass the test for Random Movement and Random communication. It was interesting that GPSR was not better at the Vehicle test it would seem to be an ideal routing algorithm for this kind of test. It seems that if you are moving knowing your location doesn't gain you as much as we thought it might.

Illustration 2: Shows the (dataDeliver*packetsize)/totalBytesSent as a percentage. This gives an indication of how efficient the algorithms were in delivering data. This could be very important for and Ad-hoc network that will not have a lot of bandwidth to spare. This really shows why the flooding algorithm really cannot be used in many networks it just adds way to much overhead. DSR really does an excellent job when the network is one source and receiver in a static network. This is expected since once it sets up a route it should be able to just keep sending data over that route. The rest of the nodes can just sit there doing nothing if they are not on the path. GPSR having the ability to know where nodes are does seem to help it keep its overhead low when nodes are moving around. This chart shows that choosing a routing algorithm for and Ad-hoc can really be influenced by the type of communication that will be done on the network.

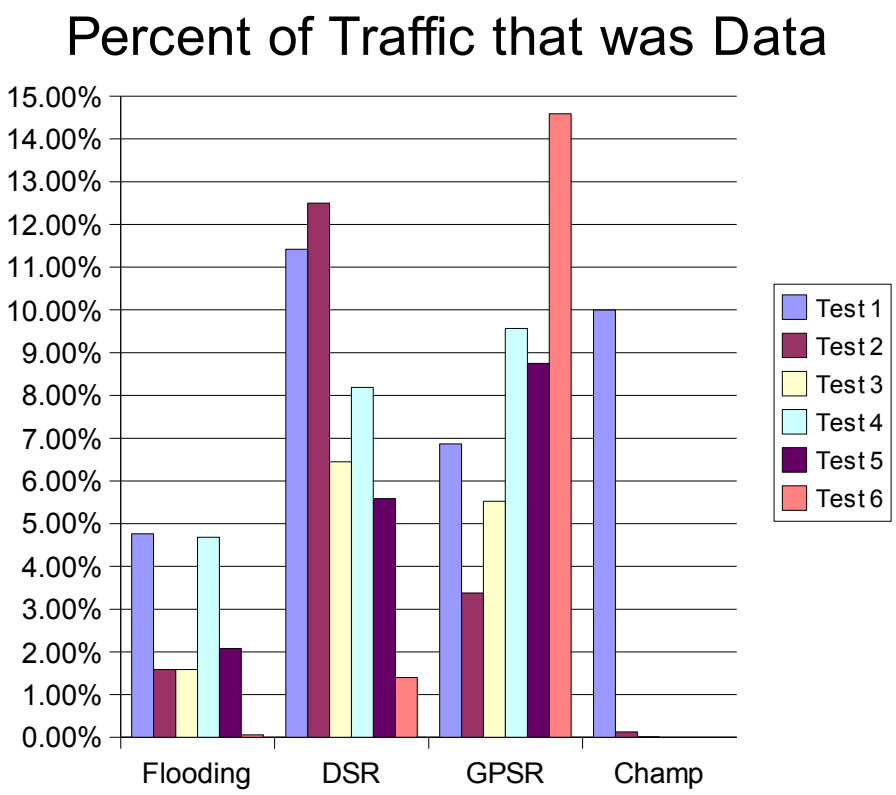
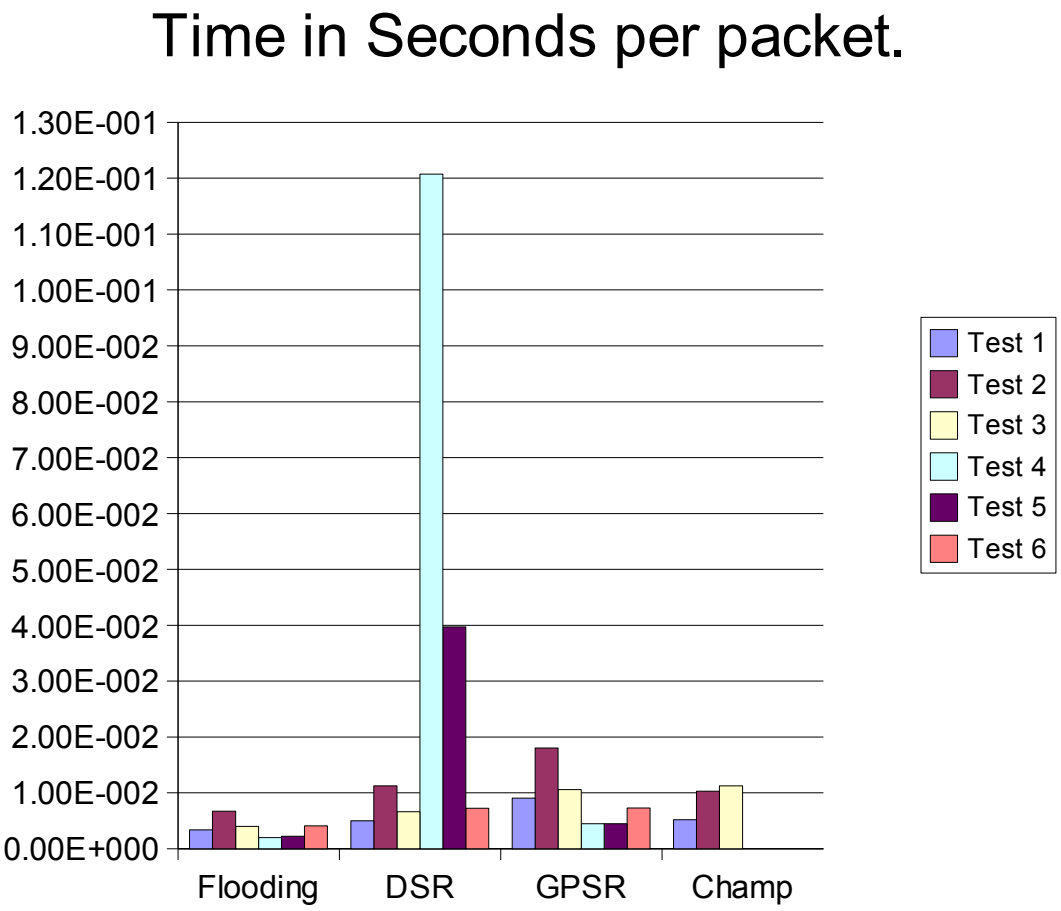


Illustration 2:

The next thing we looked at was how long the average each algorithm took to deliver a packet. The results of this are shown in Illustration 3: . It seems that once again the if you don't care about overloading the network flooding will deliver the packets the fastest. GPSR is not to far behind so this would probably be a good choice in a network that doesn't have a lot of bandwidth and has some way to find the location of each node.

The number of hops that a packet is going to travel may also be a consideration in some networks. These results are displayed in Illustration 4: flooding is not represented as it counts each copy of a packet as one and its results were off the chart around seventy hops per packet. For the first packet making it should find the minimum though since it tries all paths. It looks like for the most part this stat is a tie between DSR and GPSR they both seem to find the shortest it looks like pretty well. It looks like when nodes start to move GPSR has a slight advantage.

Illustration 3:



The results seem to show that the difference between GPSR and DSR doesn't appear to be that great. Since no algorithm has really asserted itself in all tests it seems like the important take away is that you need to know what your network to pick the best algorithm. If your network is going to have a lot of movement and you don't want to have a lot of overhead GPSR would probably be your best bet. If you don't think you will have a lot of topology changes then DSR is the best fit. Since neither one really put much distance on the other it may not make a large difference in the end though.

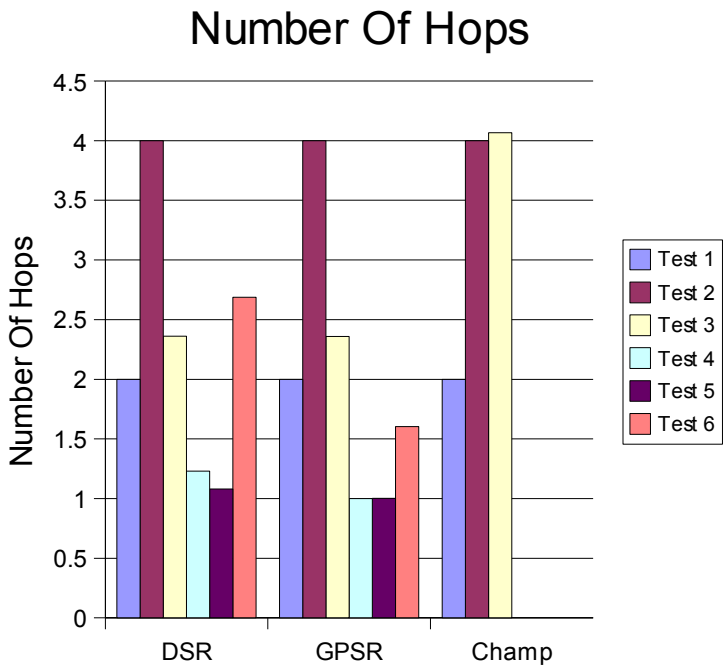


Illustration 4:

Lessons Learned

Writing a proper network simulation is difficult, for several reasons:

- How far should the simulation go in simulating the real environment? There is always a tradeoff to be drawn between implementing a more realistic simulation and the time spent on implementation. Our simulation did not simulate collisions or attempt to simulate processing time or memory used by the algorithms. Such a test might prove not useful anyway, since a real implementation on network hardware would likely not involve a high-level language such as Java and would be optimized for CPU and memory performance (our code was not).
- It is difficult to validate a simulation, because not only is the expected outcome not entirely known, but if a reasonable outcome does not occur it can be hard to tell whether it is due to a simulation that is incomplete (does not simulate every aspect of the real environment) or a bug in the algorithm. Additionally, once an algorithm works, tweaking its parameters for performance is sort of a self-fulfilling prophecy as the environment was designed by ourselves. Therefore, any results have to be taken with careful consideration as they may not entirely reflect a real execution.

However, there are some strong benefits for network simulations:

- A benefit of simulation in general is that a wide variety of planned and controlled environments can be designed and the simulation can be run much faster than "real-time," which allows us to examine the results of changing parameters much more quickly and easily than with real hardware and real networks.
- It is much easier to implement ideas and algorithms first in a simulation because of the capability of rapid development in a high-level language on hardware with ample resources, and the ability to simulate large groups of nodes without actually having hundreds of sets of networking equipment.

Future Work

There are many possibilities of future work with a simulator. First, a graphical user interface could be implemented. A GUI would be benefit to verify the algorithms are working appropriately as well as showing how the algorithms handle certain situations. Beyond physical appearance, many more environment considerations could be added. A comprehensive model for collisions should be added, especially since this is highly important within a wireless network. Enhanced power consumption can be added to accommodate longer sending ranges to consume more power.

Some algorithms were not completed in the fullest, and some were completed with less than all available features. Completing these would enhance the results of this simulation.

References

Alvin C. Valera, Winston K.G. Seah and S.V. Rao, CHAMP: A Highly Resilient and Energy-Efficient Routing Protocol for Mobile Ad hoc Networks. In Proceedings of the 5th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002), Stockholm, Sept 9-11, 2002.

Johnson, David B., Maltz, David A., Broch, Josh. 2001. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. in Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pp. 139-172, Addison-Wesley, 2001. Obtained from http://www.monarch.cs.rice.edu/monarch-papers/dsr-chapter00.ps

Karp, B. and Kung, H. T. 2000. GPSR: greedy perimeter stateless routing for wireless networks. In Proceedings of the 6th Annual international Conference on Mobile Computing and Networking (Boston, Massachusetts, United States, August 06 - 11, 2000). MobiCom '00. ACM Press, New York, NY, 243-254. DOI= http://doi.acm.org/10.1145/345910.345953