ENSC427: COMMUNICATION NETWORKS

SPRING 2011

GROUP #10

FINAL PROJECT

IMPROVED ANT ROUTING FOR WIRELESS AD-HOC MESH NETWORKS

http://www.sfu.ca/~ela6/

Jack Qaio
301025357
hqa@sfu.ca

Kwang-Young Lee
301026697
ela6@sfu.ca

Abstract

A mobile ad-hoc network (MANET) can be used to set up a temporary mean of communication by having nodes behave as hosts and routers, with packets hopping between nodes to reach the destination node. This makes MANET suitable for rapid deployment, but due to the dynamic changes in network configurations, quickly finding the optimum route is difficult. In this project, a biology-inspired ant routing algorithm with GPS assisted location awareness is used to reduce routing overhead and to improve convergence compared to an ant routing without location awareness.

1. Introduction

A MANET is an architecture-less network with nodes acting as both hosts and routers, and are usually wireless. Some can also communicate through existing fixed infrastructure. Since a MANET does not require an established infrastructure, it can be used in areas where establishing a structured architecture network is too cost-prohibitive or unavailable. Some examples include emergency disaster relief efforts, military communication, and One Laptop Per Child donation program.

Unlike other structured architecture network, a MANET's topology changes dynamically due to the mobile nature of nodes and unreliable link conditions due to the environment. Because of the dynamic topology of MANETs, a great deal of research effort is spent on developing a good routing algorithm. One of the principal requirements for a good routing algorithm is to quickly adapt to changing network topology and conditions.

One of the implementations for routing in MANET is Antnet, first developed by Lavina Jain and later updated by Richardson Lima[1]. Inspired by the foraging behaviour of ants when they search for food and the swarm intelligence to the optimized path between a food source and colony, each node in a MANET stores routing information for the neighbouring nodes and their usage as pheromone values and routes incoming packets to the path with the highest pheromone value. However, during initialization and route discovery phase, all the pheromone values are set equal and nodes route incoming packets randomly until the pheromone value converges to the optimum values.

This project proposes using another layer of information with global positioning system (GPS) during route discovery phase in order to improve the end-to-end delay and therefore converge to the optimum routing solution. Using both regular gridded network topology and random network topology, the viability and the effectiveness of using GPS information during route discovery phase when Antnet implementation cannot make a good decision based on available pheromone information will be explored.

First, some related work in MANET routing, both conventional and ant-like, will be discussed. After, the routing algorithm of GPS-assisted Antnet will be examined in detail. Lastly, some simulation scenarios and their results will be discussed and some conclusion will be drawn based on the simulation findings.

2. MANET Routing Related Work

In this section, some related work on MANET routing algorithms, both conventional and ant-inspired, will be discussed.

2.1 Dynamic Source Routing (DSR)

When a source node wishes to communicate to a certain destination node, it broadcasts route request message packets (RREQ) with the destination node address. When the neighbouring

nodes receive the RREQ message packet, they forward the packets with their address attached to the message. So until the RREQ message reaches the correct destination node, the message packet increases in size with the necessary information of the path the message packet travelled.

When the destination node receives the RREQ message packet(s), it destroys the messages, calculates best route between the destination node and the source node, and sends a replay message back to the source through the best route with the best routing information. Once the source node receives the reply message, it starts sending data packets through the best route, completing the route discovery phase.

While this algorithm scales and performs well in small MANETs with a small number of nodes, M. Bouhorma, H. Bentaouit and A. Boudhir found that when the network size is large, the routing overhead becomes larger [2].

2.2 Ad-hoc On-demand Distance Vector Routing (AODV)

The AODV is a distance vector routing for MANET [3]. When a node wishes to establish a connection to a destination node, it first broadcasts RREQ message, and if another node has a recent, up-to-date information to route to the destination, or if the message reaches the destination node, a replay message is sent back to the source node [4]. While the node replies, it creates a reverse route entry with information regarding the number of hops to the source and the address of the node where the reply message came from. A limited lifetime is assigned to entry and the entry is removed when the lifetime expires.

Unlike DSV, where the RREQ message becomes bigger as it propagates through the network, AODV uses nodes to store the necessary routing information and properly route messages through.

2.3 Ant-Inspired Routing Algorithm (ARA)

In this section, the biological ant's foraging behaviour to search for food wil be explained, then one of the ARAs, Antnet developed by Lavina Jain and updated by Richardson Lima [1], will be examined in detail. After, some previous works related to ARAs and MANETs will be discussed.

2.3.1 Biological Ant Foraging Behaviour

When an ant searches for food, it first randomly disperses from the colony until a food source is discovered. While searching, each ant leaves a trail of pheromone and it uses the trail of pheromone it left behind as a guide back to the colony, while it strengthens the existing trail by leaving more pheromone on it. Therefore, the path to the food source has stronger pheromone trail than other trails. This is the path discovery phase.

Other ants can use the stronger level of pheromone as an indication of a good path to a food source, and follow the trail while leaving their own pheromone behind. This maintains and strengthens the previously existing pheromone trail. This mechanism is also employed to finding the best path between the colony and the food source.

If two paths for the same food source with different lengths are established, assuming the pheromone levels are equal, ants will split equally to both paths. However, since one of the paths are shorter than the other path, the ants on the shorter paths will have returned back to the colony while the other ants on the longer path are still en-route. This will cause the pheromone level on the shorter path to be stronger than that on the longer path, leading more ants to the shorter path, strengthening the shorter path's pheromone level. And since the longer path is not used regularly, the pheromone evaporates, removing the pheromone trail. This allows ants to converge on the optimized path from their colony to a food source. The figure below shows the pheromone level difference between a longer path and a shorter path to the food source.

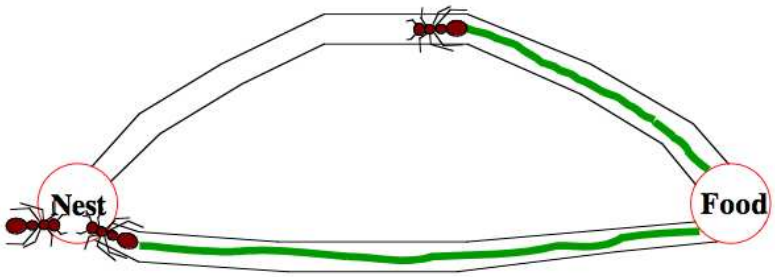


Fig 1: Biological ant's pheromone level difference between two paths with unequal length. The top path takes longer for a complete travel while the shorter path will take less.

2.3.2. Previous ARA Related Work

Inspired by the behaviour of real ants and the emergent swarm intelligence of finding the optimum path between a colony and a food source, various ant inspired algorithms were made and its applications grew wider to various computational optimization problems. One of the applications where the ant routing algorithm is used is on routing in MANETs.

D. Câmara and A. A.F. Loureiro developed a novel approach, GPS ant-like routing algorithm (GPSAL), to use GPS information to route information in MANETs while using ant-like agents to disseminate routing information between nodes and compared their algorithm to Location-Aided Routing (LAR) [5]. While this approach uses ant-like agents to populate the GPS coordinates and routing information, GPSAL mostly uses GPS information to make routing decisions and travelling packets do not affect the routing decision like Antnet does.

E. Osagie, P. Thulasiraman and R.K. Thulasiram developed improved ant colony optimization routing algorithm (PACONET) based on the foraging behaviour of real ants [6]. While their implementation is similar to Antnet, they have made a number of modifications in the route discovery algorithm to prevent packets looping and to explore all possible routes. PACONET performed similar to AODV in terms of average end-to end delay but improved in routing control overhead.

2.3.3. Antnet

In the Antnet algorithm, the general behaviour of each packet used for routing table generation can be described as follows:

Forward mode, no pheromone trail	In forward mode, the ant packet searches for the destination node. Choose random next destination that is not the parent node. Record a pheromone vector in the current node's routing table.
Forward mode, pheromone trail found	Follow the pheromone vector in the local node's routing table. If multiple possible next nodes are found, choose the next node based on the strength of its pheromone vector, where the pheromone strength determines the probability that it will be chosen. Multiply the pheromone vector that was used by a scaling factor, and reduce all other pheromone vector entries such that the total probability adds to one.
Reverse mode	In the reverse mode, the ant packet has arrived at the destination and must make its way back to the source. Follow the reverse pheromone vector in the current node's routing table. If multiple candidates are found, use a probability distribution as previously described. Scale the pheromone vector that was used.

The pheromone vectors in each node are recorded as probability values in the range from 0 to 1, the sum of which must add to one. A simple 4 node ring topology may have the following pheromone table:

For Node 1

Node 2	Node 2: 1
Node 3	Node 2: 0.5 Node 4: 0.5
Node 4	Node 4: 1

Because two possible routes are possible from 1->4, the probabilities of both are recorded in the pheromone table. Depending on network congestion and other factors, one route may be favoured over the other.

3. GPS-Assisted Antnet

3.1. Algorithm Implementation and Assumptions

For this project, the original Antnet code developed by Lavina Jain was used as a starting base. The routing algorithm was modified such that if there existed paths with pheromone values difference less than $\mathbf{P_d}$, and if it is small enough, instead of choosing the route destination randomly as Antnet would, the algorithm will compare the node's relative orientation with respect to the neighbouring nodes and the destination node for the packet, and send the packet to the neighbouring node closest to the destination node. Then the routing decision changes the node's pheromone table as normal Antnet routing would. In other cases where $\mathbf{P_d}$ is larger than the preset threshold, the normal Antnet routing algorithm based on the pheromone levels in the routing table.

Due to technical difficulties of modifying the original Antnet code to fully integrate wireless links and dynamic network topology changes, a simulated wireless network with varying delay wired links (512 Mbps, variable delay, Drop-Tail) are used between the nodes. Since it is a simulated wireless link, the distance between two nodes are used to calculate the link delay. The maximum link distance, that is the distance between two nodes, is **20m**. The minimum link delay between nodes is 5ms and the maximum link delay is 20ms. The link delay is calculated using the following equation.

$$\mathbf{delay_{link}} = 5 + \frac{20-5}{20-0} * \mathbf{X_{distance}} \quad (1)$$

For this project, the modified code was expected to improve the first routing decision based on the information provided by GPS. Therefore, the biggest difference in performance was expected during initialization and route establishment stage, where the individual routing packet sizes are small and do not carry any payload other than some routing information. Because of this, it was assumed that the throughput is not important during initialization and only the end-to-end delay was observed relative to time.

Another assumption was made such that the initialization stage is short enough such that the mobility of the nodes can be ignored and can be assumed to be stationary.

3.2. Simulation Setup

For this project, the modified GPS-assisted Antnet code is compared to the original Antnet code for possible improvements in end-to-end delay. For this, two scenarios were created. The table below shows the simulation setup.

Table 1: Simulation setup for two scenarios

Scenarios	Dimension	Number of Nodes	Node Layout
1	50m x 50m	70	Randomly Distributed
2		25	Regular Grid

In order to satisfy the GPS-assisted Antnet's assumption, the simulation time was set to 2.5 seconds, where the initialization stage, that is when all the nodes broadcast routing packets to all other nodes, started at 1.1 seconds and ended at 2.1 seconds.

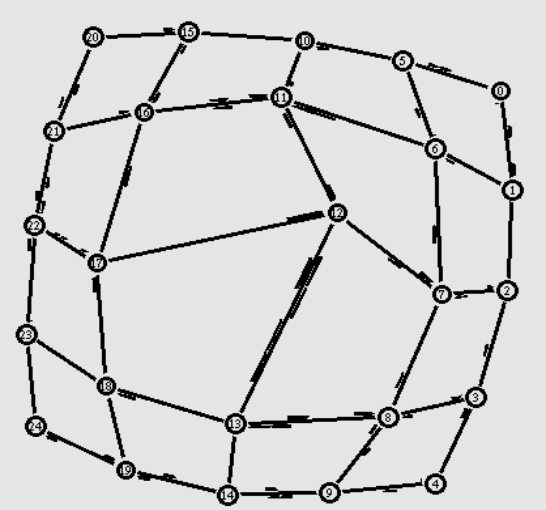


Fig 2. The grid topology

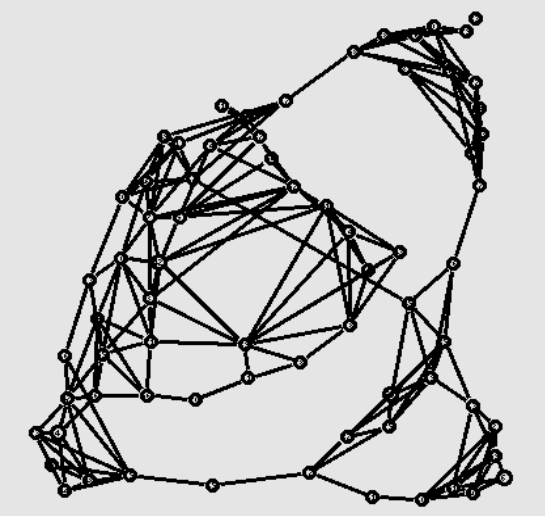


Fig 3. The random mesh topology

To maintain consistency, the random number generator used in the mesh topology is seeded with a constant value so that the same random topology is generated each time.

4. Simulation Results/Discussion

The main measure of the algorithm performance is end-to-end delay. Other metrics such as throughput and jitter were deemed not critical in gauging the performance of the routing algorithm, especially in the initialization stage where each packet does not carry any payload.

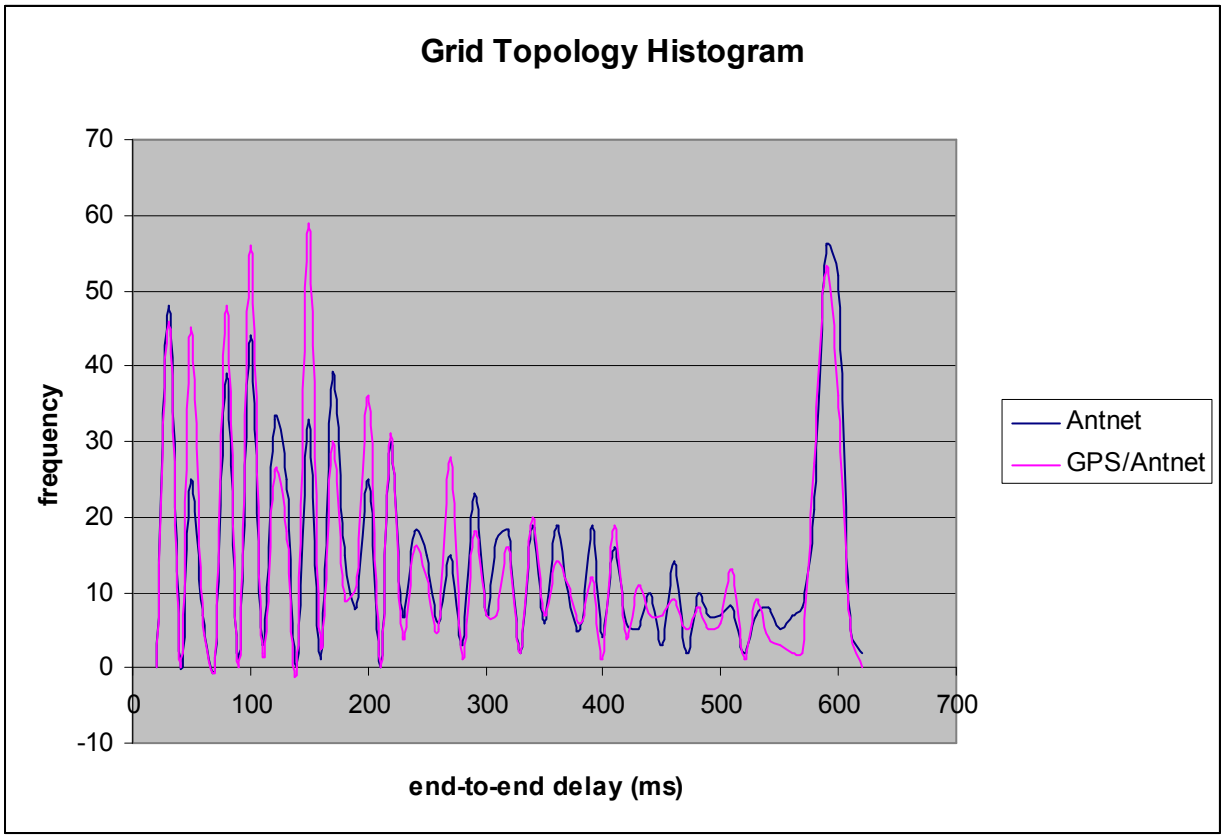
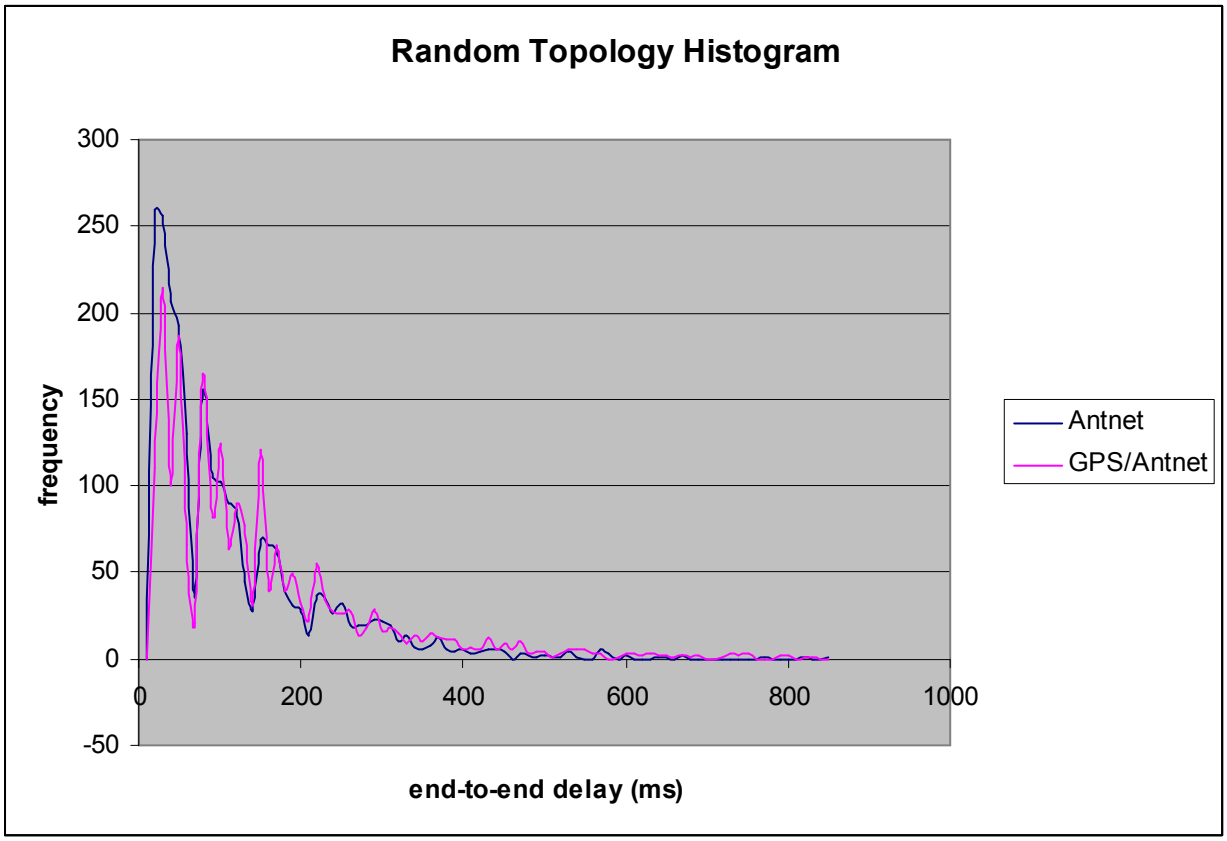


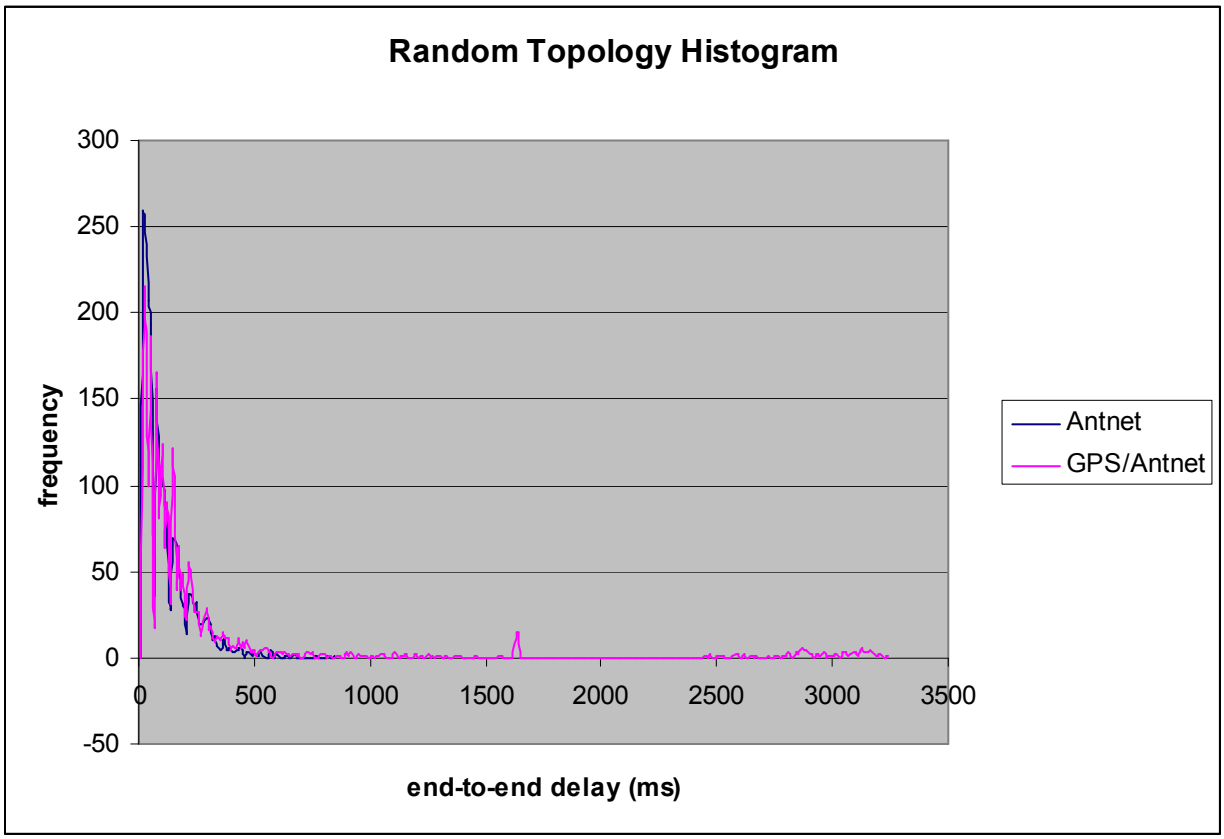
Fig4: A comparison histogram of the overall end-to-end delay for gridded gridded network

Initial results seem promising. The original Antnet algorithm ends with an average end-to-end delay of 293ms, and the GPS-assisted Antnet 271ms - A 7% improvement on the original.

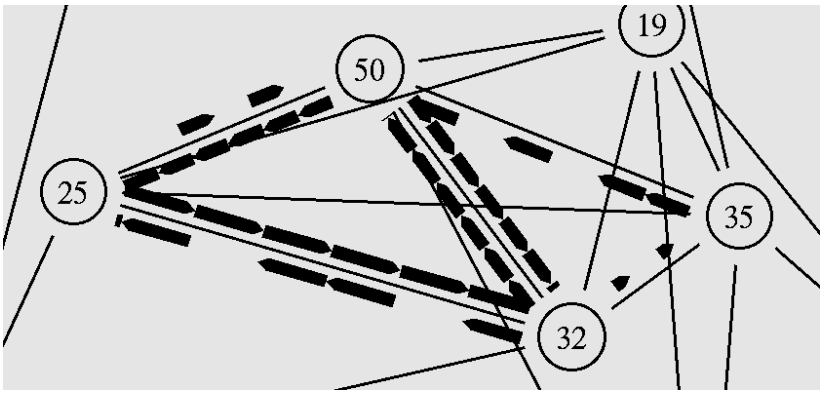
These results are consistent with our expectations, as the regular topology ensures that the GPS algorithm will always yield the shortest route to the destination. This assumption may not hold up in a more random scenario, where the ideal route may not always be the shortest, and where the GPS approach does not always yield the shortest route.

As can be expected, results were less ideal for the randomly generated topology.


Random Topology Histogram
300
250
200
150
100
50
0
-50
frequency
0
200
400
600
800
1000
end-to-end delay (ms)
Antnet
GPS/Antnet


Random Topology Histogram
300
250
200
150
100
50
0
-50
frequency
0
500
1000
1500
2000
2500
3000
3500
end-to-end delay (ms)
Antnet
GPS/Antnet

It is apparent that a small number of outliers exist for the GPS/Antnet case that exhibit anomalously large delays. While the majority of the packets in the GPS case arrive with a delay similar to that of the original Antnet case, a small number of packets are caught in a loop, resulting in large delays. Upon further investigation, a design flaw in the algorithm is discovered. When the GPS directive conflicts with an Antnet directive, a feedback loop occurs.



Eg:

At Node 1:

- no pheromone found, GPS: proceed to node 2

At Node 2:

- pheromone vector points to node 1, Antnet: go to node 1

These loops usually form in groups of 3 or more nodes, and drastically reduce the overall performance of the algorithm. Because the loop may persist for several seconds, it also causes legitimate packets to be dropped as the feedback loop consumes all available bandwidth in a single link.

5. Future Work

Although we have simulated a location-based wired network, the greatest application of this type of algorithm is in low-throughput, low-reliability wireless networks. Future work may expand the link types to such wireless protocols as ZigBee and 802.11.

The major problem of GPS/Antnet conflict is yet to be resolved, and will need to be fixed for this algorithm to be useful in real MANETs. In order to do so, some degree of communication is required between the GPS and Antnet algorithms. A possible solution is to store in each packet the addresses of a number of previously visited nodes, so that a loop cannot form and harm the performance of the network.

The current implementation of the algorithm may be useful in regular networks, where nodes are arranged in pre-designed patterns that avoid the offending corner cases that tend to generate feedback loops. In these cases a GPS device may not be necessary, as the location of each of the nodes may be programmed beforehand.

6. Conclusion

While the GPS/Antnet algorithm does not perform ideally, it is a promising idea that could be developed further to enhance the capabilities of current networking technologies.

From the data we have collected, the greatest gains in using a location-aware approach comes when a regular topology is employed. While a number of efficient routing algorithms exist for regular network topologies, none are as robust as the Antnet algorithm. Using a location-aware approach combined with Antnet may provide the best of both worlds – fast convergence during packet flooding and instant, adaptive re-routing during congestion or node failure.

As for the case of ad-hoc wireless networks, it is doubtful that the application of GPS in combination with Antnet will yield significant gains in network performance with respect to the cost of investing in the GPS devices.

7. References

[1] Richardson Lima. "Download Ns2.33-Antnet1.0 by Richardson Lima << ACO routing algorithm in practice". Internet: www.antnet.wordpress.com/2009/09/11/download-ns2-23-antnet1-0-by-richardson-lima/, Sep. 11, 2009. [Mar. 18, 2011]
[2] M. Bouhorma, H. Bentaouit, A. Boudhir. Performance comparison of routing protocols AODV and DSR, International Conference on Multimedia Computing and Systems, Ouarzazate, Morocco, Apr. 2-4 2009.
[3] M. Gunes, M. Kahmer, I. Bouazizi. "Ant-Routing-Algorithm (ARA) for Mobile Multi-hop Ad-hoc Networks - New Features and Results", Med-Hoc Net 2003 Workshop, Mahdia, Tunisia, Jun. 25-27 2003.
[4] C. Perkins, E. Belding-Royer, S. Das. "Ad hoc on-demand distance vector (AODV) routing." IETF RFC 3561. Jul. 2003.
[5] D. Camara, A.A.F. Loureiro. "A GPS/Ant-Like Routing Algorithm for Ad Hoc Networks", 2000 IEEE Wireless Communications and Networking Conference, page 1232-1236, Chicago, IL, USA, September 23-28 2000
[6] E. Osagie, P. Thulasiraman, R. K. Thulasiram. "PACONET: Improved Ant Colony Optimization Routing Algorithm for Mobile Ad-hoc Networks", 22nd International Conference on Advanced Information Networking and Applications, Ginowan, Okinawa, Japan, Mar. 25-28 2008.

Appendix I – Modifications to Antnet and ns-2

In ns2.34antnet/antnet_rtable.cc (bolded text highlights custom additions or changes)

```
//////////////////////////////////////////////////////////////////////

/// Method to implement AntNet algorithm

/// Returns next hop node address

/// Parameters:

/// - source node address

/// - destination node address

/// - parent node (to avoid loopback)

//////////////////////////////////////////////////////////////////////

nsaddr_t antnet_rtable::calc_next(nsaddr_t source, nsaddr_t dest, nsaddr_t parent) {

        nsaddr_t next, nextn;

        double thisph;

        double thisqueue;

        double thisprob;

        double maxprob = 0.0;

        double maxph = 0.0;

        double lrange = 0.0, urange = 0.0;


        // find routing table entry for destination node

        rtable_t::iterator iter = rt_.find(dest);

        double qtotal = 0.0;

        if(DEBUG)

                fprintf(stdout,"in calc_next at source %d dest %d parent %d\n",source,dest,parent);
```

```
	if(iter != rt_.end()) {

		pheromone_matrix vect_pheromone;



		//printf("\n");



		// read vector of pheromone values for the destination node

		vect_pheromone = (*iter).second;

		for(pheromone_matrix::iterator iterPh = vect_pheromone.begin(); iterPh !=
vect_pheromone.end(); iterPh++) {

			next = (*iterPh).neighbor;

			Node *node1 = node1->get_node_by_address(source);

			Node *node2 = node2->get_node_by_address(next);



			int temp_len = get_queue_length(node1,node2);

			qtotal += temp_len;

		}

		if(qtotal == 0.0) {

			qtotal = 1.0;

		}




		// calculate probability range for parent link

		lrange = 0.0;

		urange = 0.0;

		for(pheromone_matrix::iterator iterPh = vect_pheromone.begin(); iterPh !=
vect_pheromone.end(); iterPh++) {

			thisph = (*iterPh).phvalue;
```

```
		next = (*iterPh).neighbor;
		Node *node1 = node1->get_node_by_address(source);
		Node *node2 = node2->get_node_by_address(next);
		int thisqueue = get_queue_length(node1,node2);
		thisprob = (thisph + ALPHA*(1 - thisqueue/qtotal)) / (1 + ALPHA*(N-1));

		//printf("\nx: %g, y: %g", node1->meshx,node1->meshy);

		if(next == parent) {
			urange = lrange + (thisph);
			break;
		}
		lrange += (thisph);
}
if(urange == 0.0)
		urange = 1.0;

//printf("lrange %d urange %d\n",lrange,urange);
// dead end, loopback
if(lrange == 0.0 && urange == 1.0) {
//		printf("return parent %d\n",parent);
		return parent;
}

bool geolocate = false;
```

```
                double geoscale = 0.0000001;

                if(geolocate){

                        double totalphero = 0;

                        double averagephero = 0;

                        double icounter = 0;

                        vect_pheromone = (*iter).second;



                        //printf("\nsource: %d, dest: %d, parent: %d, neighbors:
",source,dest,parent); //useful for debugging!

                        for(pheromone_matrix::iterator iterPh = vect_pheromone.begin(); iterPh !=
vect_pheromone.end(); iterPh++) {

                                totalphero += (*iterPh).phvalue;

                                icounter++;

                                //printf("\t#%d",(*iterPh).neighbor); // debug code

                        }



                        if(icounter > 0){

                                averagephero = totalphero/icounter;



                                // calculate pheromone variance

                                double variance = 0;

                                for(pheromone_matrix::iterator iterPh = vect_pheromone.begin();
iterPh != vect_pheromone.end(); iterPh++) {

                                        double phero = (*iterPh).phvalue - averagephero;

                                        if(phero > variance){

                                                variance = phero;

                                        }
```

```
                }
                if(variance < geoscale){
                        Node *node1 = node1->get_node_by_address(dest);
                        printf("\nx: %g, y: %g", node1->meshx,node1->meshy);
                        // no strong pheromone detected, use location detection
                        double destx = node1->meshx;
                        double desty = node1->meshy;
                        nsaddr_t closest;
                        double shortestdistance = 100000;
                        for(pheromone_matrix::iterator iterPh =
vect_pheromone.begin(); iterPh != vect_pheromone.end(); iterPh++) {
                                Node *node2 = node2-
>get_node_by_address((*iterPh).neighbor);
                                double locx = node2->meshx;
                                double locy = node2->meshy;

                                // square roots are expensive, we only need the
distance squared for comparison purposes
                                double distance = (destx-locx)*(destx-locx) + (desty-
locy)*(desty-locy);
                                if(distance < shortestdistance &&
(*iterPh).neighbor != parent){
                                        shortestdistance = distance;
                                        closest = (*iterPh).neighbor;
                                }
                        }

                        if(shortestdistance != 100000){
```

```
                                                //printf("\nsource: %d dest: %d
next: %d\n",source,dest,closest); // outputs list of source and destinations

                                                return closest;

                                        }

                                }

                        }

                }


                // generate random probability value, out of range of parent link

                double tmp_double;

                do {

                        tmp_double = rnum->uniform(1.0);

                }while(tmp_double >= lrange && tmp_double < urange);


                // find next hop node corresponding to this range of probability

                lrange = 0.0;

                urange = 0.0;

                for(pheromone_matrix::iterator iterPh = vect_pheromone.begin(); iterPh !=
vect_pheromone.end(); iterPh++) {

                        thisph = (*iterPh).phvalue;

                        next = (*iterPh).neighbor;

                        Node *node1 = node1->get_node_by_address(source);

                        Node *node2 = node2->get_node_by_address(next);

                        int thisqueue = get_queue_length(node1,node2);

                        thisprob = (thisph + ALPHA*(1 - thisqueue/qtotal)) / (1 + ALPHA*(N-1));

                        urange += (thisph);

                        if(tmp_double >= lrange && tmp_double < urange) {
```

```
                        //printf("return next %d\n",next);
                        return next;
                    }
                    lrange = urange;
                }
            }
}
```

In *ns2.34/common/node.h*

```
        //routines for supporting routing
        void route_notify (RoutingModule *rtm);
        void unreg_route_notify(RoutingModule *rtm);
        void add_route (char *dst, NsObject *target);
        void delete_route (char *dst, NsObject *nullagent);
        void set_table_size(int nn);
        void set_table_size(int level, int csize);

        // built-in location causes segmentation fault. implement our own coordinate system
        double meshx;
        double meshy;
protected:
        LIST_ENTRY(Node) entry;  // declare list entry structure
        int address_;
        int nodeid_;                    // for nam use
```

In ns2.34/common/node.cc

```
else if (strcmp(argv[1], "add-neighbor") == 0) {
                        Node * node = (Node *)TclObject::lookup(argv[2]);
                        if (node == 0) {
                                tcl.resultf("Invalid node %s", argv[2]);
                    return (TCL_ERROR);
                        }
                        addNeighbor(node);
                        return TCL_OK;
                }
        }

        if(strcmp(argv[1], "mesh-location") == 0){
                this->meshx = atof(argv[2]);
                this->meshy = atof(argv[3]);
                return TCL_OK;
        }

        return ParentNode::command(argc,argv);
```

Appendix II – Random Topology Generator

```
# tcl script for AntNet algorithm on a pseudo-random mesh topology

# number of total nodes
set population 70

# radius of communication between each node
set noderadius 9

# minimum delay - ms (when two nodes are close together)
set mindelay 5

# maximum delay (used when two nodes are at maximum connection distance)
set maxdelay 20

# size of simulation environment
set worldwidth 50
set worldlength 50

# seed for pseudo-random generator (same mesh will result for each seed value)
set seed [expr srand(1)]

#Create event Schedular
set ns [ new Simulator ]

#Open the Trace file
set tf [open antnet_trace.out w]
$ns trace-all $tf
```

```
set nf [open out.nam w]
$ns namtrace-all $nf

#Create the nodes
for {set i 0} {$i < $population} {incr i} {
        set n($i) [$ns node]
}

#set node location using rand
for {set i 0} {$i < $population} {incr i} {
        set x [expr rand()*$worldwidth]
        set y [expr rand()*$worldlength]
        puts $x
        #save x and y in an array for later use
        set nodex($i) $x
        set nodey($i) $y

        #use custom mesh-location function to identify location
        $n($i) mesh-location $x $y

        #$n($i) set X_ $x
        #$n($i) set Y_ $y
        #$n($i) set Z_ 0.0
}

#Create links between the nodes that are within noderadius
#brute force with O(n^2)!
for {set i 0} {$i < $population} {incr i} {
        for {set j 0} {$j < $population} {incr j} {
                if {$i != $j} {
```

```
                        # square roots are expensive, so we just compare squares
                        set dx [expr $nodex($i)-$nodex($j)]
                        set dy [expr $nodey($i)-$nodey($j)]
                        set d2 [expr ($dx*$dx) + ($dy*$dy)]
                        if {$d2 < [expr $noderadius*$noderadius]} {
                                # make link delay proportional to node distance (inverse square
function)
                                #set delay [expr ($maxdelay-
$mindelay)*($d2/($noderadius*$noderadius)) + $mindelay]
                                set delay 10
                                set ms ms
                                $ns duplex-link $n($i) $n($j) 128Kb $delay$ms DropTail
                        }
                }
        }
}

#Create Antnet agents
for {set i 0} {$i < $population} {incr i} {
        set   nn($i)  [ new Agent/Antnet $i]
}

#Attach each node with Antnet agent
for {set i 0} {$i < $population} {incr i} {
        $ns attach-agent  $n($i)  $nn($i)
}

for {set i 0} {$i < $population} {incr i} {
        for {set j 0} {$j < $population} {incr j} {
                if {$i != $j} {
                        # square roots are expensive, so we just compare squares
```

```
                        set dx [expr $nodex($i)-$nodex($j)]
                        set dy [expr $nodey($i)-$nodey($j)]
                        set d2 [expr ($dx*$dx) + ($dy*$dy)]
                        if {$d2 < [expr $noderadius*$noderadius]} {
                                $ns connect $nn($i) $nn($j)
                                $ns connect $nn($j) $nn($i)
                                $ns at now "$nn(0) add-neighbor $n($i) $n($j)"
                        }
                }
        }
}

#Set parameters and start time for AntNet algorithm
for {set i 0} {$i < $population} {incr i} {
        #$nn($i) set num_nodes_x_ $sz_x
        #$nn($i) set num_nodes_y_ $sz_y
        $nn($i) set num_nodes_ $population
        $nn($i) set timer_ant_ 0.03
        $nn($i) set r_factor_ 0.01
        $ns  at  1.1  "$nn($i) start"
}

#Set stop time for AntNet algorithm
for {set i 0} {$i < $population} {incr i} {
        $ns  at 2.1 "$nn($i) stop"
}

#Print routing tables generated by AntNet
for {set i 0} {$i < $population} {incr i} {
        $ns at 4.2 "$nn($i) print_rtable"
```

```
}

# Final Wrap up
proc Finish {} {
        global ns tf
        $ns   flush-trace
        #Close the Trace file
      close $tf
}

$ns  at 8.4  "Finish"

# Start the simulator
$ns  run
```

Appendix III – Delay Data Generator (AWK script)

```
BEGIN {
	recvdSize = 0
}

{
	event = $1
	time = $2
	if (event == "+" || event == "-") node_id = $3
	if (event == "r" || event == "d") node_id = $4
	flow_id = $8
	pkt_id = $12
	pkt_size = $6
	flow_t = $5

	if(sendTime[pkt_id] == 0 && (event == "+" || event == "s")) {
		sendTime[pkt_id] = time
	}

	if(sourceNode[pkt_id] == 0 && event == "+"){
		sourceNode[pkt_id] = node_id
	}

	if (event == "r") {
		# Store packet's reception time
		recvTime[pkt_id] = time
	}
}

END{
```

```
        delay = avg_delay = recvdNum = 0
        printf("delay(ms)\n")
        for (i in recvTime) {
                delay += recvTime[i] - sendTime[i]
                recvdNum++

                printf("%g\n",(recvTime[i] - sendTime[i])*1000)
        }
}

function vint(y){
if (y < 0){
if(int(y) == y )
return int(y)
else
return int(y)-1
}
else
return int(y)
}

function abs(x){return (((x < 0.0) ? -x : x) + 0.0)}
```

Appendix IV – Delay vs Distance Statistics

```
BEGIN {
        recvdSize = 0
}

{
        event = $1
        time = $2
        if (event == "+" || event == "-") node_id = $3
        if (event == "r" || event == "d") node_id = $4
        flow_id = $8
        pkt_id = $12
        pkt_size = $6
        flow_t = $5

        if(sendTime[pkt_id] == 0 && (event == "+" || event == "s")) {
                sendTime[pkt_id] = time
        }

        if(sourceNode[pkt_id] == 0 && event == "+"){
                sourceNode[pkt_id] = node_id
        }

        if (event == "r") {
                # Store packet's reception time
                recvTime[pkt_id] = time
                dest = node_id
                if(sourceNode[pkt_id] != 0){
                        src = sourceNode[pkt_id]
                        srcx = vint(src/5)
```

```
                srcy = src%5
                destx = vint(dest/5)
                desty = dest%5
                distance = abs(destx - srcx) + abs(desty - srcy)
                if(maxDistance[pkt_id] < distance){
                        maxDistance[pkt_id] = distance
                }
        }
    }
}

END{
    delay = avg_delay = recvdNum = 0
    printf("distance, delay(ms)\n")
    for (i in recvTime) {
        delay += recvTime[i] - sendTime[i]
        recvdNum++



        printf("%g,%g\n", maxDistance[i], (recvTime[i] - sendTime[i])*1000)
    }
}

function vint(y){
if (y < 0){
if(int(y) == y )
return int(y)
else
return int(y)-1
```

```
}
else
return int(y)
}

function abs(x){return (((x < 0.0) ? -x : x) + 0.0)}
```