ROBUST ANT COLONY BASED ROUTING ALGORITHM FOR MOBILE AD-HOC NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Arush S. Sharma

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2019

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Dongsoo Kim, Chair
Department of Electrical and Computer Engineering
Dr. Brian King
Department of Electrical and Computer Engineering
Dr. Mohamed El-Sharkawy
Department of Electrical and Computer Engineering

Approved by:
Dr. Brian King
Head of Graduate Program

This work is dedicated to my parents who are the pillars of my life. Without their continued guidance and moral support, this work wouldn't have been possible. I will always be grateful to them.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Dongsoo Kim and my thesis committee members Dr. Brian King and Dr. Mohamed El-Sharkawy for their continued guidance and advice during the course of this work. Many thanks to Mrs. Sherrie for her tireless efforts in reading this thesis. Special thanks to Dr. Kim for being a good mentor and for guiding me on the right path and Electrical and Computer Engineering department at Indiana University Purdue University Indianapolis (IUPUI) for their support.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

SYMBOLS

τ	pheromone
h	hop count
α	pheromone coefficient
β	hop count coefficient
δ	energy coefficient
γ	goodness RSSI coefficient

ABBREVIATIONS

ACO	Ant Colony Optimization
SI	Swarm Intelligence
AODV	Ad hoc On Demand Distance Vector
MANET	Mobile Ad hoc Networks
MAC	Media Access Control
WSN	Wireless Sensor Networks
RSSI	Received Signal Strength Indicator

ABSTRACT

Sharma, Arush S. M.S.E.C.E., Purdue University, August 2019. Robust Ant Colony Based Routing Algorithm For Mobile Ad-hoc Networks. Major Professor: Dongsoo S. Kim Professor.

This thesis discusses about developing a routing protocol of mobile ad hoc networks in a bio inspired manner. Algorithms inspired by collective behaviour of social insect colonies, bird flocking, honey bee dancing, etc., promises to be capable of catering to the challenges faced by tiny wireless sensor networks. Challenges include but are not limited to low bandwidth, low memory, limited battery life, etc. This thesis proposes an energy efficient multi-path routing algorithm based on foraging nature of ant colonies and considers many other meta-heuristic factors to provide good robust paths from source node to destination node in a hope to overcome the challenges posed by resource constrained sensors.

1. INTRODUCTION

Wireless Sensor Networks (WSN) consist of a large number of nodes equipped with sensing capabilities; communication interfaces which has limited memory and energy resources. WSN nodes are statically deployed over large areas. However, they can also be mobile interacting with the environment. WSNs have wide spectrum of applications which includes environmental sensing, health care, traffic control, tracking of wild life animals, etc. Usually individual sensor nodes send their data towards base station node (commonly known as sink node). Intermediate nodes perform relaying of sensed data towards the destination.

1.1 Design Challenges

Following are the design challenges faced by WSNs Routing Protocols.

a) Low Computational and memory requirements

Sensor nodes are equipped with a low end CPU and have limited memory. Therefore, it is mandatory that the routing algorithm has minimum overhead to make its execution feasible and effective.

b) Self-organization

Wireless Sensor Network is expected to remain active for considerable period of time. Within that time period, few new nodes might be added to the network, while other nodes might die due to energy depletion or may become operational.

A routing protocol therefore should be robust to such dynamic and unpredictable events. It should be empowered with self-organizing properties to let the network function as an autonomous system.

c) Energy Efficiency

Nodes are equipped with small non-rechargeable batteries. Therefore, the efficient battery usage of a sensor node is very important aspect to support the extended operational lifetime of network. The routing protocol is expected to forward the data packets across multiple paths, so that all nodes can deplete their battery source at a comparable rate. This results in load balance of network and increases the network lifetime.

d) Scalability

In WSN applications, hundreds of nodes are generally deployed that have short communication ranges and high failure rates. Hence a routing protocol should be able to cope with the above challenges. It is imperative to design such routing protocols which can cater to the aforementioned challenges and are robust and adaptive. Nature provides us examples of mobile, independently working agents which seamlessly work together to perform tasks efficiently, for example, flight of migratory birds, ant colony optimization, etc. Nature inspired algorithms also known as Swarm Intelligence (SI) [1] are based on collective behavior of social insect colonies and other animal societies for solving different types of communication problems.

1.1.1 Background

Ant algorithms are special class of SI algorithms [2], which consist of population of simple agents (ants) which interact locally and with their environment. The foraging behavior of ant colony inspires Ant Colony algorithm. Initially, ants randomly wander around the nest searching for food. When the food is found, they take it back to the colony and leave a trail of pheromones on the way. Other ants can sense the pheromone concentration and prefer to follow directions with higher pheromone density. Since shorter paths can be traversed faster, they will eventually outweigh the less optimal routes in terms of pheromone concentration. Additionally, pheromones evaporate over time, so ants are less likely to follow an older path which makes them search for newer paths simultaneously. In a case where an obstacle gets in their way, ants again initiates the route discovery process by randomly selecting the next hop until the ants converge on the paths with relatively higher concentration of pheromone.

1.1.2 Overview

The remainder of thesis is given as follows. At first general introduction and challenges are given which gives us the motivating factors to pursue research. Section 2 discusses about the related work done in the field of ant routing algorithms. Section 3 describes the working of well established AODV routing protocol. The description of AODV routing protocol is vital for the readers as ACO routing algorithm is compared against AODV in later section. Section 4 gives an overview of control packet (layer 3 control message) formats. Section 5 briefly describes about traditional ant colony routing algorithm. Section 6 talks about the novel ideas implemented in ant routing algorithm. Section 7 shows the robustness of proposed ant routing algorithm against various test scenarios. Section 8 does a comparison study against AODV routing protocol. Finally we conclude by giving a short summary and an outlook on future work.

2. LITERATURE REVIEW

AntNet proposed by Di Caro and Dorigo [3] is a routing technique which is applied for best-effort IP networks. Optimizing the performance of entire network is its main aim according to the principles of Ant Colony Optimization, AntNet is based on a greedy stochastic policy, where each node maintains a routing table and an additional table containing statistics about the traffic distribution over the network. The routing table maintains for each destination and for each next hop a measure of the goodness of using the next hop to forward data packets to destination. These goodness measures, called pheromone variables, are normalized on the stochastic policy. This algorithm uses forward ants and backward ants to update the routing table. The forward ants use heuristic based on the routing table to move between a pair of given nodes and are used to collect information about the traffic distribution over the network. The backward ant stochastically follows the path of forward ants in reverse direction. At each node, the backward ant updates the routing table and the additional table which contains traffic statistics of the network.

The energy-efficient ant-based routing algorithm (EEABR) is a routing protocol for WSNs and extends AntNet proposed by Tiago Camilo et. al., [4]. EEABR tries to minimize memory requirements as well as the overall energy consumption of the original AntNet algorithm. The ants retain information of only last two visited nodes because it takes into account the size of ant packet to update pheromone trail. In the typical ant-based algorithm, each ant carries the information of all the visited nodes. Then, in a network consisting of very large number of sensor nodes, the size of information would cause considerable energy to send ants through the network. Each node keeps the information of the received and sent ants in its memory.

Each memory record contains the previous node, the forward node, the ant identification, and a timeout value. The transmission probability considers the artificial pheromone value and the remaining energy of the possible next hop.

Ladder Diffusion Algorithm proposed by Ho et al. [5] addresses the energy consumption and routing problem in WSNs. The algorithm tries to reduce the energy consumption and processing time to build the routing table and avoid the route loop. In this algorithm, the sink node broadcasts the ladder creating packet with the grade value of one. The grade value of one means that the sensor node receiving this ladder creating packet transmits data to the sink node requires only one hop. Then sensor nodes increments the grade value of ladder creating packet and broadcast the modified ladder-creating packet. A grade value of two means that the sensor node receiving this ladder-creating packet sends data to the sink node requires two hop counts. And this step repeats until all the sensor nodes get the ladder-creating packet. The ladder diffusion algorithm assures that the direction of data transfer always occurs from a high grade value to a low grade value, which means each relay is forwarded to the sink node since each sensor node records the grade value of relay nodes in the ladder table. The path decision is based on the estimated energy consumption of path and the pheromone.

Energy-Aware Ant Routing in Wireless Multi-Hop Networks proposed by Michael Frey et. al., [6] provides new mechanisms for estimating the fitness of a path and energy information dissemination thus enabling to prolong the network lifetime. The network lifetime is the time span a network can fulfill its service. Traditional Ant Routing Algorithm considers the pheromone value in its probabilistic routing decision process. This approach favours shortest paths over non-shortest paths which is not suitable for energy constrained networks. EARA extends the ant routing algorithm with an energy heuristic for determining the nodes residual energy and scheme for estimating a paths energy. EARA algorithm considers the residual energy of a node as an additional heuristic.

Since the residual energy of a node changes over time, periodic Energy Ants are released for updating the energy values in the nodes routing table. Periodic Energy Ants are sent occasionally as it can be a costly operation in terms of consumed energy.

Ant Colony and Load Balancing Optimizations for AODV Routing Protocol proposed by Ahmed M. Abd Elmoniem et al., [7] discusses about improving the AODV routing protocol by taking the Ant Colony Optimization approach. Forward ant agents are sent as a part of route establishment request to find the route to destination. This route establishment phase is very much similar to Route Request (RREQ) phase of AODV routing protocol except for the fact that if the route to destination does not exist and there exist no neighbour, then the ant is broadcasted. Otherwise, if the active neighbour exists with highest pheromone, the forward ant is sent to that neighbour. In case of destination node receiving forward ant, backward ant is sent to the source node with a route to destination which comes under the part of Route establishment reply phase. The pheromone update policy is applied on the nodes receiving backward ants. Also it is applied differently depending upon whether the node is an source node or intermediate node or is destination node. Once the source node receives backward ant, the Data transmission phase begins. Each node receiving data packets forwards it to neighbor according to the pheromone values. Neighbor node having greater pheromone receives more data than those having less pheromone which leads to load balancing. If the route does not exist at all, a Route Error (RERR) packet is sent to the source node. If the routing table entry of the destination doesn't exist in source node, it deletes the route and again initiates the route discovery process.

Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions presented by Kwang Mong Sim et al., [8] provides comparison of the approaches for solving the convergence problem in ACO algorithms. When the network reaches its equilibrium state, the already discovered optimal path is given more preference over other paths by the ants which leads to many problems such as congestion, reduction of probability for selecting other paths, network failure, etc. In order to mitigate this problem, some of the approaches include evaporation, aging, pheromone smoothing and limiting, privileged pheromone laying etc. Evaporation of pheromone is a technique to prevent the ants of favoring the older or stale paths which makes an ant to concurrently search for fresh paths. Aging refers to quantity of pheromone deposited by the ant. Older ant will deposit less pheromone compared to its young contemporary since they take more time in reaching destination. Limiting and Smoothing Pheromone refers to limiting the pheromone deposit by placing an upper bound which reduces preference of optimal paths over non-optimal paths In privileged pheromone laying, only certain ants are permitted to deposit extra pheromone. This makes the ant to converge to a solution by taking less time.

Ant-routing-algorithm (ARA) for mobile multi-hop ad-hoc networks- new features and results explored by Mesut Gunes et al., [9] is based on ant algorithms which makes it highly adaptive and efficient. The routing algorithm consists of three phases. Route Discovery Phase requires use of forward ant (FANT) and backward ant (BANT) control agents. FANT establishes the pheromone trail back to the source node. Similarly BANT establishes pheromone track back to the destination node. Node receiving FANT for the first time creates an entry in its routing table consisting of destination address which is the origin of FANT, next hop which is address of the previous node from which it received FANT and pheromone value which is computed based on the number of hops the FANT took to reach the node. The node forwards the FANT to its neighbors. Once the destination node receives FANT, it sends BANT back to the source node. Once the source nodes receives BANT from the destination node, the path is established and data packets can then be sent which comes under the Route Maintenance Phase. When data packets are relayed to destination by a node, it increases the pheromone value of the routing table entry. The last phase of ARA handles the routing failure caused by the mobility of node which are very common in MANETs. ARA assumes IEEE 802.11 on the MAC layer which enables routing algorithm to recognize the failure of route through a missing acknowledgement on the MAC layer. Node deactivates the link by setting the pheromone value to 0. The node then searches for an alternative link in its routing table. If there exist a route to destination in its routing table, it sends the packet via this path. If there exist multiple en-tries in the routing table, the node will not send any data packets. Instead it informs the source node which has to initiate the route discovery process again.

3. WORKING OF AODV ROUTING PROTOCOL

Ad hoc On Demand Distance Vector (AODV) routing protocol [10] enables multi hop routing between source and destination node. It is reactive, i.e. on-demand routing protocol where the source node(s) doesn't initiate the route discovery process unless the route to destination is required. On the other hand routing protocols which come under the category of proactive routing continues to maintain routes between source and destination even when the route to destination is not required. AODV routing protocol consists of two phases: i) route discovery and ii) route maintenance.

When a source node wishes to communicate with some destination node, it first seeks for a route in its routing table. If the route exists, then the communication between two starts immediately. If not, then route discovery process is initiated. The route discovery process consists of broadcasting a route request (RREQ) message. If one of the intermediate node receiving RREQ message has a valid route to destination, it replies back with a route reply (RREP) message. Otherwise RREQ message is broadcasted by intermediate nodes until it reaches the destination node. The intermediate node while handling RREQ message increments the hop count value in RREQ packet by one. This accounts for hop count required to reach the source node. Additionally intermediate nodes create a routing table entry which contains address of source node, total number of hops required to reach the source and the next hop's address which is IPV4 address of neighbor node from whom it received the message. Each routing table entry is associated with lifetime, i.e. if the route entry is not used within the lifetime, t will be deleted from routing table. In this way the destination node becomes aware of source node and generates RREP message.

The RREP is unicast to the next hop towards the originator of RREQ message which is indicated by routing table entry for the source node. Intermediate nodes receiving RREP packet increments the hop count field in routing table by one. When

RREP is received by source node, the hop count field represents the total distance, in terms of hops, to reach destination node. This completes the route discovery phase.

The second phase of the protocol is route maintenance. It is performed by source node when the destination or an intermediate node moves. A route error message (RERR) is sent to the source node. Intermediate nodes receiving RERR message update their routing table entry for destination by setting the hop count field to infinity. The source node receiving RERR initiates the route discovery process again.

4. MESSAGE FORMATS

4.1 Foraging ant request format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type       |    Hop Count    | Time to live  |  Void         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            RREQ ID                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Destination IP address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Originator IP address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4.1. Format of foraging ant

The format of the foraging ant control message is illustrated above and contains the following fields:

Type	2
Hop Count	Number of hop counts between the node receiving the foraging ant and the originator node which initiated the request
Time to Live	Counter mechanism to limit the lifetime of control packet

RREQ ID	A sequence number identifying the foraging ant when taken in consideration with the originating node's IP address.
Destination IP Address	The IP Address of destination node
Originator IP Address	The IP Address of source node which initiated the route discovery process

4.2 Reply ant message format

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type       |    Hop Count    | Time to live  |     Void      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Destination IP address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Originator IP address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4.2. Reply Message Format

The format of the reply ant control message is shown above, and contains the following fields:

Type	3
Hop Count	Number of hops between the node receiving reply message and destination node which initiated the reply message
Time to Live	Counter mechanism to limit the life-time of control packet

Destination IP Address — The IP Address of destination node

Originator IP Address — The IP Address of source node which initiated the route discovery process

4.3 Hello Messages

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type       | Time to Live  |     Hop Count  |     Void      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Destination IP address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Originator IP address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Residual Energy (32 bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Lifetime    (32 bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4.3. Hello Message Format

The format of hello control message is shown below and contains the following fields:

Type — 1

Time to Live — Counter mechanism to limit the lifetime of control packet. It is initialized to 1.

Hop Count — 0

Destination IP Address	The node's IP Address
Originator IP Address	Neighbor's IP Address
Residual Energy	Remaining energy of node from whom it received the hello message
Lifetime	Time till which the link between neighbor node is considered to be active. If it doesn't receive Hello message within that time period, the node assumes that its active link to neighbor is lost

5. WORKING OF TRADITIONAL ANT COLONY ROUTING PROTOCOL

The basic ACO takes a reactive probabilistic approach of finding good robust paths between source and destination. The algorithm follows: At regular intervals, a foraging ant is launched with a mission to find a path to destination. This establishes back-ward pheromone trail from destination to source. When an intermediate node receives a foraging ant for the first time, it creates an entry in its routing table. The entry consists of destination address which is the source address of the foraging ant, next hop which implies the node from which it received the packet, and pheromone value. The intermediate node increments the pheromone value as given by following equation:

$$\tau_n^d(t) = \tau(t - \delta t) + \delta\tau \tag{5.1}$$

where $\tau_n^d(t)$ is the current pheromone value present at the routing table of node n to reach destination node d, δt is the time duration for which it received the last pheromone, $\delta\tau$ is the incremental pheromone. The foraging ant probabilistically selects the next hop to reach destination using the formula given below.

$$P_{n,d} = \frac{(\tau_{n,d})^\beta}{\sum_{j\epsilon N}(\tau_{j,d})^\beta} \tag{5.2}$$

where $P_{n,d}$ is a probability to select neighbor n as a next hop towards destination d, $\tau_{n,d}$ is a pheromone value at neighbor n to reach destination d, N is the set of neighbors and β is a pheromone coefficient constant.

Duplicate foraging ants are then removed by identifying their unique sequence ID. Once a forward ant reaches its destination, it initiates uni-cast forwarding of backward ants which the destination will send to the source node. The backward ant establishes the pheromone trail from source to destination. After calculating the selection probabilities, the node will forward the data packets to that neighbor node which has been selected based on the distribution of Equation 5.2. The data packet is sent to the selected relayed node and is further relayed towards the destination node. The selected relay nodes increments their pheromone value by a specific amount. Like their natural counterpart, artificial pheromones decay over time. The evaporation process provides a negative feedback in the system which helps ant avoid the stale paths in the network. The evaporation of pheromones takes place constantly by equation given below.

$$\tau_n^d(t) = \tau_n^d(\delta t) * e^{-(t-\delta t)\rho} \tag{5.3}$$

where ρ is constant called evaporation rate of pheromone.

The procedure finishes once the data packet reaches the destination node.

Nodes maintain the neighbor entity in its neighbor table by sending Hello Packet periodically to each other. Hello packet sending interval can be different according to different mobility scenarios. If a node doesn't receive Hello packet from a neighbor for a certain period of time, it then deletes the neighbor information from its neighbor table.

6. PROPOSED IDEA

This section discusses about various heuristic factors which are considered in the proposed routing algorithm which makes it novel compared to already existing routing algorithms for MANETs.

Pheromone and Repellent Pheromone: Ants in nature while travelling from their nest to food source make the routing decision when they reach intersection, i.e., when more than one path is available for their next hop. In such scenario, the probability to choose that path is more which has relatively higher concentration of pheromone compared to other available paths. After the robust path is established, the ants continue to use that path until they encounter some obstacle, for example, placing a stone or pouring water on the path recently formed by ants. In such case, ants no longer use that path and instead begin exploring new paths. Pheromones with repellent property are deposited by ants so that their contemporaries no longer use the earlier efficient path. This property if incorporated in algorithm would make delay less in the network as the nodes would be aware of failed paths in the network.

RSSI: In WSN, sensor nodes are aware of the proximity of their neighbors through RSSI. If the scenario is considered where source node and destination node are placed very far apart such that the destination node barely comes under the transmission range of source node, both the nodes will receive packets with very low RSSI as power of received signal decreases with increase in distance. As a result the chances of packet drops are very high. But due to less number of hops between source and destination node, the destination node will experience less delay. Whereas if source node and destination node are connected such that there exist neighboring nodes through which the packets can be transferred in a multi-hop fashion, then the nodes will receive packets with very high value of RSSI due to close vicinity with each other.

Nevertheless, the delay experienced by the packet will be more as the packet would have travelled with more number of hops from source to destination.

Therefore, it is understood that extreme values of RSSI is not appreciated for our pro-posed system. With the logic of RSSI explained above, it is vital that the goodness of RSSI closely follows the Gaussian distribution [11] as shown in the figure below.

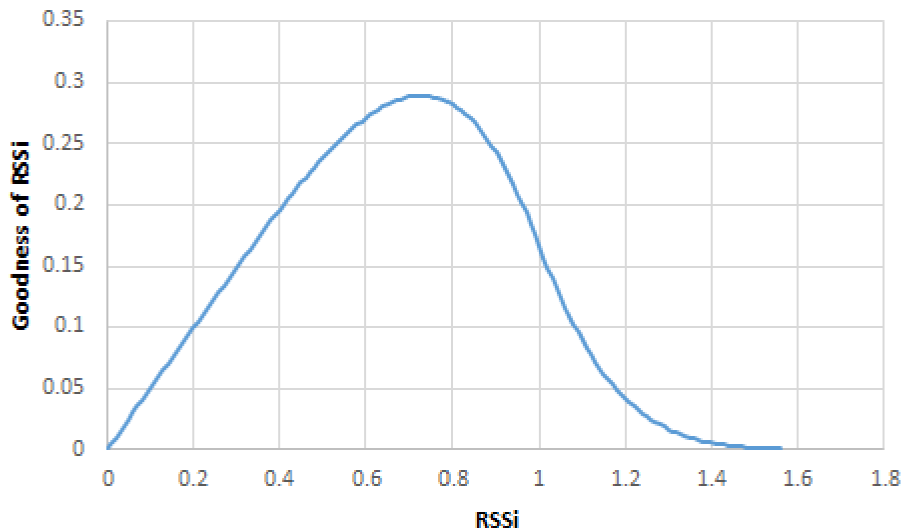


Fig. 6.1. Goodness of RSSI against varying range of RSSI

With reference to [12], if two nodes (source and destination) are placed such that the distance between them is less than the transmission radius R, the total expected hop count from source node to destination node is given by

$$\frac{d}{x}\left[\frac{1}{p(x)[(1-(1-p(x))^{u})]}+\frac{u}{[1-(1-p(x))^{u})]}\right] \tag{6.1}$$

where x is the distance between two consecutive nodes. *p(x)* is the probability of receiving packet and u is a constant and is selected as 1. *p(x)* is dependent on several measurements such as signal strengths, delay, etc. It is approximated by the following equation:

$$P(x) = \begin{cases} 1 - \left(\frac{x}{R}\right)^{2\beta}/2, & \text{for } x < R \\ ((2R - x)/R)^{2\beta}/2, & \text{for } x \geq R \end{cases} \tag{6.2}$$

Where β is constant and is selected as 2. In the experiment, R is selected such that the packet delivery ratio at destination node is 80%.

In order to make equation 6.1 independent of particular distance d, it is optimized which is given as follows:

$$h(x, \mu, \beta, R) = \frac{R}{x}\left[\frac{1}{[p(x)(1-(1-p(x))^u)]} + \frac{u}{[(1-(1-p(x))^u)]}\right] \tag{6.3}$$

By plugging in the value of u in equation 6.3, it is simplified by following equation

$$h(x) = \frac{R(1+p(x))}{xp^2(x)} \tag{6.4}$$

Taking the inverse of above equation, we claim that the Goodness of RSSI [13] is achieved which follows the graph in Figure 6.1. Hence

$$f(S) = \frac{1}{h(x)} = \frac{xp^2(x)}{R(1+p(x))} \tag{6.5}$$

Where *f(S)* is known as Goodness RSSI. After the goodness of RSSI is calculated, the probabilistic formulae and goodness RSSI follows linear relationship. Each node then maintains a neighbor table where RSSI records are maintained against every neighbor. Also it is widely known that RSSI fluctuates too often even when nodes are static [14], the exponential weighted moving average (EWMA) approach chosen by us helps in smoothening the RSSI value.

Residual Energy: If the nodes among the discovered robust path are going to be used extensively for data packet transmission, their battery will deplete faster compared to nodes on non-efficient paths. This will result in creation of void nodes in the network which may lead to network partition. Inclusion of residual energy of node [15] in the routing decision will help in exploring paths other than already discovered robust paths. This technique will improve the lifetime of the network.

Hops: The routing tables at each node gets modified by information from the incoming packets. Through the backward learning the node learns the identity of source as well as destination node and also the total hops required to reach them. If the previous value of hop count stored in the routing table of node is better than the current one then nothing is done but if the current value of hop count is better than the previous one, then the value is updated for future use.

The proposed routing algorithm is an extended version of traditional ant colony based routing algorithm in which the main objective is to maximize the network lifetime. Traditional ant colony routing algorithm considers the pheromone value alone in its probabilistic routing decision process which is not favorable for energy con-strained networks. With additional heuristics discussed above, the extended probabilistic formula is defined as:

$$P_{n,d} = \frac{(\tau_{n,d})^{\alpha}(h_{n,d})^{-\beta}(E_n)^{\delta}(f(S)_n)^{\gamma}}{\sum_{j \epsilon N}(\tau_{j,d})^{\alpha}(h_{j,d})^{-\beta}(E_{j,d})^{\delta}(f(S)_{j,d})^{\gamma}} \tag{6.6}$$

where n is the next hop selected by an ant to reach destination d, $\tau_{n,d}$ is a pheromone value from neighbor n to reach destination d. h is the number of hops taken by an ant to reach destination node d. N is the set of neighbors of node. E is the remaining energy in the node, *f(S)* is the goodness RSSI which follows Gaussian distribution as shown in Figure 6.1. α,β,δ and γ are the factors to adjust the relative importance of pheromone concentration, hops, residual energy and goodness RSSI respectively.

As discussed in Section 2, nodes receiving the ants update the pheromone value in their routing table by depositing a constant value of pheromone in their routing table which acts as a positive feedback. As a result, an impulsive response is observed with regards to pheromone whenever a node receives an ant. Similar to the biological ants, the pheromone value is a function of time which means pheromone value decreases exponentially as the time progresses which makes it volatile.

7. EXPERIMENTAL RESULTS

7.1 Simulation Parameters

Ant Colony routing algorithm is implemented on open source network simulator (ns-3) [16]. Nodes are initially laid out in regular hexagonal structure to account for hexagon shape used for radio coverage in cellular communication system. The deployment of nodes are then made random by adding Gaussian Random Variable with variance (99) to its x and y coordinates. This deployment of nodes can be named as hexagonal randomized placement. As shown in Figure 7.1, the source and destination nodes are placed along the diagonal so that they are far apart. By default 802.11b is used as underlying MAC protocol (Layer-2). UDP Echo is used as the application layer protocol. Simulation time for each experiment is set to at-least 500 seconds. Performance of routing protocol is carried out by measuring the performance metrics as described in the next section.

7.2 Performance Metrics

a) Throughput

It is the rate at which the data packets are delivered successfully by the network to destination node from source node. Also known as goodput, it is represented by bits/bytes per second. The throughput is affected by various factors such as background traffic/noise, bandwidth of physical medium, processing power, end user medium, etc. In any communication based network, higher throughput is always desired.

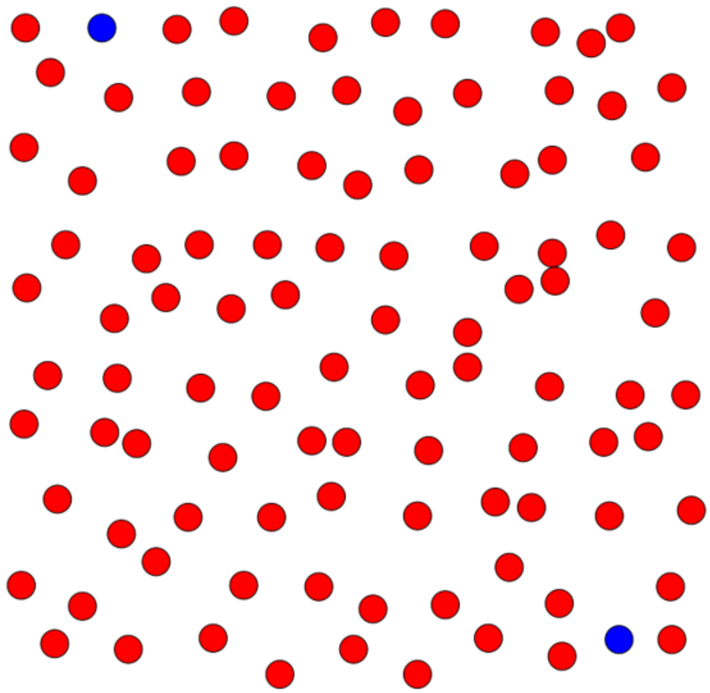


Fig. 7.1. Placement of nodes

b) Mean Delay

Mean Delay is the time consumed by the data packet to reach the destination node from source node. With reference to [17] mean delay is calculated by taking the ratio of delay sum to the total number of received packets at the destination node.

c) Packet Loss

Packet Loss results when one or more data packets fail to reach the destination node due to various reasons such as dropping of packets, error in data transmission, network congestion due to overwhelming loads.

d) Network Lifetime

It is defined as the time span a network can fulfill its service whereby source node and destination node communicate with each other by exchanging data packets and other control packets. Also in various literature's, it is the time at which the first node in the network becomes dead.

e) Partition Time

Partition Time is defined as the time beyond which communication between source and destination no longer takes place. This happens when the network becomes disconnected due to energy depleted nodes.

f) Mean Hop Count

Mean Hop Count is the average of hops taken by the data packet to reach the destination node from source node.

7.2.1 Simulation Results and Analysis

The performance of routing protocol is analyzed under different test scenarios and topologies to account for the robustness.

a) Background Traffic

The logic of introducing background traffic in simulation is to mimic real life network scenarios where there's a disturbance or white noise along with regular traffic flow. At every regular time interval (50 seconds) a pair of random nodes are chosen as background source and destination nodes which does the job of background traffic flow. At the end of simulation, there are 10 pairs of nodes which contribute to the background traffic. During simulation run-time, every node from MAC (L-2) layer is able to keep track of how many packets it has sent, how many packets it has received and how many packets it has dropped during transmission and reception of packet. These results are analyzed at that time when the pair of nodes are chosen randomly for background traffic flow and then the average of resultant is calculated. Hence these attributes are vital for background traffic calculation.

The topology of graph is shown in Figure 7.2. The minimum average degree of node is 4.94 for the graph to be connected. AML2FIG software [18] is used to show the topology of the network.

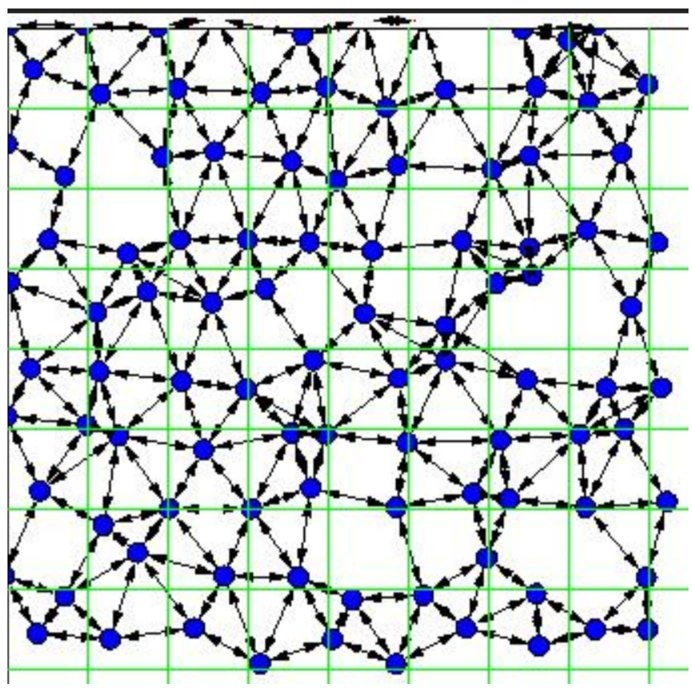


Fig. 7.2. Connectivity of the graph

a1) Effect of throughput against background traffic

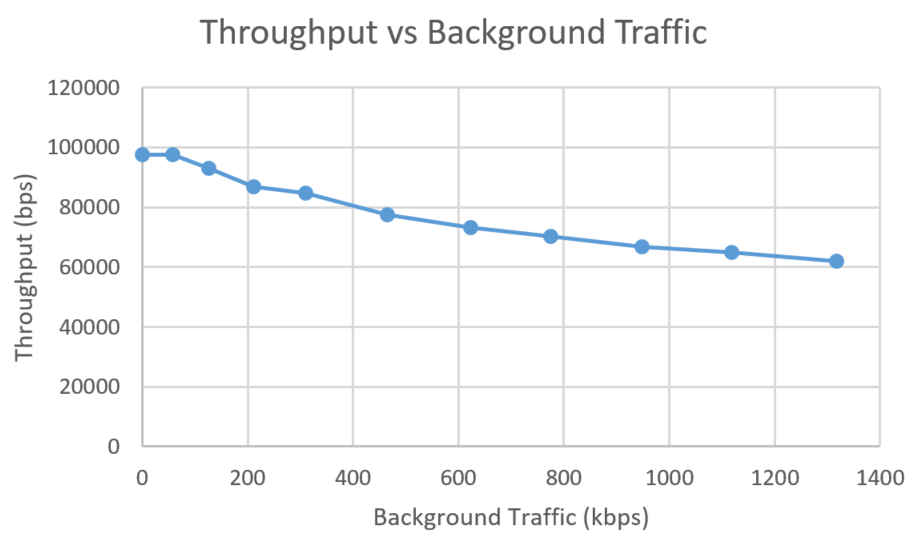


Fig. 7.3. Throughput against background traffic

The traffic flow from genuine source and destination nodes as shown in Figure 7.1 are probed throughout the simulation time. It is observed that as the background traffic overwhelms the network resources, Ant Colony algorithm becomes more sensitive to background traffic. As a result, there's a dip in throughput when background traffic increases.

a2) Effect of delay against background traffic

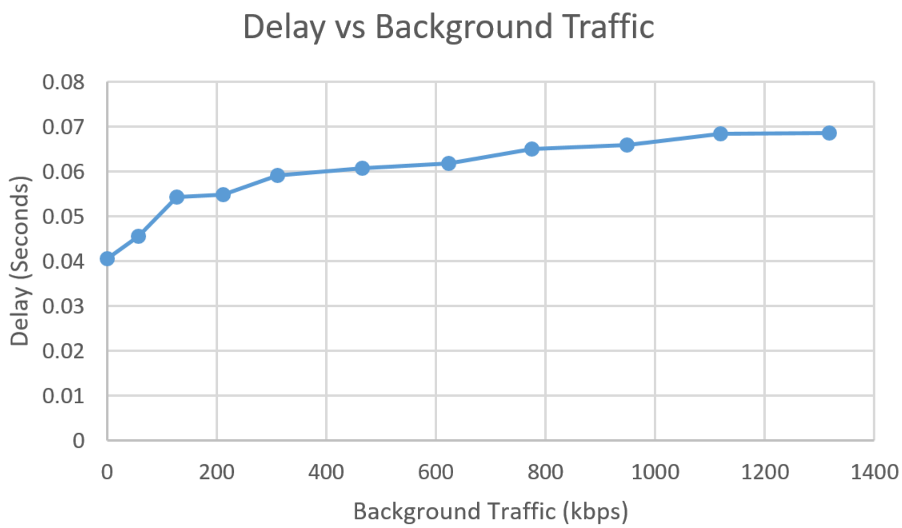


Fig. 7.4. Delay against background traffic

Relationship between delay and background traffic is very much linear. Since ant colony forms multiple paths with varying hop counts between source and destination pair, these paths are affected by background traffic which brings in the latency in the system.

a3) Effect of Packet loss against background traffic

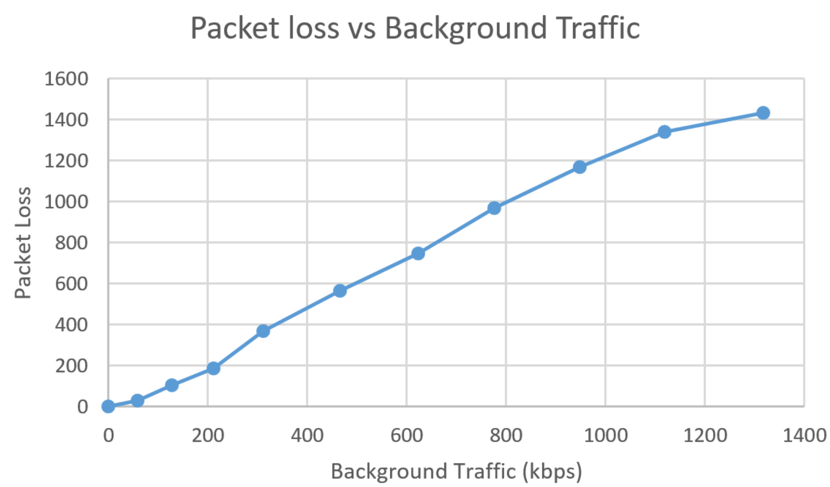
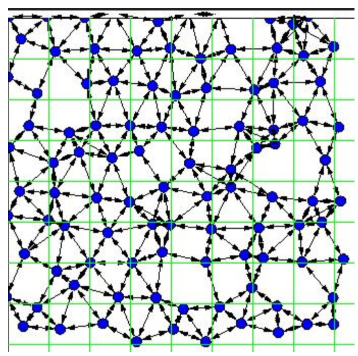


Fig. 7.5. Packet loss against background traffic

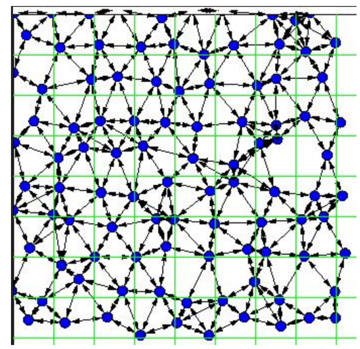
Packet loss exponentially increases with increase in background traffic. Conditions like packet collision, congestion, overhearing due to background load affects multiple paths between source and destination pair which makes Packet losses more sensitive to background traffic.

b) Average Degree

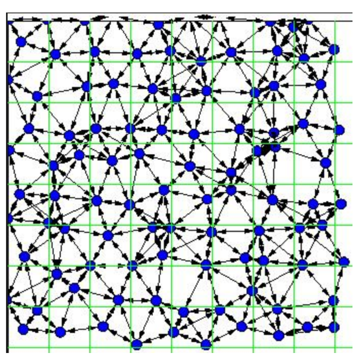
The topology and complexity of network varies on the transmission range of nodes. Before proceeding for the analysis, it is vital for us to understand the topology changes and degree distribution when the transmission range is varied. In this experiment, the transmission range is varied from 71 meters to 120 meters. Figure 7.6 - 7.7 shows the connectivity of graph as well as its complexity when the transmission range is varied in increasing order.



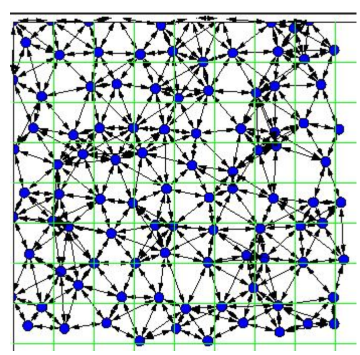
(a) Average Degree 4.94 (Tx Range: 71 m)



(b) Average Degree 5.34 (Tx Range: 75 m)

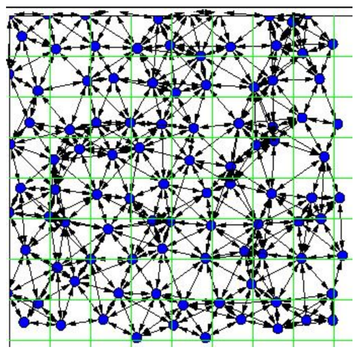


(c) Average Degree 5.94 (Tx Range: 81 m)

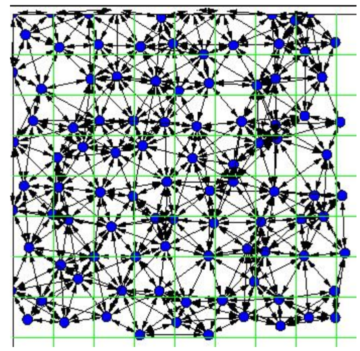


(d) Average Degree 6.94 (Tx Range: 87 m)

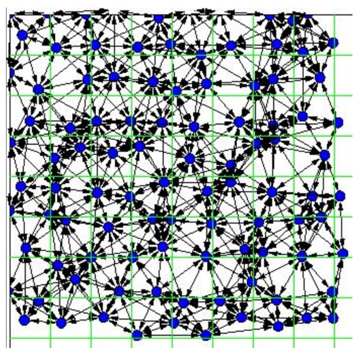
Fig. 7.6. Topology and connectivity of network.



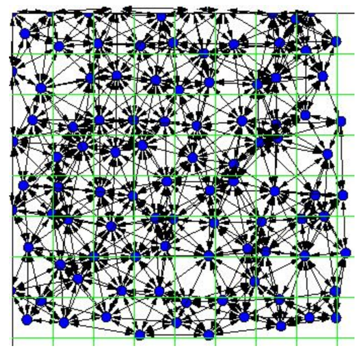
(a) Average Degree 8.03 (Tx Range: 93 m)



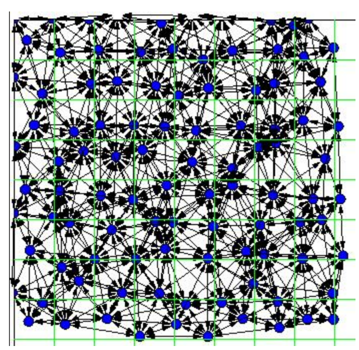
(b) Average Degree 9.42 (Tx Range: 99 m)



(c) Average Degree 10.54 (Tx Range: 105 m)



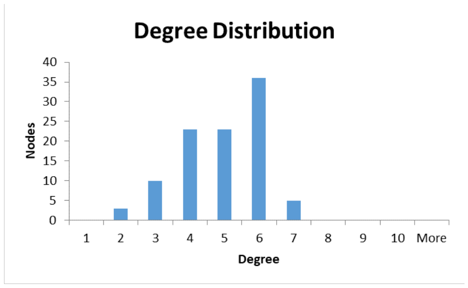
(d) Average Degree 11.63 (Tx Range: 111 m)



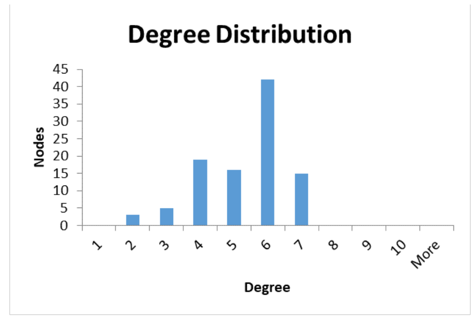
(e) Average Degree 13.43 (Tx Range: 120 m)

Fig. 7.7. Topology and connectivity of network continued.

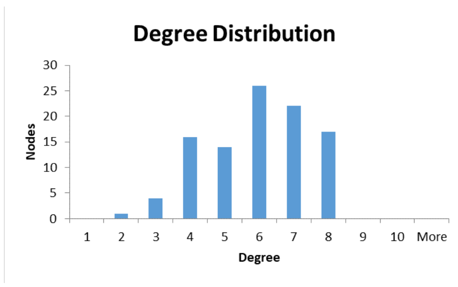
In the following figures the degree distribution will be shown against every average degree of network.



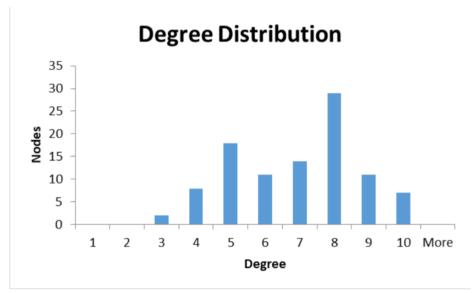
(a) Average Degree 4.94



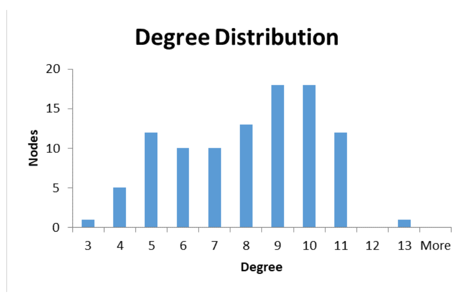
(b) Average Degree 5.34



(c) Average Degree 5.94

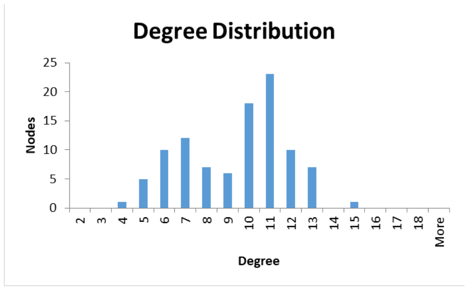


(d) Average Degree 6.94

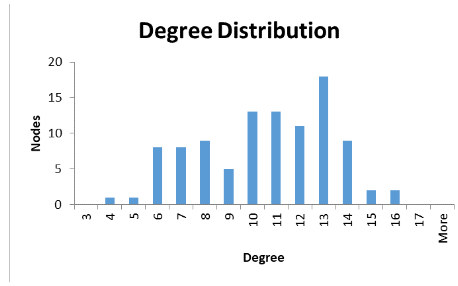


(e) Average Degree 8.03

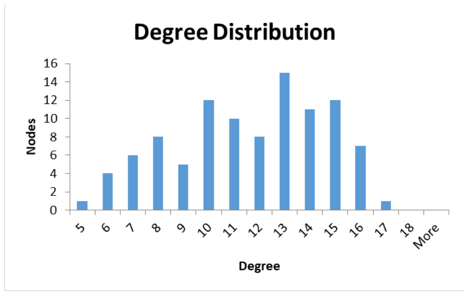
Fig. 7.8. Degree Distribution against varying transmission range



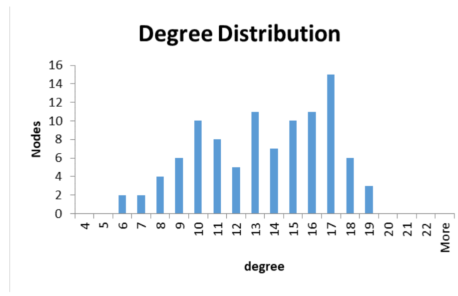
(a) Average Degree 9.42



(b) Average Degree 10.54



(c) Average Degree 11.63



(d) Average Degree 13.43

Fig. 7.9. Degree Distribution against varying transmission range (continued)

b1) Analysis of Throughput against average degree

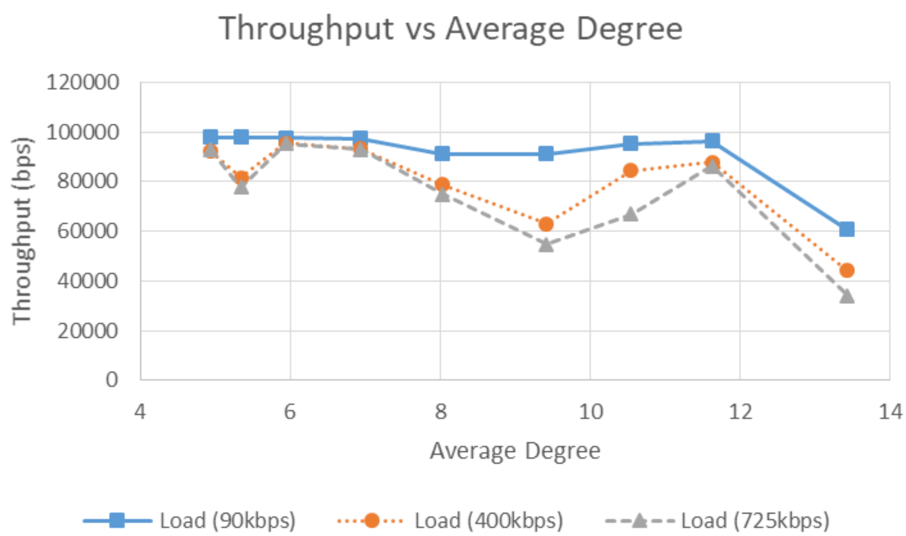


Fig. 7.10. Throughput vs Average Degree

The Throughput decreases with increase in average degree of the network. As the transmission range increases, the complexity of network increases as shown in Figures 7.6- 7.7, which leads to packet collisions and high interference. It has been observed that when the average degree of network is close to 6, the network experiences its best performance as the throughput of network is unaffected by the background load possibly due to best connectivity in the network.

b2) Analysis of Delay against average degree

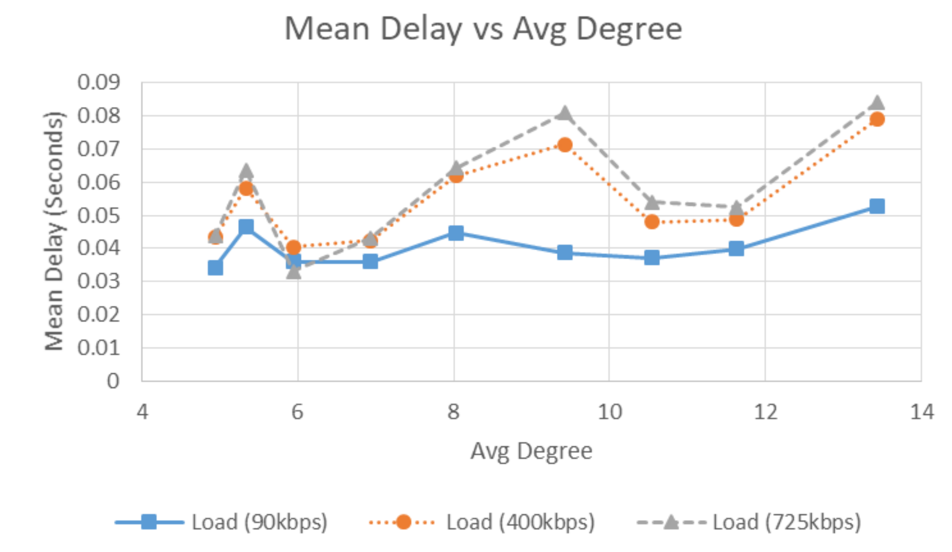


Fig. 7.11. Delay vs Average Degree

It is observed that mean delay is directly proportional to average degree of network. With increase in average degree of node, the node density becomes higher, which means a node can access more number of neighbors around itself. This leads to increase in overhearing and congestion which is the root cause for increase in delay. However the optimal average degree of network is close to 6 as the delay of network is hardly affected.

b3) Analysis of Packet Loss against average degree

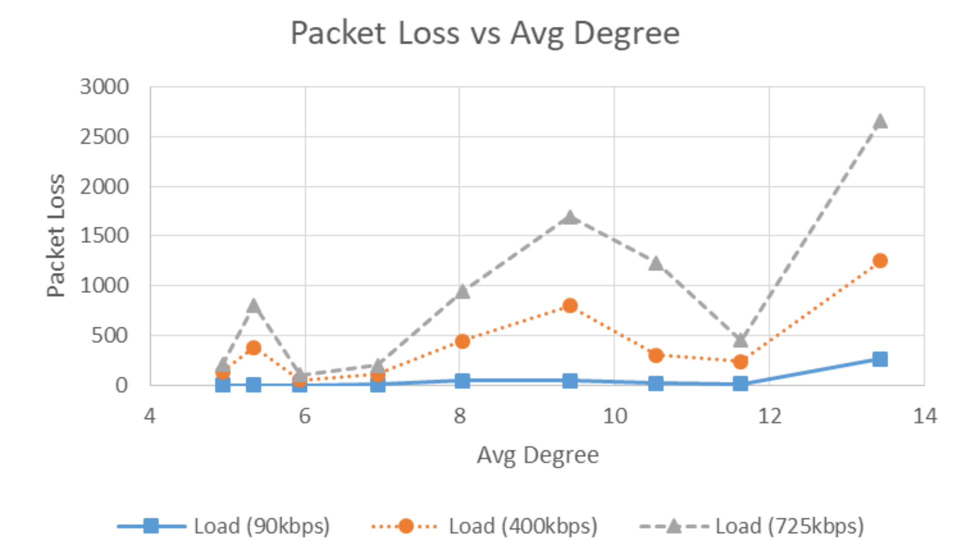


Fig. 7.12. Packet Loss vs Average Degree

Packet loss is expected to increase with increase in node density due to overhearing and congestion. The packet loss is found to be the least when the average degree of network is around 6 as it offers best connectivity.

7.2.2 Randomness Property of Ant Colony Routing Protocol in terms of Hop Count

This section claims about the random hop count property of our routing algorithm by showing the 95% confidence interval. Against every minimum path between source and destination node, the mean hop between them is calculated along with the confidence interval that gives a range of hop count values with 95% surety.

The nodes here are placed on 10x10 2-D grid as shown in Figure 7.13. This uniform topology of network is chosen because it makes sure that majority of the nodes have equal degree around itself.

Otherwise in case of random placement of nodes, it is possible to have a bias in hop count performance as the degree of nodes are not consistent.

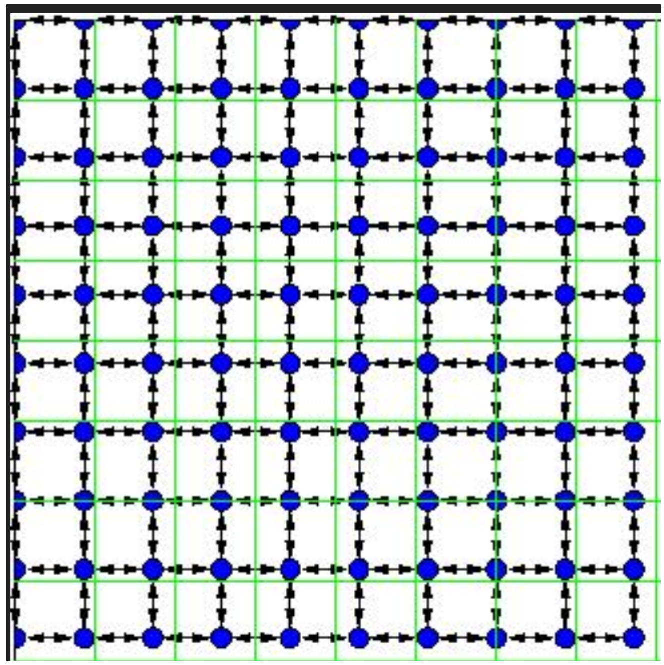


Fig. 7.13. Placement of nodes in a square grid

The Figure 7.14 shows the confidence interval of hop counts between the source and destination node.

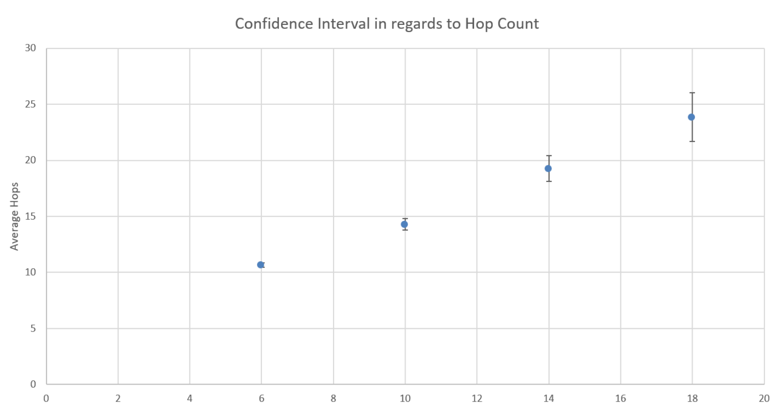
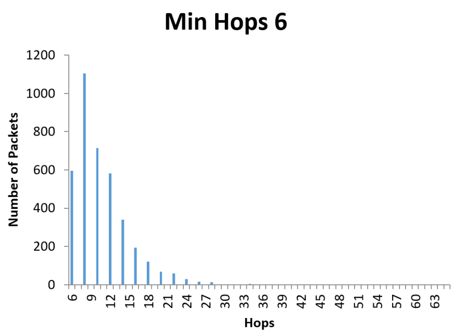


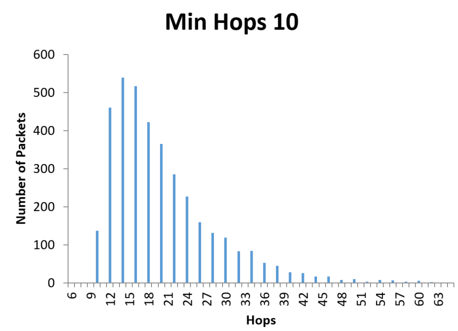
Fig. 7.14. Confidence interval in regards to hop count

It can be inferred that when the source node and destination node are placed such that the minimum number of hops between them, let's say is 18, there's 95% chance that the packets received by the destination node will have hop count in the range from 22 to 27 hops.

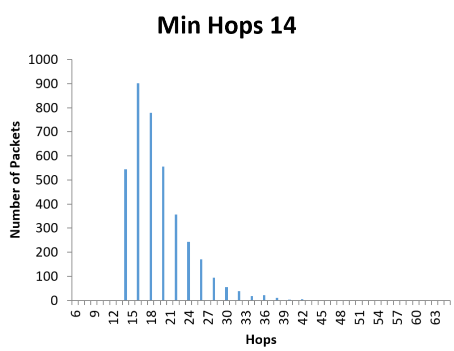
The Figure 7.15 shows the hop count distribution for every minimum hops shown in Figure 7.14.



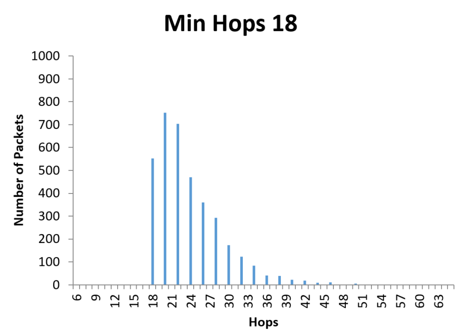
(a) Hop Count Distribution for min hops 6



(b) Hop Count Distribution for min hops 10



(c) Hop Count Distribution for min hops 14



(d) Hop Count Distribution for min hops 18

Fig. 7.15. Hop Count Distribution against minimum hops

7.3 Importance of impact factors in Ant Colony Algorithm

This section discusses about the need of individual impact factors taken into the consideration as described in section 6.

7.3.1 RSSI as an impact factor

Using RSSI is a well known technique to measure distance between source and receiver. The distance is estimated by using the strength of received wireless signal. The relationship between RSSI and distance is inversely proportional as shown in Appendix A. Low values of RSSI cause more number of packet losses which in turn makes failure of transmissions of packets. The motive behind this analysis is to find the total number of failed transmission of data packet by varying the weightage of RSSI coefficient. Each node gathers this data from MAC (L-2) layer.

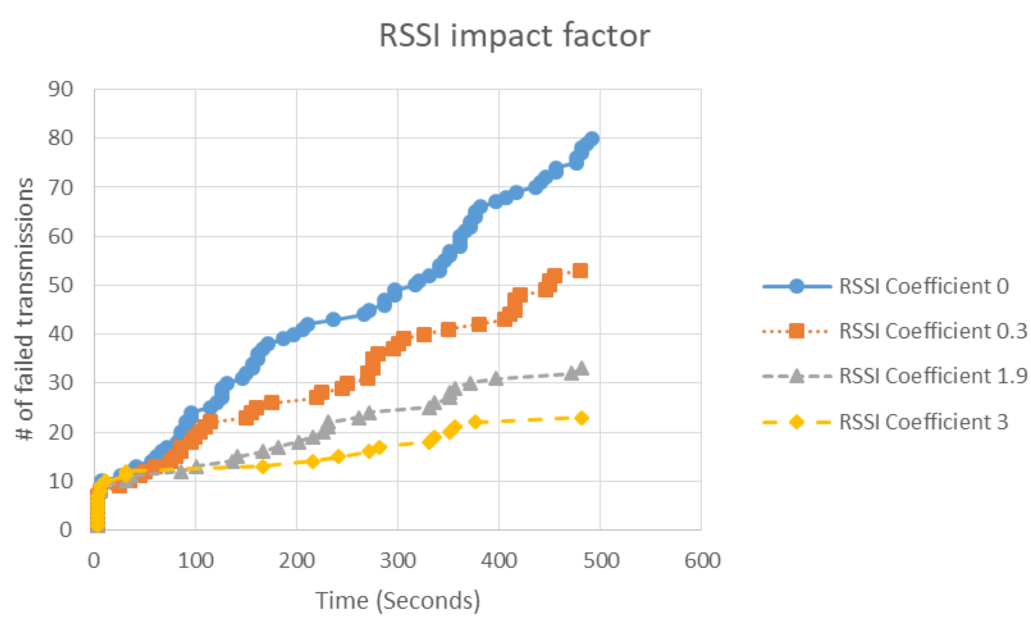


Fig. 7.16. Importance of RSSI Coefficient factor

The figure above shows how the failed transmission of packets can be controlled with RSSI impact factor. RSSI Coefficient 0 signifies there's no consideration of RSSI in the routing algorithm due to which maximum failure of packet transmissions are observed. When the RSSI is taken into consideration by increasing the weightage of RSSI coefficient, the number of failed transmissions of data packets drops down.

7.3.2 Hop Count as an impact factor

Hop Count information is used by the nodes to learn about the how far they are from source and destination. This knowledge gathered from this data helps to reduce the end-to-end delay in routing as much as possible.

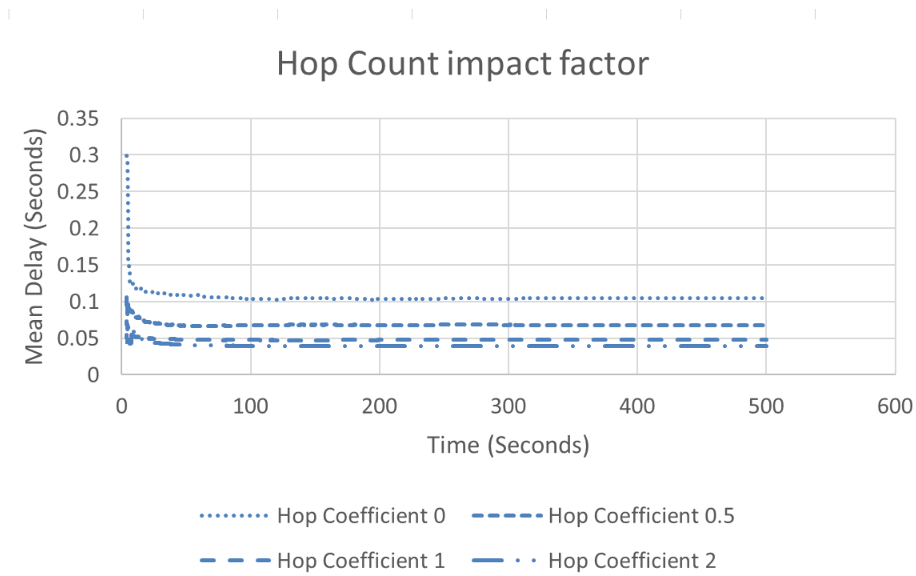


Fig. 7.17. Importance of Hop Count factor

With reference to the figure given above, highest delay is observed when hop count factor is turned off (hop coefficient 0) as the routing protocol has no knowledge about the source and destination.

When the preference to hop count factor is increased by changing the hop count coefficient, the nodes are aware of source and destination in terms of hop counts and the reduction in end-to-end delay is expected. Overall 60% reduction in delay is observed.

7.3.3 Residual Energy as an impact factor

The inclusion of residual energy as an impact factor in Layer 3 (IP layer) routing makes the protocol energy aware as the routing decision considers the residual energy among neighboring nodes while forwarding the data packet . If this impact factor is excluded from routing, the reduction in network lifetime is expected.

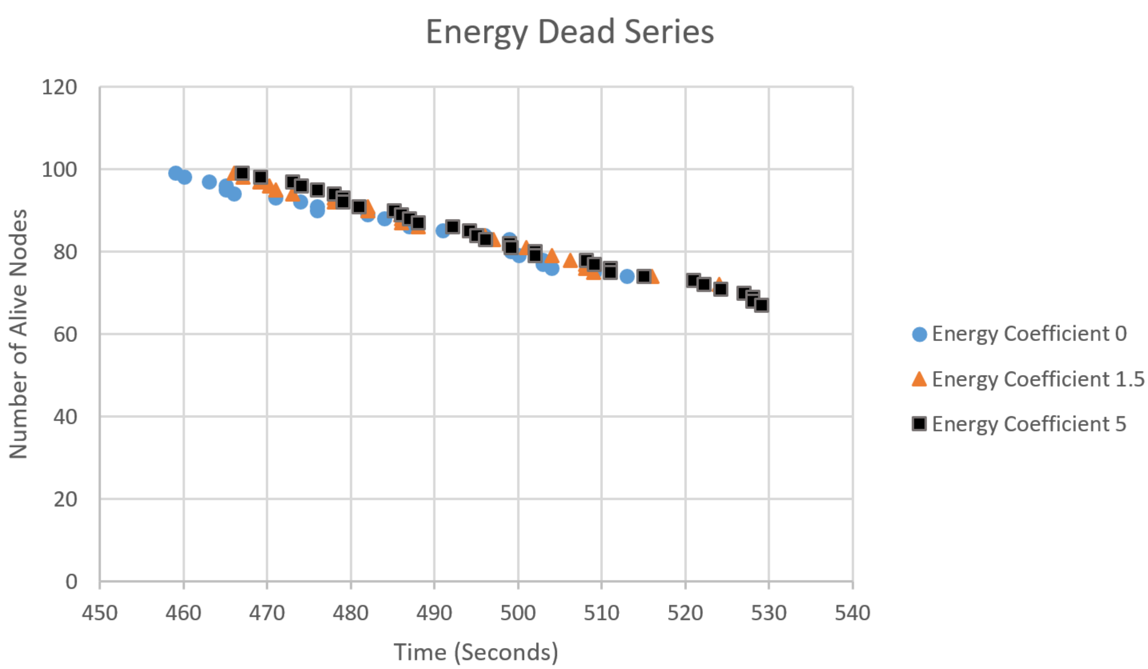


Fig. 7.18. Importance of Energy factor

In the figure given above, the time at which the node dies is recorded for every node which gives us the relationship between number of alive node versus time. The source transmits data at every second. When the energy impact factor is not considered (Energy Coefficient 0), the first node dies at 459 seconds and the communication link breakage occurs at around 515 seconds, also known as network partition time where source and destination are no longer able to communicate with each other.

When slight weightage is given to Energy impact factor (Energy coefficient 1.5), network partition occurs at 524.14 seconds and when energy coefficient is 5, highest energy related performance is observed where the network partition time is around 530 seconds.

Overall 15 seconds of improvement in network partition time is observed when energy impact factor is taken into consideration.

8. COMPARATIVE ANALYSIS

This section compares Ant Colony Routing Algorithm against the AODV Protocol. Various test scenarios such as background traffic analysis, varying transmission range, energy analysis, etc. are considered for comparative purposes.

8.1 Background Traffic

8.1.1 Throughput effect against background traffic

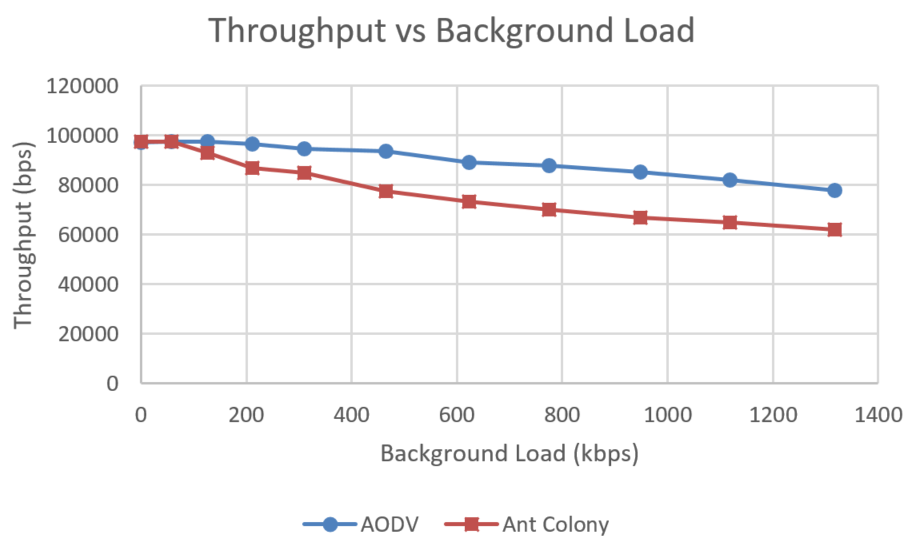


Fig. 8.1. Throughput Comparison

It is deduced that Ant Colony is more sensible to background traffic as compared to AODV as it is evident from the throughput trend. Since AODV forms path between source and destination pair with least number of hops unlike Ant Colony, it suffers less packet loss comparatively which makes the throughput performance better than Ant Colony.

8.1.2 Effect of Mean Delay against background traffic

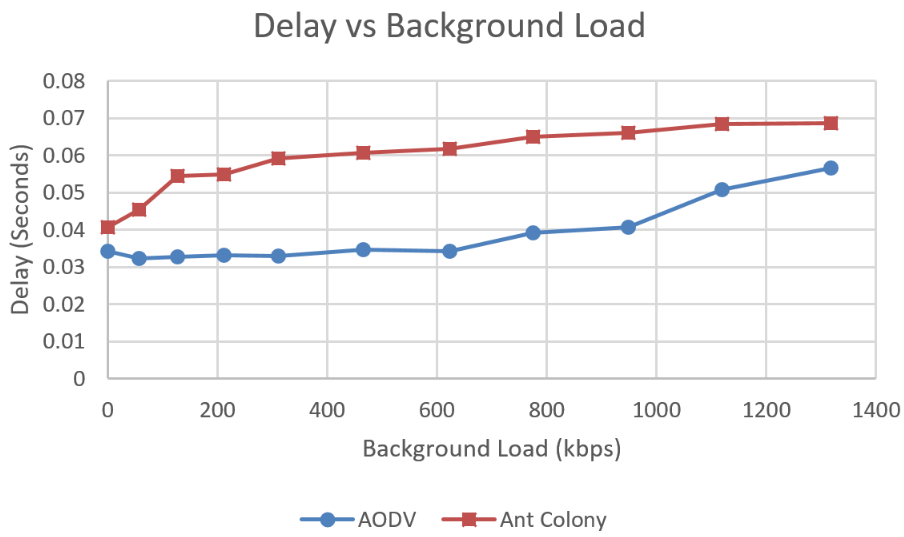


Fig. 8.2. Mean Delay Comparison

AODV experiences lesser delay as compared to Ant Colony protocol. The reason is that AODV forms the shortest path between source and destination. As a result, even though delay in AODV increases with background traffic, its going to be lesser than that of Ant Colony. Ant Colony Algorithm is not designed for routing over the best path between source and destination which makes the performance of AODV better than Ant Colony in terms of delay.

8.1.3 Effect of Packet Loss against background traffic

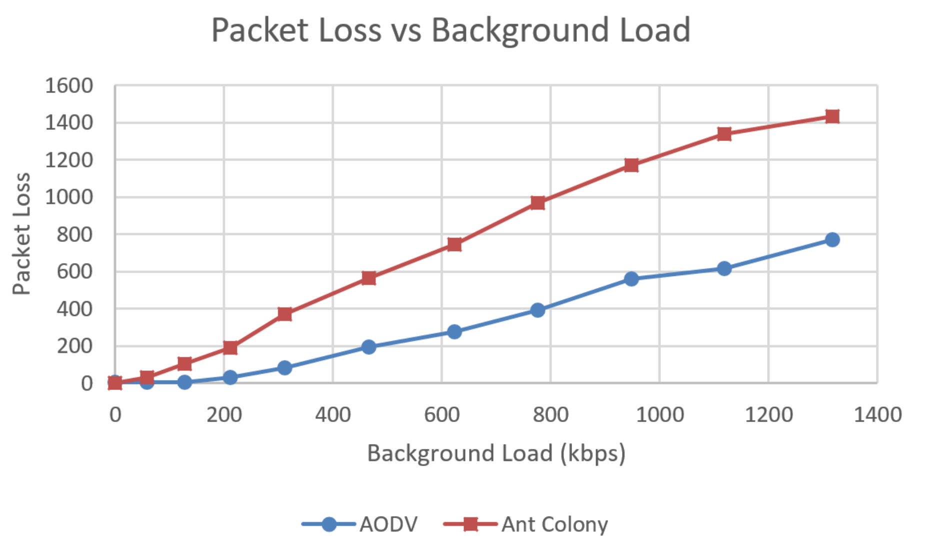


Fig. 8.3. Packet Loss Comparison

Ant Colony suffers more packet loss against increase in background traffic when compared to AODV. Along with optimal hop path between source and destination, Ant Colony constructs multiple paths with higher hop counts. Therefore the probability of packet losses along higher hop count paths is more than AODV which only constructs hop optimal path. This makes Ant Colony more prone to packet losses.

8.2 Average Degree

As explained in previous Section 7.2.1, the topology is made dense by changing the transmission range of all the nodes and the performance metrics are measured.

8.2.1 Throughput effect against average degree

By including the background load in the system, the throughput is measured against varying transmission range and then the following observations are made.

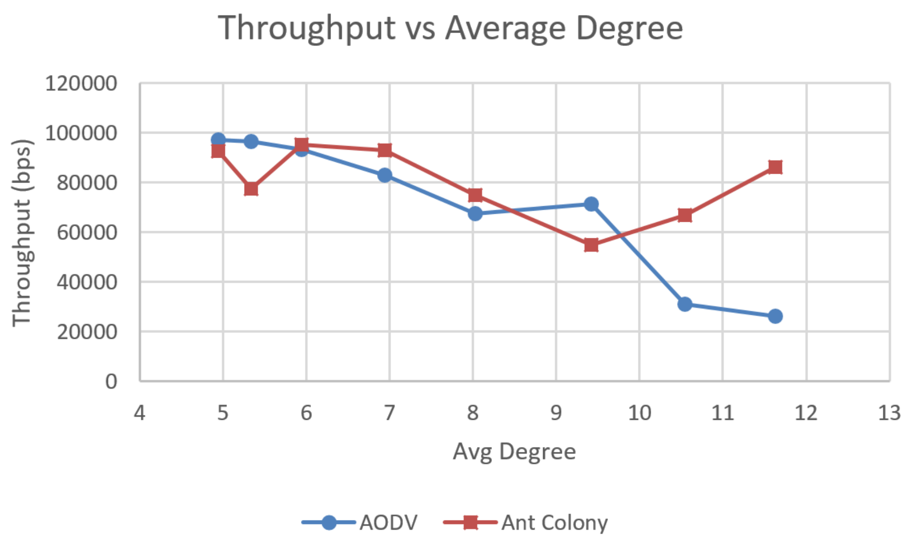


Fig. 8.4. Throughput Comparison against average degree

Ant Colony attains maximum throughput at average degree of 6 whereas AODV has its maximum throughput at average degree of 4.94. During increase in transmission range, AODV experiences more packet loss as shown in figure 8.5 due to network congestion which makes the throughput performance lower than that of Ant Colony.

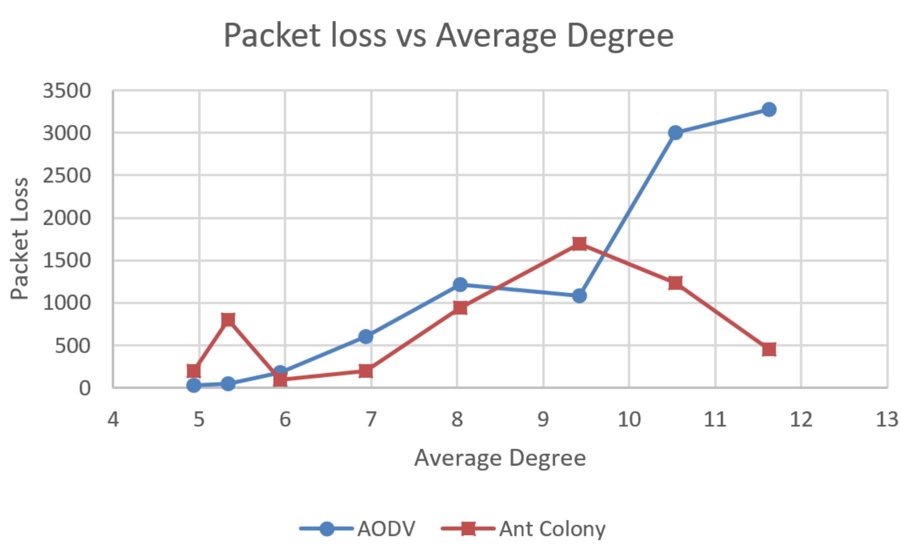


Fig. 8.5. Packet loss against average degree

8.2.2 Effect of Mean Delay against Average Degree

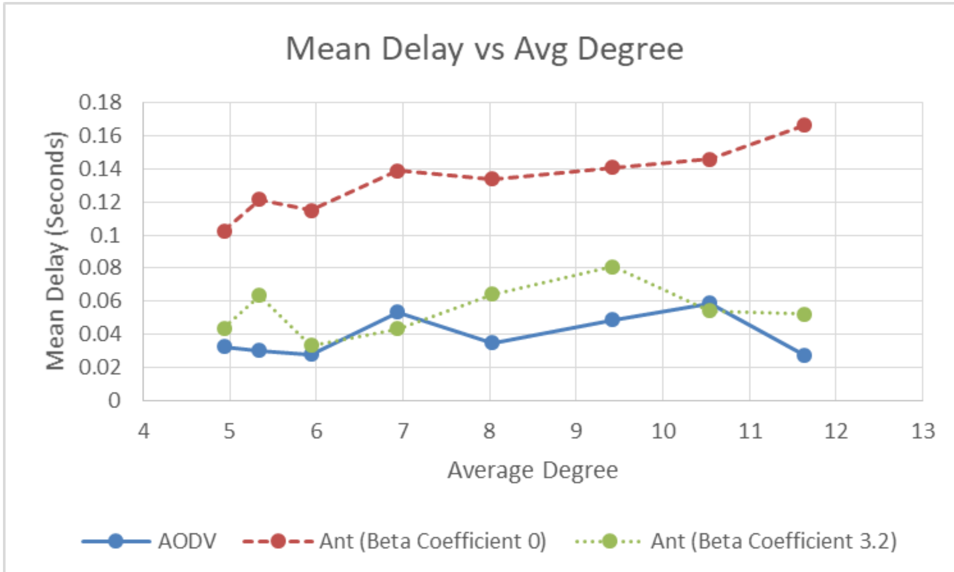


Fig. 8.6. Mean Delay comparison against average degree

AODV outperforms Ant Colony when it comes to delay against increase in transmission range. As AODV forms shortest path between source and destination, with increase in transmission range, packets from source node can reach destination with fewer number of hops making the delay comparatively less. Also in case of Ant Colony, maximum delay is observed that when the hop count impact factor is ignored as the nodes ignore the hop count information in its routing table.

8.3 Energy Comparison

In both AODV and Ant Colony, all nodes are equipped with battery. With reference to data-sheet of FRDM KW41Z [19] micro-controller, the transmitting current has been configured to 6.1 mA, the receiving current is set to 6.8 mA. The micro-controller uses a single coin cell battery [20] which has a idle current capacity of 0.19 mA. The goal in this section is to study the energy analysis and lifetime of the network.

8.3.1 Energy Depletion Series

Communication between source and destination takes place at an interval of 2 minutes in order to mimic the wireless sensor network applications. Figure 8.7 shows the order at which the nodes become energy depleted when two different routing protocols are used.

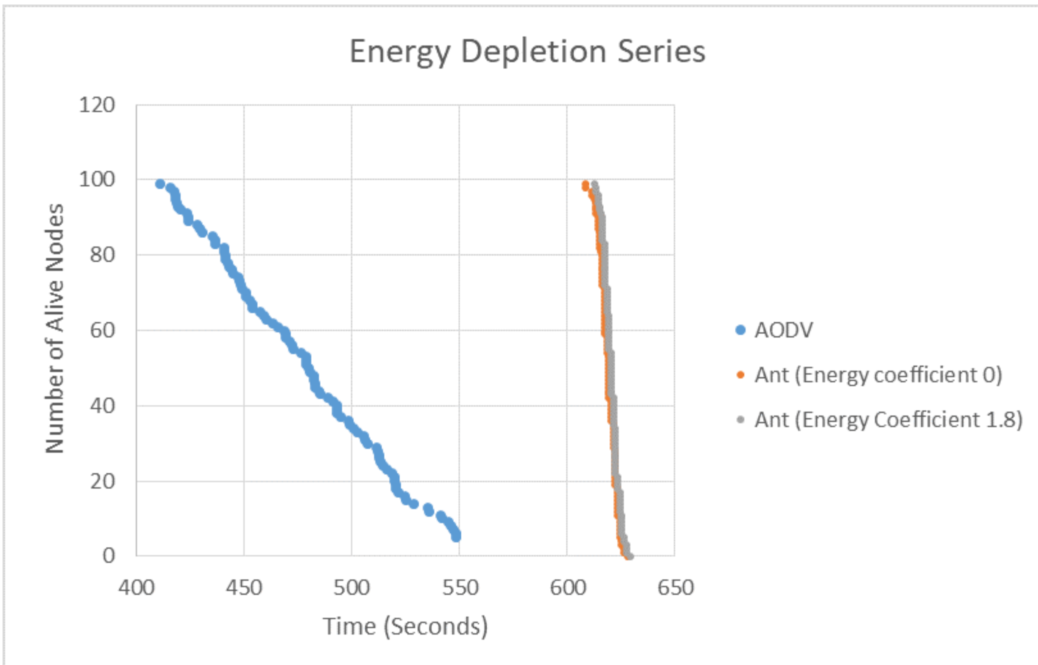


Fig. 8.7. Comparison of dead node series

The above figure shows that while using AODV routing protocol the time at which the first dead node is observed at approximately 411 seconds whereas in Ant Colony, the time at which the first dead node (when δ is 1.8) is observed at approximately 612 seconds. When δ is 0, time at which the first dead node is observed at 608 seconds. Partition time when AODV is used is observed at 549 seconds whereas when using Ant Colony, partition time is 629 seconds. Thus Ant Colony experiences 49% improvement compared to AODV. The inclusion of residual energy as an impact factor in ant colony routing algorithm makes it more better than AODV in terms of energy performance.

8.3.2 Comparison of Remaining Energy over different Time-stamps

The following figures have been drawn using MATLAB [21] color plots where RGB value is varied according to the residual energy of node at that time instant. The colorbar indicates darker color as full energy and as the node looses energy, the color gradually changes to brighter shade.

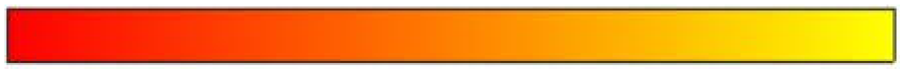
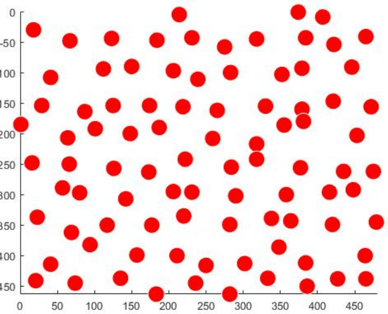
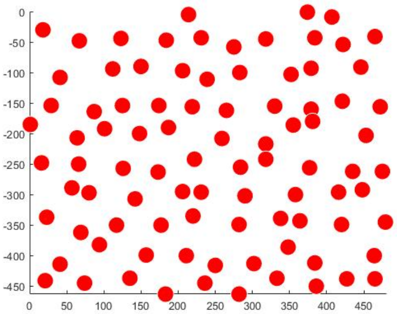


Fig. 8.8. Color bar of remaining energy

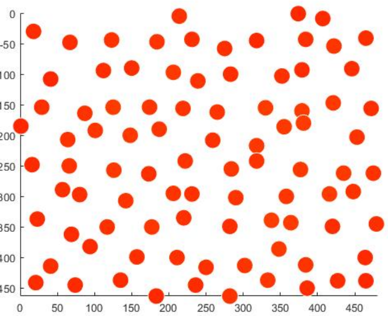
This Section visualizes the remaining energy of nodes over different time intervals which gives the reader a rough idea about energy consumption when using different routing protocols.



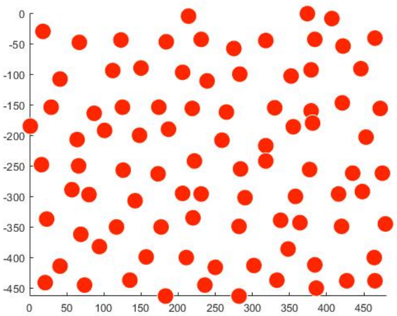
(a) AODV at 0 seconds



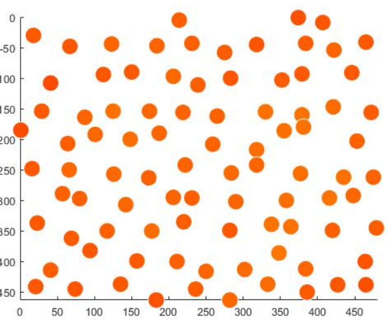
(b) Ant Colony at 0 seconds



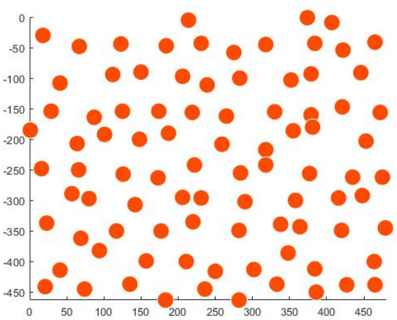
(c) AODV at 100 seconds



(d) Ant Colony at 100 seconds



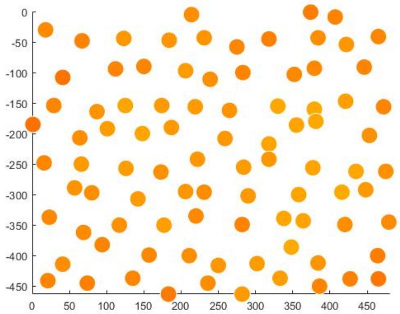
(e) AODV at 200 seconds



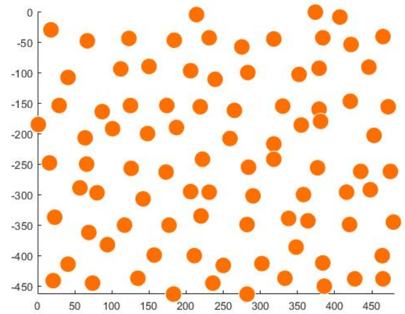
(f) Ant Colony at 200 seconds

Fig. 8.9. Remaining Energy over different time stamps

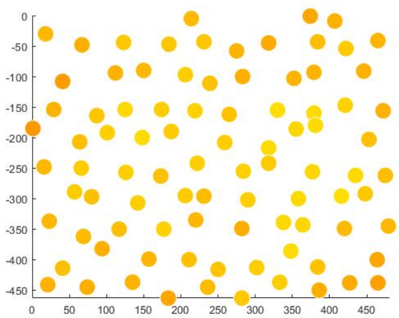
Till 100 seconds, it is hard to observe changes in the residual energy of nodes. When around 200 seconds, nodes begin to fall under middle region of colorbar shown in Figure 8.8 which means that energy level of nodes is going down.



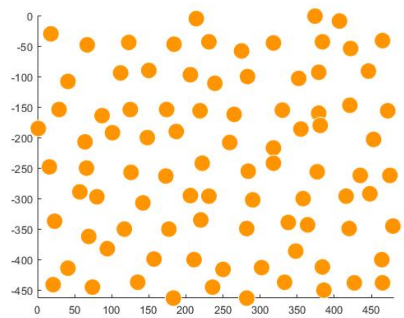
(a) AODV at 300 seconds



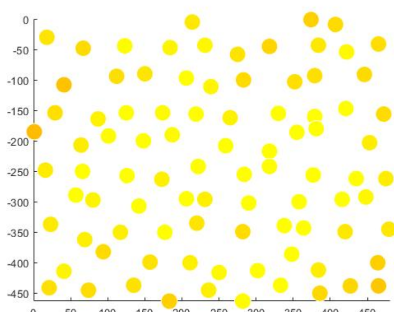
(b) Ant Colony at 300 seconds



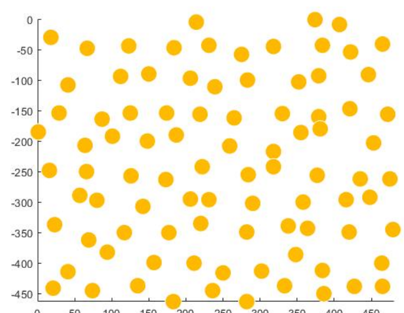
(c) AODV at 400 seconds



(d) Ant Colony at 400 seconds



(e) AODV at 500 seconds



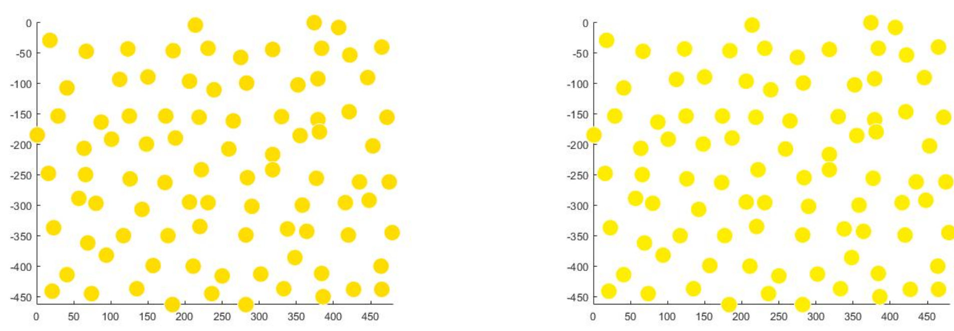
(f) Ant Colony at 500 seconds

Fig. 8.10. Remaining Energy over different time stamps (continued)

At 400 seconds, it is observed that in case of AODV protocol, the nodes around the middle of topology have average remaining energy of 0.0699 Joules with standard deviation of 0.0127 Joules as compared to the rest of nodes which has average remaining energy of 0.1265 Joules with standard deviation of 0.0173 Joules. Whereas in ant colony, the nodes around middle region have average remaining energy of 0.166 Joules with standard deviation of 0.0008 Joules as compared to the rest of nodes having average residual energy of 0.1683 Joules with standard deviation of 0.001 Joules. At around 500 seconds, most of the nodes using AODV have been energy depleted except few of them on the extreme sides. In case of ant colony, the nodes can communicate for longer time due to relatively high residual energy.

Two things can be claimed with reference to the Figures 8.9 and 8.10. While using Ant Colony protocol, the nodes at any time instant and irrespective of their location (extreme corners, middle of topology) have evenness in regards to residual energy. This is due to the fact that Ant Colony uses remaining energy as one of the impact factor during packet forwarding process. Also the ability to establish multiple paths between source and destination makes it consume less energy unlike AODV which follows hop optimal approach leading to discrepancy in terms of residual energy of nodes.

Figures below show the remaining energy of nodes at 600 and 640 seconds when using Ant Colony routing protocol. AODV could not run for longer period of time because of high energy consumption.



(a) Ant Colony at 600 seconds (b) Ant Colony at 640 seconds

Fig. 8.11. Ant Colony time stamps at 600 and 640 seconds

When time is at 640 seconds, all the nodes are uniformly used and are energy depleted as can be seen in the figure shown above.

8.3.3 Comparison of standard deviation of residual energy

Starting from time 50 seconds, at regular intervals of 25 seconds, the standard deviation of residual energy for all 100 nodes in the network is calculated and is plotted against time as shown in Figure 8.12 below.

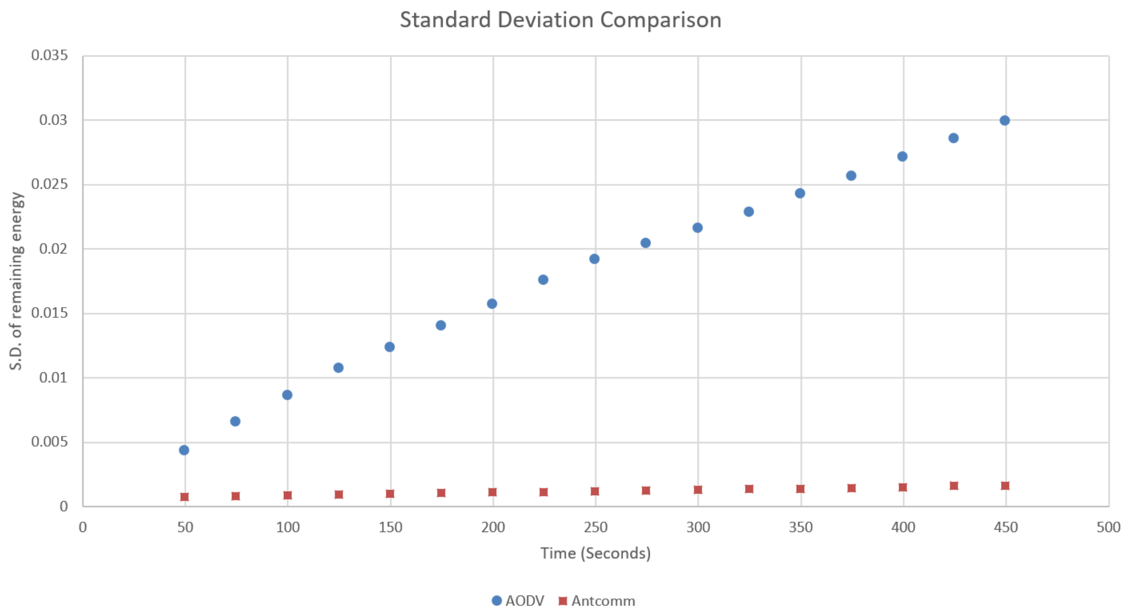


Fig. 8.12. Comparison of Standard Deviation of Residual Energy

It is observed that standard deviation of AODV is very high compared to standard deviation of Ant Colony Routing. The reason is in AODV routing, there's only one path between source and destination node. As a result, the nodes along the chosen path are used more compared to the rest of the nodes in the network. This creates a disparity among nodes in terms of residual energy.

Whereas in Ant Colony Routing, there are multiple paths between source and destination. Due to this, most of the nodes are used uniformly during the communication process which makes the standard deviation low due to evenness in remaining energy.

8.3.4 Comparison of Average Remaining Energy

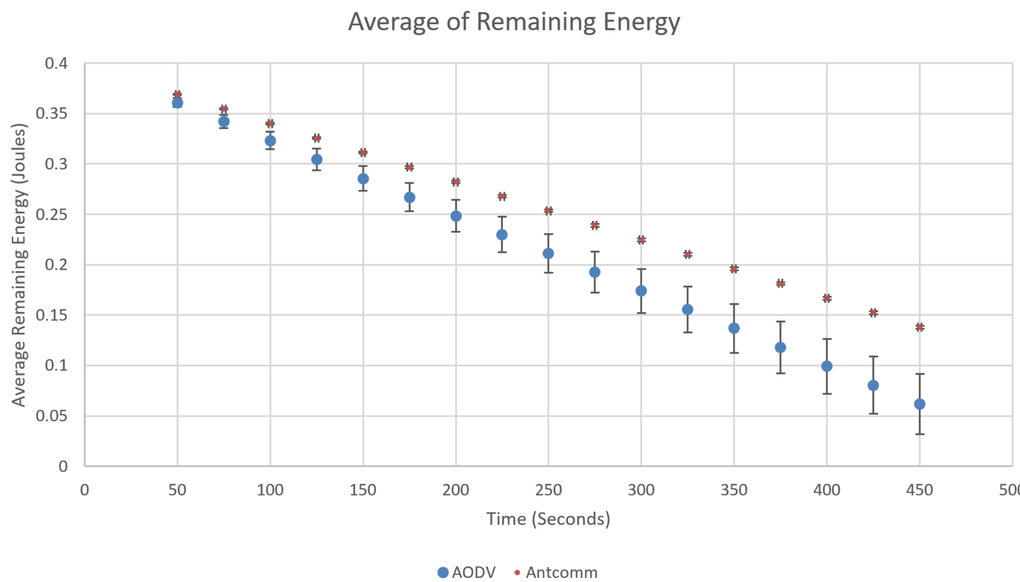


Fig. 8.13. Comparison of average remaining energy

The difference in average remaining energy of ant colony algorithm when compared against AODV is quite high. AODV, being a single path routing algorithm, consumes more energy over a period of time unlike Ant Colony algorithm which constructs multiple paths and has a packet forwarding mechanism which selects the next hop relative to the proportion of impact factors as explained in Section 6. The error bars around the average energy data point shows the standard deviation of residual energy.

8.4 Hop Count Comparison

Hop Count analysis is studied in this section each time by changing the transmission range of nodes. As shown in the figure below, for every time interval, the average of hop counts for first 30 data packets received by destination are calculated and is compared.

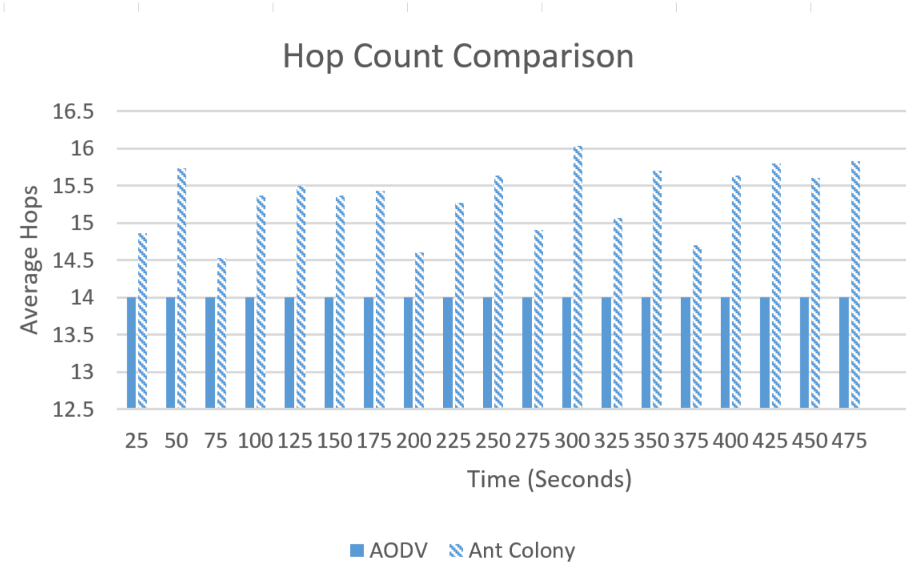


Fig. 8.14. Hop Comparison when Avg degree is 4.94

It can be claimed from these figures that while using AODV routing protocol, once source establishes route towards destination, it uses the same path no matter how long the communication takes place because it follows hop optimal approach.

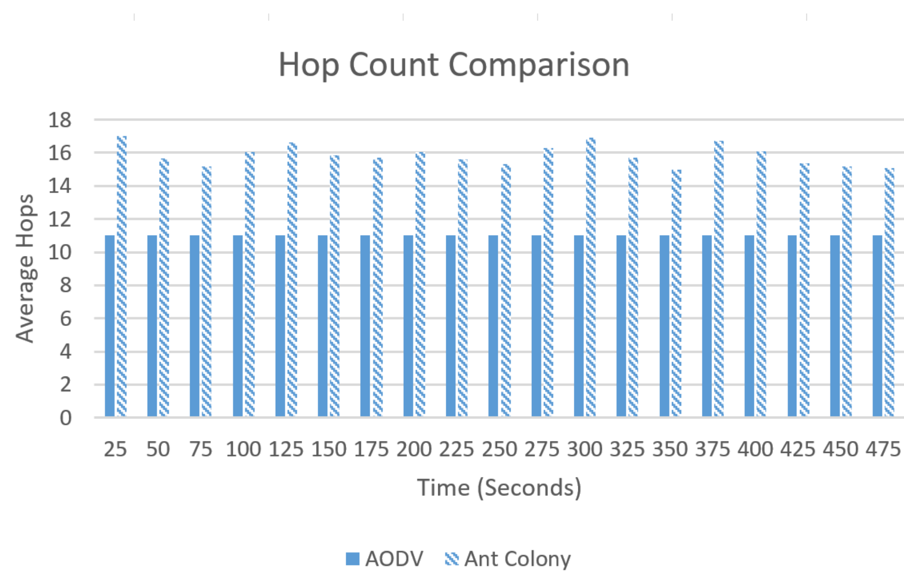


Fig. 8.15. Hop Comparison when Avg degree is 5.94

With increase in average degree of network it is observed that when AODV protocol is used, there is reduction in average hops of packets from source to destination as AODV uses single shortest path route which makes it hop optimal. Whereas Ant Colony protocol selects the next hop relative to the proportion of impact factors which involves randomness. Hence ant colony protocol is not hop optimal.

9. CONCLUSION AND FUTURE WORK

Comprehensive performance analysis of Ant Colony routing algorithm under various load conditions, density of network, etc is carried out in this research. First the challenges faced by sensor networks are described in Section 1.1. The methods to overcome these challenges are elaborated in Section 6 and in Section 7.3 the performance of network is shown by varying the weightage of individual impact factors. Hop distribution and its confidence interval gives the idea of how data packets are forwarded across multiple paths between source and destination.

Performance of Ant Colony routing algorithm was analyzed and compared against AODV routing by considering various performance metrics such as throughput, delay, packet loss, residual energy, etc. In terms of energy analysis, Ant Colony outperforms AODV as the goal for Ant Colony routing is to extend the lifetime of network by constructing multiple paths. There are cases where AODV performs better than Ant Colony routing algorithm since latter involves the decision to forward the packet randomly which doesn't always favour the optimal path unlike AODV, which forms shortest paths between source and destination.

It should be noted that the current version of NS-3 simulator doesn't have the module for Ant Colony routing algorithm. Development of this module required good understanding of C/C++ Programming as well as solid networking background.

In future, error handling mechanism can be introduced in IEEE 802.11 MAC layer by including repellent pheromone as one of the impact factors. Ants detour from their established paths and give emergency signals to their peers when they encounter emergency situations such as an obstacle (stone) placed in between the paths, water flowing through the paths, etc. Repellent pheromone can act like a signal to the nodes in which the failed transmission of data packets can be minimized by preventing the forwarding of packet to bad neighboring node.

REFERENCES

REFERENCES

[1] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, G. Theraulaz *et al.*, *Swarm intelligence: from natural to artificial systems.* Oxford university press, 1999, no. 1.

[2] M. Dorigo and T. Stützle, "Ant colony optimization. 2004," *Massachusetts Institute of Technology*, 2004.

[3] G. Di Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

[4] T. Camilo, C. Carreto, J. S. Silva, and F. Boavida, "An energy-efficient ant-based routing algorithm for wireless sensor networks," in *International workshop on ant colony optimization and swarm intelligence.* Springer, 2006, pp. 49–59.

[5] J.-H. Ho, H.-C. Shih, B.-Y. Liao, and S.-C. Chu, "A ladder diffusion algorithm using ant colony optimization for wireless sensor networks," *Information Sciences*, vol. 192, pp. 204–212, 2012.

[6] M. Frey, F. Große, and M. Günes, "Energy-aware ant routing in wireless multi-hop networks," in *2014 IEEE International Conference on Communications (ICC).* IEEE, 2014, pp. 190–196.

[7] A. M. A. Elmoniem, H. M. Ibrahim, M. H. Mohamed, and A.-R. Hedar, "Ant colony and load balancing optimizations for aodv routing protocol," *International Journal of Sensor Networks and Data Communications*, vol. 1, pp. 1–14, 2012.

[8] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 33, no. 5, pp. 560–572, 2003.

[9] M. Günes, M. Kähmer, and I. Bouazizi, "Ant-routing-algorithm (ara) for mobile multi-hop ad-hoc networks-new features and results," in *Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networks (Med-Hoc-Net03)*, 2003, pp. 9–20.

[10] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," Tech. Rep., 2003.

[11] Z. Zinonos, V. Vassiliou, and T. Christofides, "Radio propagation in industrial wireless sensor network environments: From testbed to simulation evaluation," in *Proceedings of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks.* ACM, 2012, pp. 125–132.

[12] J. Kuruvila, A. Nayak, and I. Stojmenovic, "Hop count optimal position-based packet routing algorithms for ad hoc wireless networks with a realistic physical layer," *IEEE Journal on selected areas in communications*, vol. 23, no. 6, pp. 1267–1275, 2005.

[13] A. Sharma and D. S. Kim, "Robust bio-inspired routing protocol in manets using ant approach," in *International Conference on Ubiquitous Information Management and Communication*. Springer, 2019, pp. 97–110.

[14] Y. Chapre, P. Mohapatra, S. Jha, and A. Seneviratne, "Received signal strength indicator and its analysis in a typical wlan system (short paper)," in *38th Annual IEEE Conference on Local Computer Networks*. IEEE, 2013, pp. 304–307.

[15] S.-g. Jung, B. Kang, S. Yeoum, and H. Choo, "Trail-using ant behavior based energy-efficient routing protocol in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 12, no. 4, p. 7350427, 2016.

[16] N. S. ns3, *Network Simulator*, 2011 (accessed July 2, 2019). [Online]. Available: https://www.nsnam.org/

[17] G. Carneiro, P. Fortuna, and M. Ricardo, "Flowmonitor: a network monitoring framework for the network simulator 3 (ns-3)," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST (Institute for Computer Sciences, Social-Informatics and , 2009, p. 1.

[18] P. D. S. Kim, *AML2FIG*, 2007 (accessed July 3, 2019). [Online]. Available: http://www.engr.iupui.edu/ dskim/Distribution/aml2fig/index.php?section=main

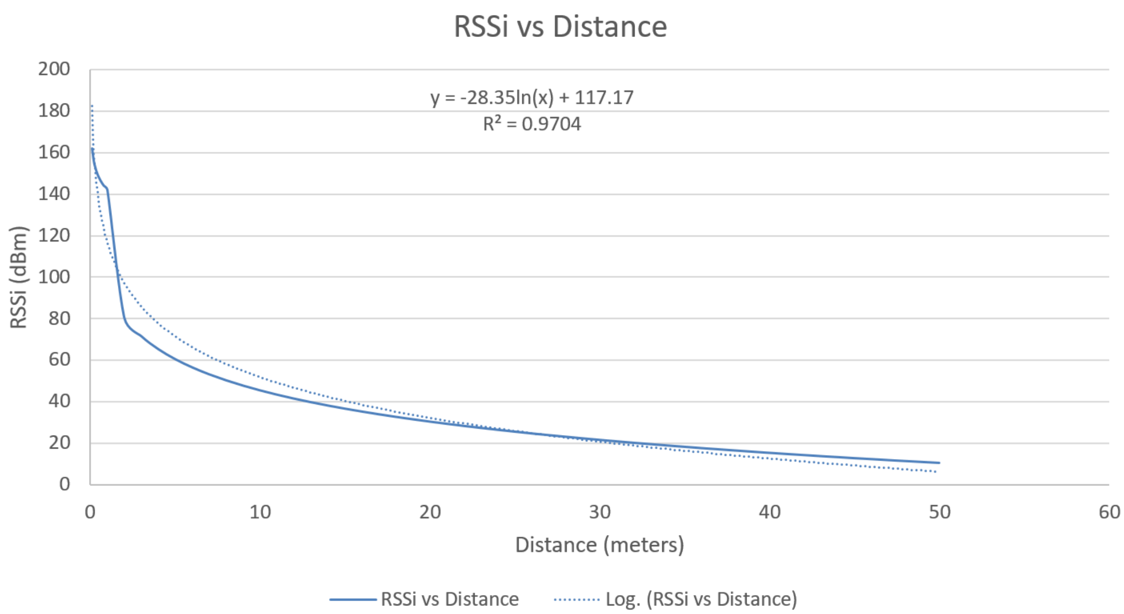
[19] N. Semiconductors, *MKW41Z/31Z/21Z Data Sheet*, 2018 (accessed July 2, 2019). [Online]. Available: https://www.nxp.com/docs/en/datasheet/MKW41Z512.pdf

[20] Energizer, *ENERGIZER CR2032*, (accessed July 2, 2019). [Online]. Available: http://data.energizer.com/pdfs/cr2032.pdf

[21] Mathworks, *MATLAB*, 2019 (accessed July 15, 2019). [Online]. Available: https://www.mathworks.com/products/matlab.html

APPENDICES

A. APPENDIX RSSI VS DISTANCE RELATIONSHIP



```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */

#ifndef antcomm_DPD_H
#define antcomm_DPD_H

#include "antcomm-id-cache.h"
#include "ns3/nstime.h"
#include "ns3/packet.h"
#include "ns3/ipv4-header.h"

namespace ns3
{
namespace antcomm
{
/**
 * \ingroup antcomm
 *
 * \brief Helper class used to remember already seen packets and detect duplicates.
 *
 * Currently duplicate detection is based on uinique packet ID given by Packet::GetUid ()
 * This approach is known to be weak and should be changed.
 */
class DuplicatePacketDetection
{
public:
  /// C-tor
  DuplicatePacketDetection (Time lifetime) : m_idCache (lifetime) {}
  /// Check that the packet is duplicated. If not, save information about this packet.
  bool IsDuplicate (Ptr<const Packet> p, const Ipv4Header & header);
  /// Set duplicate records lifetimes
  void SetLifetime (Time lifetime);
  /// Get duplicate records lifetimes
  Time GetLifetime () const;
private:
  /// Impl
  IdCache m_idCache;
};

}
}

#endif /* antcomm_DPD_H */
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
                      Arush Sharma <sharmaas@iupui.edu>
 */

#include "antcomm-dpd.h"

namespace ns3
{
namespace antcomm
{

bool
DuplicatePacketDetection::IsDuplicate (Ptr<const Packet> p, const Ipv4Header & header)
{
  return m_idCache.IsDuplicate (header.GetSource (), p->GetUid () );
}
void
DuplicatePacketDetection::SetLifetime (Time lifetime)
{
  m_idCache.SetLifetime (lifetime);
}

Time
DuplicatePacketDetection::GetLifetime () const
{
  return m_idCache.GetLifeTime ();
}

}
}
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory, IUPUI
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>, Arush Sharma <sharmaas@iupui.edu>
 */

#include "antcomm-helper.h"
#include "antcomm-protocol.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/energy-module.h"
#include "ns3/wifi-radio-energy-model-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/gnuplot.h"
#include <iostream>
#include <fstream>
#include <cmath>
#include <sstream>
#include "ns3/netanim-module.h"
#include <cstdlib>
#include <vector>
#include <string>
#include "ns3/object.h"
#include "ns3/uinteger.h"
#include "ns3/traced-value.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/v4ping-helper.h"
#include "ns3/packet-sink.h"
#include "ns3/trace-helper.h"
#include "ns3/snr-tag.h"


using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("antcomm");

class antcommExample {
public:
  antcommExample (uint32_t index, uint32_t size);
  /// Configure script parameters, \return true on successful configuration
  bool Configure (int argc, char **argv);
  /// Run simulation
  void Run ();
  /// Report results
  void Report (std::ostream & os);

private:

  // parameters
  uint32_t index;

  /// Number of nodes
  uint32_t size;

  /// Simulation time, seconds
  double totalTime;

  double TxRange; // transmission range to set

  /// Write per-device PCAP traces if true
  bool pcap;
  /// Print routes if true
  bool printRoutes;

  uint32_t explorationTime;

  double distanceForTimer;

  uint32_t packetSize; // bytes
  // Convert to time object
  uint32_t numPackets; // number of packets to send

  double interval; // seconds

  std::string topologyFilePath;

```

```
  // network
  NodeContainer nodes;
  NetDeviceContainer devices;
  Ipv4InterfaceContainer interfaces;
  antcommHelper antcomm;

private:
  void CreateNodes ();
  void CreateDevices ();
  void SetPositions ();
  void InstallInternetStack ();
  void SetDistances();
  void InstallApplications ();
  void MakeCoordinateFile ();
  void RandomNodes ();
  void BackgroundTraffic ();
  void PrintNodesInformation ();
};
void GetStatistics(Ptr<FlowMonitor> flowmon,FlowMonitorHelper& flowmonHelper)
{
  flowmon->CheckForLostPackets ();
  flowmon->SerializeToXmlFile("myproject", true, true);
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmonHelper.GetClassifier () );
  std::map<FlowId, FlowMonitor::FlowStats> stats = flowmon->GetFlowStats ();

  uint32_t txPacketsum = 0;
  uint32_t rxPacketsum = 0;
  uint32_t DropPacketsum = 0;
  uint32_t LostPacketsum = 0;
  double Delaysum = 0;

  std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i;

  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
  {
    txPacketsum += i->second.txPackets;
    rxPacketsum += i->second.rxPackets;
    LostPacketsum += i->second.lostPackets;
    DropPacketsum += i->second.packetsDropped.size();
    Delaysum += i->second.delaySum.GetSeconds();

    std::cout << "Inside␣for␣loop"<<"\n";
          Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);

    std::cout << "Flow␣" << i->first << "␣(" << t.sourceAddress << "␣->␣" << t.destinationAddress << ")";
    std::cout << "Flow␣" << i->first << "␣(" << t.sourceAddress << "␣->␣" << t.destinationAddress << ")\n";
    std::cout << "␣␣Tx␣Bytes:␣␣␣" << i->second.txBytes << "\n";
    std::cout << "␣␣Rx␣Bytes:␣␣␣" << i->second.rxBytes << "\n";
    std::cout << "␣␣Throughput:␣" << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.GetSeconds()) << "␣bps\n";
    std::cout << "␣␣Lost␣Packets:␣␣␣" << i->second.lostPackets << "\n";
    std::cout << "␣␣Dropped␣Packets:␣␣␣" << i->second.packetsDropped.size() << "\n";
    std::cout << "␣␣Bytes␣Dropped:␣␣" << i->second.bytesDropped.size() << "\n";
    std::cout << "␣␣Total␣received␣packets:␣" << i->second.rxPackets << "\n";
    std::cout << "␣␣Total␣transmitted␣packets:␣" << i->second.txPackets <<"\n";
    std::cout << "␣␣Delay␣Sum␣is:␣" << i->second.delaySum.GetSeconds () << "\n";
    std::cout << "␣␣Mean␣Delay:␣␣␣" << i->second.delaySum.GetSeconds () / i->second.rxPackets << "\n";
    std::cout << "␣␣Average␣Hop␣Count:␣" << 1 + (double)i->second.timesForwarded / (double)i->second.rxPackets << "\n";
    std::cout << "␣␣Packet␣Delivery␣Ratio:␣" << i->second.rxPackets *100 / i->second.txPackets <<"%" << "\n";
    std::cout << "␣␣Mean␣jitter:␣␣␣" << i->second.jitterSum.GetSeconds () / (i->second.rxPackets - 1) << "\n";
    std::cout << "␣␣Time:␣␣" << Simulator::Now().GetSeconds() << "\n";
    std::cout << "\n\n";
  }

   std::cout << "␣␣All␣Tx␣Packets:␣" << txPacketsum << "\n";
   std::cout << "␣␣All␣Rx␣Packets:␣" << rxPacketsum << "\n";
   std::cout << "␣␣All␣Delay:␣" << Delaysum / txPacketsum << "\n";
   std::cout << "␣␣All␣Lost␣Packets:␣" << LostPacketsum << "\n";
   std::cout << "␣␣All␣Drop␣Packets:␣" << DropPacketsum << "\n";
   std::cout << "␣␣Packets␣Delivery␣Ratio:␣" << ((rxPacketsum * 100) / txPacketsum) << "%" << "\n";
   std::cout << "␣␣Packets␣Lost␣Ratio:␣" << ((LostPacketsum * 100) / txPacketsum) << "%" << "\n";
}
```

```
void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<FlowMonitor> flowMon)
{
        // p is for previous
  double localThrou=0;
  double delay = 0;
  double avgdelay = 0;
  uint32_t txPacketsum = 0;
  static uint32_t ptxPacketsum = 0;
  uint32_t rxPacketsum = 0;
  static uint32_t prxPacketsum = 0;
  uint32_t DropPacketsum = 0;
  uint32_t LostPacketsum = 0;
  uint64_t packetforwarded = 0;
  uint64_t bytes_Dropped = 0;
  uint64_t rx_Bytes = 0;
  uint64_t tx_Bytes = 0;
  double delaysum = 0;
  static double pdelaysum = 0;
  double hopcount;
  static int count = 0;

  std::map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon->GetFlowStats();
  Ptr<Ipv4FlowClassifier> classing = DynamicCast<Ipv4FlowClassifier> (fmhelper->GetClassifier());
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator stats = flowStats.begin (); stats != flowStats.end (); ++stats)
  {
    Ipv4FlowClassifier::FiveTuple fiveTuple = classing->FindFlow (stats->first);
        if (fiveTuple.destinationPort == 114/*fiveTuple.sourceAddress == "10.0.0.22" && fiveTuple.destinationAddress == "10.0.0.11"*/) // fiveTuple.destinationPort == 114
        {
                if (count > 0)
                {
                        // Do something
                        txPacketsum = stats->second.txPackets /*- ptxPacketsum*/; // modified on 18th feb 2019
                        //printf("Current guy %d Previous guy %d\n", txPacketsum, ptxPacketsum);
                        rxPacketsum = stats->second.rxPackets /*- prxPacketsum*/; // modified on 18th feb 2019
                        delaysum = stats->second.delaySum.GetSeconds () - pdelaysum;
        }
        else
        {
                txPacketsum = stats->second.txPackets;
                rxPacketsum = stats->second.rxPackets;
                delaysum = stats->second.delaySum.GetSeconds ();
        }

          ptxPacketsum = stats->second.txPackets;
          prxPacketsum = stats->second.rxPackets;
          pdelaysum = stats->second.delaySum.GetSeconds ();
      LostPacketsum = stats->second.lostPackets;
      DropPacketsum = stats->second.packetsDropped.size();
      packetforwarded =stats->second.timesForwarded;
      localThrou=(stats->second.rxBytes * 8.0 / (stats->second.timeLastRxPacket.GetSeconds()-stats->second.timeFirstTxPacket.GetSeconds()));
      delay = delaysum /*/ rxPacketsum*/; // modified on 18th feb 2019
      avgdelay = stats->second.delaySum.GetSeconds() / stats->second.rxPackets;
      hopcount = 1 + (double)packetforwarded/(double)(stats->second.rxPackets);
      bytes_Dropped = stats->second.bytesDropped.size();
      rx_Bytes = stats->second.rxBytes;
      tx_Bytes = stats->second.txBytes;

      std::stringstream ss;
      ss << "statistics" << ".csv";
      static std::fstream f (ss.str().c_str(), std::ios::out);

       //f << "Throughput\t\t" <<"Delay\t\t\t\t\t" << "AvgDelay\t\t" << "Bytes_Dropped\t\t" << "Tx_Pkt\t\t" << "Rcvd_Pkt\t\t" << "Lst_Pkts\t\t" << "Dpd_Pkts\t\t" << "Hop Count\t\t" << "Rcvd Bytes\t\t" << "Tx Bytes\t\t" << "Time" << std::endl;
       f <<
                localThrou << "," <<
                delay << "," <<
                avgdelay << "," <<
                bytes_Dropped << "," <<
                txPacketsum << "," <<
                rxPacketsum << "," <<
                LostPacketsum << "," <<
                DropPacketsum << "," <<
                hopcount << "," <<
                rx_Bytes << "," <<
                tx_Bytes << "," <<
```

```
                Simulator::Now().GetSeconds() <<
                std::endl;
        //break;
    }
        }
        count++;
        printf("The counter value is %d\n", count);
        Simulator::Schedule(Seconds(120), &ThroughputMonitor, fmhelper, flowMon);//0.125
        flowMon->SerializeToXmlFile ("ThroughputMonitor.xml", true, true);

}

/// Trace function for remaining energy at node.
template <int node>
void RemainingEnergyTrace (double oldValue, double newValue)
{
  std::stringstream ss;
  ss << "remaining_energy_" << node << ".log";

  static std::fstream f (ss.str().c_str(), std::ios::out);

  f << Simulator::Now().GetSeconds() << " " << newValue << std::endl;
}

/// Trace function for total energy consumption at node.
void
TotalEnergy (double oldValue, double totalEnergy)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds ()
                 << "s Total energy consumed by radio = " << totalEnergy << "J");
}

void EnergyDepletion ()
{
  std::stringstream ss;
  ss << "energy_" << ".log";

  static std::fstream f (ss.str().c_str(), std::ios::out);

  f /*<< "I am dead at Time "*/ << Simulator::Now().GetSeconds() << std::endl;

        NS_LOG_UNCOND (" I am dead at Time " << Simulator::Now ().GetSeconds () << "s");
}

void TTLchecker(Ptr <const Packet> p, Ptr<Ipv4> ipv4, uint32_t test)
{
        static int i = 0;
  static double hopsquared = 0;
  static double hopstotal = 0;
  Ipv4Header header;
  p->PeekHeader (header);
  uint8_t ttl = header.GetTtl ();
  int hops = 64 - (int)ttl + 1;
  if (header.GetSource () == Ipv4Address ("10.0.0.2") && header.GetDestination () == Ipv4Address ("
      10.0.0.99"))
  {
    i++;
    hopsquared += hops*hops;
    hopstotal += hops;
    std::cout << "Total number of packets received " << i << std::endl;
    std::stringstream ss;
    ss << "hopcount" << ".log";

    static std::fstream f (ss.str().c_str(), std::ios::out);

    f << Simulator::Now().GetSeconds ()<<" " /*<< "Received Packet with hopCount "*/ << 64-(int)ttl +
        1 /*<< "\t" << hopstotal << "\t" << hopsquared*/ << std::endl;
  }
}

void SendingPacket (Ptr<const Packet> packet)
{
        static int i = 0;
        std::stringstream ss;
        ss << "packets_sent" << ".txt";

        static std::fstream f (ss.str().c_str(), std::ios::out);
        i++;

        f << "At time " << Simulator::Now().GetSeconds () << " packet " << i << " is sent with size "
            << packet->GetSize () << "\n";
```

```
}

void ReceivingPacket (Ptr<const Packet> packet)
{
       static int i = 0;
       std::stringstream ss;
       ss << "packets_received" << ".txt";

       static std::fstream f (ss.str().c_str(), std::ios::out);
       i++;

       f << "At␣time␣" << Simulator::Now().GetSeconds () << "␣packet␣" << i << "␣is␣received␣with␣
            size␣" << packet->GetSize () << "\n";
}

void ReceivingDrop (Ptr<const Packet> packet)
{
       static int i = 0;
       std::stringstream ss;
       ss << "drop_rx" << ".txt";

       static std::fstream f (ss.str().c_str(), std::ios::out);
       i++;

       f << "At␣time␣" << Simulator::Now().GetSeconds () << "␣packet␣" << i << "␣is␣dropped␣while␣
            receiving␣with␣size␣" << packet->GetSize () << "\n";
}

void SendingDrop (Ptr<const Packet> packet)
{
       static int i = 0;
       std::stringstream ss;
       ss << "drop_tx" << ".txt";

       static std::fstream f (ss.str().c_str(), std::ios::out);
       i++;

       f << "At␣time␣" << Simulator::Now().GetSeconds () << "␣packet␣" << i << "␣is␣dropped␣while␣
            sending␣with␣size␣" << packet->GetSize () << "\n";
}

void Retransmission (Mac48Address value)
{
       static int i = 0;
       std::stringstream ss;
       ss << "transmission" << ".txt";

       static std::fstream f (ss.str().c_str(), std::ios::out);
       i++;
       f << i << "Transmission␣of␣data␣packet␣has␣failed␣at␣time␣" << Simulator::Now().GetSeconds ()
            << "\n";
}

void MaxAttempts (Mac48Address value)
{
       std::stringstream ss;
       ss << "max_tries" << ".txt";

       static std::fstream f (ss.str().c_str(), std::ios::out);
       f << "The␣transmission␣of␣a␣data␣packet␣has␣exceeded␣the␣maximum␣number␣of␣attempts␣at␣time␣"
            << Simulator::Now().GetSeconds () << "\n";
}

int main (int argc, char **argv) {
  uint32_t start = 100;
  uint32_t index = 1;
  antcommExample test(index, start);
  if (!test.Configure (argc, argv))
    NS_FATAL_ERROR ("Configuration␣failed.␣Aborted.");
  test.Run ();
  test.Report (std::cout);
  return 0;
}
void ReceivePacket (Ptr<Socket> socket)
{
  while (socket->Recv ())
    {
      NS_LOG_UNCOND ("Received␣one␣packet!");
    }
}
```

```
static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                             uint32_t pktCount, Time pktInterval )
{
  if (pktCount > 0)
  {
          //Ptr <UniformRandomVariable> x = CreateObject <UniformRandomVariable> ();
          //pktSize = x->GetInteger(200, 1400);
          std::stringstream ss;
      ss << "packet_size" << ".log";

      static std::fstream f (ss.str().c_str(), std::ios::out);

      //f << Simulator::Now().GetSeconds() << " packet size=" << pktSize << std::endl;
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &GenerateTraffic,
                           socket, pktSize, pktCount-1, pktInterval);
  }
  else
  {
      socket->Close ();
   }
}

//-----------------------------------------------------------------------------
antcommExample::antcommExample (uint32_t index, uint32_t size) :
  index (index),
  size (size),
  totalTime (500),
  TxRange (93),
  pcap (true),
  printRoutes (true),
  explorationTime (50),
  distanceForTimer(0.5),
  packetSize(1200),
  numPackets(500000),
  interval(0.125){ // 0.16, 0.12, 0.14

}

bool
antcommExample::Configure (int argc, char **argv)
{
  // Enable AODV logs by default. Comment this if too noisy
   //LogComponentEnable("antcommProtocol", LOG_LEVEL_INFO);
   //LogComponentEnable("antcommRoutingTable", LOG_LEVEL_INFO);

  SeedManager::SetSeed(12345); // 12345
  SeedManager::SetRun(10);

  CommandLine cmd;

  cmd.AddValue ("pcap", "Write␣PCAP␣traces.", pcap);
  cmd.AddValue ("printRoutes", "Print␣routing␣table␣dumps.", printRoutes);
  cmd.AddValue ("size", "Number␣of␣nodes.", size);
  cmd.AddValue ("TxRange", "Transmission␣Range␣to␣set.", TxRange);
  cmd.AddValue ("totalTime", "Simulation␣time,␣s.", totalTime);
  cmd.AddValue ("explorationTime", "Exploation␣time,␣s.", explorationTime);
  cmd.AddValue ("topologyFilePath", "Topology␣file␣path", topologyFilePath);
  cmd.AddValue ("distanceForTimer", "The␣distance␣for␣timer", distanceForTimer);
  cmd.AddValue ("packetSize", "The␣packet␣size", packetSize);
  cmd.AddValue ("numPackets", "The␣number␣of␣packets", numPackets);
  cmd.Parse (argc, argv);


  std::ostringstream s;
  s << "output/topology_case181_" << index << "_" << size << ".csv";
  topologyFilePath = s.str();
  return true;
}

void antcommExample::Run () {
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", UintegerValue (2200)); // enable rts cts all the time.

  CreateNodes ();
  SetPositions();
  CreateDevices ();
  BasicEnergySourceHelper basicSourceHelper;
  // I am changing the battery capacity in order to make the simulation run for full 500 seconds
```

```
basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (0.4)); // 470 1102. 1270 is for 120 meters and 1237 for 99 meters

EnergySourceContainer sources = basicSourceHelper.Install (nodes);

WifiRadioEnergyModelHelper radioEnergyHelper;

radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0061)); //0.0061 and 0.4 joules as initial energy
radioEnergyHelper.Set ("RxCurrentA", DoubleValue (0.0068)); //0.0068
radioEnergyHelper.Set ("IdleCurrentA", DoubleValue (0.00019)); //0.00019

radioEnergyHelper.SetDepletionCallback (MakeCallback (&EnergyDepletion));
DeviceEnergyModelContainer deviceModels = radioEnergyHelper.Install (devices, sources);
InstallInternetStack ();
MakeCoordinateFile ();
Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/RemoteStationManager/MacTxDataFailed", MakeCallback (&Retransmission));
Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/RemoteStationManager/MacTxFinalDataFailed", MakeCallback (&MaxAttempts));
Config::ConnectWithoutContext ("/NodeList/98/$ns3::Ipv4L3Protocol/Rx", MakeCallback(&TTLchecker));//99
Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacTx", MakeCallback (&SendingPacket));
Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacRx", MakeCallback (&ReceivingPacket));
Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacRxDrop", MakeCallback (&ReceivingDrop));
Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/MacTxDrop", MakeCallback (&SendingDrop));
//Simulator::Schedule (Seconds(50), &antcommExample::RandomNodes, this);
//Simulator::Schedule (Seconds(100), &antcommExample::RandomNodes, this);
//Simulator::Schedule (Seconds(150), &antcommExample::RandomNodes, this);
//Simulator::Schedule (Seconds(200), &antcommExample::RandomNodes, this);
//RandomNodes ();
InstallApplications ();
/*for (uint32_t i = 0 ; i < size; i++)
{
     // energy source
  Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (sources.Get (i));
  basicSourcePtr->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback (&RemainingEnergy));

  // device energy model
  Ptr<DeviceEnergyModel> basicRadioModelPtr = basicSourcePtr->FindDeviceEnergyModels ("ns3::WifiRadioEnergyModel").Get (0);
  NS_ASSERT (basicRadioModelPtr != NULL);
  basicRadioModelPtr->TraceConnectWithoutContext ("TotalEnergyConsumption", MakeCallback (&TotalEnergy));

}*/
// Tried the loop way, but it doesn't work. If there's another solution to it, that would be good
sources.Get (0)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<0>));
sources.Get (1)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<1>));
sources.Get (2)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<2>));
sources.Get (3)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<3>));
sources.Get (4)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<4>));
sources.Get (5)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<5>));
sources.Get (6)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<6>));
sources.Get (7)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<7>));
sources.Get (8)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<8>));
sources.Get (9)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<9>));
sources.Get (10)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<10>));
sources.Get (11)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<11>));
sources.Get (12)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<12>));
sources.Get (13)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<13>));
```

```
sources.Get (14)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<14>));
sources.Get (15)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<15>));
sources.Get (16)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<16>));
sources.Get (17)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<17>));
sources.Get (18)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<18>));
sources.Get (19)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<19>));
sources.Get (20)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<20>));
sources.Get (21)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<21>));
sources.Get (22)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<22>));
sources.Get (23)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<23>));
sources.Get (24)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<24>));
sources.Get (25)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<25>));
sources.Get (26)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<26>));
sources.Get (27)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<27>));
sources.Get (28)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<28>));
sources.Get (29)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<29>));
sources.Get (30)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<30>));
sources.Get (31)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<31>));
sources.Get (32)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<32>));
sources.Get (33)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<33>));
sources.Get (34)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<34>));
sources.Get (35)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<35>));
sources.Get (36)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<36>));
sources.Get (37)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<37>));
sources.Get (38)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<38>));
sources.Get (39)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<39>));
sources.Get (40)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<40>));
sources.Get (41)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<41>));
sources.Get (42)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<42>));
sources.Get (43)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<43>));
sources.Get (44)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<44>));
sources.Get (45)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<45>));
sources.Get (46)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<46>));
sources.Get (47)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<47>));
sources.Get (48)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<48>));
sources.Get (49)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<49>));
sources.Get (50)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<50>));
sources.Get (51)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<51>));
sources.Get (52)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<52>));
sources.Get (53)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<53>));
```

```
sources.Get (54)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<54>));
sources.Get (55)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<55>));
sources.Get (56)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<56>));
sources.Get (57)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<57>));
sources.Get (58)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<58>));
sources.Get (59)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<59>));
sources.Get (60)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<60>));
sources.Get (61)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<61>));
sources.Get (62)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<62>));
sources.Get (63)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<63>));
sources.Get (64)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<64>));
sources.Get (65)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<65>));
sources.Get (66)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<66>));
sources.Get (67)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<67>));
sources.Get (68)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<68>));
sources.Get (69)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<69>));
sources.Get (70)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<70>));
sources.Get (71)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<71>));
sources.Get (72)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<72>));
sources.Get (73)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<73>));
sources.Get (74)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<74>));
sources.Get (75)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<75>));
sources.Get (76)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<76>));
sources.Get (77)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<77>));
sources.Get (78)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<78>));
sources.Get (79)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<79>));
sources.Get (80)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<80>));
sources.Get (81)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<81>));
sources.Get (82)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<82>));
sources.Get (83)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<83>));
sources.Get (84)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<84>));
sources.Get (85)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<85>));
sources.Get (86)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<86>));
sources.Get (87)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<87>));
sources.Get (88)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<88>));
sources.Get (89)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<89>));
sources.Get (90)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<90>));
sources.Get (91)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<91>));
sources.Get (92)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<92>));
sources.Get (93)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<93>));
```

```
  sources.Get (94)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<94>));
  sources.Get (95)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<95>));
  sources.Get (96)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<96>));
  sources.Get (97)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<97>));
  sources.Get (98)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<98>));
  sources.Get (99)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace<99>));


  std::cout << "Starting␣simulation␣antcomm␣case181␣No.␣" << index << "␣for␣" << totalTime << "␣s␣...\n";

  //std::ostringstream s;
  //s << "animation_" << index << "_" << size << ".xml";
  /*Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowHelper;
  flowMonitor = flowHelper.InstallAll();*/
  FlowMonitorHelper* flowmonHelper = new FlowMonitorHelper();
  Ptr<FlowMonitor> flowmon = flowmonHelper->InstallAll ();

  ThroughputMonitor(flowmonHelper, flowmon); // just changed this

  Simulator::Stop (Seconds (totalTime));
  //AnimationInterface anim (s.str());
  AnimationInterface anim ("animation.xml");
  anim.UpdateNodeColor(nodes.Get(1), 0, 0, 255);
  anim.UpdateNodeColor(nodes.Get(98), 0 ,0, 255);
  Simulator::Run ();
  /*for (DeviceEnergyModelContainer::Iterator iter = deviceModels.Begin (); iter != deviceModels.End (); iter ++)
  {
    double energyConsumed = (*iter)->GetTotalEnergyConsumption ();
    NS_LOG_UNCOND ("End of simulation (" << Simulator::Now ().GetSeconds () << "s) Total energy consumed by radio = " << energyConsumed << "J");
  }*/
  PrintNodesInformation();
  antcommHelper::EmptyAllInformation();
  //ThroughputMonitor(flowmonHelper, flowmon);
  GetStatistics(flowmon,*flowmonHelper);
  Simulator::Destroy ();
}

void antcommExample::Report (std::ostream &) {
}

void antcommExample::CreateNodes () {
  NS_LOG_UNCOND ("creating␣the␣nodes");
  // NodeContainer sinkNode;
  // NodeContainer sensorNodes;
  std::cout << "Creating␣" << (unsigned)size << "␣nodes.\n";
  // sinkNode.Create(1);
  // sensorNodes.Create(size);

  nodes.Create(size);

  //Name sink
  /* std::ostringstream os;
        os << "sink";
        Names::Add (os.str (), nodes.Get (0));

        // Name sensor nodes
        for (uint32_t i = 1; i < size; ++i) {
        std::ostringstream os;
        os << "node-" << i;
        Names::Add (os.str (), nodes.Get (i));
        }

        // nodes = NodeContainer(sinkNode, sensorNodes); */
}

void antcommExample::CreateDevices () {
  NS_LOG_UNCOND ("setting␣the␣default␣phy␣and␣channel␣parameters");
  //double txp = 87; // 87 // 60
  WifiMacHelper wifiMac;
  wifiMac.SetType ("ns3::AdhocWifiMac");
```

```
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  //wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
  //wifiPhy.Set ("TxPowerEnd",DoubleValue (150)); // 95
  //wifiPhy.Set ("TxPowerLevels", UintegerValue (60));
  //wifiPhy.Set ("ChannelWidth", UintegerValue (160));
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  //wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
  /*For the current topology, 63 meter is minimum transmission range and 71 meter for background traffic*/
  wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel", "MaxRange", DoubleValue(TxRange)); //15 22
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
  wifiPhy.SetChannel (wifiChannel.Create ());
  WifiHelper wifi;
  //wifi.SetStandard (WIFI_PHY_STANDARD_80211g);
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringValue ("OfdmRate6Mbps")); // OfdmRate6Mbps
  devices = wifi.Install (wifiPhy, wifiMac, nodes);
  if (pcap) {
    wifiPhy.EnablePcapAll (std::string ("antcomm"));
  }
}

void antcommExample::SetPositions() {

  NS_LOG_UNCOND ("Assigning mobility");
  MobilityHelper mobility;
  int px, py/*, dpx, dpy*/;
  Ptr<ListPositionAllocator> node_coordinates = CreateObject<ListPositionAllocator> ();
  Ptr<NormalRandomVariable> x = CreateObject<NormalRandomVariable> ();
  x->SetAttribute ("Variance", DoubleValue (99));
  FILE *stream/*, *stream2*/;
  stream = fopen("/export/home1/sharmaas/newworkspace/ns-allinone-3.25/ns-3.25/scratch/antcomm/cellular_layout.txt", "r"); // grid_layout
  //stream2 = fopen("/export/home1/sharmaas/newworkspace/ns-allinone-3.25/ns-3.25/scratch/antcomm/cellular_layout.txt", "w");
  if (stream == NULL)
    printf ("FILE not found\n");
  for(uint32_t i = 0; i < size; i++)
  {
    fscanf(stream, "%d %d", &px, &py);
    //dpx = px + x->GetInteger ();
    //dpy = py + x->GetInteger ();
    //fprintf(stream2, "%d %d\n", dpx, dpy);
    node_coordinates->Add (Vector (px, py, 0.0));
  }
  fclose(stream);
  //fclose(stream2);
  mobility.SetPositionAllocator(node_coordinates);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  /*double distance = 50;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (distance),
                                 "DeltaY", DoubleValue (distance),
                                 "GridWidth", UintegerValue (10),
                                 "LayoutType", StringValue ("RowFirst"));*/
  mobility.Install (nodes);

}


void antcommExample::PrintNodesInformation(){

  std::ostringstream s;
  s << "antcomm.nodes.information_case181_" << index << "_" << size;
  Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> (s.str(), std::ios::out);
  antcomm.PrintNodesInformation(routingStream);
}

void antcommExample::InstallInternetStack () {
  // you can configure AODV attributes here using aodv.Set(name, value)
  //antree.Set("m_sink", );
  //antree.Set("TransmissionRange", UintegerValue (55));
  antcommHelper antcomm;
  InternetStackHelper stack;
  antcomm.Set("DistanceForTimer", DoubleValue(distanceForTimer));
  antcomm.Set("Alpha", DoubleValue(0.5));
  antcomm.Set("Beta", DoubleValue(0.8));
```

```
  antcomm.Set("Gamma", DoubleValue(0.8));
  stack.SetRoutingHelper (antcomm); // has effect on the next Install ()
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);

  if (printRoutes) {
    Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("antree.routes", std::ios::out);
    antcomm.PrintRoutingTableAllAt (Seconds (totalTime), routingStream);
  }
}

void antcommExample::RandomNodes ()
{
  Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable> ();
  x->SetAttribute ("Min", DoubleValue (0));
  x->SetAttribute ("Max", DoubleValue (size-1));
  uint32_t sourceNode = x->GetInteger();
  uint32_t sinkNode = x->GetInteger();
  while (sourceNode==sinkNode || sourceNode==98 || sourceNode==1 || sinkNode==98 || sinkNode==1)
  {
    sourceNode = x->GetInteger();
    sinkNode = x->GetInteger();
  }
    UdpServerHelper myServer (14);
    ApplicationContainer serverApp = myServer.Install (nodes.Get (sinkNode));
    serverApp.Start (Seconds (1.0));
    serverApp.Stop (Seconds (totalTime));

    UdpClientHelper client (interfaces.GetAddress (sinkNode), 14);
    client.SetAttribute ("MaxPackets", UintegerValue (14000000));
    client.SetAttribute ("Interval", TimeValue (Time ("0.125"))); // 0.125 packets/s
    client.SetAttribute ("PacketSize", UintegerValue (1500)); //bytes

    ApplicationContainer clientApp = client.Install (nodes.Get (sourceNode));
    clientApp.Start (Seconds (4.0));//3 4
    clientApp.Stop (Seconds (totalTime));
  /*Time interPacketInterval = Seconds (interval);
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> recvSink = Socket::CreateSocket (nodes.Get (sinkNode), tid);
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 14);
  recvSink->Bind (local);
  recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

  Ptr<Socket> source = Socket::CreateSocket (nodes.Get (sourceNode), tid);
  InetSocketAddress remote = InetSocketAddress (interfaces.GetAddress (sinkNode, 0), 14);
  source->Connect (remote);
  Simulator::Schedule (Seconds (5.0), &GenerateTraffic, source, packetSize, numPackets, interPacketInterval);*/

  //Simulator::Schedule(Seconds(50), &antcommExample::RandomNodes, this);
}

void antcommExample::InstallApplications () {

  NS_LOG_UNCOND ("Traffic␣generator␣for␣data␣packet␣generation");
  UdpServerHelper myServer (114);
  ApplicationContainer serverApp = myServer.Install (nodes.Get (98)); //96
  serverApp.Start (Seconds (1.0));
  serverApp.Stop (Seconds (totalTime));

  UdpClientHelper client (interfaces.GetAddress (98), 114);
  client.SetAttribute ("MaxPackets", UintegerValue (14000000));
  client.SetAttribute ("Interval", TimeValue (Time ("120"))); // 0.125 0.08 packets/s
  client.SetAttribute ("PacketSize", UintegerValue (1500)); //bytes

  ApplicationContainer clientApp = client.Install (nodes.Get (1)); //4 or 9
  clientApp.Start (Seconds (3.0));//3
  clientApp.Stop (Seconds (totalTime));
  /*uint32_t sourceNode = 1;
  uint32_t sinkNode = 98;
  Time interPacketInterval = Seconds (interval);
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> recvSink = Socket::CreateSocket (nodes.Get (sinkNode), tid);
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 114);
  recvSink->Bind (local);
  recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

```

```
  Ptr<Socket> source = Socket::CreateSocket (nodes.Get (sourceNode), tid);
  InetSocketAddress remote = InetSocketAddress (interfaces.GetAddress (sinkNode, 0), 114);
  source->Connect (remote);
  Simulator::Schedule (Seconds (3.0), &GenerateTraffic, source, packetSize, numPackets,
       interPacketInterval);*/
}
void antcommExample::MakeCoordinateFile ()
{
  FILE *fp;
  fp = fopen ("id2coordinates.txt", "w");
  uint32_t i;
  for (i=0; i < size; i++)
  {
    uint32_t id, x1, y1;
    Ptr <Node> node = nodes.Get (i);
    id = node->GetId ();
    Ptr <MobilityModel> mob = node->GetObject<MobilityModel> ();
    Vector position = mob->GetPosition ();
    x1 = position.x;
    y1 = position.y;
    fprintf (fp, "%d\t%d\t%d\n", id, x1, y1);
  }
  fclose (fp);
}
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

#include "antcomm-helper.h"
#include "antcomm-protocol.h"
#include "ns3/core-module.h"
#include "ns3/node-list.h"
#include "ns3/names.h"
#include "ns3/ptr.h"
#include "ns3/ipv4-list-routing.h"

namespace ns3 {

  // std::map<std::string, double> antcommHelper::m_distances;
  //Line commented out by me

  // std::map<uint16_t, uint16_t> antcommHelper::m_hopCount;
  // Line commented out by me

  // std::map<uint16_t, double> antcommHelper::m_energy;
 // Line commented out by me

  // std::map<uint16_t, double> antcommHelper::m_pheromone;
  // Line commented out by me

  // std::map<uint16_t, double> antcommHelper::m_pheromoneUpdatedTime; // Line commented out by me

  // std::map<uint32_t, Time > antcommHelper::m_startTimestamps; // Line commented out by me

  uint32_t antcommHelper::dataPacketID = 0;


  antcommHelper::antcommHelper() : Ipv4RoutingHelper () {
    m_agentFactory.SetTypeId ("ns3::antcomm::antcommProtocol");
  }

  antcommHelper* antcommHelper::Copy (void) const {
    return new antcommHelper (*this);
  }

  Ptr<Ipv4RoutingProtocol> antcommHelper::Create (Ptr<Node> node) const {
    Ptr<antcomm::antcommProtocol> agent = m_agentFactory.Create<antcomm::antcommProtocol> ();
    node->AggregateObject (agent);
    return agent;
  }

  void antcommHelper::Set (std::string name, const AttributeValue &value) {
    m_agentFactory.Set (name, value);
  }


  int64_t antcommHelper::AssignStreams (NodeContainer c, int64_t stream) {
    int64_t currentStream = stream;
    Ptr<Node> node;
    for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i) {
      node = (*i);
      Ptr<Ipv4> ipv4 = node->GetObject<Ipv4> ();
      NS_ASSERT_MSG (ipv4, "Ipv4␣not␣installed␣on␣node");
      Ptr<Ipv4RoutingProtocol> proto = ipv4->GetRoutingProtocol ();
      NS_ASSERT_MSG (proto, "Ipv4␣routing␣not␣installed␣on␣node");
      Ptr<antcomm::antcommProtocol> antcomm = DynamicCast<antcomm::antcommProtocol> (proto);
      if (antcomm) {
        currentStream += antcomm->AssignStreams (currentStream);
        continue;
      }
      // antcomm may also be in a list
      Ptr<Ipv4ListRouting> list = DynamicCast<Ipv4ListRouting> (proto);
      if (list) {
        int16_t priority;
        Ptr<Ipv4RoutingProtocol> listProto;
        Ptr<antcomm::antcommProtocol> listantcomm;
        for (uint32_t i = 0; i < list->GetNRoutingProtocols (); i++) {
          listProto = list->GetRoutingProtocol (i, priority);
          listantcomm = DynamicCast<antcomm::antcommProtocol> (listProto);
          if (listantcomm) {
            currentStream += listantcomm->AssignStreams (currentStream);
            break;
          }
        }
      }
    }
```

```
    return (currentStream - stream);
  }

  void antcommHelper::PrintNodesInformation(Ptr<OutputStreamWrapper> stream) {

    *stream->GetStream () << "ID\tHopCount\tPheromone\tEnergy\tRecvPacketCount\tHopDistance\
        tEndToEndDelay\n";
    for (uint32_t i = 0; i < NodeList::GetNNodes (); i++) {
      Ptr<Node> node = NodeList::GetNode (i);
      Ptr<Ipv4> ipv4 = node->GetObject<Ipv4> ();
      Ptr<Ipv4RoutingProtocol> proto = ipv4->GetRoutingProtocol ();
      Ptr<antcomm::antcommProtocol> antcomm = DynamicCast<antcomm::antcommProtocol> (proto);
      if (antcomm) {
        antcomm->PrintInformation(stream);
      }
    }
  }



}
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
#ifndef antcomm_HELPER_H
#define antcomm_HELPER_H

#include "ns3/object-factory.h"
#include "ns3/node.h"
#include "ns3/node-container.h"
#include "ns3/ipv4-routing-helper.h"
#include "ns3/wifi-module.h"
#include "ns3/energy-module.h"

namespace ns3 {

    /**
     * \ingroup antcomm
     * \brief Helper class that adds antcomm routing to nodes.
     */
    class antcommHelper : public Ipv4RoutingHelper {
                public:
                        antcommHelper();

                        /**
                        * \returns pointer to clone of this OlsrHelper
                        *
                        * \internal
                        * This method is mainly for internal use by the other helpers;
                        * clients are expected to free the dynamic memory allocated by this method
                        */
                        antcommHelper* Copy (void) const;

                        /**
                        * \param node the node on which the routing protocol will run
                        * \returns a newly-created routing protocol
                        *
                        * This method will be called by ns3::InternetStackHelper::Install
                        *
                        * \todo support installing AODV on the subset of all available IP interfaces
                        */
                        virtual Ptr<Ipv4RoutingProtocol> Create (Ptr<Node> node) const;
                        /**
                        * \param name the name of the attribute to set
                        * \param value the value of the attribute to set.
                        *
                        * This method controls the attributes of ns3::aodv::RoutingProtocol
                        */
                        void Set (std::string name, const AttributeValue &value);
                        /**
                        * Assign a fixed random variable stream number to the random variables
                        * used by this model. Return the number of streams (possibly zero) that
                        * have been assigned. The Install() method of the InternetStackHelper
                        * should have previously been called by the user.
                        *
                        * \param stream first stream index to use
                        * \param c NodeContainer of the set of nodes for which AODV
                        * should be modified to use a fixed stream
                        * \return the number of stream indices assigned by this helper
                        */
                        int64_t AssignStreams (NodeContainer c, int64_t stream);

                        void PrintNodesInformation (Ptr<OutputStreamWrapper> stream);


                    static void PutDistance(uint16_t sourceID, uint16_t destinationID, double distance){

                                std::stringstream buffer;
                                buffer << sourceID << "|" << destinationID;
                                antcommHelper::m_distances[buffer.str()] = distance; // Line commented out by me
                    }


                    static void PutPheromone(uint16_t nodeID, double pheromone){
                        //antcommHelper::m_pheromone[nodeID] = pheromone;
                        PutPheromoneUpdatedTime(nodeID, Simulator::Now ());
               }

                        static double GetPheromone(Ipv4Address dst/*uint16_t nodeID*/){

                                //std::map<uint16_t,double>::iterator result = antcommHelper::m_pheromone.find(nodeID);
```

```
                        //if (result != antcommHelper::m_pheromone.end()){
                                Time offset = Simulator::Now() - Time(GetPheromoneUpdatedTime(dst/*nodeID*/));
                                //return exp( (-1)*offset.GetSeconds()*0.5 ) * result->second;
                        //}else{
                                //return -1;
    return 0; //Line inserted by me
                        }
                //}

            static void PutEnergy(uint16_t nodeID, double energy){
            // antcommHelper::m_energy[nodeID] = energy; //Line commented out by me
      //BasicEnergySourceHelper basicSourceHelper;

      //basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (2.0));

      //EnergySourceContainer sources = basicSourceHelper.Install (nodes);
      //Device Energy Model
      //WifiRadioEnergyModelHelper radioEnergyHelper;

      //radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0174));

      //DeviceEnergyModelContainer deviceModels = radioEnergyHelper.Install (devices, sources);
            }

                static double GetEnergy(Ipv4Address dst/*uint16_t nodeID*/){
                        /*std::map<uint16_t,double>::iterator result = antcommHelper::m_energy.find(nodeID);
                        if (result != antcommHelper::m_energy.end()){
                                return result->second;
                        }else{
                                return -1;
                        } */
  return 0;
                }

            static void PutPheromoneUpdatedTime(uint16_t nodeID, Time time){
            // antcommHelper::m_pheromoneUpdatedTime[nodeID] = time.GetDouble();
            }

                static double GetPheromoneUpdatedTime(Ipv4Address dst/*uint16_t nodeID*/){
                        /*std::map<uint16_t,double>::iterator result = antcommHelper::m_pheromoneUpdatedTime.find(nodeID);
                        if (result != antcommHelper::m_pheromoneUpdatedTime.end()){
                                return result->second;
                        }
    else{
                                return -1;
                        } */
  return 0;
                }


        static void PutHopCount(uint16_t nodeID, uint16_t hopCount){
   // antcommHelper::m_hopCount[nodeID] = hopCount; // Line commented out by me
            }

                static uint16_t GetHopCount(Ipv4Address dst/*uint16_t nodeID*/){
                        /*std::map<uint16_t,uint16_t>::iterator result = antcommHelper::m_hopCount.find(nodeID);
                        if (result != antcommHelper::m_hopCount.end()){
                                return result->second;
                        }
   else{
                                return -1;
                        }*/
  return 0; //Line inserted by me as its a non void function
                }



                static void EmptyAllInformation(){
                        // m_distances.clear(); //Line commented out by me
                        // m_hopCount.clear(); //Line commented out by me
                        // m_energy.clear(); //Line commented out by me
                        // m_pheromone.clear(); //Line commented out by me
                        // m_pheromoneUpdatedTime.clear(); // Line commented out by me
                }
```

```
                    static uint32_t PutStartTimestamp(){
                        m_startTimestamps[dataPacketID] = Simulator::Now();
                                return dataPacketID++;
                    }

                    static double GetEndToEndDelay(uint32_t packetID){
                                //std::map<uint32_t,Time>::iterator result = m_startTimestamps.find(packetID);
                                //return Simulator::Now().GetMilliSeconds() - result->second.GetMilliSeconds() ;
         return 0;
                    }

                private:
                        /** the factory to create antcomm routing object */
                        ObjectFactory m_agentFactory;

                        static std::map<std::string, double> m_distances;

                        static std::map<uint16_t, uint16_t> m_hopCount;

                        static std::map<uint16_t, double> m_energy;

                        static std::map<uint16_t, double> m_pheromone;

                        static std::map<uint16_t, double> m_pheromoneUpdatedTime;

                        static std::map<uint32_t, Time > m_startTimestamps;

                  static uint32_t dataPacketID;
    };


}

#endif /* antcomm_HELPER_H */
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Based on
 * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * antcomm-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/antcomm-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */
#include "antcomm-id-cache.h"
#include <algorithm>

namespace ns3
{
namespace antcomm
{
bool
IdCache::IsDuplicate (Ipv4Address addr, uint32_t id)
{
  Purge ();
  for (std::vector<UniqueId>::const_iterator i = m_idCache.begin ();
       i != m_idCache.end (); ++i)
    if (i->m_context == addr && i->m_id == id)
      return true;
  struct UniqueId uniqueId =
  { addr, id, m_lifetime + Simulator::Now () };
  m_idCache.push_back (uniqueId);
  return false;
}
void
IdCache::Purge ()
{
  m_idCache.erase (remove_if (m_idCache.begin (), m_idCache.end (),
                              IsExpired ()), m_idCache.end ());
}

uint32_t
IdCache::GetSize ()
{
  Purge ();
  return m_idCache.size ();
}

}
}
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Based on
 * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * antcomm-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/antcomm-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */

#ifndef antcomm_ID_CACHE_H
#define antcomm_ID_CACHE_H

#include "ns3/ipv4-address.h"
#include "ns3/simulator.h"
#include <vector>

namespace ns3
{
namespace antcomm
{
/**
 * \ingroup antcomm
 *
 * \brief Unique packets identification cache used for simple duplicate detection.
 */
class IdCache
{
public:
  /// c-tor
  IdCache (Time lifetime) : m_lifetime (lifetime) {}
  /// Check that entry (addr, id) exists in cache. Add entry, if it doesn't exist.
  bool IsDuplicate (Ipv4Address addr, uint32_t id);
  /// Remove all expired entries
  void Purge ();
  /// Return number of entries in cache
  uint32_t GetSize ();
  /// Set lifetime for future added entries.
  void SetLifetime (Time lifetime) { m_lifetime = lifetime; }
  /// Return lifetime for existing entries in cache
  Time GetLifeTime () const { return m_lifetime; }
private:
  /// Unique packet ID
  struct UniqueId
  {
    /// ID is supposed to be unique in single address context (e.g. sender address)
    Ipv4Address m_context;
    /// The id
    uint32_t m_id;
    /// When record will expire
    Time m_expire;
  };
  struct IsExpired
  {
    bool operator() (const struct UniqueId & u) const
    {
      return (u.m_expire < Simulator::Now ());
    }
  };
  /// Already seen IDs
  std::vector<UniqueId> m_idCache;
  /// Default lifetime for ID records
```

```
  Time m_lifetime;
};

}
}
#endif /* antcomm_ID_CACHE_H */
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Based on
 * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * antcomm-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/antcomm-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */

#include "antcomm-neighbor.h"
#include "ns3/log.h"
#include <algorithm>


namespace ns3
{

NS_LOG_COMPONENT_DEFINE ("antcommNeighbors");

namespace antcomm
{
Neighbors::Neighbors (Time delay) :
  m_ntimer (Timer::CANCEL_ON_DESTROY)
{
  m_ntimer.SetDelay (delay);
  m_ntimer.SetFunction (&Neighbors::Purge, this);
  m_txErrorCallback = MakeCallback (&Neighbors::ProcessTxError, this);
}

bool
Neighbors::IsNeighbor (Ipv4Address addr)
{
  Purge ();
  for (std::vector<Neighbor>::const_iterator i = m_nb.begin ();
       i != m_nb.end (); ++i)
    {
      if (i->m_neighborAddress == addr)
        return true;
    }
  return false;
}

Time
Neighbors::GetExpireTime (Ipv4Address addr)
{
  Purge ();
  for (std::vector<Neighbor>::const_iterator i = m_nb.begin (); i
       != m_nb.end (); ++i)
    {
      if (i->m_neighborAddress == addr)
        return (i->m_expireTime - Simulator::Now ());
    }
  return Seconds (0);
}

void
Neighbors::Update (Ipv4Address addr, double snr, double energy, Time expire)
{
  for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
    if (i->m_neighborAddress == addr)
      {
```

```
        i->m_expireTime
          = std::max (expire + Simulator::Now (), i->m_expireTime);
        if (i->m_hardwareAddress == Mac48Address ())
          i->m_hardwareAddress = LookupMacAddress (i->m_neighborAddress);
        return;
      }

  NS_LOG_LOGIC ("Open␣link␣to␣" << addr);
  Neighbor neighbor (addr, LookupMacAddress (addr), snr, energy, expire + Simulator::Now ());
  m_nb.push_back (neighbor);
  Purge ();
}

int Neighbors::Degree ()
{
  return m_nb.size();
}

Ipv4Address Neighbors::GetAddress (int i)
{
  return m_nb[i].m_neighborAddress;
}

double Neighbors::GetEnergy (Ipv4Address addr)
{
  for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
  {
    if (i->m_neighborAddress == addr)
      return i->residualEnergy;
  }
  NS_LOG_LOGIC ("Neighbor␣not␣found␣yet␣so␣return␣zero␣energy");
  return 0;

}

double Neighbors::GetRssi (Ipv4Address addr)
{
        for (std::vector<Neighbor>::iterator i = m_nb.begin (); i !=m_nb.end (); ++i)
        {
                if (i->m_neighborAddress == addr)
                        return i->signalnoiseratio;
        }
        NS_LOG_LOGIC ("Neighbor␣not␣found␣yet␣so␣RSSi␣would␣be␣0");
        return 0;
}
struct CloseNeighbor
{
  bool operator() (const Neighbors::Neighbor & nb) const
  {
    return ((nb.m_expireTime < Simulator::Now ()) || nb.close);
  }
};

void
Neighbors::Purge ()
{
  if (m_nb.empty ())
    return;

  CloseNeighbor pred;
  if (!m_handleLinkFailure.IsNull ())
    {
      for (std::vector<Neighbor>::iterator j = m_nb.begin (); j != m_nb.end (); ++j)
        {
          if (pred (*j))
            {
              NS_LOG_LOGIC ("Close␣link␣to␣" << j->m_neighborAddress);
              m_handleLinkFailure (j->m_neighborAddress);
            }
        }
    }
  m_nb.erase (std::remove_if (m_nb.begin (), m_nb.end (), pred), m_nb.end ());
  m_ntimer.Cancel ();
  m_ntimer.Schedule ();
}

void
Neighbors::ScheduleTimer ()
{
  m_ntimer.Cancel ();
```

```
  m_ntimer.Schedule ();
}

void
Neighbors::AddArpCache (Ptr<ArpCache> a)
{
  m_arp.push_back (a);
}

void
Neighbors::DelArpCache (Ptr<ArpCache> a)
{
  m_arp.erase (std::remove (m_arp.begin (), m_arp.end (), a), m_arp.end ());
}

Mac48Address
Neighbors::LookupMacAddress (Ipv4Address addr)
{
  Mac48Address hwaddr;
  for (std::vector<Ptr<ArpCache> >::const_iterator i = m_arp.begin ();
       i != m_arp.end (); ++i)
    {
      ArpCache::Entry * entry = (*i)->Lookup (addr);
      if (entry != 0 && (entry->IsAlive () || entry->IsPermanent ()) && !entry->IsExpired ())
        {
          hwaddr = Mac48Address::ConvertFrom (entry->GetMacAddress ());
          break;
        }
    }
  return hwaddr;
}

void
Neighbors::ProcessTxError (WifiMacHeader const & hdr)
{
  Mac48Address addr = hdr.GetAddr1 ();

  for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
    {
      if (i->m_hardwareAddress == addr)
        i->close = true;
    }
  Purge ();
}

void
Neighbors::PrintTable (Ptr<OutputStreamWrapper> stream) const
{
  int counter = 0;
  for (std::vector<Neighbor>::const_iterator i = m_nb.begin(); i != m_nb.end(); ++i)
  {
    *stream->GetStream () << i->m_neighborAddress <<"\t" << i->signalnoiseratio << "\t" << i->
         residualEnergy << "\t" << ++counter << "\n";
  }
}
}
}
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2009 IITP RAS
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Based on
 * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
 * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
 *
 * antcomm-UU implementation by Erik Nordstrm of Uppsala University
 * http://core.it.uu.se/core/index.php/antcomm-UU
 *
 * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
 * Pavel Boyko <boyko@iitp.ru>
 */

#ifndef antcommNEIGHBOR_H
#define antcommNEIGHBOR_H

#include "ns3/simulator.h"
#include "ns3/timer.h"
#include "ns3/ipv4-address.h"
#include "ns3/callback.h"
#include "ns3/wifi-mac-header.h"
#include "ns3/arp-cache.h"
#include <vector>

namespace ns3
{
namespace antcomm
{
class antcommProtocol;
/**
 * \ingroup antcomm
 * \brief maintain list of active neighbors
 */
class Neighbors
{
public:
  /// c-tor
  Neighbors (Time delay);
  /// Neighbor description
  struct Neighbor
  {
    Ipv4Address m_neighborAddress;
    Mac48Address m_hardwareAddress;
    double signalnoiseratio;
    double residualEnergy;
    Time m_expireTime;
    bool close;

    Neighbor (Ipv4Address ip, Mac48Address mac, double snr, double energy, Time t) :
      m_neighborAddress (ip), m_hardwareAddress (mac), signalnoiseratio (snr), residualEnergy (energy)
          , m_expireTime (t),
      close (false)
    {
    }
  };
  /// Return expire time for neighbor node with address addr, if exists, else return 0.
  Time GetExpireTime (Ipv4Address addr);
  /// Check that node with address addr is neighbor
  bool IsNeighbor (Ipv4Address addr);
  /// Update expire time for entry with address addr, if it exists, else add new entry
  void Update (Ipv4Address addr, double snr, double energy, Time expire);
  /// Get the degree of node
  int Degree ();
  /// Get the Ipv4Address of neighbor i in <vector> Neighbor, where i is an index
  Ipv4Address GetAddress (int i);
```

```
  /// Return the energy of neighbor node
  double GetEnergy (Ipv4Address addr);
  /// Return the received strength signal of addr
  double GetRssi (Ipv4Address addr);
  /// Remove all expired entries
  void Purge ();
  /// Schedule m_ntimer.
  void ScheduleTimer ();
  /// Remove all entries
  void Clear () { m_nb.clear (); }

  /// Add ARP cache to be used to allow layer 2 notifications processing
  void AddArpCache (Ptr<ArpCache>);
  /// Don't use given ARP cache any more (interface is down)
  void DelArpCache (Ptr<ArpCache>);
  /// Get callback to ProcessTxError
  Callback<void, WifiMacHeader const &> GetTxErrorCallback () const { return m_txErrorCallback; }
  /// Print the neighbor table
  void PrintTable (Ptr<OutputStreamWrapper> stream) const;

  /// Handle link failure callback
  void SetCallback (Callback<void, Ipv4Address> cb) { m_handleLinkFailure = cb; }
  /// Handle link failure callback
  Callback<void, Ipv4Address> GetCallback () const { return m_handleLinkFailure; }

private:
  /// link failure callback
  Callback<void, Ipv4Address> m_handleLinkFailure;
  /// TX error callback
  Callback<void, WifiMacHeader const &> m_txErrorCallback;
  /// Timer for neighbor's list. Schedule Purge().
  Timer m_ntimer;
  /// vector of entries
  std::vector<Neighbor> m_nb;
  /// list of ARP cached to be used for layer 2 notifications processing
  std::vector<Ptr<ArpCache> > m_arp;

  /// Find MAC address by IP using list of ARP caches
  Mac48Address LookupMacAddress (Ipv4Address);
  /// Process layer 2 TX error notification
  void ProcessTxError (WifiMacHeader const &);
};

}
}

#endif /* antcommNEIGHBOR_H */
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#include "antcomm-packet.h"
#include "ns3/address-utils.h"
#include "ns3/packet.h"
#include "ns3/ethernet-header.h"


namespace ns3
{


  namespace antcomm
  {
    TypeHeader::TypeHeader (MessageType t):m_type (t), m_valid (true)
    {
    }

    NS_OBJECT_ENSURE_REGISTERED (TypeHeader);

    TypeId TypeHeader::GetTypeId ()
    {
      static TypeId tid =
        TypeId ("ns3::antcomm::TypeHeader").SetParent < Header >
        ().SetGroupName ("antcomm").AddConstructor < TypeHeader > ();
      return tid;
    }

    TypeId TypeHeader::GetInstanceTypeId () const
    {
      return GetTypeId ();
    }

    uint32_t TypeHeader::GetSerializedSize () const
    {
      return 1;
    }

    void TypeHeader::Serialize (Buffer::Iterator i) const
    {
      i.WriteU8 ((uint8_t) m_type);
    }

    uint32_t TypeHeader::Deserialize (Buffer::Iterator start)
    {
      Buffer::Iterator i = start;
      uint8_t type = i.ReadU8 ();
      m_valid = true;
      switch (type)
        {
        case HELLO:
        case EXPL:
        case REPA:
        case RE_REPA:
        //case DATA_PACKET:
          {
          m_type = (MessageType) type;
          break;
          }
        default:
          m_valid = false;
        }
      uint32_t dist = i.GetDistanceFrom (start);
      NS_ASSERT (dist == GetSerializedSize ());
      return dist;
    }

    void TypeHeader::Print (std::ostream & os) const
    {
      switch (m_type)
        {
        case HELLO:
          {
            os << "HELLO";
            break;
          }
        case EXPL:
```

```
        {
          os << "EXPL";
          break;
        }
      case REPA:
        {
          os << "REPA";
          break;
        }
      case RE_REPA:
        {
          os << "RE_REPA";
          break;
        }
      //case DATA_PACKET:
        //{
          //os << "DATA_PACKET";
          //break;
        //}
      default:
        os << "UNKNOWN_TYPE";
      }
  }

  bool TypeHeader::operator== (TypeHeader const &o) const
  {
    return (m_type == o.m_type && m_valid == o.m_valid);
  }

  std::ostream & operator<< (std::ostream & os, TypeHeader const &h)
  {
    h.Print (os);
    return os;
  }

  antcommEXPLHeader::antcommEXPLHeader (uint8_t hopCount, uint8_t timeToLive, uint32_t requestID, Ipv4Address origin, Ipv4Address dst):
    m_hopCount (hopCount), m_timeToLive (timeToLive), m_requestID (requestID), m_origin (origin), m_dst (dst)
  {
  }

  NS_OBJECT_ENSURE_REGISTERED (antcommEXPLHeader);

  TypeId antcommEXPLHeader::GetTypeId ()
  {
    static TypeId tid =
      TypeId ("ns3::antcomm::antcommEXPLHeader").SetParent < Header >
      ().SetGroupName ("antcomm").AddConstructor < antcommEXPLHeader > ();
    return tid;
  }

  TypeId antcommEXPLHeader::GetInstanceTypeId () const
  {
    return GetTypeId ();
  }

  uint32_t antcommEXPLHeader::GetSerializedSize () const
  {
    return 14;
  }

  void antcommEXPLHeader::Serialize (Buffer::Iterator i) const
  {
    i.WriteU8 (m_hopCount);
    i.WriteU8 (m_timeToLive);
    i.WriteU32 (m_requestID);
    WriteTo (i, m_origin);
    WriteTo (i, m_dst);
  }

  uint32_t antcommEXPLHeader::Deserialize (Buffer::Iterator start)
  {
    Buffer::Iterator i = start;

    m_hopCount = i.ReadU8 ();
    m_timeToLive = i.ReadU8 ();
    m_requestID = i.ReadU32 ();
    ReadFrom (i, m_origin);
    ReadFrom (i, m_dst);
```

```
  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void antcommEXPLHeader::Print (std::ostream & os) const
{
  os << "antcomm - ";
  os << " Hop count : " << m_hopCount << "TTL " << m_timeToLive << "AntID" << m_requestID << " Source ID :  " << m_origin << "Destination : " << m_dst;
}

bool antcommEXPLHeader::operator== (antcommEXPLHeader const &o) const
{
  return (m_hopCount == o.m_hopCount && m_timeToLive == o.m_timeToLive && m_requestID == o.m_requestID
       && m_origin == o.m_origin && m_dst == o.m_dst );
}

std::ostream & operator<< (std::ostream & os, antcommEXPLHeader const &h)
{
  h.Print (os);
  return os;
}


antcommHelloHeader::antcommHelloHeader (uint8_t timeToLive, uint8_t hopCount, double residualEnergy, Ipv4Address origin, Ipv4Address dst, Time lifeTime):
  m_timeToLive (timeToLive), m_hopCount (hopCount), m_residualEnergy (residualEnergy),
  m_origin (origin), m_dst (dst)
{
  m_lifeTime = uint32_t (lifeTime.GetMilliSeconds ());
}

NS_OBJECT_ENSURE_REGISTERED (antcommHelloHeader);

TypeId antcommHelloHeader::GetTypeId ()
{
  static TypeId tid =
    TypeId ("ns3::antcomm::antcommHelloHeader").SetParent < Header >
    ().SetGroupName ("antcomm").AddConstructor < antcommHelloHeader > ();
  return tid;
}

TypeId antcommHelloHeader::GetInstanceTypeId () const
{
  return GetTypeId ();
}

uint32_t antcommHelloHeader::GetSerializedSize () const
{
  return 18 /*10 */ ;
}

void antcommHelloHeader::Serialize (Buffer::Iterator i) const
{


  i.WriteU8 (m_timeToLive);
  i.WriteU8 (m_hopCount);
  //i.WriteU32 (m_pheromone*100000000);
  i.WriteU32 (m_residualEnergy*100000000);
  WriteTo (i, m_origin);
  WriteTo (i, m_dst);
  i.WriteHtonU32 (m_lifeTime);
}

uint32_t antcommHelloHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;


  m_timeToLive = i.ReadU8 ();
  m_hopCount = i.ReadU8 ();
  // m_pheromone = i.ReadU32 ()/100000000.0; //
  m_residualEnergy = i.ReadU32 ()/100000000.0; //
  ReadFrom (i, m_origin);
  ReadFrom (i, m_dst);
  m_lifeTime = i.ReadNtohU32 ();
```

```

  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void antcommHelloHeader::Print (std::ostream & os) const
{
  os << "TTL: " << (int) m_timeToLive << ", Hops:" << m_hopCount
     << ",Res. Energy" << m_residualEnergy << ", Src: " << m_origin << ", Dst: "<<m_dst << ", lifeTime" << m_lifeTime;
  /* " Residual Energy : " << m_residualEnergy << */
}

void antcommHelloHeader::SetLifeTime (Time t)
{
  m_lifeTime = t.GetMilliSeconds ();
}

Time antcommHelloHeader::GetLifeTime () const
{
  Time t (MilliSeconds (m_lifeTime));
  return t;
}


bool antcommHelloHeader::operator== (antcommHelloHeader const &o) const
{
  return (m_timeToLive == o.m_timeToLive && m_hopCount == o.m_hopCount
          && m_residualEnergy == o.m_residualEnergy && m_origin == o.m_origin && m_dst == o.m_dst && m_lifeTime == o.m_lifeTime);
}

std::ostream & operator<< (std::ostream & os, antcommHelloHeader const &h)
{
  h.Print (os);
  return os;
}

antcommREPAHeader::antcommREPAHeader (uint8_t hopCount,uint8_t timeToLive, Ipv4Address origin, Ipv4Address dst ): //
  m_hopCount (hopCount), m_timeToLive (timeToLive), m_origin (origin), m_dst (dst)
{
}

NS_OBJECT_ENSURE_REGISTERED (antcommREPAHeader);

TypeId antcommREPAHeader::GetTypeId ()
{
  static TypeId tid =
    TypeId ("ns3::antcomm::antcommREPAHeader").SetParent < Header >
    ().SetGroupName ("antcomm").AddConstructor < antcommREPAHeader > ();
  return tid;
}

TypeId antcommREPAHeader::GetInstanceTypeId () const
{
  return GetTypeId ();
}

uint32_t antcommREPAHeader::GetSerializedSize () const
{
  return 10 ;
}

void antcommREPAHeader::Serialize (Buffer::Iterator i) const
{
  i.WriteU8 (m_hopCount);
  i.WriteU8 (m_timeToLive);
  //i.WriteU16 (m_sourceID);
  WriteTo (i, m_origin);
  WriteTo (i, m_dst);
}

uint32_t antcommREPAHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;

  m_hopCount = i.ReadU8 ();
  m_timeToLive = i.ReadU8 ();
```

```
  //m_sourceID = i.ReadU16 ();
  ReadFrom (i, m_origin);
  ReadFrom (i, m_dst);

  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void antcommREPAHeader::Print (std::ostream & os) const
{
  os << "antcomm␣-␣";
  os << "␣Hop␣count␣:␣" << m_hopCount << "TTL" << m_timeToLive << "␣Source␣ID␣:␣" << m_origin << "
       Destination" << m_dst ;
}

bool antcommREPAHeader::operator== (antcommREPAHeader const &o) const
{
  return (m_hopCount == o.m_hopCount && m_timeToLive == o.m_timeToLive && m_origin == o.m_origin
       && m_dst == o.m_dst);
}

std::ostream & operator<< (std::ostream & os, antcommREPAHeader const &h)
{
  h.Print (os);
  return os;
}



antcommREREPAHeader::antcommREREPAHeader (uint8_t hopCount, uint8_t timeToLive, double pheromone,
     double residualEnergy, Ipv4Address origin, Ipv4Address dst):
  m_hopCount (hopCount), m_timeToLive (timeToLive), m_pheromone (pheromone),
  m_residualEnergy (residualEnergy),
  m_origin (origin), m_dst (dst)
{
}

NS_OBJECT_ENSURE_REGISTERED (antcommREREPAHeader);

TypeId antcommREREPAHeader::GetTypeId ()
{
  static TypeId tid =
    TypeId ("ns3::antcomm::antcommREREPAHeader").SetParent < Header >
    ().SetGroupName ("antcomm").AddConstructor < antcommREREPAHeader > ();
  return tid;
}

TypeId antcommREREPAHeader::GetInstanceTypeId () const
{
  return GetTypeId ();
}

uint32_t antcommREREPAHeader::GetSerializedSize () const
{
  return 18 /*32 */ ;
}

void antcommREREPAHeader::Serialize (Buffer::Iterator i) const
{
  i.WriteU8 (m_hopCount);
  i.WriteU8 (m_timeToLive);
  i.WriteU32 (m_pheromone * 100000000);
  i.WriteU32 (m_residualEnergy * 100000000);
  //i.WriteU16 (m_sourceID);
  WriteTo (i, m_origin);
  WriteTo (i, m_dst);
}

uint32_t antcommREREPAHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;

  m_hopCount = i.ReadU8 ();
  m_timeToLive = i.ReadU8 ();
  m_pheromone = i.ReadU32 () / 100000000.0;
  m_residualEnergy = i.ReadU32 () / 100000000.0;
  //m_sourceID = i.ReadU16 ();
  ReadFrom (i, m_origin);
  ReadFrom (i, m_dst);
```

```
    uint32_t dist = i.GetDistanceFrom (start);
    NS_ASSERT (dist == GetSerializedSize ());
    return dist;
  }

  void antcommREREPAHeader::Print (std::ostream & os) const
  {
    os << "antcomm␣-␣";
    os << "␣Hop␣count␣:␣" << m_hopCount << "TTL" << m_timeToLive << "␣Pheromone␣:␣" << m_pheromone <<
      "␣Residual␣Energy␣:␣" << m_residualEnergy << "␣Source␣ID␣:␣" <<
      m_origin << "Destination" << m_dst ;
  }

  bool antcommREREPAHeader::operator== (antcommREREPAHeader const &o) const
  {
    return (m_hopCount == o.m_hopCount && m_pheromone == o.m_pheromone
            && m_residualEnergy == o.m_residualEnergy
            && m_residualEnergy == o.m_residualEnergy
            && m_origin == o.m_origin && m_dst ==o.m_dst && m_timeToLive == o.m_timeToLive );
  }

  std::ostream & operator<< (std::ostream & os,
                             antcommREREPAHeader const &h)
  {
    h.Print (os);
    return os;
  }


  antcommDataPacketHeader::antcommDataPacketHeader (Ipv4Address origin, Ipv4Address dst, uint8_t hopCount, uint8_t timeToLive, uint16_t hopDistance, uint32_t packetID): //
    m_origin (origin ), m_dst (dst ), m_hopCount (hopCount), m_timeToLive (timeToLive), m_hopDistance (hopDistance), m_packetID (packetID)
  {
  }

  NS_OBJECT_ENSURE_REGISTERED (antcommDataPacketHeader);

  TypeId antcommDataPacketHeader::GetTypeId ()
  {
    static TypeId tid =
      TypeId ("ns3::antcomm::antcommDataPacketHeader").SetParent < Header >
      ().SetGroupName ("antcomm").AddConstructor < antcommDataPacketHeader >
      ();
    return tid;
  }

  TypeId antcommDataPacketHeader::GetInstanceTypeId () const
  {
    return GetTypeId ();
  }

  uint32_t antcommDataPacketHeader::GetSerializedSize () const
  {
    return 512 ;
  }

  void antcommDataPacketHeader::Serialize (Buffer::Iterator i) const
  {
    //i.WriteU16 (m_sourceID);
    //i.WriteU16 (m_destinationID);
    WriteTo (i, m_origin);
    WriteTo (i, m_dst);
    i.WriteU8 (m_hopCount);
    i.WriteU8 (m_timeToLive);
    i.WriteU16 (m_hopDistance);
    i.WriteU32 (m_packetID);
  }

  uint32_t antcommDataPacketHeader::Deserialize (Buffer::Iterator start)
  {
    Buffer::Iterator i = start;

    //m_sourceID = i.ReadU16 ();
    //m_destinationID = i.ReadU16 ();
    ReadFrom (i, m_origin);
    ReadFrom (i, m_dst);
```

```
      m_hopCount = i.ReadU8 ();
      m_timeToLive = i.ReadU8 ();
      m_hopDistance = i.ReadU16 ();
      m_packetID = i.ReadU32();


      uint32_t dist = i.GetDistanceFrom (start);
      return dist;
    }

    void antcommDataPacketHeader::Print (std::ostream & os) const
    {
      os << "antcomm - ";
      os << " Source ID : " << m_origin << " Destination ID : " << m_dst << "Hopcount" << m_hopCount
           << "TTL" <<m_timeToLive << "HopDistance" << m_hopDistance << "PacketID" << m_packetID ;
    }

    bool antcommDataPacketHeader::operator== (antcommDataPacketHeader const &o) const
    {
      return (m_origin == o.m_origin && m_dst == o.m_dst &&m_hopCount == o.m_hopCount && m_timeToLive
           == o.m_timeToLive && m_hopDistance == o.m_hopDistance && m_packetID == o.m_packetID );
    }

    std::ostream & operator<< (std::ostream & os,
                               antcommDataPacketHeader const &h)
    {
      h.Print (os);
      return os;
    }
  }
}
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#ifndef antcommPACKET_H
#define antcommPACKET_H

#include <iostream>
#include "ns3/header.h"
#include "ns3/enum.h"
#include "ns3/ipv4-address.h"
#include <map>
#include "ns3/nstime.h"
#include "ns3/ethernet-header.h"

namespace ns3
{

  namespace antcomm
  {

    enum MessageType
    {
      HELLO = 1,  //!< EXPL
      EXPL = 2,   //!< REPA
      REPA = 3,   //!< RE_REPA
      RE_REPA = 4,
      //DATA_PACKET = 5
    };
                /**
                * \ingroup antcomm
                * \brief antcomm types
                */
    class TypeHeader:public Header
    {
    public:
      /// c-tor
      TypeHeader (MessageType t = EXPL);

      // Header serialization/deserialization
      static TypeId GetTypeId ();
      TypeId GetInstanceTypeId () const;
      uint32_t GetSerializedSize () const;
      void Serialize (Buffer::Iterator start) const;
      uint32_t Deserialize (Buffer::Iterator start);
      void Print (std::ostream & os) const;

      /// Return type
      MessageType Get () const
      {
        return m_type;
      }
      /// Check that type if valid
      bool IsValid () const
      {
        return m_valid;
      }
      bool operator== (TypeHeader const &o) const;
    private:
      MessageType m_type;
      bool m_valid;
    };

      std::ostream & operator<< (std::ostream & os, TypeHeader const &h);


    /**
* \ingroup antcomm
* \brief antcomm EXPL Message Format
  \verbatim
  0 1 2 3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Type | Hop Count | Time to live | Void |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | RREQ ID |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Destination IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
  | Originator IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


  \endverbatim
*/

    class antcommEXPLHeader:public Header
    {
    public:
      /// c-tor
      antcommEXPLHeader (uint8_t hopCount = 0, uint8_t timeToLive = 0, uint32_t requestID = 0,
           Ipv4Address origin = Ipv4Address (), Ipv4Address dst = Ipv4Address () );

      // Header serialization/deserialization
      static TypeId GetTypeId ();
      TypeId GetInstanceTypeId () const;
      uint32_t GetSerializedSize () const;
      void Serialize (Buffer::Iterator start) const;
      uint32_t Deserialize (Buffer::Iterator start);
      void Print (std::ostream & os) const;

      // Fields
      void SetHopCount (uint8_t count)
      {
        m_hopCount = count;
      }
      uint8_t GetHopCount () const
      {
        return m_hopCount;
      }
      void SetId (uint32_t id)
      {
        m_requestID = id;
      }
      uint32_t GetId () const
      {
        return m_requestID;
      }
      void SetOrigin (Ipv4Address a)
      {
            m_origin = a;
      }
      Ipv4Address GetOrigin () const
      {
            return m_origin;
      }
      void SetDst (Ipv4Address a)
      {
      m_dst = a;
      }
      Ipv4Address GetDst () const
      {
      return m_dst;
      }
      void SetTimeToLive (uint8_t timeToLive)
      {
            m_timeToLive = timeToLive;
      }
      uint8_t GetTimeToLive () const
      {
            return m_timeToLive;
      }

      //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
      //uint16_t GetSourceID () const { return m_sourceID; } //


      bool operator== (antcommEXPLHeader const &o) const;

    private:
      uint8_t m_hopCount; ///< Hop Count
      uint8_t m_timeToLive;
      uint32_t m_requestID;
      Ipv4Address m_origin;
      Ipv4Address m_dst;



    };

```

```
    std::ostream & operator<< (std::ostream & os, antcommEXPLHeader const &h);



    /**
* \ingroup antcomm
* \brief antcomm Hello Message Format
  \verbatim
  0 1 2 3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Type | Time to Live | Hop Count | Void |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Destination IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Originator IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Residual Energy (32 bits) |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Lifetime (32 bits) |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

  \endverbatim
*/
    class antcommHelloHeader:public Header
    {
    public:
      /// c-tor
      antcommHelloHeader (uint8_t timeToLive = 0, uint8_t hopCount = 0, double residualEnergy = 0,
           Ipv4Address origin = Ipv4Address (), Ipv4Address dst = Ipv4Address (), Time lifetime =
           MilliSeconds (0));

      // Header serialization/deserialization
      static TypeId GetTypeId ();
      TypeId GetInstanceTypeId () const;
      uint32_t GetSerializedSize () const;
      void Serialize (Buffer::Iterator start) const;
      uint32_t Deserialize (Buffer::Iterator start);
      void Print (std::ostream & os) const;


      // Fields

      void SetTimeToLive (uint8_t timeToLive)
      {
        m_timeToLive = timeToLive;
      }
      uint8_t GetTimeToLive () const
      {
        return m_timeToLive;
      }
      void SetHopCount (uint8_t hopCount)
      {
        m_hopCount = hopCount;
      }
      uint8_t GetHopCount () const
      {
        return m_hopCount;
      }
      void SetResidualEnergy ( double residualEnergy)
      {
 m_residualEnergy = residualEnergy;
      }
      double GetResidualEnergy() const
      {
 return m_residualEnergy;
      }
      void SetOrigin (Ipv4Address a)
      {
        m_origin = a;
      }
      Ipv4Address GetOrigin () const
      {
        return m_origin;
      }

      void SetDst (Ipv4Address a)
      {
  m_dst = a;
      }
      Ipv4Address GetDst () const
```

```
      {
  return m_dst;
      }
      void SetLifeTime (Time t);
  Time GetLifeTime () const;


      bool operator== (antcommHelloHeader const &o) const;

    private:


      uint8_t m_timeToLive;
      uint8_t m_hopCount;
      double m_residualEnergy;
      Ipv4Address m_origin;
      Ipv4Address m_dst;
      uint32_t m_lifeTime;
    };

    std::ostream & operator<< (std::ostream & os,
                               antcommHelloHeader const &h);


   /**
* \ingroup antcomm
* \brief antcomm REPA Message Format
  \verbatim
  0 1 2 3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Type | Hop Count | Time to live | Void |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Destination IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Originator IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


  \endverbatim
*/
    class antcommREPAHeader:public Header
    {
    public:
      /// c-tor
      antcommREPAHeader (uint8_t hopCount = 0, uint8_t timeToLive = 0, Ipv4Address origin =
           Ipv4Address (), Ipv4Address dst = Ipv4Address () );

      // Header serialization/deserialization
      static TypeId GetTypeId ();
      TypeId GetInstanceTypeId () const;
      uint32_t GetSerializedSize () const;
      void Serialize (Buffer::Iterator start) const;
      uint32_t Deserialize (Buffer::Iterator start);
      void Print (std::ostream & os) const;

      void SetHopCount (uint8_t hopCount)
      {
           m_hopCount = hopCount;
      }
      uint16_t GetHopCount () const
      {
           return m_hopCount;
      }
      void SetOrigin (Ipv4Address a)
      {
           m_origin = a;
      }
      Ipv4Address GetOrigin () const
      {
           return m_origin;
      }
      void SetDst (Ipv4Address a)
      {
      m_dst = a;
      }
      Ipv4Address GetDst () const
      {
      return m_dst;
      }
      void SetTimeToLive (uint8_t timeToLive)
```

```
      {
            m_timeToLive = timeToLive;
      }
      uint8_t GetTimeToLive () const
      {
            return m_timeToLive;
      }
      //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
      //uint16_t GetSourceID () const { return m_sourceID; } //

      bool operator== (antcommREPAHeader const &o) const;

    private:
      uint8_t m_hopCount; ///< Hop Count
      uint8_t m_timeToLive;
      Ipv4Address m_origin;
      Ipv4Address m_dst;

    };

    std::ostream & operator<< (std::ostream & os, antcommREPAHeader const &h);


     /**
* \ingroup antcomm
* \brief antcomm REREPA Message Format
  \verbatim
  0 1 2 3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Type | Hop Count | Time to live | Void |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Destination IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Originator IP address |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Payload |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


  \endverbatim
*/
    class antcommREREPAHeader:public Header
    {
    public:
      /// c-tor
      antcommREREPAHeader (uint8_t hopCount = 0, uint8_t timeToLive = 0, double pheromone = 0.0,
           double residualEnergy = 0.0, Ipv4Address origin = Ipv4Address (), Ipv4Address dst =
           Ipv4Address ());

      // Header serialization/deserialization
      static TypeId GetTypeId ();
      TypeId GetInstanceTypeId () const;
      uint32_t GetSerializedSize () const;
      void Serialize (Buffer::Iterator start) const;
      uint32_t Deserialize (Buffer::Iterator start);
      void Print (std::ostream & os) const;

      void SetHopCount (uint8_t hopCount)
      {
            m_hopCount = hopCount;
      }
      uint16_t GetHopCount () const
      {
            return m_hopCount;
      }
      void SetPheromone (double pheromone)
      {
            m_pheromone = pheromone;
      }
      double GetPheromone () const
      {
            return m_pheromone;
      }
      void SetResidualEnergy (double residualEnergy)
      {
            m_residualEnergy = residualEnergy;
      }
      double GetResidualEnergy () const
      {
            return m_residualEnergy;
```

```
      }
      void SetOrigin (Ipv4Address a)
      {
           m_origin = a;
      }
      Ipv4Address GetOrigin () const
      {
           return m_origin;
      }
      void SetDst (Ipv4Address a)
      {
      m_dst = a;
      }
      Ipv4Address GetDst () const
      {
      return m_dst;
      }
      void SetTimeToLive (uint8_t timeToLive)
      {
           m_timeToLive = timeToLive;
      }
      uint8_t GetTimeToLive () const
      {
           return m_timeToLive;
      }
      //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
      //uint16_t GetSourceID () const { return m_sourceID; } //

      bool operator== (antcommREREPAHeader const &o) const;

    private:
      uint8_t m_hopCount; ///< Hop Count
      uint8_t m_timeToLive;
      double m_pheromone;
      double m_residualEnergy;
      Ipv4Address m_origin;
      Ipv4Address m_dst;






    };

    std::ostream & operator<< (std::ostream & os,
                               antcommREREPAHeader const &h);


    /**
 * \ingroup antcomm
 * \brief antcomm DATAPacket Message Format
   \verbatim
   0 1 2 3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Type | Hop Count | Time to live | Void |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Destination IP address |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Originator IP address |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | PacketID |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Hop Distance |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

   \endverbatim
 */
    class antcommDataPacketHeader:public Header
    {
    public:
      /// c-tor
      antcommDataPacketHeader (Ipv4Address origin = Ipv4Address (), Ipv4Address dst = Ipv4Address (),
           uint8_t hopCount = 0, uint8_t timeToLive = 0, uint16_t hopDistance = 0, uint32_t packetID =
            0); // 0

      // Header serialization/deserialization
      static TypeId GetTypeId ();
      TypeId GetInstanceTypeId () const;
      uint32_t GetSerializedSize () const;
```

```
      void Serialize (Buffer::Iterator start) const;
      uint32_t Deserialize (Buffer::Iterator start);
      void Print (std::ostream & os) const;

      // Fields
      //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
      //uint16_t GetSourceID () const { return m_sourceID; } //
      //void SetDestinationID (uint16_t destinationID) { m_destinationID = destinationID; } //
      //uint16_t GetDestinationID () const { return m_destinationID; } //

      void SetOrigin (Ipv4Address a)
      {
        m_origin = a;
      }
      Ipv4Address GetOrigin () const
      {
        return m_origin;
      }

      void SetDst (Ipv4Address a)
      {
      m_dst = a;
      }
      Ipv4Address GetDst () const
      {
      return m_dst;
      }
      void SetTimeToLive (uint8_t timeToLive)
      {
            m_timeToLive = timeToLive;
      }
      uint8_t GetTimeToLive () const
      {
            return m_timeToLive;
      }
      void SetHopCount (uint8_t hopCount)
      {
            m_hopCount = hopCount;
      }
      uint16_t GetHopCount () const
      {
            return m_hopCount;
      }
      void SetHopDistance (uint16_t hopDistance) { m_hopDistance = hopDistance; }
            uint16_t GetHopDistance () const { return m_hopDistance; }
            void SetPacketID (uint32_t packetID) { m_packetID = packetID; }
            uint32_t GetPacketID () const { return m_packetID; }

      bool operator== (antcommDataPacketHeader const &o) const;

    private:
      //uint16_t m_sourceID; ///< Source ID //
      //uint16_t m_destinationID; ///< Destination ID //
      Ipv4Address m_origin;
      Ipv4Address m_dst;
      uint8_t m_hopCount;
      uint8_t m_timeToLive;
      uint16_t m_hopDistance;
      uint32_t m_packetID;

    };

    std::ostream & operator<< (std::ostream & os,
                               antcommDataPacketHeader const &h);

  }
}

#endif /* antcommPACKET_H */
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#define NS_LOG_APPEND_CONTEXT      \
       if (m_ipv4) { std::clog << "[node␣" << m_ipv4->GetObject<Node> ()->GetId () << "]␣"; }

#include "antcomm-helper.h"
#include "antcomm-protocol.h"
#include "ns3/core-module.h"
#include "ns3/log.h"
#include "ns3/boolean.h"
#include "ns3/random-variable-stream.h"
#include "ns3/inet-socket-address.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/udp-socket-factory.h"
#include "ns3/wifi-net-device.h"
#include "ns3/adhoc-wifi-mac.h"
#include "ns3/string.h"
#include "ns3/pointer.h"
#include "ns3/energy-module.h"
#include "ns3/propagation-loss-model.h"
#include "ns3/snr-tag.h"
#include "ns3/constant-position-mobility-model.h"
#include "ns3/random-variable-stream.h"
#include "ns3/ipv4-interface.h"
#include <algorithm>
#include <limits>
#include <math.h>
#include <iostream>
#include <stdio.h>
#include <iomanip>
#define LAMBDA 0.7

namespace ns3
{

  NS_LOG_COMPONENT_DEFINE ("antcommProtocol");

  namespace antcomm
  {
    NS_OBJECT_ENSURE_REGISTERED (antcommProtocol);

    /// UDP Port for antcomm control traffic
    const uint32_t antcommProtocol::antcomm_PORT = 285;

    //-----------------------------------------------------------------------------
    /// Tag used by antcomm implementation

    class DeferredRouteOutputTag:public Tag
    {

    public:
    DeferredRouteOutputTag (int32_t o = -1):Tag (), m_oif (o) {}

      static TypeId GetTypeId ()
      {
        static TypeId tid = TypeId ("ns3::antcomm::DeferredRouteOutputTag")
        .SetParent < Tag > ()
        .SetGroupName ("antcomm")
        .AddConstructor <DeferredRouteOutputTag > ()
        ;
          return tid;
      }

      TypeId GetInstanceTypeId () const
      {
        return GetTypeId ();
      }

      int32_t GetInterface () const
      {
        return m_oif;
      }

      void SetInterface (int32_t oif)
      {
        m_oif = oif;
      }
```

```

    uint32_t GetSerializedSize () const
    {
      return sizeof (int32_t);
    }

    void Serialize (TagBuffer i) const
    {
      i.WriteU32 (m_oif);
    }

    void Deserialize (TagBuffer i)
    {
      m_oif = i.ReadU32 ();
    }

    void Print (std::ostream & os) const
    {
      os << "DeferredRouteOutputTag:␣output␣interface␣=␣" << m_oif;
    }

  private:
    /// Positive if output device is fixed in RouteOutput
      int32_t m_oif;
  };

  NS_OBJECT_ENSURE_REGISTERED (DeferredRouteOutputTag);

  //-----------------------------------------------------------------------------

antcommProtocol::antcommProtocol ():
  m_numberOfReceivedPacket (0),
    EXPLRateLimit (10),
    m_averageHopDistanceOfReceivedPackets (0.0),
    m_averageEndToEndDelays (0.0),
    m_numberOfReceivedAnts (0.0),
    ActiveRouteTimeout (Seconds (3)),
    NetDiameter (35),
    NodeTraversalTime (MilliSeconds (40)),
    NetTraversalTime (Time ((2 * NetDiameter) * NodeTraversalTime)),
    PathDiscoveryTime ( Time (2 * NetTraversalTime)),
    m_currentPhase (EXPLORATION),
    HelloInterval (Seconds (0.5)),
    AllowedHelloLoss (2),
    m_explorationTime (EXPLORATION_TIME),
    m_explorationTimer (Timer::CANCEL_ON_DESTROY),
    m_repairTime (REPAIR_TIME),
    m_repairTimer (Timer::CANCEL_ON_DESTROY),
    m_transmissionRange (TRANSMISSION_RANGE),
    m_distanceForTimer (DISTANCE_FOR_TIMER),
    m_pheromone (INITIAL_PHEROMONE),
    m_pheromoneDecayRate (PHEROMONE_DECAY_RATE),
    m_updatedTimeOfPheromone (Seconds (0)),
    m_epsilon (EPSILON),
    m_timeMinForWaitingTimer (TIME_MIN_FOR_WAITING_TIMER),
    m_timeMaxForWaitingTimer (TIME_MAX_FOR_WAITING_TIMER), m_alpha (ALPHA),
    m_beta (BETA), m_gamma (GAMMA), m_routingTable (), MaxQueueLen (64),
    MaxQueueTime (Seconds (30)), m_queue (MaxQueueLen, MaxQueueTime),
    //m_helloInterval (TIME_FOR_HELLO_TIMER),
    m_requestId (0),
    m_helloTimer (Timer::CANCEL_ON_DESTROY),
    m_rreqIdCache (PathDiscoveryTime),
    m_dpd (PathDiscoveryTime),
    m_explCount (0),
    m_neighborTable (HelloInterval),
    m_lastBcastTime (Seconds (0)),
    m_forwardTimer (Timer::CANCEL_ON_DESTROY),
    m_backwardTimer (Timer::CANCEL_ON_DESTROY),
    m_retryTimer (Timer::CANCEL_ON_DESTROY),
    numberOfDeferredPacket (0), m_waitingTimer (Timer::CANCEL_ON_DESTROY)
  {
  }


  TypeId antcommProtocol::GetTypeId (void)
  {
    static TypeId tid = TypeId ("ns3::antcomm::antcommProtocol")
    .SetParent <Ipv4RoutingProtocol > ()
    .SetGroupName ("antcomm")
    .AddConstructor <antcommProtocol > ()
```

```
        .AddAttribute ("ExplorationTime", "The exploration time.",
                                    UintegerValue (EXPLORATION_TIME),
                                    MakeUintegerAccessor(&antcommProtocol::m_explorationTime),
                                    MakeUintegerChecker < uint32_t >())
        .AddAttribute ("HelloInterval", "HELLO messages emission interval.",
                     TimeValue (Seconds (5)),
                     MakeTimeAccessor (&antcommProtocol::HelloInterval),
                     MakeTimeChecker ())
        .AddAttribute ("NetDiameter", "Net diameter measures the maximum possible number of hops between two nodes in the network",
                                    UintegerValue (35),
                                    MakeUintegerAccessor(&antcommProtocol::NetDiameter),
                                    MakeUintegerChecker <uint32_t>())
        .AddAttribute ("NodeTraversalTime", "Conservative estimate of the average one hop traversal time for packets and should include "
                                   "queuing delays, interrupt processing times and transfer times.",
                                    TimeValue (MilliSeconds (40)),
                                    MakeTimeAccessor(&antcommProtocol::NodeTraversalTime),
                                    MakeTimeChecker ())
        .AddAttribute ("NetTraversalTime", "Estimate of the average net traversal time = 2 * NodeTraversalTime * NetDiameter",
                        TimeValue (Seconds (2.8)),
                        MakeTimeAccessor (&antcommProtocol::NetTraversalTime),
                        MakeTimeChecker ())
        .AddAttribute ("ActiveRouteTimeout", "Period of time during which the route is considered to be valid",
                    TimeValue (Seconds (3)),
                    MakeTimeAccessor (&antcommProtocol::ActiveRouteTimeout),
                    MakeTimeChecker ())
        .AddAttribute ("DistanceForTimer", "The distance for timer.",
                                    DoubleValue (0.5),
                                    MakeDoubleAccessor(&antcommProtocol::m_distanceForTimer),
                                    MakeDoubleChecker <double >())
        .AddAttribute ("PathDiscoveryTime", "Estimate of maximum time needed to find route in network = 2 * NetTraversalTime",
                    TimeValue (Seconds (5.6)),
                    MakeTimeAccessor (&antcommProtocol::PathDiscoveryTime),
                    MakeTimeChecker ())
        .AddAttribute ("MaxQueueLen", "Maximum number of packets that we allow a routing protocol to buffer.",
                        UintegerValue (64),
                        MakeUintegerAccessor (&antcommProtocol::SetMaxQueueLen,
                                              &antcommProtocol::GetMaxQueueLen),
                        MakeUintegerChecker <uint32_t>())
        .AddAttribute ("MaxQueueTime", "Maximum time packets can be queued (in seconds)",
                                    TimeValue (Seconds (30)),
                                    MakeTimeAccessor (&antcommProtocol:: SetMaxQueueTime,
                                                      &antcommProtocol:: GetMaxQueueTime),
                                    MakeTimeChecker ())
        .AddAttribute ("AllowedHelloLoss", "Number of hello messages which may be loss for valid link.",
                     UintegerValue (2),
                     MakeUintegerAccessor (&antcommProtocol::AllowedHelloLoss),
                     MakeUintegerChecker<uint16_t> ())
        .AddAttribute ("PheromoneDecayRate", "The pheromone decay rate.",
                        DoubleValue (0.5),
                        MakeDoubleAccessor (&antcommProtocol::m_pheromoneDecayRate),
                        MakeDoubleChecker <double>())
        .AddAttribute ("Alpha", "The alpha.",
                                    DoubleValue (0.33),
                                    MakeDoubleAccessor(&antcommProtocol::m_alpha),
                                    MakeDoubleChecker<double>())
        .AddAttribute ("Beta", "The beta.",
                        DoubleValue (0.33),
                        MakeDoubleAccessor (&antcommProtocol::m_beta),
                        MakeDoubleChecker < double >())
        .AddAttribute ("Gamma", "The gamma.",
                                    DoubleValue (0.33),
                                    MakeDoubleAccessor(&antcommProtocol::m_gamma),
                                    MakeDoubleChecker<double>())
        .AddAttribute ("EnableBroadcast", "Indicates whether a broadcast data packets forwarding enable.",
                        BooleanValue (true),
                        MakeBooleanAccessor (&antcommProtocol::SetBroadcastEnable,
                                             &antcommProtocol::GetBroadcastEnable),
                        MakeBooleanChecker ())
        .AddAttribute ("UniformRv", "Access to the underlying UniformRandomVariable",
                                    StringValue ("ns3::UniformRandomVariable"),
                                    MakePointerAccessor(&antcommProtocol::m_uniformRandomVariable),
                                    MakePointerChecker<UniformRandomVariable>())
        ;
```

```
  return tid;
}

antcommProtocol::~antcommProtocol ()
{
}
void antcommProtocol::SetMaxQueueLen (uint32_t len)
{
  MaxQueueLen = len;
  m_queue.SetMaxQueueLen (len);
}

void antcommProtocol::SetMaxQueueTime (Time t)
{
  MaxQueueTime = t;
  m_queue.SetQueueTimeout (t);
}

void antcommProtocol::DoDispose ()
{
  NS_LOG_FUNCTION (this);
  m_ipv4 = 0;
  for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::iterator iter = m_socketAddresses.begin (); iter != m_socketAddresses.end (); iter++)
      {
           iter->first->Close ();
      }
  m_socketAddresses.clear ();
  for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::iterator iter = m_socketSubnetBroadcastAddresses.begin (); iter != m_socketSubnetBroadcastAddresses.end (); iter++)
  {
    iter->first->Close ();
      }
  m_socketSubnetBroadcastAddresses.clear ();
  Ipv4RoutingProtocol::DoDispose ();
}

void antcommProtocol::UpdateCurrentPhase (Phase type)
{
  NS_LOG_FUNCTION (this);
  m_currentPhase = type;
}

int64_t antcommProtocol::AssignStreams (int64_t stream)
{
  NS_LOG_FUNCTION (this << stream);
  m_uniformRandomVariable->SetStream (stream);
  return 1;
}


void antcommProtocol::SetIpv4 (Ptr < Ipv4 > ipv4)
{
  NS_LOG_FUNCTION (this);
  NS_ASSERT (ipv4 != 0);
  NS_ASSERT (m_ipv4 == 0);
  m_ipv4 = ipv4;

  // Create lo route. It is asserted that the only one interface up for now is loopback
  NS_ASSERT (m_ipv4->GetNInterfaces () == 1 && m_ipv4->GetAddress (0,0).GetLocal () == Ipv4Address ("127.0.0.1"));
  m_lo = m_ipv4->GetNetDevice (0);
  NS_ASSERT (m_lo != 0);
  /** This is how every node has access to its own energy
      through the ipv4 layer 3. */
  Ptr <Node> node = m_ipv4->GetObject< Node > ();
  Ptr<EnergySourceContainer> EnergySourceContrainerOnNode = node->GetObject<EnergySourceContainer> ();
  Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> ( EnergySourceContrainerOnNode->Get(0));
  /**************************************************/
  // Remember lo route
  antcommRoutingTableEntry rt (m_lo, /*dst= */ Ipv4Address::GetLoopback (), /*hops= */ 1, m_pheromone, /*energy=*/basicSourcePtr->GetRemainingEnergy(), /*RSSI=*/ 0, Ipv4Address::GetLoopback (),
                              Ipv4InterfaceAddress (Ipv4Address::GetLoopback (), Ipv4Mask("255.0.0.0")),
                                              /*lifetime= */ Simulator::GetMaximumSimulationTime ());
  m_routingTable.AddRoute (rt);
```

```
  Simulator::ScheduleNow (&antcommProtocol::Start, this);
}


void antcommProtocol::NotifyInterfaceUp (uint32_t i)
{
  NS_LOG_DEBUG("NotifyInterfaceUp function");
  NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
  Ptr <Ipv4L3Protocol> l3 = m_ipv4->GetObject <Ipv4L3Protocol> ();
  if (l3->GetNAddresses (i) > 1)
      {
           NS_LOG_WARN ("antcomm does not work with more then one address per each interface.");
      }
  Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
  if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
      {
           return;
      }

  // Create a socket to listen only on this interface
  Ptr < Socket > socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
  NS_ASSERT (socket != 0);
  socket->SetRecvCallback (MakeCallback (&antcommProtocol::Recvantcomm, this));
  socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), antcomm_PORT));
  socket->BindToNetDevice (l3->GetNetDevice (i));
  socket->SetAllowBroadcast (true);
  socket->SetAttribute ("IpTtl", UintegerValue (1));
  m_socketAddresses.insert (std::make_pair (socket, iface));

  // create also a subnet broadcast socket
  socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
  NS_ASSERT (socket != 0);
  socket->SetRecvCallback (MakeCallback (&antcommProtocol::Recvantcomm, this));
  socket->Bind (InetSocketAddress (iface.GetBroadcast (), antcomm_PORT));
  socket->BindToNetDevice (l3->GetNetDevice (i));
  socket->SetAllowBroadcast (true);
  socket->SetAttribute ("IpTtl", UintegerValue (1));
  m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

  // Add local broadcast record to the routing table
  Ptr < NetDevice > dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
  /** This is how every node has access to its own energy
      through the ipv4 layer 3. */

  Ptr <Node> node = m_ipv4->GetObject< Node > ();
  Ptr<EnergySourceContainer> EnergySourceContrainerOnNode = node->GetObject<EnergySourceContainer>();
  Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (EnergySourceContrainerOnNode->Get(0));

  antcommRoutingTableEntry rt (dev,/*dst= */ iface.GetBroadcast (),
                                            /*hops= */ 1, m_pheromone, /*Energy=*/basicSourcePtr->GetRemainingEnergy(), /*RSSI=*/ 0, /*Next Hop*/ iface.GetBroadcast (),
                                            /*Interface*/ iface, /*lifetime= */
                                            Simulator::GetMaximumSimulationTime ());
  m_routingTable.AddRoute (rt);

  if (l3->GetInterface (i)->GetArpCache ())
  {
    m_neighborTable.AddArpCache (l3->GetInterface (i)->GetArpCache ());
  }
  // Allow neighbor manager use this interface for layer 2 feedback if possible
  Ptr < WifiNetDevice > wifi = dev->GetObject < WifiNetDevice > ();
  if (wifi == 0)
    return;
  Ptr < WifiMac > mac = wifi->GetMac ();
  if (mac == 0)
    return;
  mac->TraceConnectWithoutContext ("TxErrHeader", m_neighborTable.GetTxErrorCallback ());

}

void antcommProtocol::NotifyInterfaceDown (uint32_t i)
{
  NS_LOG_DEBUG ("NotifyInterfaceDown function");
  NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
  // Disable layer 2 link state monitoring (if possible)
```

```
  Ptr <Ipv4L3Protocol> l3 = m_ipv4->GetObject <Ipv4L3Protocol>();
  Ptr <NetDevice> dev = l3->GetNetDevice (i);
  Ptr <WifiNetDevice> wifi = dev->GetObject <WifiNetDevice>();
  if (wifi != 0)
      {
           Ptr <WifiMac> mac = wifi->GetMac ()->GetObject <AdhocWifiMac> ();
           if (mac != 0)
        {
           mac->TraceDisconnectWithoutContext ("TxErrHeader", m_neighborTable.GetTxErrorCallback ());
    m_neighborTable.DelArpCache (l3->GetInterface (i)->GetArpCache ());
        }
      }
  // Close socket
  Ptr < Socket > socket = FindSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
  NS_ASSERT (socket);
  socket->Close ();
  m_socketAddresses.erase (socket);
  // Close socket
  socket = FindSubnetBroadcastSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
  NS_ASSERT (socket);
  socket->Close ();
  m_socketSubnetBroadcastAddresses.erase (socket);

  if (m_socketAddresses.empty ())
      {
           NS_LOG_LOGIC ("No␣antcomm␣interfaces");
        m_helloTimer.Cancel ();
  m_neighborTable.Clear ();
        return;
      }
}

void antcommProtocol::NotifyAddAddress (uint32_t i, Ipv4InterfaceAddress address)
{
  NS_LOG_DEBUG ("NotifyAddAddress␣function");
  NS_LOG_FUNCTION (this << "␣interface␣" << i << "␣address␣" << address);
  Ptr < Ipv4L3Protocol > l3 = m_ipv4->GetObject < Ipv4L3Protocol > ();
  if (!l3->IsUp (i))
    return;
  if (l3->GetNAddresses (i) == 1)
      {
           Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
           Ptr < Socket > socket = FindSocketWithInterfaceAddress (iface);
           if (!socket)
        {
          if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
            return;
          // Create a socket to listen only on this interface
          Ptr < Socket > socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
          NS_ASSERT (socket != 0);
          socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
          socket->Bind (InetSocketAddress (iface.GetLocal (), antcomm_PORT));
          socket->BindToNetDevice (l3->GetNetDevice (i));
          socket->SetAllowBroadcast (true);
          m_socketAddresses.insert (std::make_pair (socket, iface));

          // create also a subnet directed broadcast socket
          socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
          NS_ASSERT (socket != 0);
          socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
          socket->Bind (InetSocketAddress(iface.GetBroadcast (), antcomm_PORT));
          socket->BindToNetDevice (l3->GetNetDevice (i));
          socket->SetAllowBroadcast (true);
          socket->SetAttribute ("IpTtl", UintegerValue (1));
          m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

          // Add local broadcast record to the routing table
          Ptr < NetDevice > dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
    /** This is how every node has access to its own energy
      through the ipv4 layer 3. */
    Ptr <Node> node = m_ipv4->GetObject< Node > ();
    Ptr<EnergySourceContainer> EnergySourceContrainerOnNode = node->GetObject< EnergySourceContainer> ();
    Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> ( EnergySourceContrainerOnNode->Get(0));

```

```
            antcommRoutingTableEntry rt (dev, /*dst= */iface.GetBroadcast (), /*hops= */ 1,
                 m_pheromone, basicSourcePtr->GetRemainingEnergy(),
                                  /*RSSI=*/ 0, iface.GetBroadcast (), iface, /*lifetime= */Simulator::
                                       GetMaximumSimulationTime ());
            m_routingTable.AddRoute (rt);
          }
      }
    else
      {
        NS_LOG_LOGIC("AODV␣does␣not␣work␣with␣more␣then␣one␣address␣per␣each␣interface.␣Ignore␣added␣
             address");
      }
  }

   void antcommProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
   {
     NS_LOG_DEBUG ("NotifyRemoveAddress␣function");
     NS_LOG_FUNCTION (this);
     Ptr < Socket > socket = FindSocketWithInterfaceAddress (address);
     if (socket)
         {
              socket->Close ();
              m_socketAddresses.erase (socket);

              Ptr < Socket > unicastSocket = FindSubnetBroadcastSocketWithInterfaceAddress (address);
           if (unicastSocket)
           {
              unicastSocket->Close ();
              m_socketAddresses.erase (unicastSocket);
           }
         Ptr < Ipv4L3Protocol > l3 = m_ipv4->GetObject < Ipv4L3Protocol > ();
         if (l3->GetNAddresses (i))
           {
              Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
              // Create a socket to listen only on this interface
              Ptr <Socket> socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::
                   GetTypeId ());
              NS_ASSERT (socket != 0);
              socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));

              // Bind to any IP address so that broadcasts can be received
              socket->Bind (InetSocketAddress (iface.GetLocal (), antcomm_PORT));
              socket->BindToNetDevice (l3->GetNetDevice (i));
              socket->SetAllowBroadcast (true);
              socket->SetAttribute ("IpTtl", UintegerValue (1));
              m_socketAddresses.insert (std::make_pair (socket, iface));

              // create also a unicast socket
              socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
              NS_ASSERT (socket != 0);
              socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
              socket->Bind (InetSocketAddress(iface.GetBroadcast (), antcomm_PORT));
              socket->BindToNetDevice (l3->GetNetDevice (i));
              socket->SetAllowBroadcast (true);
              socket->SetAttribute ("IpTtl", UintegerValue (1));
              m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

              // Add local broadcast record to the routing table
              Ptr <NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.
                   GetLocal ()));
        /** This is how every node has access to its own energy
          through the ipv4 layer 3. */
        Ptr <Node> node = m_ipv4->GetObject< Node > ();
        Ptr<EnergySourceContainer> EnergySourceContrainerOnNode = node->GetObject<
             EnergySourceContainer> ();
        Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
             EnergySourceContrainerOnNode->Get(0));

              antcommRoutingTableEntry rt (dev, /*dst= */iface.GetBroadcast (), /*hops= */
                                                         1, m_pheromone, /*energy=*/basicSourcePtr
                                                              ->GetRemainingEnergy(),
                                  /*RSSi=*/ 0, iface.GetBroadcast (), iface,
                                                         /*lifetime= */Simulator::
                                                              GetMaximumSimulationTime ());
                  m_routingTable.AddRoute (rt);
           }
           if (m_socketAddresses.empty ())
           {
             NS_LOG_LOGIC ("No␣antcomm␣interfaces");
             m_helloTimer.Cancel ();
```

```
      m_neighborTable.Clear ();
          return;
        }
      }
    else
      {
        NS_LOG_LOGIC("Remove␣address␣not␣participating␣in␣antcomm␣operation");
      }
  }


  void antcommProtocol::PrintRoutingTable (Ptr <OutputStreamWrapper>stream) const
  {
    *stream->GetStream () << std::fixed << std::setprecision (10);
    *stream->GetStream () << "Node:␣" << m_ipv4->GetObject <Node>()->GetId () << "␣Time:␣" <<
         Simulator::Now ().GetSeconds () << "s␣";
    //m_routingTable.Print (stream);
    m_neighborTable.PrintTable (stream);
  }


  void antcommProtocol::PrintInformation (Ptr <OutputStreamWrapper>stream) const
  {
    *stream->GetStream () << "Node:␣" << m_ipv4->GetObject<Node>()->GetId ()
                          << ";␣Time:␣" << Now ().As (Time::S)
                          << ",␣Local␣time:␣" << GetObject<Node>()->GetLocalTime ().As (Time::S)
                          << ",␣antcomm␣Routing␣table" << std::endl;

    //m_routingTable.Print (stream);
    m_neighborTable.PrintTable (stream);
    *stream->GetStream () << std::endl;
  }

  Ptr <Socket>antcommProtocol::FindSocketWithInterfaceAddress (Ipv4InterfaceAddress addr) const
  {
    NS_LOG_FUNCTION (this << addr);
    for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
         begin (); j != m_socketAddresses.end (); ++j)
        {
              Ptr < Socket > socket = j->first;
              Ipv4InterfaceAddress iface = j->second;
              if (iface == addr)
                     return socket;
        }
        Ptr < Socket > socket;
    return socket;
  }

  Ptr <Socket>antcommProtocol::FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress
       addr) const
  {
    NS_LOG_FUNCTION (this << addr);
    for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j =
         m_socketSubnetBroadcastAddresses.begin ();
         j != m_socketSubnetBroadcastAddresses.end (); ++j)
      {
              Ptr < Socket > socket = j->first;
              Ipv4InterfaceAddress iface = j->second;
              if (iface == addr)
                     return socket;
        }
        Ptr < Socket > socket;
        return socket;
  }
  //Modified RouteOutput function
  Ptr <Ipv4Route> antcommProtocol::RouteOutput (Ptr <Packet> p, const Ipv4Header &header,
                                                                     Ptr < NetDevice > oif, Socket::
                                                                         SocketErrno & sockerr)
  {
      NS_LOG_DEBUG ("This␣is␣RouteOutput␣functionality");
      NS_LOG_FUNCTION (this << header << (oif ? oif->GetIfIndex () : 0));
      if (!p)
      {
              NS_LOG_DEBUG ("Packet␣is␣==␣0");
              return LoopbackRoute (header, oif); // later
        }
        if (m_socketAddresses.empty ())
        {
            sockerr = Socket::ERROR_NOROUTETOHOST;
            NS_LOG_LOGIC ("No␣antcomm␣interfaces");
```

```
              Ptr <Ipv4Route> route;
              return route;
            }
            sockerr = Socket::ERROR_NOTERROR;
      Ptr < Ipv4Route > route;
      Ipv4Address dst = header.GetDestination ();

/*
        check a valid route with the destination address
        The function looks for the destination address. If there are multiple neighbors,
        it selects one of them by considering phermone, energy, hop count, etc. ,
        and return the result as rt as if LookupValidRoute(dst, rt) function does in aodv


*/

      antcommRoutingTableEntry rt;
      if (m_routingTable.LookupRoute2 (dst, rt))
      {
        route = rt.GetRoute ();
        NS_ASSERT (route != 0);
        NS_LOG_DEBUG ("The␣destination␣through␣route␣is␣" << dst );
        NS_LOG_DEBUG ("Exist␣route␣to␣" << route->GetDestination () << "␣from␣interface␣" << route->GetSource ());
        if (oif != 0 && route->GetOutputDevice () != oif)
        {
          NS_LOG_DEBUG ("Output␣device␣doesn't␣match.␣Dropped.");
          sockerr = Socket::ERROR_NOROUTETOHOST;
          return Ptr < Ipv4Route > ();
        }
       UpdateRouteLifeTime (dst, ActiveRouteTimeout);
       UpdateRouteLifeTime (route->GetGateway (), ActiveRouteTimeout);
       return route;
      }
      // Valid route not found, in this case we return loopback.
      // Actual route request will be deferred until packet will be fully formed,
      // routed to loopback, received from loopback and passed to RouteInput (see below)
      uint32_t iif = (oif ? m_ipv4->GetInterfaceForDevice (oif) : -1);
      DeferredRouteOutputTag tag (iif);
      NS_LOG_DEBUG ("Valid␣Route␣not␣found");
      if (!p->PeekPacketTag (tag))
      {
        p->AddPacketTag (tag);
      }
      return LoopbackRoute (header, oif);
    }


    Ptr <Ipv4Route> antcommProtocol::LoopbackRoute (const Ipv4Header & hdr, Ptr <NetDevice> oif) const
    {
      NS_LOG_FUNCTION (this << hdr);
      NS_ASSERT (m_lo != 0);
      Ptr < Ipv4Route > rt = Create < Ipv4Route > ();
      rt->SetDestination (hdr.GetDestination ());
      std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.begin ();
      if (oif)
            {
        // Iterate to find an address on the oif device
        for (j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
              {
          Ipv4Address addr = j->second.GetLocal ();
          int32_t interface = m_ipv4->GetInterfaceForAddress (addr);
          if (oif == m_ipv4->GetNetDevice (static_cast < uint32_t > (interface)))
                        {
            rt->SetSource (addr);
            break;
                        }
             }
            }
      else
            {
        rt->SetSource (j->second.GetLocal ());
            }

      NS_ASSERT_MSG (rt->GetSource () != Ipv4Address (), "Valid␣antcomm␣source␣address␣not␣found");
      rt->SetGateway (Ipv4Address ("127.0.0.1"));
      rt->SetOutputDevice (m_lo);
      return rt;
    }
```

```
void antcommProtocol::DeferredRouteOutput (Ptr < const Packet > p, const Ipv4Header & header,
                                            UnicastForwardCallback ucb,
                                                ErrorCallback ecb)
{
  NS_LOG_FUNCTION (this << p << header);
  NS_ASSERT (p != 0 && p != Ptr < Packet > ());

  QueueEntry newEntry (p, header, ucb, ecb);
  bool result = m_queue.Enqueue (newEntry);
  if (result)
      {
    NS_LOG_LOGIC ("Add packet " << p->GetUid () << " to queue. Protocol " << (uint16_t)header.
        GetProtocol ());
    antcommRoutingTableEntry rt;
    bool result = m_routingTable.LookupRoute2 (header.GetDestination (), rt);
    if(!result)
    {
      NS_LOG_LOGIC ("Send new RREQ for outbound packet to " <<header.GetDestination ());
      SendEXPL (header.GetDestination ());
    }
       }

}

bool antcommProtocol::IsMyOwnAddress (Ipv4Address src)
{
  NS_LOG_FUNCTION (this << src);
  for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j =
           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
      {
    Ipv4InterfaceAddress iface = j->second;
    if (src == iface.GetLocal ())
        {
      return true;
        }
      }
  return false;
}

bool antcommProtocol::RouteInput (Ptr < const Packet > p, const Ipv4Header &header,
                                  Ptr < const NetDevice > idev,
                                      UnicastForwardCallback ucb,
                                  MulticastForwardCallback mcb,
                                      LocalDeliverCallback lcb,
                                  ErrorCallback ecb)
{
  NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->GetAddress ());
  NS_LOG_DEBUG ("antcomm node " << m_ipv4->GetObject<Node> ()->GetId () << " from RouteInput 
      received a packet ");
  if (m_socketAddresses.empty ())
      {
    NS_LOG_LOGIC ("No antcomm interfaces");
    return false;
      }

  NS_ASSERT (m_ipv4 != 0);
  NS_ASSERT (p != 0);
  // Check if input device supports IP
  NS_ASSERT (m_ipv4->GetInterfaceForDevice (idev) >= 0);
  int32_t iif = m_ipv4->GetInterfaceForDevice (idev);

  Ipv4Address dst = header.GetDestination ();
  Ipv4Address origin = header.GetSource ();

  NS_LOG_DEBUG ("dst through RouteInput is " << dst);
  NS_LOG_DEBUG ("origin through RouteInput is " << origin);

  // Deferred route request
  if (idev == m_lo)
      {
    DeferredRouteOutputTag tag;
    if (p->PeekPacketTag (tag))
        {
      DeferredRouteOutput (p, header, ucb, ecb);
      return true;
        }
      }
  // Duplicate of own packet
  if (IsMyOwnAddress (origin))
```

```
        {
      return true;
        }
    if (dst.IsMulticast ())
        {
      return false;
        }

    for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j =
           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
        {
      Ipv4InterfaceAddress iface = j->second;
      if (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
          {
        if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
                  {
                    if (m_dpd.IsDuplicate (p, header))
                    {
                            NS_LOG_DEBUG ("Duplicated␣packets␣" << p->GetUid() << "␣from␣" << origin << "␣.Drop␣");
                            return true;
                    }
          UpdateRouteLifeTime (origin, ActiveRouteTimeout);
          Ptr < Packet > packet = p->Copy ();
          if (lcb.IsNull () == false)
                    {
            NS_LOG_LOGIC ("Broadcast␣local␣delivery␣to␣" << iface.GetLocal ());
            lcb (p, header, iif);
                    }
          else
                    {
            NS_LOG_ERROR("Unable␣to␣deliver␣packet␣locally␣due␣to␣null␣callback␣"<< p->GetUid () << "␣from␣" << origin);
            ecb (p, header, Socket::ERROR_NOROUTETOHOST);
                    }
          if (!EnableBroadcast)
                    {
            return true;
                    }
          if (header.GetTtl () > 1)
                    {
            NS_LOG_LOGIC ("Forward␣broadcast.␣TTL␣" << (uint16_t)header.GetTtl ());
            antcommRoutingTableEntry toBroadcast;
            if (m_routingTable.LookupRoute2 (dst, toBroadcast))
                            {
              Ptr < Ipv4Route > route = toBroadcast.GetRoute ();
              ucb (route, packet, header);
                            }
            else
                            {
              NS_LOG_DEBUG ("No␣route␣to␣forward␣broadcast.␣Drop␣packet␣" << p->GetUid ());
                            }
                    }
          else
          {
             NS_LOG_DEBUG ("TTL␣exceeded.␣Drop␣packet␣" << p->GetUid ());
          }
          return true;
                  }
         }
      }
    // Unicast local delivery
    if (m_ipv4->IsDestinationAddress (dst, iif))
        {
      double x = 0;
      SnrTag tag;
      if (p->PeekPacketTag(tag))
      {
        NS_LOG_DEBUG ("the␣snr␣of␣the␣packet␣is␣" << tag.Get() << "dBm");
        x = tag.Get();
      }

      UpdateRouteLifeTime (origin, ActiveRouteTimeout);
      antcommRoutingTableEntry toOrigin;
      if (m_routingTable.LookupRoute2 (origin, toOrigin))
      {

        UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);
        double rssi = m_neighborTable.GetRssi (toOrigin.GetNextHop ());
        double energy = m_neighborTable.GetEnergy (toOrigin.GetNextHop ());
```

```
          m_neighborTable.Update (toOrigin.GetNextHop (), rssi, energy, ActiveRouteTimeout);
        }
        if (lcb.IsNull () == false)
              {
          NS_LOG_LOGIC ("Unicast local delivery to " << dst);
          lcb (p, header, iif);
              }
        else
              {
          NS_LOG_ERROR ("Unable to deliver packet locally due to null callback " << p->GetUid () << " from " << origin);
          ecb (p, header, Socket::ERROR_NOROUTETOHOST);
              }
        return true;
            }

      // Check if input device supports IP forwarding
      if (m_ipv4->IsForwarding (iif) == false)
      {
        NS_LOG_LOGIC ("Forwarding disabled for this interface");
        ecb (p, header, Socket::ERROR_NOROUTETOHOST);
        return true;
      }

    // Forwarding
      return Forwarding (p, header, ucb, ecb);
  }

    bool antcommProtocol::Forwarding (Ptr < const Packet > p, const Ipv4Header & header,
                                                            UnicastForwardCallback ucb, ErrorCallback ecb)
    {
      NS_LOG_DEBUG("Forwarding function ");
      NS_LOG_FUNCTION (this);
      Ipv4Address dst = header.GetDestination ();
      Ipv4Address origin = header.GetSource ();

      antcommRoutingTableEntry toDst;
      if (m_routingTable.LookupRoute2 (dst, toDst))
            {
        if (toDst.GetPheromone() < 0.5)
          toDst.SetPheromone(INITIAL_PHEROMONE);
        Ptr < Ipv4Route > route = toDst.GetRoute ();
        NS_LOG_LOGIC (route->GetSource () << " forwarding to " << dst << " from " << origin << " packet " << p->GetUid ());

        toDst.SetPheromone (GetPheromone() + 0.5);
        UpdateRouteLifeTime (origin, ActiveRouteTimeout);
        UpdateRouteLifeTime (dst, ActiveRouteTimeout);
        UpdateRouteLifeTime (route->GetGateway (), ActiveRouteTimeout);

        antcommRoutingTableEntry toOrigin;
        m_routingTable.LookupRoute2 (origin, toOrigin);
        UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);

        m_neighborTable.Update (route->GetGateway (), m_neighborTable.GetRssi (route->GetGateway ()), m_neighborTable.GetEnergy (route->GetGateway ()), ActiveRouteTimeout);
        m_neighborTable.Update (toOrigin.GetNextHop (), m_neighborTable.GetRssi (toOrigin.GetNextHop ()), m_neighborTable.GetEnergy (toOrigin.GetNextHop ()), ActiveRouteTimeout);

        ucb (route, p, header);
        return true;
            }
      else
            {
        NS_LOG_DEBUG ("Drop packet " << p->GetUid () << " because no route to forward it.");
        return false;
            }
      //return false;
    }


    void antcommProtocol::Start ()
    {
      NS_LOG_FUNCTION (this);
      //m_explorationTimer.SetFunction (&antcommProtocol::SendHello, this);
      //m_explorationTimer.Schedule (Seconds (m_explorationTime));
      m_neighborTable.ScheduleTimer ();
      m_forwardTimer.SetFunction (&antcommProtocol::ExplRateLimitTimerExpire, this);
      m_forwardTimer.Schedule (Seconds (1));
      //m_backwardTimer.SetFunction (&antcommProtocol::SendREPA, this);
```

```
    //m_backwardTimer.Schedule (Seconds (1));
  }

  void antcommProtocol::SendTo (Ptr <Socket> socket, Ptr <Packet> packet, Ipv4Address destination)
  {
    NS_LOG_FUNCTION (this);

    //std::cout << "Sent a data packet " << this << "\n";
    socket->SendTo (packet, 0, InetSocketAddress (destination, antcomm_PORT));
  }

  void antcommProtocol::Recvantcomm (Ptr < Socket > socket)
  {
    Address sourceAddress;
    Ptr < Packet > packet = socket->RecvFrom (sourceAddress);
    InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
    Ipv4Address sender = inetSourceAddr.GetIpv4 ();
    Ipv4Address my;
    if (m_socketAddresses.find (socket) != m_socketAddresses.end ())
          {
      my = m_socketAddresses[socket].GetLocal ();
          }
    else if (m_socketSubnetBroadcastAddresses.find (socket) != m_socketSubnetBroadcastAddresses.end
         ())
          {
      my = m_socketSubnetBroadcastAddresses[socket].GetLocal ();
          }
    else
          {
      NS_ASSERT_MSG (false, "Received␣a␣packet␣from␣an␣unknown␣socket");
          }
    NS_LOG_DEBUG ("antcomm␣node␣" << m_ipv4->GetObject<Node> ()->GetId () << "␣received␣a␣antcomm␣
         packet␣from␣" << sender << "␣to␣" << my);

    TypeHeader tHeader (EXPL);
    packet->RemoveHeader (tHeader);
    //uint16_t myNodeID = m_ipv4->GetObject<Node> ()->GetId (); //uint16_t
    if (!tHeader.IsValid ())
          {
      NS_LOG_DEBUG ("antcomm␣message␣" << packet->GetUid () << "␣with␣unknown␣type␣received:␣" <<
           tHeader.Get () << ".␣Drop");
      return;  // drop
          }
    //if (IsMyOwnAddress("10.0.0.5")) {
    //NS_LOG("lsdfkjasldfkj");
    //}
    if (tHeader.Get () == EXPL)
      NS_LOG_INFO ("The␣packet␣was␣tagged␣as␣EXPL␣and␣then␣it␣was␣received");

    if (tHeader.Get () == REPA && IsMyOwnAddress ("10.0.0.5"))
      NS_LOG_INFO ("The␣packet␣was␣tagged␣as␣REPA␣and␣hopefully␣it␣should␣be␣received");



    switch (tHeader.Get ())
          {
      case HELLO:
            {
        RecvHELLO (packet, my, sender);
        //MakeRoutingTable (my);
        break;
            }
      case EXPL:
            {
        RecvEXPL (packet, my, sender);
        break;
            }
      case REPA:
            {
        RecvREPA (packet, my, sender);
        break;
            }
      case RE_REPA:
            {
        antcommREREPAHeader headerRerepa;
        packet->RemoveHeader (headerRerepa);
        //RecvREREPA (headerRerepa, my, headerRerepa.GetOrigin (), sender);
        break;
            }

```

```
        }
}

bool antcommProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
{
  NS_LOG_FUNCTION (this << addr << lifetime);
  antcommRoutingTableEntry rt;
  if (m_routingTable.LookupRoute2 (addr, rt))
  {
      //NS_LOG_DEBUG ("Updating VALID route");
      rt.SetUpdatedTime (std::max (lifetime, rt.GetUpdatedTime ()));
      m_routingTable.Update (rt); // update
      return true;

  }
  return false;
 }

 void antcommProtocol::MakeRoutingTable (Ipv4Address origin, Ipv4Address my, Ipv4Address src) // modifying right now
 {
  int degree = 0;
  degree = m_neighborTable.Degree();
  antcommRoutingTableEntry toOrigin;

  while (degree-- > 0)
  {
    //std::cout << "***" << my << " " << degree+1 << ": " << m_neighborTable.GetAddress (degree) << std::endl;
    if (src.IsEqual (m_neighborTable.GetAddress (degree)))
    {
      if (m_routingTable.LookupRoute (origin, m_neighborTable.GetAddress (degree), toOrigin) == 0)
    {
      Ptr <NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my));
      antcommRoutingTableEntry newEntry (dev, origin, 1, 0.2, m_neighborTable.GetEnergy (src),
                                    m_neighborTable.GetRssi (src), src,
                                    m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (my), 0),
                                    Time ((2* NetTraversalTime - 2 * NodeTraversalTime)));
      m_routingTable.AddRoute (newEntry);
    }
    }


  }
  //return;

  //std::cout << "*** while done\n";
 }

/// Receive HELLO
void antcommProtocol::RecvHELLO (Ptr < Packet > p, Ipv4Address my, Ipv4Address sender)
{
  antcommHelloHeader headerHello;
  p->RemoveHeader (headerHello);
  NS_LOG_INFO ("HELLO␣from␣" << sender << ",␣␣origin␣" << headerHello.GetOrigin () << ",␣to␣" << my);
  headerHello.SetTimeToLive (headerHello.GetTimeToLive () - 1);
  headerHello.SetHopCount (headerHello.GetHopCount () + 1);
  double x = 0;
  double distance;
  double probability;
  double goodness_rssi = 0;
  //double rssi_previous = 0;
  SnrTag tag;
  if (p->PeekPacketTag(tag))
  {
    NS_LOG_DEBUG ("the␣snr␣of␣the␣packet␣is␣" << tag.Get() << "dBm");
    x = tag.Get();
  }
  distance = exp((x - 117.17)/(-28.35));
  if (distance/50.92 < 1)
  {
     probability = 1- 0.5 * pow ((distance/50.92), 4);
  }
  else
    probability = 0.5 * pow(((2*50.92-distance)/50.92), 4);

  goodness_rssi = distance * pow (probability, 2) / (50.92 * (1 + probability));

```

```
    //std::cout << "The residual energy of node in hello function is " << headerHello.GetResidualEnergy () << std::endl;
    /* Higher RSSI value (meaning higher number of hops) and lower RSSI value (higher packet drop chances)
       is not good for our probabilistic system. Hence our goodness_rssi should follow that curve that
       has minima on extreme ends and maxima in a middle. The curve a*xe^(-x) is goodness rssi where "a" is
       any constant and x is rssi value from physical layer. In our case I have taken "a" as 7
    */
    //goodness_rssi = 7*x*exp(-x);
    //m_rssi = goodness_rssi;

    /* std::cout << " receiving hello message\n" <<
       " from "<<(Ipv4Address)sender<<std::endl<<
       " origin "<<(Ipv4Address)headerHello.GetOrigin()<<std::endl<<
       " to "<<(Ipv4Address)my<<std::endl; */
    m_neighborTable.Update (sender, goodness_rssi, headerHello.GetResidualEnergy(), Time (AllowedHelloLoss * HelloInterval));

  }

  void antcommProtocol::RecvEXPL (Ptr <Packet> p, Ipv4Address my, Ipv4Address src)
  {
    NS_LOG_FUNCTION (this);
    antcommEXPLHeader headerEXPL;
    p->RemoveHeader (headerEXPL);

    /* std::cout << " I am receiving Request message\n" <<
       " from "<<(Ipv4Address)sender<<std::endl<<
       " originator "<<(Ipv4Address)headerEXPL.GetOrigin()<<std::endl<<
       " My IP address is "<<(Ipv4Address)my<<std::endl<<
       " The intended destination is "<<(Ipv4Address)headerEXPL.GetDst()<<std::endl; */

    NS_LOG_INFO ("Request␣EXPL␣from␣" << src << ",␣␣origin␣" << headerEXPL.GetOrigin () << ",␣to␣" << my);
    Ipv4Address destination = headerEXPL.GetDst ();
    std::cout << destination << std::endl; // 10.0.0.5
    Ipv4Address origin = headerEXPL.GetOrigin ();
    std::cout << origin << std::endl; // 10.0.0.1

    MakeRoutingTable (origin, my, src);

    uint32_t id = headerEXPL.GetId ();
    if (m_rreqIdCache.IsDuplicate (origin, id))
    {
      NS_LOG_DEBUG ("Ignoring␣request␣message␣due␣to␣duplication");
      return;
    }

    //Increment RREQ hop Count
    uint16_t hop = headerEXPL.GetHopCount () + 1;
    headerEXPL.SetHopCount (hop);
    headerEXPL.SetTimeToLive (headerEXPL.GetTimeToLive () - 1);
    //std::cout<<"I am this hop "<<(double)headerEXPL.GetHopCount() << " away from "<<(Ipv4Address)headerEXPL.GetOrigin()<<std::endl;
    antcommRoutingTableEntry toOrigin; // Discuss this methodology, AODV has the same
    if (m_routingTable.LookupRoute (origin, src, toOrigin))
          //If the routing table entry exists, set the parameters and update the routing table
          {
      if(toOrigin.GetPheromone() < 0.5)
        toOrigin.SetPheromone(INITIAL_PHEROMONE);
      toOrigin.SetPheromone (GetPheromone() + 0.2);
      toOrigin.SetHopCount (hop);
      toOrigin.SetRssi (m_neighborTable.GetRssi (src));
      toOrigin.SetNextHop (src);
      toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress(my)));
      toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (my), 0));
      toOrigin.SetResidualEnergy (m_neighborTable.GetEnergy (src));
      toOrigin.SetUpdatedTime (std::max (Time (2 * NetTraversalTime -2 * hop * NodeTraversalTime), toOrigin.GetUpdatedTime ()));
      m_routingTable.Update (toOrigin); // update
          }
    else   // otherwise make a new entry and add it in routing table
          {
      Ptr <NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my));

      antcommRoutingTableEntry newEntry (dev, origin, hop, INITIAL_PHEROMONE, m_neighborTable.GetEnergy (src),
                                                                            m_neighborTable.GetRssi (src), src,
```

```
                                                    m_ipv4->GetAddress (m_ipv4->
                                                        GetInterfaceForAddress(my), 0),
                                                    Time ((2 * NetTraversalTime - 2 *
                                                        hop * NodeTraversalTime)));
        //std::cout << "Adding route (" << src << "," << headerEXPL.GetOrigin() <<")\n";
        NS_LOG_INFO ("Adding␣route␣(␣" <<src << ",␣" << origin << "\n");
        m_routingTable.AddRoute (newEntry);
            }

      if (IsMyOwnAddress (headerEXPL.GetDst ()))
      {
        m_routingTable.LookupRoute2 (origin, toOrigin);
        NS_LOG_DEBUG ("Send␣reply␣since␣I␣am␣the␣destination");
        SendREPA (headerEXPL, toOrigin);
        return;
      }
      //else scan the entries from my socket
      for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
          begin (); j != m_socketAddresses.end (); ++j)
      {
        Ptr < Socket > socket = j->first;
        Ipv4InterfaceAddress iface = j->second;
        Ptr < Packet > packet = Create < Packet > ();
        packet->AddHeader (headerEXPL);
        TypeHeader tHeader (EXPL);
        packet->AddHeader (tHeader);
        // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
        Ipv4Address destination;
        if (iface.GetMask () == Ipv4Mask::GetOnes ())
  {
          destination = Ipv4Address ("255.255.255.255");
  }
        else
  {
          destination = iface.GetBroadcast ();
  }

        m_lastBcastTime = Simulator::Now ();
        Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
                            &antcommProtocol::SendTo, this, socket, packet, destination);

       }



    }
    void antcommProtocol::SendReplyByIntermediateNode (antcommRoutingTableEntry & toDst,
        antcommRoutingTableEntry & toOrigin)
    {
      NS_LOG_FUNCTION (this);
      antcommREPAHeader headerRepa (/*hops=*/ toDst.GetHopCount (), /*Time To Live*/ 255, /*origin=*/
          toOrigin.GetDestination (), /*dst=*/ toDst.GetDestination ());

      m_routingTable.Update (toDst);
      m_routingTable.Update (toOrigin);

      Ptr<Packet> packet = Create<Packet> ();
      packet->AddHeader (headerRepa);
      TypeHeader tHeader (REPA);
      packet->AddHeader (tHeader);
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
      NS_ASSERT (socket);
      socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), antcomm_PORT));

      // Generating gratuitous RREPs (add later if required, otherwise no need)

    }
    /// Receive REPA
    void antcommProtocol::RecvREPA (Ptr < Packet > p, Ipv4Address my, Ipv4Address sender)
    {
      NS_LOG_FUNCTION (this << "neighbor␣is" << sender);
      antcommREPAHeader headerRepa;
      p->RemoveHeader (headerRepa);

      NS_LOG_INFO ("Reply␣from␣" << sender << ",␣␣origin␣" << headerRepa.GetDst () << ",␣to␣" << my);
      //std::cout<<" I have received the reply packet" << std::endl;
      Ipv4Address dst = headerRepa.GetDst ();

      std::cout << "destination␣through␣REPA␣header␣is␣" << dst << std::endl; // 10.0.0.5

```

```
  std::cout << "origin through REPA header is " << headerRepa.GetOrigin() << std::endl; //10.0.0.1

  uint16_t hop = headerRepa.GetHopCount () + 1;
  headerRepa.SetHopCount (hop);
  headerRepa.SetTimeToLive (headerRepa.GetTimeToLive () - 1);
  /*Each node receiving REPA will check the routing table entry
     for destination (generator of EXPL message). If the entry exist,
     update the entries otherwise add */
  Ptr < NetDevice > dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my));

  antcommRoutingTableEntry newEntry (dev, dst, hop, INITIAL_PHEROMONE, m_neighborTable.GetEnergy (
       sender),
                                                              m_neighborTable.GetRssi (sender),
                                                                   sender, m_ipv4->GetAddress (
                                                                   m_ipv4->GetInterfaceForAddress(my
                                                                   ), 0), Simulator::Now ());
  antcommRoutingTableEntry toDst;
  if (m_routingTable.LookupRoute (dst, sender, toDst))
        {
    if (toDst.GetPheromone() < 0.5)
      toDst.SetPheromone(INITIAL_PHEROMONE);
    toDst.SetPheromone (GetPheromone() + 0.2);
    toDst.SetHopCount (hop);
    toDst.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my)));
    toDst.SetNextHop (sender);
    toDst.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (my), 0));
    toDst.SetResidualEnergy (m_neighborTable.GetEnergy (sender));
    toDst.SetRssi (m_neighborTable.GetRssi (sender));
    m_routingTable.Update (toDst); // update
        }
  else
    m_routingTable.AddRoute (newEntry);

  if (IsMyOwnAddress (headerRepa.GetOrigin ()))
        {
    std::cout << headerRepa.GetOrigin() << std::endl;
    NS_LOG_DEBUG ("Route establishment phase completed since I am the source and have received 
         REPA packet");
    m_routingTable.LookupRoute2 (dst, toDst);
    SendPacketFromQueue (dst, toDst.GetRoute ());
    return;
        }
  //SendREREPA();

  antcommRoutingTableEntry toOrigin;
  if (m_routingTable.LookupRoute2 (headerRepa.GetOrigin (), toOrigin))
  {
     toOrigin.SetUpdatedTime (std::max (ActiveRouteTimeout, toOrigin.GetUpdatedTime ()));
     m_routingTable.Update (toOrigin); // update
  }
  else
    return;
  Ptr < Packet > packet = Create < Packet > ();
  packet->AddHeader (headerRepa);
  TypeHeader tHeader (REPA);
  packet->AddHeader (tHeader);
  Ptr < Socket > socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
  NS_ASSERT (socket);
  socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), antcomm_PORT));
}

void antcommProtocol::SendPacketFromQueue (Ipv4Address dst, Ptr <Ipv4Route> route)
{
  NS_LOG_FUNCTION (this);
  std::cout << "test case to see whether packets from buffer are being sent or not" << std::endl;
  QueueEntry queueEntry;
  while (m_queue.Dequeue (dst, queueEntry))
        {
    DeferredRouteOutputTag tag;
    Ptr <Packet> p = ConstCast <Packet> (queueEntry.GetPacket ());
    if (p->RemovePacketTag (tag) &&
        tag.GetInterface () != -1 &&
        tag.GetInterface () != m_ipv4->GetInterfaceForDevice (route->GetOutputDevice ()))
          {
      NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
      return;
          }
  UnicastForwardCallback ucb = queueEntry.GetUnicastForwardCallback ();
  Ipv4Header header = queueEntry.GetIpv4Header ();
  header.SetSource (route->GetSource ());
```

```
    header.SetTtl (header.GetTtl () + 1); // compensate extra TTL decrement by fake loopback routing
    ucb (route, p, header);
        }
  }

  void antcommProtocol::SendEXPL (Ipv4Address dst)
  {

    if (m_explCount == EXPLRateLimit)
        {
      Simulator::Schedule (m_forwardTimer.GetDelayLeft () + MicroSeconds (100),
                           &antcommProtocol::SendEXPL, this, dst);
      return;
        }
    else
      m_explCount++;
    //Create RREQ header
    antcommEXPLHeader headerEXPL;
    headerEXPL.SetDst (dst);
    antcommRoutingTableEntry rt;
    if (m_routingTable.LookupRoute2 (dst, rt))
        {
      headerEXPL.SetHopCount (rt.GetHopCount ());
      m_routingTable.Update (rt); // update
        }
    else
        {
      Ptr < NetDevice > dev = 0;
      Ptr <Node> node = m_ipv4->GetObject< Node > ();
      Ptr<EnergySourceContainer> EnergySourceContrainerOnNode = node->GetObject<
           EnergySourceContainer> ();
      Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
           EnergySourceContrainerOnNode->Get(0));
      antcommRoutingTableEntry newEntry (dev, dst, 1/*0*/, GetPheromone(), basicSourcePtr->
           GetRemainingEnergy(), 0, Ipv4Address (), Ipv4InterfaceAddress (), Seconds (0));
      m_routingTable.AddRoute (newEntry);
        }

    //UpdateCurrentPhase (EXPLORATION);
        m_requestId++;
        headerEXPL.SetId (m_requestId);
    headerEXPL.SetHopCount (0);
    headerEXPL.SetTimeToLive (255);
    for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
         begin (); j != m_socketAddresses.end (); ++j)
        {
      Ptr < Socket > socket = j->first;
      Ipv4InterfaceAddress iface = j->second;

        headerEXPL.SetOrigin (iface.GetLocal ());
        m_rreqIdCache.IsDuplicate (iface.GetLocal (), m_requestId);
        Ptr < Packet > packet = Create < Packet > ();
  packet->AddHeader (headerEXPL);
        TypeHeader tHeader (EXPL);
        packet->AddHeader (tHeader);
        Ipv4Address destination;
        if (iface.GetMask () == Ipv4Mask::GetOnes ())
          {
            destination = Ipv4Address ("255.255.255.255");
          }
        else
          {
            destination = iface.GetBroadcast ();
          }
        NS_LOG_DEBUG ("Send RREQ with id " << headerEXPL.GetId () << " to socket");
        m_lastBcastTime = Simulator::Now ();
        //SendTo (socket, packet, Ipv4Address ("255.255.255.255"), headerEXPL.GetSerializedSize ());
        Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
                             &antcommProtocol::SendTo, this, socket, packet, destination);
          }

}


  void antcommProtocol::SendREPA (antcommEXPLHeader const & headerEXPL, antcommRoutingTableEntry
       const & toOrigin)
  {
    NS_LOG_FUNCTION (this << toOrigin.GetDestination ());
    NS_LOG_INFO ("This is a function sending reply packets");
    std::cout << "The dst from rreq packet is " << headerEXPL.GetDst() << std::endl;
```

```
  std::cout << "The␣origin␣I␣read␣from␣toOrigin␣routingTable␣entry␣is␣" << toOrigin.GetDestination () << std::endl;
  antcommREPAHeader headerRepa (0, /*Time to Live*/255, toOrigin.GetDestination (), /*myNodeID */ headerEXPL.GetDst ());
  Ptr < Packet > packet = Create < Packet > ();
  packet->AddHeader (headerRepa);
  TypeHeader tHeader (REPA);
  packet->AddHeader (tHeader);
  Ptr < Socket > socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
  NS_ASSERT (socket);
  std::cout << "Socket␣is␣confirmed␣" << std::endl;
  socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), antcomm_PORT));
  //SendTo (socket, packet, Ipv4Address ("255.255.255.255"), headerRepa.GetSerializedSize ());
  //}
}

void antcommProtocol::SendREREPA ()
{
  NS_LOG_FUNCTION (this);

  for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
      {
    Ptr < Socket > socket = j->first;
    //uint16_t myNodeID = m_ipv4->GetObject<Node> ()->GetId (); //testing by commenting out
    antcommREREPAHeader header (header.GetHopCount (), 255, header.GetPheromone (), 0, header.GetOrigin (), header.GetDst ()); // the same
    Ptr < Packet > packet = Create < Packet > ();
    packet->AddHeader (header);
    TypeHeader tHeader (RE_REPA);
    packet->AddHeader (tHeader);
    //SendTo (this, socket, packet, Ipv4Address ("255.255.255.255"));
      }

}




void antcommProtocol::SendHello ()
{
  NS_LOG_FUNCTION (this);
  NS_LOG_DEBUG("Hello␣message␣sent");
  Ptr <Node> node = m_ipv4->GetObject< Node > ();
  Ptr<EnergySourceContainer> EnergySourceContrainerOnNode = node->GetObject<EnergySourceContainer> ();
  // Convert the type from EnergySourceContainer to BasicEnergy Source by using Dynamic Cast
  Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (EnergySourceContrainerOnNode->Get(0));

  NS_LOG_DEBUG ("Remaining␣energy␣in␣send␣hello␣is␣" << basicSourcePtr->GetRemainingEnergy ());
  if (basicSourcePtr->GetRemainingEnergy () == 0 || basicSourcePtr->GetRemainingEnergy () < 0)
  {
    //Get the radio model from BasicEnergySource installed on node
    Ptr<DeviceEnergyModel> basicRadioModelPtr = basicSourcePtr->FindDeviceEnergyModels ("ns3::WifiRadioEnergyModel").Get(0);
    // Set all the currents to zero such that the dead node doesn't consume energy
    basicRadioModelPtr->SetAttribute("RxCurrentA", DoubleValue (0.0));
    basicRadioModelPtr->SetAttribute("TxCurrentA", DoubleValue (0.0));
    basicRadioModelPtr->SetAttribute("IdleCurrentA", DoubleValue (0.0));
    basicRadioModelPtr->SetAttribute("CcaBusyCurrentA", DoubleValue (0.0));
    NS_LOG_DEBUG ("Shutting␣down␣the␣interface␣at␣time␣" << Simulator::Now ().GetSeconds ());
    std::cout << "Shutting␣down␣the␣interface␣at␣time␣" << Simulator::Now ().GetSeconds () << std::endl;
    // 1 in the SetDown parameter is the Loopback interface index
    m_ipv4->SetDown (1);
  }
  for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
      {
    Ptr < Socket > socket = j->first;
    Ipv4InterfaceAddress iface = j->second;
    antcommHelloHeader helloHeader (1, 0, basicSourcePtr->GetRemainingEnergy (), iface.GetLocal (), iface.GetLocal (), Time (AllowedHelloLoss * HelloInterval)); // for source ip, it is iface.getlocal ()
    Ptr < Packet > packet = Create < Packet > ();
    packet->AddHeader (helloHeader);
    TypeHeader tHeader (HELLO);
    packet->AddHeader (tHeader);
    Ipv4Address destination;
```

```
        if (iface.GetMask () == Ipv4Mask::GetOnes ())
              {
          destination = Ipv4Address ("255.255.255.255");
              }
        else
              {
          destination = iface.GetBroadcast ();
              }
        Simulator::Schedule ( Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 100))),
                                       &antcommProtocol::SendTo, this, socket, packet, destination);

            }

    }

    void antcommProtocol::HelloTimerExpire ()
    {
      NS_LOG_FUNCTION (this);
      Time offset = Time (Seconds (0));
      if (m_lastBcastTime > Time (Seconds (0)))
            {
        offset = Simulator::Now () - m_lastBcastTime;
        NS_LOG_DEBUG ("Hello␣deferred␣due␣to␣last␣bcast␣at:" << m_lastBcastTime);
            }
      else
            {
        SendHello ();
            }
      m_helloTimer.Cancel ();
      Time diff = HelloInterval - offset;
      m_helloTimer.Schedule (std::max (Time (Seconds (0)), diff));
      //m_helloTimer.Schedule(Seconds (m_helloInterval));
      m_lastBcastTime = Time (Seconds (0));
    }
    void antcommProtocol::ExplRateLimitTimerExpire ()
    {
      NS_LOG_FUNCTION (this);
      m_explCount = 0;
      m_forwardTimer.Schedule (Seconds (1));
    }

    double antcommProtocol::GetPheromone ()
    {
      Time tm = Simulator::Now ();
      Time offset = tm - m_updatedTimeOfPheromone;
      double x = 0.0;
      x = exp (-offset.GetSeconds () * m_pheromoneDecayRate) * m_pheromone;
      m_pheromone = std::max (x, m_epsilon);
      m_updatedTimeOfPheromone = tm;
      return m_pheromone;
    }

    void antcommProtocol::DoInitialize (void)
    {
      NS_LOG_FUNCTION (this);
      uint32_t startTime;
      m_helloTimer.SetFunction (&antcommProtocol::HelloTimerExpire, this);
      startTime = m_uniformRandomVariable->GetInteger (0, 100);
      //std::cout << "Starting at time" << startTime << "ms" << std::endl;
      NS_LOG_DEBUG ("Starting␣at␣time␣" << startTime << "ms");
      m_helloTimer.Schedule (MilliSeconds (startTime));

      Ipv4RoutingProtocol::DoInitialize ();
    }

  }    //namespace antcomm
}    //namespace ns3
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#ifndef antcommPROTOCOL_H
#define antcommPROTOCOL_H

#define INITIAL_HOP_COUNT   9999
//50m
#define TRANSMISSION_RANGE   60.0
#define DISTANCE_FOR_TIMER   0.5

#define INITIAL_PHEROMONE   1.0
#define INCREMENTAL_PHEROMONE   1E-5
#define PHEROMONE_DECAY_RATE  0.5
#define EPSILON 1.0E-6
#define PHEROMONE_THRESHOLD 3*INITIAL_PHEROMONE
#define TIME_MIN_FOR_WAITING_TIMER 30
#define TIME_MAX_FOR_WAITING_TIMER 300
#define ALPHA       0.33
#define BETA       0.33
#define GAMMA       0.33

#define TIME_FOR_HELLO_TIMER  4
//s
#define EXPLORATION_TIME    10
#define REPAIR_TIME     900

#include "antcomm-packet.h"
#include "antcomm-rtable.h"
#include "antcomm-rqueue.h"
#include "antcomm-dpd.h"
#include "antcomm-neighbor.h"
#include "ns3/node.h"
#include "randomx.h"
#include "ns3/random-variable-stream.h"
#include "ns3/output-stream-wrapper.h"
#include "ns3/ipv4-routing-protocol.h"
#include "ns3/ipv4-interface.h"
#include "ns3/ipv4-l3-protocol.h"
#include "ns3/network-module.h"
#include <map>
#include <iostream>

namespace ns3
{
  namespace antcomm
  {

    enum Phase
    {
      EXPLORATION = 1,  //!< Exploration
      FORAGING = 2,  //!< Foraging
      REPAIRING = 3  //!< Repairing
    };

    enum PheromoneUpdateType
    {
      EVENT = 1,
      RECALCULATION = 2
    };

                /**
                * \ingroup antcomm
                *
                * \brief antcomm protocol
                */
    class antcommProtocol:public Ipv4RoutingProtocol
    {

    public:

      static TypeId GetTypeId (void);
      static const uint32_t antcomm_PORT;

      antcommProtocol ();

      virtual ~ antcommProtocol ();
      virtual void DoDispose ();
```

```

    // Inherited from Ipv4RoutingProtocol
    Ptr < Ipv4Route > RouteOutput (Ptr < Packet > p, const Ipv4Header & header, Ptr < NetDevice > oif, Socket::SocketErrno & sockerr);
    bool RouteInput (Ptr < const Packet > p, const Ipv4Header & header, Ptr < const NetDevice > idev ,
                                  UnicastForwardCallback ucb, MulticastForwardCallback mcb,
                     LocalDeliverCallback lcb, ErrorCallback ecb);
    Ptr < Ipv4Route > LoopbackRoute (const Ipv4Header & hdr, Ptr < NetDevice > oif) const;
    virtual void NotifyInterfaceUp (uint32_t interface);
    virtual void NotifyInterfaceDown (uint32_t interface);
    virtual void NotifyAddAddress (uint32_t interface, Ipv4InterfaceAddress address);
    virtual void NotifyRemoveAddress (uint32_t interface, Ipv4InterfaceAddress address);
    virtual void SetIpv4 (Ptr < Ipv4 > ipv4);
    virtual void PrintRoutingTable (Ptr < OutputStreamWrapper > stream) const;

    Time GetMaxQueueTime () const { return MaxQueueTime; }
    void SetMaxQueueTime (Time t);
    uint32_t GetMaxQueueLen () const { return MaxQueueLen; }
    void SetMaxQueueLen (uint32_t len);
    void SetBroadcastEnable (bool f) { EnableBroadcast = f; }
    bool GetBroadcastEnable () const { return EnableBroadcast; }

    int64_t AssignStreams (int64_t stream);


    void UpdateCurrentPhase (Phase phase);

    void SendDATAPACKET (bool isFirstPacket, uint16_t count,
                         uint32_t packetID);

    void PrintInformation (Ptr < OutputStreamWrapper > stream) const;


  protected:
      virtual void DoInitialize (void);


  private:

    uint32_t m_numberOfReceivedPacket;

    uint16_t EXPLRateLimit; ///< Maximum number of EXPL per second.
    double m_averageHopDistanceOfReceivedPackets;
    double m_averageEndToEndDelays;

    uint16_t m_numberOfReceivedAnts;
    Time ActiveRouteTimeout;

    uint32_t NetDiameter; ///< Net diameter measures the maximum possible number of hops between two nodes in the network
    /**
     * NodeTraversalTime is a conservative estimate of the average one hop traversal time for packets
     * and should include queuing delays, interrupt processing times and transfer times.
     */
    Time NodeTraversalTime;
    Time NetTraversalTime; ///< Estimate of the average net traversal time.
    Time PathDiscoveryTime; ///< Estimate of maximum time needed to find route in network.

    //Phase
    Phase m_currentPhase;

    Time HelloInterval; // just now added

    uint32_t AllowedHelloLoss;

    uint32_t m_explorationTime;
    /// Exprolation timer
    Timer m_explorationTimer;
    /// Repair timer
    uint32_t m_repairTime;
    Timer m_repairTimer;


    double m_transmissionRange;
    double m_distanceForTimer;
    bool EnableBroadcast;
    /// IP protocol
      Ptr < Ipv4 > m_ipv4;
    /// Loopback device used to defer
      Ptr < NetDevice > m_lo;
```

```
    /// Raw unicast socket per each IP interface, map socket -> iface address (IP + mask)
      std::map < Ptr < Socket >, Ipv4InterfaceAddress > m_socketAddresses;
    /// Raw subnet directed broadcast socket per each IP interface, map socket -> iface address (IP + mask)
      std::map < Ptr < Socket >,Ipv4InterfaceAddress > m_socketSubnetBroadcastAddresses;


    //Pheromone
    double m_pheromone;

    //Initial pheromone
    double m_initialPheromone;

    //Pheromone decay rate
    float m_pheromoneDecayRate;
    //Time of pheromone updated
    Time m_updatedTimeOfPheromone;
    double m_epsilon;

    //IP4 Address of sink
    //Ipv4Address m_sink;

    //Minimum time (ms) for waiting timer
    uint32_t m_timeMinForWaitingTimer;
    //Maximum time (ms) for waiting timer
    uint32_t m_timeMaxForWaitingTimer;
    //Weight parameters for selection probability
    double m_alpha;
    double m_beta;
    double m_gamma;

    /// Routing table
    RoutingTable m_routingTable;
    /// Used by the routing layer to buffer packets to which it does not have a route.
    uint32_t MaxQueueLen; ///< The maximum number of packets that we allow a routing protocol to buffer.

    Time MaxQueueTime; ///< The maximum period of time that a routing protocol is allowed to buffer a packet for.

    RequestQueue m_queue;

    // Hello Interval
    uint32_t m_helloInterval;

    /// Broadcast ID
    uint32_t m_requestId;
    /// Hello timer
    Timer m_helloTimer;
    /// Handle duplicated RREQ
    IdCache m_rreqIdCache;
    /// Handle duplicated broadcast/multicast packets
    DuplicatePacketDetection m_dpd;

    /// Number of EXPLs used for EXPL rate control
    uint16_t m_explCount;

    Neighbors m_neighborTable;

    /// Keep track of the last bcast time
    Time m_lastBcastTime;

    Timer m_forwardTimer;

    Timer m_backwardTimer;

    Timer m_retryTimer;


     uint32_t numberOfDeferredPacket;

    /// Waiting timer
    Timer m_waitingTimer;



    /// Provides uniform random variables.
      Ptr < UniformRandomVariable > m_uniformRandomVariable;


    private:
```

```

        /// Start protocol operation
        void Start ();

        bool UpdateRouteLifeTime (Ipv4Address addr, Time lt);

        /// Check that packet is send from own interface
        bool IsMyOwnAddress (Ipv4Address src);


        /// Receive and process control packet
        void Recvantcomm (Ptr < Socket > socket);
        /// Receive HELLO
        void RecvHELLO (Ptr<Packet> p, Ipv4Address my, Ipv4Address sender);
        /// Make the routing table after hello messages have been exchanged
        /// Ipv4Address origin is originator of foraging ant
        /// Ipv4Address my is Ipv4Address of myself
        /// Ipv4Address src is Ipv4Address of my neighbor
        void MakeRoutingTable (Ipv4Address origin, Ipv4Address my, Ipv4Address src);
        /// Receive EXPL
        void RecvEXPL (Ptr<Packet> p, Ipv4Address my, Ipv4Address src);

        /// Receive REPA
        void RecvREPA (Ptr<Packet> p, Ipv4Address my, Ipv4Address sender);
        /** Send RREP by intermediate node
        * \param toDst routing table entry to destination
        * \param toOrigin routing table entry to originator
        */
        void SendReplyByIntermediateNode (antcommRoutingTableEntry & toDst, antcommRoutingTableEntry &
             toOrigin);

        bool Forwarding (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb,
             ErrorCallback ecb);
        /// Queue packet and send route requeste
        void DeferredRouteOutput (Ptr < const Packet > p, const Ipv4Header & header,
             UnicastForwardCallback ucb, ErrorCallback ecb);

        /// Find unicast socket with local interface address iface
          Ptr < Socket > FindSocketWithInterfaceAddress (Ipv4InterfaceAddress iface) const;
        /// Find subnet directed broadcast socket with local interface address iface
          Ptr < Socket > FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress iface)
               const;

        /// For antcomm's packet sending
        void SendPacketFromQueue (Ipv4Address dst, Ptr < Ipv4Route > route);

        void SendHello ();

        void SendEXPL (Ipv4Address dst);

        void SendREPA (antcommEXPLHeader const & EXPLHeader, antcommRoutingTableEntry const & toOrigin);
        void SendREREPA ();
        void SendTo (Ptr < Socket > socket, Ptr < Packet > packet, Ipv4Address destination);
        /// Schedule next send of hello message
        void HelloTimerExpire ();

        /// Reset EXPL count and schedule EXPL rate limit timer with delay 1 sec.
        void ExplRateLimitTimerExpire ();

        double GetPheromone();

      };

    }
}
#endif     /* antcommPROTOCOL_H */
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#include "antcomm-rqueue.h"
#include <algorithm>
#include <functional>
#include "ns3/ipv4-route.h"
#include "ns3/socket.h"
#include "ns3/log.h"

namespace ns3
{

  NS_LOG_COMPONENT_DEFINE ("antcommRequestQueue");

  namespace antcomm
  {

    uint32_t RequestQueue::GetSize ()
    {
      Purge ();
      return m_queue.size ();
    }

    bool RequestQueue::Enqueue (QueueEntry & entry)
    {
      Purge ();
      for (std::vector < QueueEntry >::const_iterator i = m_queue.begin ();
           i != m_queue.end (); ++i)
        {
          if ((i->GetPacket ()->GetUid () == entry.GetPacket ()->GetUid ())
              && (i->GetIpv4Header ().GetDestination () ==
                  entry.GetIpv4Header ().GetDestination ()))
            return false;
        }
      entry.SetExpireTime (m_queueTimeout);
      if (m_queue.size () == m_maxLen)
        {
          Drop (m_queue.front (), "Drop the most aged packet"); // Drop the most aged packet
          m_queue.erase (m_queue.begin ());
        }
      m_queue.push_back (entry);
      return true;
    }

    void RequestQueue::DropPacketWithDst (Ipv4Address dst)
    {
      NS_LOG_FUNCTION (this << dst);
      Purge ();
      for (std::vector < QueueEntry >::iterator i = m_queue.begin ();
           i != m_queue.end (); ++i)
        {
          if (IsEqual (*i, dst))
            {
              Drop (*i, "DropPacketWithDst ");
            }
        }
      m_queue.erase (std::remove_if (m_queue.begin (), m_queue.end (),
                                     std::bind2nd (std::
                                                   ptr_fun (RequestQueue::
                                                            IsEqual), dst)),
                     m_queue.end ());
    }

    bool RequestQueue::Dequeue (Ipv4Address dst, QueueEntry & entry)
    {
      Purge ();
      for (std::vector < QueueEntry >::iterator i = m_queue.begin ();
           i != m_queue.end (); ++i)
        {
          if (i->GetIpv4Header ().GetDestination () == dst)
            {
              entry = *i;
              m_queue.erase (i);
              return true;
            }
        }
      return false;
```

```
  }

  bool RequestQueue::Find (Ipv4Address dst)
  {
    for (std::vector < QueueEntry >::const_iterator i = m_queue.begin ();
         i != m_queue.end (); ++i)
      {
        if (i->GetIpv4Header ().GetDestination () == dst)
          return true;
      }
    return false;
  }

  struct IsExpired
  {
    bool operator () (QueueEntry const &e) const
    {
      return (e.GetExpireTime () < Seconds (0));
    }
  };

  void RequestQueue::Purge ()
  {
    IsExpired pred;
    for (std::vector < QueueEntry >::iterator i = m_queue.begin ();
         i != m_queue.end (); ++i)
      {
        if (pred (*i))
          {
            Drop (*i, "Drop␣outdated␣packet␣");
          }
      }
    m_queue.erase (std::remove_if (m_queue.begin (), m_queue.end (), pred),
                   m_queue.end ());
  }

  void RequestQueue::Drop (QueueEntry en, std::string reason)
  {
    NS_LOG_LOGIC (reason << en.GetPacket ()->GetUid () << "␣" << en.
                  GetIpv4Header ().GetDestination ());
    en.GetErrorCallback ()(en.GetPacket (), en.GetIpv4Header (),
                           Socket::ERROR_NOROUTETOHOST);
    return;
  }

 }
}
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#ifndef antcomm_RQUEUE_H
#define antcomm_RQUEUE_H

#include <vector>
#include "ns3/ipv4-routing-protocol.h"
#include "ns3/simulator.h"


namespace ns3 {
namespace antcomm {

/**
 * \ingroup antcomm
 * \brief antcomm Queue Entry
 */
class QueueEntry {
    public:
        typedef Ipv4RoutingProtocol::UnicastForwardCallback UnicastForwardCallback;
        typedef Ipv4RoutingProtocol::ErrorCallback ErrorCallback;
        /// c-tor
        QueueEntry (Ptr<const Packet> pa = 0, Ipv4Header const & h = Ipv4Header (),
                  UnicastForwardCallback ucb = UnicastForwardCallback (),
                  ErrorCallback ecb = ErrorCallback (), Time exp = Simulator::Now ()) :
        m_packet (pa), m_header (h), m_ucb (ucb), m_ecb (ecb),
        m_expire (exp + Simulator::Now ()) {
        }

        /**
        * Compare queue entries
        * \return true if equal
        */
        bool operator== (QueueEntry const & o) const {
            return ((m_packet == o.m_packet) && (m_header.GetDestination () == o.m_header.
                 GetDestination ()) && (m_expire == o.m_expire));
        }

        // Fields
        UnicastForwardCallback GetUnicastForwardCallback () const { return m_ucb; }
        void SetUnicastForwardCallback (UnicastForwardCallback ucb) { m_ucb = ucb; }
        ErrorCallback GetErrorCallback () const { return m_ecb; }
        void SetErrorCallback (ErrorCallback ecb) { m_ecb = ecb; }
        Ptr<const Packet> GetPacket () const { return m_packet; }
        void SetPacket (Ptr<const Packet> p) { m_packet = p; }
        Ipv4Header GetIpv4Header () const { return m_header; }
        void SetIpv4Header (Ipv4Header h) { m_header = h; }
        void SetExpireTime (Time exp) { m_expire = exp + Simulator::Now (); }
        Time GetExpireTime () const { return m_expire - Simulator::Now (); }

    private:

        /// Data packet
        Ptr<const Packet> m_packet;
        /// IP header
        Ipv4Header m_header;
        /// Unicast forward callback
        UnicastForwardCallback m_ucb;
        /// Error callback
        ErrorCallback m_ecb;
        /// Expire time for queue entry
        Time m_expire;
    };
    /**
     * \ingroup aodv
     * \brief antcomm route request queue
     *
     * Since antcomm is an on demand routing we queue requests while looking for route.
     */
    class RequestQueue {
        public:
            /// Default c-tor
            RequestQueue (uint32_t maxLen, Time routeToQueueTimeout) :
            m_maxLen (maxLen), m_queueTimeout (routeToQueueTimeout) {
            }
            /// Push entry in queue, if there is no entry with the same packet and destination address
                 in queue.
```

```
            bool Enqueue (QueueEntry & entry);
            /// Return first found (the earliest) entry for given destination
            bool Dequeue (Ipv4Address dst, QueueEntry & entry);
            /// Remove all packets with destination IP address dst
            void DropPacketWithDst (Ipv4Address dst);
            /// Finds whether a packet with destination dst exists in the queue
            bool Find (Ipv4Address dst);
            /// Number of entries
            uint32_t GetSize ();

            // Fields
            uint32_t GetMaxQueueLen () const { return m_maxLen; }
            void SetMaxQueueLen (uint32_t len) { m_maxLen = len; }
            Time GetQueueTimeout () const { return m_queueTimeout; }
            void SetQueueTimeout (Time t) { m_queueTimeout = t; }

        private:

            std::vector<QueueEntry> m_queue;
            /// Remove all expired entries
            void Purge ();
            /// Notify that packet is dropped from queue by timeout
            void Drop (QueueEntry en, std::string reason);
            /// The maximum number of packets that we allow a routing protocol to buffer.
            uint32_t m_maxLen;
            /// The maximum period of time that a routing protocol is allowed to buffer a packet for,
                 seconds.
            Time m_queueTimeout;
            static bool IsEqual (QueueEntry en, const Ipv4Address dst) { return (en.GetIpv4Header ().
                 GetDestination () == dst); }
        };


    }
}

#endif /* antcomm_RQUEUE_H */
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */

#include "antcomm-rtable.h"
#include "antcomm-helper.h"
#include <algorithm>
#include <math.h>
#include <iomanip>
#include "ns3/simulator.h"
#include "ns3/random-variable-stream.h"
#include "ns3/log.h"
#define MAX 100

namespace ns3
{

  NS_LOG_COMPONENT_DEFINE ("antcommRoutingTable");

  namespace antcomm
  {

    /*
       The antcomm Information Table
      */
    antcommRoutingTableEntry::antcommRoutingTableEntry (Ptr<NetDevice> dev, Ipv4Address dst, uint16_t hopCount, double pheromone, double residualEnergy, double rssi, Ipv4Address nextHop, Ipv4InterfaceAddress iface, Time updatedTime):
      m_hopCount (hopCount), m_pheromone (pheromone), m_residualEnergy (residualEnergy), m_rssi (rssi), m_iface (iface), m_updatedTime (updatedTime + Simulator::Now ())
    {
      m_ipv4Route = Create<Ipv4Route> ();
      m_ipv4Route->SetDestination (dst);
      m_ipv4Route->SetGateway (nextHop);
      m_ipv4Route->SetSource (m_iface.GetLocal ());
      m_ipv4Route->SetOutputDevice (dev);
    }

    antcommRoutingTableEntry::~antcommRoutingTableEntry ()
    {
    }

    bool antcommRoutingTableEntry::InsertPrecursor (Ipv4Address id)
   {
    NS_LOG_FUNCTION (this << id);
    if (LookupPrecursor (id))
    {
      return false;
    }
    else
     m_precursorList.push_back (id);
     return true;
   }

   bool antcommRoutingTableEntry::LookupPrecursor (Ipv4Address id)
   {
    NS_LOG_FUNCTION (this << id);
    for (std::vector<Ipv4Address>::const_iterator i = m_precursorList.begin (); i
       != m_precursorList.end (); ++i)
    {
      if (*i == id)
        {
          NS_LOG_LOGIC ("Precursor␣" << id << "␣found");
          return true;
        }
    }
     NS_LOG_LOGIC ("Precursor␣" << id << "␣not␣found");
     return false;
   }
    void antcommRoutingTableEntry::Print (Ptr < OutputStreamWrapper > stream) const
    {
      *stream->GetStream () << m_ipv4Route->GetGateway ()/*m_dest*/ << "\t"
      << m_hopCount << "\t" << m_pheromone << "\t" << m_rssi << "\t"
      << m_residualEnergy << "\t" << std::setw (14) << m_updatedTime.GetSeconds () << "\n";
    }


    /*
```

```
   The Routing Table
  */
bool RoutingTable::LookupRoute (Ipv4Address dst, Ipv4Address neighbor, antcommRoutingTableEntry & nt)
{

  NS_LOG_FUNCTION (this << dst << neighbor);
  RTable::const_iterator it;
  Neighbor::const_iterator jt;
  if(m_routes.empty())
  {
    NS_LOG_LOGIC("Backward␣learning␣Route␣to" << dst << "␣not␣found;␣m_routes␣is␣empty");
    return false;
  }
  it = m_routes.find (dst);
  if (it != m_routes.end ()) {
    pout << "Backward␣learning␣Destination␣Exists:␣" << it->first << std::endl;
    jt = it->second.find (neighbor);
    if (jt != it->second.end ())
    {
      nt = m_routes[it->first][jt->first];
      NS_LOG_LOGIC ("Backward␣learning␣Route␣to␣" << dst << "␣found");
      pout << "Backward␣Neighbor␣Exists:␣" << jt->first << std::endl;
      return true;
    }
  }
  NS_LOG_LOGIC ("Backward␣learning␣Route␣to␣" << dst << "␣not␣found");
  return false;
}

bool RoutingTable::LookupRoute2 (Ipv4Address dst, antcommRoutingTableEntry & nt)
{
    /* Rules for finding the good neighbor:
    a) find the destination node
    b) If the iterator is pointing towards end then it means destination not found
    c) Otherwise scan neighbor
    d) Use probabilistic rule to select the good neighbor
    e) Return its entry and set the parameter, "nt" to the entry you found above */
    NS_LOG_FUNCTION(this << dst);
    RTable::const_iterator it;
    Neighbor::const_iterator jt;
  antcommRoutingTableEntry entry[MAX];
  double probability[MAX];
  int i;
  double cum = 0;
  int counter = 0;
  double sum = 0;
  double pvt;
    if(m_routes.empty())
    {
            NS_LOG_LOGIC("Route␣to" << dst << "not␣found;␣m_routes␣is␣empty");
            return false;
    }
    it = m_routes.find(dst);
    if(it == m_routes.end())
    {
            NS_LOG_LOGIC("Route␣to␣" << dst << "␣not␣found");
            return false;
    }
    if(it->second.begin() == it->second.end())
    {
            NS_LOG_LOGIC("There␣are␣no␣neighbors␣present,␣hence␣can't␣forward␣the␣packet");
            return false;
    }
  for (jt = it->second.begin(); jt != it->second.end(); ++jt)
  {
    probability[counter] = pow((jt->second).GetPheromone(), 1)*pow((jt->second).GetHopCount(), 3.2)*
    pow((jt->second).GetResidualEnergy(), 1.8)*pow((jt->second).GetRssi(), 2.3);//2.8 2.3 is current
    entry[counter] = jt->second;
    sum += probability[counter];
    counter++;
  }
  NS_LOG_DEBUG ( "Neighbor␣counter␣is" << counter);
  NS_LOG_INFO ( "␣Number␣of␣Neighbors␣are␣" << counter );
  Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable>();
  pvt = x->GetValue(0.0, sum);
// for (cum = probability[i=0]; i < counter && pvt >= cum; cum += probability[++i]) // changed ++ here
// ;
```

```
    for (i=0; i < counter; ++i)
    {
      cum += probability[i];
      if (pvt <= cum)
        break;
    }

    NS_ASSERT(i != counter);

    //std::cout << i<<"th Node ID is selected as neighbor"<< std::endl;

    nt = entry[i];
    NS_LOG_LOGIC ("Route␣to␣" << dst << "␣found");
    /*jt = it->second.begin ();

    while (i-- > 0)
      jt++;

    nt = jt->second;*/
    //nt = m_routes[it->first][jt->first];
    return true;
  }

  bool RoutingTable::DeleteRoute (Ipv4Address dst)
  {
    NS_LOG_FUNCTION (this << dst );
    if (m_routes.erase (dst) != 0)
          {
            NS_LOG_LOGIC ("Route␣deletion␣of␣" << dst << "␣successful");
            return true;
          }
    NS_LOG_LOGIC ("Route␣deletion␣of␣" << dst << "␣not␣successful");
    return false;
  }


  bool RoutingTable::AddRoute (antcommRoutingTableEntry & it)
  {
    NS_LOG_FUNCTION (this);
    //m_routes [it.GetDestination ()].insert (std::make_pair (it.GetNextHop (), it));
    //pout << "Adding Route ( " <<it.GetDestination() << " , " << it.GetNextHop() <<" )\n";
    m_routes[it.GetDestination()][it.GetNextHop()] = it;
    return true;
  }


  bool RoutingTable::Update (antcommRoutingTableEntry & nt)
  {
    NS_LOG_FUNCTION (this);
    RTable::iterator it;
    Neighbor::iterator jt;
    it = m_routes.find (nt.GetDestination());
    if (it == m_routes.end ())
    {
      return false;
    }
    jt = it->second.find(nt.GetNextHop());
    m_routes[it->first][jt->first] = nt; //jt->second = nt;
    return true;

  }

  Ipv4Address RoutingTable::NextHopforDst (antcommRoutingTableEntry & nt)
  {
    RTable::const_iterator it;
    Neighbor::const_iterator jt;
    double probability[MAX];
    Ipv4Address none = "0.0.0.0";
    int counter = 0;
    int i;
    double sum = 0;
    double cum = 0;
    double pvt;
    if(m_routes.empty())
    {
      NS_LOG_LOGIC("Route␣to" << nt.GetDestination () << "not␣found;␣m_routes␣is␣empty");
      return none;
    }
    it = m_routes.find(nt.GetDestination ());
    if(it == m_routes.end())
    {
```

```
        NS_LOG_LOGIC("Route␣to␣" << nt.GetDestination () << "not␣found");
        return none;
      }

      for (jt = it->second.begin(); jt != it->second.end(); ++jt)
      {
        probability[counter] = pow((jt->second).GetPheromone(), 1) * pow((jt->second).GetHopCount(),
             2);
        sum += probability[counter];
        counter++;
      }
      Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable> ();
      pvt = x->GetValue(0.0, sum);
      for (cum = probability[i=0]; i < counter && pvt>=cum; cum+=probability[++i])
        ;
      jt = it->second.begin();
      while (i-- > 0)
      {
        jt++;
      }

      return jt->first;
    }

    void RoutingTable::Print (Ptr < OutputStreamWrapper > stream) const
    {

         RTable::const_iterator ot;
         Neighbor::const_iterator it;
           *stream->GetStream () << "\nantcomm␣information␣table\n"
         << "ID\tHopCount\tPheromone\tRSSI\tResidualEnergy\tUpdated␣Time\n";

         /*std::cout << "map size= "<< m_routes.size() << std::endl;
         ot = m_routes.begin();
         std::cout << "map2 first size= " <<(ot->second).size() << std::endl;
         ot = m_routes.end(); ot--;
         std::cout << "map2 last size= " <<(ot->second).size() << std::endl;*/
         for (ot = m_routes.begin (); ot != m_routes.end (); ++ot)
         {
          //std::cout << "Printing the destinations " << ot->first << std::endl;
           for (it = ot->second.begin (); it != ot->second.end (); ++it)


         {
         it->second.Print (stream);
         }
         }
         *stream->GetStream () << "\n";

    }

  }
}
```

```
/*
 * Copyright (c) 2015 SKKU Networking Laboratory, IUPUI
 *
 * Authors: Soon-gyo Jung <soongyo@skku.edu>
 */
#ifndef antcomm_RTABLE_H
#define antcomm_RTABLE_H

#define ARUSH_DEBUG_COUT //comment this if you want to suppress the cout-s

#ifdef ARUSH_DEBUG_COUT
#define pout std::cout
#else
#define pout if(0) std::cout
#endif

#include <stdint.h>
#include <cassert>
#include <sys/types.h>
#include <vector>
#include "ns3/ipv4.h"
#include "ns3/ipv4-route.h"
#include "ns3/timer.h"
#include "ns3/net-device.h"
#include "ns3/wifi-mac-header.h"
#include "ns3/output-stream-wrapper.h"

namespace ns3
{

  namespace antcomm
  {

              /**
               * \ingroup antcomm
               * \brief The table entry used by antcomm protocol
               */

    class antcommRoutingTableEntry
    {
    public:

      antcommRoutingTableEntry (Ptr<NetDevice> dev = 0, Ipv4Address dst = Ipv4Address (),
                             uint16_t hopCount = 1, double pheromone = 0.0, double residualEnergy =
                                  0.0, double rssi = 0.0, Ipv4Address nextHop = Ipv4Address (),
                                  Ipv4InterfaceAddress iface = Ipv4InterfaceAddress (), Time
                                  updatedTime = Simulator::Now ());

       ~antcommRoutingTableEntry ();
    /**
    * Insert precursor in precursor list if it doesn't yet exist in the list
    * \param id precursor address
    * \return true on success
    */
  bool InsertPrecursor (Ipv4Address id);
  /**
   * Lookup precursor by address
   * \param id precursor address
   * \return true on success
   */
  bool LookupPrecursor (Ipv4Address id);



      Ipv4Address GetDestination () const
      {
        return m_ipv4Route->GetDestination ();
      }
      Ptr<Ipv4Route> GetRoute () const
      {
        return m_ipv4Route;
      }
      void SetRoute (Ptr<Ipv4Route> r)
      {
        m_ipv4Route = r;
      }
      void SetHopCount (uint16_t hopCount)
      {
        m_hopCount = hopCount;
      }
```

```
  uint16_t GetHopCount () const
  {
    return m_hopCount;
  }
  void SetPheromone (double pheromone)
  {
    m_pheromone = pheromone;
  }
  double GetPheromone () const
  {
    return m_pheromone;
  }
  void SetResidualEnergy (double residualEnergy)
  {
    m_residualEnergy = residualEnergy;
  }
  double GetResidualEnergy () const
  {
    return m_residualEnergy;
  }
  void SetRssi (double rssi)
  {
    m_rssi = rssi;
  }
  double GetRssi () const
  {
    return m_rssi;
  }
  void SetNextHop (Ipv4Address nextHop)
  {
    //m_nexthop = nextHop;
    m_ipv4Route->SetGateway (nextHop);
  }
  Ipv4Address GetNextHop () const
  {
    //return m_nexthop;
    return m_ipv4Route->GetGateway ();
  }
  void SetOutputDevice (Ptr<NetDevice> dev)
  {
    m_ipv4Route->SetOutputDevice (dev);
  }
  Ptr<NetDevice> GetOutputDevice () const
  {
    return m_ipv4Route->GetOutputDevice ();
  }
  Ipv4InterfaceAddress GetInterface () const
  {
    return m_iface;
  }
  void SetInterface (Ipv4InterfaceAddress iface)
  {
    m_iface = iface;
  }
  void SetUpdatedTime (Time lt)
  {
    m_updatedTime = lt + Simulator::Now ();
  }
  Time GetUpdatedTime () const
  {
    return m_updatedTime - Simulator::Now ();
  }


                           /**
                            * \brief Compare source address
                            * \return true if equal
                            */
  bool operator == (Ipv4Address const dst ) const
  {
    //return (m_dest == dst );
    return (m_ipv4Route->GetDestination () == dst);
  }

  void Print (Ptr < OutputStreamWrapper > stream) const;

private:
```

```

    //Ipv4Address m_dest;

    Ptr<Ipv4Route> m_ipv4Route;

    uint16_t m_hopCount;

    double m_pheromone;

    std::vector<Ipv4Address> m_precursorList;

    double m_residualEnergy;

    double m_rssi;

    //Ipv4Address m_nexthop;

    Ipv4InterfaceAddress m_iface;

    Time m_updatedTime;

  };


  typedef std::map < Ipv4Address, antcommRoutingTableEntry > Neighbor;
  typedef std::map < Ipv4Address, Neighbor > RTable; // map the above entries with IPv4 address of
      destination node(s)
          /**
           * \ingroup antcomm
           * \brief The Information table used by antcomm protocol
           */
  class RoutingTable
  {
  public:

                        /**
                         * Add routing table entry if it doesn't yet exist in information table
                         * \param r information table entry
                         * \return true in success
                         */
    //bool AddRoute (antcommRoutingTableEntry & it);
    bool AddRoute (antcommRoutingTableEntry & it);

                        /**
                         * Delete Routing table entry with source address source, if it exists.
                         * \param dst source address
                         * \return true on success
                         */
    bool DeleteRoute (Ipv4Address dst );

                        /**
                         * Lookup Routing table entry with source address source
                         * \param dst source address
                         * \param rt entry with source address source, if exists
                         * \return true on success
                         */
    bool LookupRoute (Ipv4Address dst, Ipv4Address neighbor, antcommRoutingTableEntry & nt);

    bool LookupRoute2 (Ipv4Address dst, antcommRoutingTableEntry & nt);

    /// Update Routing table
    bool Update (antcommRoutingTableEntry & nt);

    Ipv4Address NextHopforDst (antcommRoutingTableEntry & nt);

    /// Print routing table
    void Print (Ptr < OutputStreamWrapper > stream) const;

  private:

    /* m_routes is a map of map
       std::map < Ipv4Address, std::map < Ipv4Address, antcommRoutingTableEntry > > m_routes;
     */
    RTable m_routes;
  };

 }
}


#endif /* antcomm_RTABLE_H */
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

// Include a header file from your module to test.
#include "ns3/antree-module.h"

// An essential include is test.h
#include "ns3/test.h"

// Do not put your test classes in namespace ns3. You may find it useful
// to use the using directive to access the ns3 namespace directly
using namespace ns3;

// This is an example TestCase.
class AntreeTestCase1 : public TestCase
{
public:
  AntreeTestCase1 ();
  virtual ~AntreeTestCase1 ();

private:
  virtual void DoRun (void);
};

// Add some help text to this case to describe what it is intended to test
AntreeTestCase1::AntreeTestCase1 ()
  : TestCase ("Antree␣test␣case␣(does␣nothing)")
{
}

// This destructor does nothing but we include it as a reminder that
// the test case should clean up after itself
AntreeTestCase1::~AntreeTestCase1 ()
{
}

//
// This method is the pure virtual method from class TestCase that every
// TestCase must implement
//
void
AntreeTestCase1::DoRun (void)
{
  // A wide variety of test macros are available in src/core/test.h
  NS_TEST_ASSERT_MSG_EQ (true, true, "true␣doesn't␣equal␣true␣for␣some␣reason");
  // Use this one for floating point comparisons
  NS_TEST_ASSERT_MSG_EQ_TOL (0.01, 0.01, 0.001, "Numbers␣are␣not␣equal␣within␣tolerance");
}

// The TestSuite class names the TestSuite, identifies what type of TestSuite,
// and enables the TestCases to be run. Typically, only the constructor for
// this class must be defined
//
class AntreeTestSuite : public TestSuite
{
public:
  AntreeTestSuite ();
};

AntreeTestSuite::AntreeTestSuite ()
  : TestSuite ("antree", UNIT)
{
  // TestDuration for TestCase can be QUICK, EXTENSIVE or TAKES_FOREVER
  AddTestCase (new AntreeTestCase1, TestCase::QUICK);
}

// Do not forget to allocate an instance of this TestSuite
static AntreeTestSuite antreeTestSuite;
```

Table A.1.: Placement of Nodes

Position X	Position Y
0	-2
55	-2
109	-1
150	-7
214	5
259	-6
302	-5
374	1
407	9
428	-2
18	30
67	48
123	44
184	47
231	43
275	58
318	45
384	43
422	54
465	41
-1	84
41	108
112	94
150	90
206	97

continued on next page

Table A.1.: *continued*

Position X	Position Y
239	111
283	100
352	103
379	93
446	91
29	154
87	164
125	154
174	154
219	156
265	162
330	155
379	160
421	147
472	156
1	185
64	207
101	192
148	200
187	190
259	208
318	217
355	186
381	180
453	203

continued on next page

Table A.1.: *continued*

Position X	Position Y
16	248
66	250
126	257
173	263
222	242
284	255
318	242
377	256
435	262
475	262
-1	283
57	289
80	297
142	307
206	295
231	296
290	302
358	300
416	296
448	292
23	337
69	362
117	350
177	350
220	335

continued on next page

Table A.1.: *continued*

Position X	Position Y
282	349
338	339
364	343
420	349
479	345
-3	399
41	414
94	382
157	399
211	400
250	416
302	413
348	386
384	412
464	400
21	441
74	445
135	437
183	463
236	445
282	463
333	437
386	450
427	438
465	438

Table A.2.
Data of Throughput against background traffic in Ant Colony

Background Traffic	**Throughput**
0	97478
57.8	97462
126.8	93082.8
211.7	86924.2
310.5	84732.9
465.7	77369
623	73137.4
775.2	70158.4
948.2	66692.6
1118.2	64796.4
1317.4	62058.7

Table A.3.
Data of Mean Delay against background traffic in Ant Colony

Background Traffic	**Mean Delay**
0	0.0406364
57.8	0.0455162
126.8	0.0544133
211.7	0.0548401
310.5	0.0592297
465.7	0.0607113
623	0.0618095
775.2	0.0649593
948.2	0.0659632
1118.2	0.0684697
1317.4	0.0686697

Table A.4.
Data of Packet Loss against background traffic in Ant Colony

Background Traffic	**Packet Loss**
0	2
57.8	31
126.8	106
211.7	189
310.5	370
465.7	564
623	747
775.2	969
948.2	1170
1118.2	1340
1317.4	1432

Table A.5.
Throughput vs Average Degree in Ant Colony

Throughput vs Average Degree			
Average Degree	Load (90 kbps)	Load (400 kbps)	Load (725 kbps)
4.94	97760.9	92220.2	92692.4
5.34	97733.9	81543	77623.1
5.94	97623.6	95599.7	95300.2
6.94	97386.9	93169.4	92840
8.03	91075.3	78756.8	74819.3
9.42	91085.5	63038.6	54744.2
10.54	95355.5	84516.5	66927.4
11.63	96353.4	87651.3	86194.6
13.43	60617.5	44129.5	34122.2

Table A.6.
Mean Delay vs Average Degree in Ant Colony

Mean Delay vs Average Degree			
Average Degree	Load (90 kbps)	Load (400 kbps)	Load (725 kbps)
4.94	0.034309	0.0432472	0.0437581
5.34	0.0463942	0.058193	0.063629
5.94	0.0360623	0.0403674	0.033113
6.94	0.0359187	0.042346	0.0431649
8.03	0.0447374	0.0619078	0.064226
9.42	0.0386595	0.0713462	0.0808875
10.54	0.0372176	0.0480497	0.0540632
11.63	0.0398859	0.0487399	0.0523276
13.43	0.0525717	0.0789278	0.0839992

Table A.7.
Packet Loss vs Average Degree in Ant Colony

Packet loss vs Average Degree			
Average Degree	Load (90 kbps)	Load (400 kbps)	Load (725 kbps)
4.94	1	133	204
5.34	1	377	803
5.94	1	52	100
6.94	3	106	201
8.03	42	442	942
9.42	46	801	1696
10.54	16	304	1234
11.63	10	237	452
13.43	265	1251	2662

Table A.8.
Throughput comparison between AODV and Ant Colony

Comparison of Throughput performance		
Background Load (kbps)	Throughput in AODV (kbps)	Throughput in Ant Colony (kbps)
0	97064.8	97478
57.8	97439	97462
126.8	97332.2	93082.8
211.7	96481	86924.2
310.5	94474.925	84732.9
465.7	93643.6	77369
623	89244.675	73137.4
775.2	87875.75	70158.4
948.2	85372.475	66692.6
1118.8	82064.225	64796.4
1317.4	77739.2	62058.7

Table A.9.
Delay Comparison between AODV and Ant Colony

Comparison of Delay performance		
Background Load (kbps)	Delay in AODV (sec)	Delay in Ant Colony (sec)
0	0.03429795	0.0406364
57.8	0.03234435	0.0455162
126.8	0.0328218	0.0544133
211.7	0.0330905	0.0548401
310.5	0.032899725	0.0592297
465.7	0.034572075	0.0607113
623	0.034309825	0.0618095
775.2	0.03908005	0.0649593
948.2	0.04073455	0.0659632
1118.8	0.05084815	0.0684697
1317.4	0.0565708	0.0686697

Table A.10.
Packet Loss Comparison between AODV and Ant Colony

Comparison of Packet Loss performance		
Background Load (kbps)	Packet Loss in AODV	Packet Loss in Ant Colony
0	3.5	2
57.8	3.5	31
126.8	4.75	106
211.7	30	189
310.5	81	370
465.7	196	564
623	275	747
775.2	392	969
948.2	558	1170
1118.8	618	1340
1317.4	772.25	1432

Table A.11.
Throughput vs Average Degree between AODV and Ant Colony

Comparison of Throughput against Average Degree		
Average Degree	Throughput in AODV	Throughput in Ant Colony
4.94	97170.9	92692.4
5.34	96532.3	77623.1
5.94	93259.6	95300.2
6.94	82998.6	92840
8.03	67433	74819.3
9.42	71188	54744.2
10.54	31164	66927.4
11.63	26226	86194.6

Table A.12.
Packet Loss vs Average Degree between AODV and Ant Colony

Comparison of Packet Loss against Average Degree		
Average Degree	Packet Loss in AODV	Packet Loss in Ant Colony
4.94	26	204
5.34	52	803
5.94	185	100
6.94	602	201
8.03	1215	942
9.42	1081	1696
10.54	3000	1234
11.63	3274	452

Table A.13.
Delay vs Average Degree between AODV and Ant Colony

Comparison of Delay against Average Degree		
Average Degree	Delay in AODV	Delay in Ant Colony
4.94	0.0325634	0.0437581
5.34	0.0302164	0.063629
5.94	0.027723	0.033113
6.94	0.053475	0.0431649
8.03	0.0347015	0.064226
9.42	0.0486191	0.0808875
10.54	0.0586493	0.0540632
11.63	0.0276077	0.0523276

Table A.14.
Std. Dev. of Residual Energy and Avg Residual Energy in AODV

Time (sec)	Std. Dev of Remaining Energy	Average Residual Energy
50	0.004362	0.360862
75	0.006586	0.342027
100	0.008627	0.32312
125	0.01075	0.304148
150	0.012362	0.285476
175	0.014043	0.266907
200	0.015738	0.24848
225	0.01761	0.22992
250	0.019177	0.211082
275	0.02043	0.192541
300	0.021627	0.173883
325	0.022866	0.15545
350	0.024275	0.136809
375	0.025676	0.118054
400	0.027143	0.099187
425	0.028562	0.080547
450	0.029961	0.061782

Table A.15.
Std. Dev. of Residual Energy and Avg Residual Energy in Ant Colony

Time (sec)	Std. Dev of Remaining Energy	Avg Residual Energy
50	0.000767	0.368974
75	0.000797	0.354565
100	0.000862	0.340102
125	0.00093	0.325639
150	0.000979	0.311195
175	0.001049	0.296738
200	0.0011	0.282297
225	0.001152	0.267879
250	0.001212	0.253415
275	0.001253	0.238983
300	0.001299	0.224578
325	0.001363	0.210151
350	0.00138	0.195768
375	0.001467	0.181294
400	0.001534	0.166846
425	0.001607	0.152396
450	0.001641	0.137967

Table A.16.
Hop Comparison between AODV and Ant Colony when Avg Degree is 4.94

Time (sec)	Hop in AODV	Hop in Ant Colony
25	14	14.86667
50	14	15.73333
75	14	14.53333
100	14	15.36667
125	14	15.5
150	14	15.36667
175	14	15.43333
200	14	14.6
225	14	15.26667
250	14	15.63333
275	14	14.9
300	14	16.03333
325	14	15.06667
350	14	15.7
375	14	14.7
400	14	15.63333
425	14	15.8
450	14	15.6
475	14	15.83333

Table A.17.
Hop Comparison between AODV and Ant Colony when Avg Degree is 5.94

Time (sec)	Hop in AODV	Hop in Ant Colony
25	11	17
50	11	15.66667
75	11	15.16667
100	11	16.03333
125	11	16.63333
150	11	15.86667
175	11	15.7
200	11	16.06667
225	11	15.6
250	11	15.33333
275	11	16.3
300	11	16.9
325	11	15.7
350	11	14.96667
375	11	16.73333
400	11	16.1
425	11	15.36667
450	11	15.2
475	11	15.1

Table A.18.: Remaining Energy of Nodes over different time stamps using Ant Colony

Node ID	T=100s	T=200s	T=300s	T=400s
0	0.341465	0.284081	0.226477	0.169066
1	0.340104	0.28191	0.22384	0.165946
2	0.340241	0.282562	0.224755	0.167184
3	0.340873	0.283225	0.225619	0.168111
4	0.339973	0.282359	0.22487	0.167369
5	0.340057	0.282507	0.224917	0.167159
6	0.341423	0.28399	0.226531	0.169262
7	0.341114	0.28355	0.225958	0.168582
8	0.340822	0.283156	0.225435	0.167847
9	0.339961	0.282386	0.224533	0.166622
10	0.340286	0.282317	0.224391	0.166539
11	0.340328	0.282636	0.22484	0.16727
12	0.338346	0.280131	0.221904	0.163669
13	0.338205	0.280205	0.222377	0.164378
14	0.338564	0.280817	0.222971	0.164953
15	0.340903	0.283007	0.225474	0.16815
16	0.341319	0.284071	0.226718	0.169274
17	0.339768	0.282104	0.224454	0.166714
18	0.33886	0.28076	0.222448	0.164438
19	0.34052	0.282796	0.225126	0.167599
20	0.341866	0.284254	0.226557	0.169001
21	0.341668	0.284423	0.227091	0.169741
22	0.340073	0.282227	0.224503	0.166746
23	0.339011	0.281157	0.22339	0.165659

continued on next page

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
24	0.339047	0.281239	0.22361	0.1658
25	0.339607	0.281412	0.22366	0.165997
26	0.341481	0.284313	0.227138	0.169855
27	0.34101	0.28346	0.225989	0.168676
28	0.340883	0.283527	0.225762	0.16809
29	0.340634	0.282945	0.225584	0.167831
30	0.340995	0.283111	0.22525	0.167542
31	0.339797	0.282075	0.224153	0.16641
32	0.33915	0.281355	0.223407	0.165405
33	0.339761	0.281694	0.223893	0.165818
34	0.33874	0.28081	0.222853	0.164829
35	0.340663	0.283041	0.225442	0.167796
36	0.340289	0.282482	0.224532	0.166912
37	0.339865	0.281701	0.223886	0.166616
38	0.339756	0.281715	0.223553	0.165703
39	0.341443	0.283802	0.226187	0.169
40	0.3421	0.284995	0.2278	0.170696
41	0.340259	0.282549	0.225119	0.167581
42	0.33995	0.282233	0.224613	0.166835
43	0.338519	0.280264	0.221804	0.163601
44	0.33931	0.281365	0.223488	0.165697
45	0.339943	0.282059	0.224144	0.16655
46	0.34042	0.282595	0.224841	0.16722
47	0.339655	0.281565	0.223491	0.165925
48	0.340484	0.282323	0.224342	0.16686

continued on next page

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
49	0.340253	0.282865	0.225655	0.167997
50	0.340409	0.282279	0.224639	0.167264
51	0.339282	0.281417	0.222721	0.165089
52	0.339174	0.281162	0.223744	0.165791
53	0.339956	0.281995	0.224009	0.166335
54	0.339224	0.281203	0.223191	0.164998
55	0.339418	0.281344	0.223595	0.165704
56	0.340576	0.282519	0.224819	0.167443
57	0.339683	0.281892	0.224323	0.166649
58	0.33883	0.280542	0.222594	0.164952
59	0.340431	0.282978	0.225458	0.168083
60	0.340429	0.282974	0.224982	0.167451
61	0.338996	0.280482	0.222542	0.164647
62	0.339826	0.28166	0.223542	0.165861
63	0.339625	0.281781	0.223928	0.166012
64	0.339233	0.281369	0.22372	0.16597
65	0.339594	0.281601	0.224028	0.166378
66	0.340425	0.282518	0.22445	0.166802
67	0.33952	0.281214	0.223188	0.165386
68	0.339325	0.281044	0.223402	0.165686
69	0.339632	0.282045	0.224571	0.166828
70	0.340365	0.282424	0.224782	0.167254
71	0.339509	0.281754	0.224265	0.166247
72	0.339972	0.282104	0.224624	0.166779
73	0.338836	0.280996	0.223185	0.165266

continued on next page

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
74	0.34104	0.283742	0.22637	0.168972
75	0.341018	0.283456	0.225973	0.168414
76	0.339781	0.281263	0.2233	0.165361
77	0.340696	0.282996	0.225085	0.167349
78	0.340678	0.282913	0.225334	0.167645
79	0.34103	0.283566	0.226195	0.168767
80	0.340725	0.283459	0.226274	0.168762
81	0.33877	0.280468	0.222561	0.164212
82	0.340178	0.282609	0.225364	0.167481
83	0.340926	0.283416	0.226041	0.16874
84	0.339943	0.282216	0.224653	0.167131
85	0.339803	0.281657	0.224145	0.166126
86	0.33945	0.281364	0.223665	0.16561
87	0.338659	0.280441	0.222078	0.163336
88	0.340223	0.28245	0.224535	0.166115
89	0.341613	0.284182	0.226766	0.169167
90	0.340371	0.282589	0.224947	0.166833
91	0.340022	0.282514	0.225245	0.167676
92	0.340522	0.28275	0.22489	0.16681
93	0.341423	0.283892	0.226625	0.169326
94	0.339687	0.281557	0.223961	0.16585
95	0.339413	0.281298	0.223589	0.165149
96	0.33921	0.280967	0.223022	0.164273
97	0.340496	0.283033	0.225082	0.167014
98	0.340525	0.282828	0.225057	0.167008

continued on next page

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
99	0.341924	0.28465	0.22737	0.170042

Table A.19.: Remaining Energy of Nodes over different time stamps using AODV

Node ID	T=100s	T=200s	T=300s	T=400s
0	0.332787	0.267067	0.202308	0.13745
1	0.328895	0.258227	0.188063	0.119458
2	0.335387	0.271528	0.206096	0.138221
3	0.332696	0.267222	0.200556	0.132197
4	0.323235	0.250811	0.177245	0.106459
5	0.322828	0.248777	0.175692	0.10443
6	0.335273	0.270925	0.204009	0.138158
7	0.334938	0.271518	0.203592	0.136369
8	0.331367	0.264843	0.192755	0.122216
9	0.328967	0.261578	0.187344	0.114485
10	0.328988	0.256726	0.183339	0.111387
11	0.329349	0.262496	0.194921	0.125296
12	0.319249	0.24041	0.161026	0.081824
13	0.318357	0.243126	0.166417	0.089973
14	0.326326	0.249293	0.165885	0.087848
15	0.324294	0.25799	0.190004	0.116952
16	0.335287	0.26851	0.198843	0.131318
17	0.323034	0.252161	0.176158	0.102115
18	0.31472	0.24066	0.161071	0.081001

continued on next page

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
19	0.327221	0.259713	0.188593	0.117693
20	0.334562	0.271483	0.207173	0.142014
21	0.339308	0.280295	0.218552	0.155836
22	0.329139	0.261511	0.191406	0.118936
23	0.327867	0.25947	0.187467	0.113637
24	0.310109	0.233083	0.155431	0.076186
25	0.321971	0.25085	0.17468	0.095426
26	0.330063	0.263083	0.192583	0.125098
27	0.323766	0.25277	0.183785	0.11618
28	0.328666	0.25909	0.187244	0.115389
29	0.327279	0.260267	0.191575	0.121981
30	0.324258	0.258334	0.18888	0.115756
31	0.32017	0.24355	0.168074	0.090938
32	0.307693	0.223347	0.140676	0.058419
33	0.316195	0.237827	0.156137	0.077571
34	0.318168	0.24142	0.15646	0.071226
35	0.317989	0.244496	0.169953	0.094536
36	0.307359	0.22082	0.13528	0.051603
37	0.307679	0.22133	0.137046	0.055047
38	0.309944	0.229927	0.15032	0.071286
39	0.325798	0.256345	0.189285	0.122804
40	0.340492	0.282147	0.223476	0.163903
41	0.321159	0.250845	0.178363	0.102225
42	0.31524	0.235316	0.156051	0.075354
43	0.309937	0.225888	0.139425	0.052846

continued on next page

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
44	0.316945	0.240377	0.159744	0.081195
45	0.32303	0.242662	0.161019	0.079047
46	0.313458	0.226555	0.142267	0.055014
47	0.309206	0.223791	0.142352	0.062921
48	0.307914	0.22082	0.135937	0.054305
49	0.319558	0.244409	0.170378	0.098469
50	0.332528	0.261635	0.187719	0.110597
51	0.319025	0.235728	0.157836	0.076375
52	0.312214	0.233849	0.155344	0.073312
53	0.324319	0.249167	0.174497	0.099221
54	0.321115	0.242626	0.162238	0.075967
55	0.313709	0.234149	0.155883	0.072326
56	0.320479	0.239469	0.161173	0.081739
57	0.31257	0.225374	0.141589	0.062089
58	0.309686	0.221738	0.13854	0.058453
59	0.321028	0.24583	0.172622	0.10034
60	0.333644	0.261181	0.188415	0.111091
61	0.325104	0.248164	0.168314	0.080323
62	0.325105	0.251012	0.175786	0.098182
63	0.318985	0.238915	0.159059	0.078646
64	0.317792	0.240924	0.162386	0.080022
65	0.323731	0.251241	0.179038	0.104283
66	0.318005	0.239739	0.16003	0.076011
67	0.315395	0.226633	0.141791	0.059918
68	0.307857	0.21822	0.132903	0.048526

continued on next page

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
69	0.320143	0.237787	0.159859	0.086817
70	0.33328	0.260481	0.187245	0.111736
71	0.331321	0.2552	0.179956	0.105992
72	0.323267	0.248646	0.174626	0.102011
73	0.314772	0.231146	0.150624	0.070472
74	0.32253	0.252828	0.183058	0.11091
75	0.32927	0.262237	0.196232	0.127453
76	0.303814	0.217649	0.13821	0.058535
77	0.308128	0.223441	0.142832	0.063998
78	0.327082	0.249309	0.17596	0.1071
79	0.324916	0.252619	0.180782	0.108547
80	0.333629	0.269467	0.205296	0.13895
81	0.326967	0.251589	0.175555	0.099256
82	0.334601	0.262149	0.192008	0.125222
83	0.327566	0.257585	0.186516	0.114563
84	0.325324	0.249858	0.175507	0.099204
85	0.323293	0.240744	0.163658	0.088853
86	0.317425	0.232421	0.153416	0.078733
87	0.308453	0.215455	0.133234	0.055484
88	0.318979	0.241224	0.16552	0.094875
89	0.333471	0.26599	0.201629	0.138348
90	0.330645	0.261805	0.19237	0.120844
91	0.331974	0.265291	0.197226	0.126145
92	0.331236	0.259782	0.189974	0.118529
93	0.335239	0.271529	0.20481	0.137616

continued on next page

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
94	0.330434	0.253798	0.179548	0.103815
95	0.320832	0.236944	0.158846	0.084013
96	0.315952	0.231257	0.151792	0.07664
97	0.329066	0.258353	0.190882	0.126378
98	0.33121	0.26496	0.197831	0.12957
99	0.336774	0.275134	0.213194	0.150685