Course ID: CS F366

FINAL REPORT

DELAY TOLERANT NETWORK FOR DATA DELIVERY IN AN IOT NETWORK

Name: Samina Shiraj Mulani
ID: 2018A7PS034P

CONTENT PAGE

1 Summary of the Research Paper titled ‘Delay-tolerant sensing data delivery for IoT network by using signal strength information’

1.1 The problem

Cellular technology can’t be used for IoT due to manufacturing cost, operation fee, limited battery lifetime, etc. (The proposed solution however does operate in an environment where cellular infrastructure like base station is available).
Internet of Things (IoT) has a large number of devices which transmit small amounts of data. All of this data is to be accumulated by the IoT gateway and sent to the cloud. Usage of short-range radios for the delivery of data from the IoT devices to the IoT gateway is better. For this scenario, Delay Tolerant Networks (DTNs) are suitable.

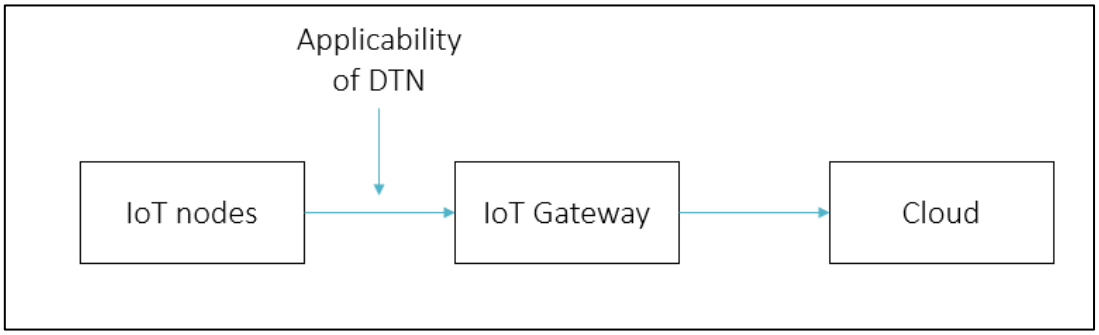


Figure 1: Use case of DTN in IoT

1.2 What is DTN?

Delay tolerant networking is an approach to networking architecture that is specifically designed for environments with intermittent connectivity. DTN has its own protocols for routing data through a network whose nodes are constantly coming in and out of contact with one another. The main difference between delay tolerant networks and the normal internet is that DTNs have a memory component and nodes are able to store data for an arbitrary amount of time, whereas nodes in the internet only relay data and all memory resources are concentrated at the end points of a connection.

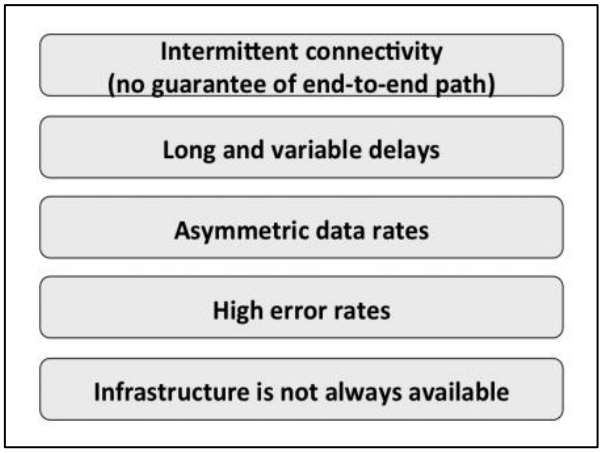


Figure 2: Features of DTN

The features of DTN present in current use case are - Intermittent connectivity, long and variable delays and the fact that infrastructure albeit available isn't suitable for direct use.

1.3 Objective

To enhance the routing efficiency in an DTN environment where IoT nodes transmit messages to IoT gateway.

1.4 Previous Works

- One class of solutions enable usage of cellular technology by adopting Power Saving Mode and extended discontinuous reception (DRX). This achieves scalability and terminal energy efficiency at the cost of communication delay.
- DTN based solutions employ different routing schemes. One category of routing schemes is the forwarding method in which a node deletes its copy of the message after forwarding it to another node. This means that there is only one copy of the message in the entire network at any point in time. The second category of routing schemes involve the replication method in which the message copy is not deleted and multiple copies exist in the network. The chances of a successful message delivery are higher with this method.
- One solution, a location-based routing scheme, uses Received Signal Strength Indicator (RSSI) between node and destination. The drawback for this method is that destination would have to transmit a beacon periodically, requiring high power.
- Another location-based routing solution uses the exact location coordinate of destination for routing. The disadvantage of this technique is that identifying the exact location is expensive (either in terms of battery power or complicated coordination among the base stations).

1.5 Proposed Solution

The proposed solution is a location-based routing scheme. GPS is avoided as it requires a large amount of power and is not usable indoors. Triangulation is also not used as it is a complex technique required a lot of coordination among the base stations. Received Signal Strength Indicator (RSSI) is used, which provides partial location knowledge.
Assumption made - Each node knows its RSSI and destination node's (IoT gateway) RSSI.
A target contour is defined as a ring of width w in which the IoT gateway is located. Routing performed in two phases.

- Approach phase: Deals with routing outside target contour. Single copy forwarding is performed in this phase. It uses RSSI information to estimate a node's distance from the destination. When two nodes approach and establish a connection, the node further away from the destination transfers its messages to the node that is closer, deleting the message copies from its buffer after transfer is completed.
- Spin phase: Deals with routing inside target contour. It uses a replication-based forwarding. This phase does not use RSSI information. Message may be flooded to all neighbours who are also within the target contour (similar to Epidemic Routing).

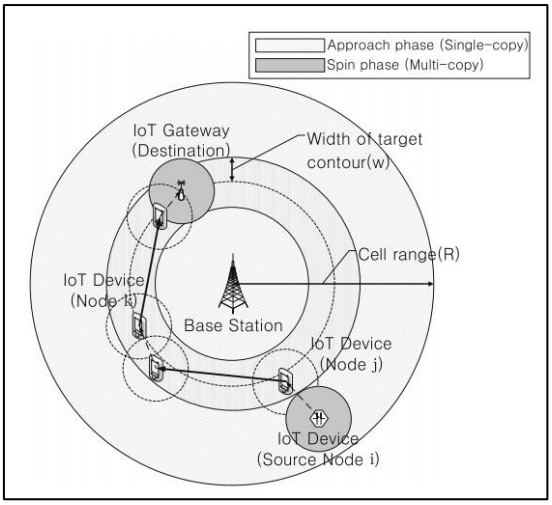


Figure 3: The proposed routing scheme for a single cell

1.6 Analytical Model

Formulae for the following have been derived.

- Expected meeting time between nodes
- Delivery Delay (Approach phase + Spin phase) (Estimated hops*Expected meeting time between nodes)
- Delivery Success Ratio (as each message has finite TTL, all messages may not reach the destination)
- Delivery Overhead (Number of copies of the message in the network – copies generated only in the spin phase)

The formulae for these parameters can be found in the Appendix.

1.7 Performance Metric Used

Delivery delay is used if TTL is infinite, otherwise, delivery ratio (probability of a successful message delivery) is used. Overhead ratio, which is the total number of message replicas in the network is another metric.

1.8 Summary of Simulation Results

Results of the analytical model agree with those of simulation.

1.8.1 Random Waypoint Model

The random waypoint model is used to simulate the movement of the nodes in a single cell.

- Delivery Delay with variation in number of nodes and contour size (TTL = infinity)
  The delivery delay in the approach phase decreases as the number of nodes increases. It is because as the number of nodes increases, the node encounter time decreases and the message is forwarded faster. When the size of target contour increases, the delay in the approach phase also decreases. It is because the distance that a message is forwarded by the approach phase decreases, as the size of target contour increases. The same as the approach phase, as the number of nodes increases, the delay in the spin phase decreases. It is because flooding within the target contour speeds up as the node encounter time decreases. As the size of target contour increases, however, the delay in the spin phase increases. It is because the distance between the destination node and the first infected node of the spin phase increases. Overall, increasing the size of target contour reduces the

delivery delay, as the delay in the approach phase decreases more sharply than the increase of the spin phase delay.

- Delivery ratio with variation in number of nodes and contour size
  As the number of nodes increases, the delivery ratio increases regardless of the target contour size. As the target contour size increases, the delivery ratio increases regardless of the user density. Higher TTL results in a higher delivery ratio.
- Distance of destination from base station
  Experimentally, it was found that the results were better if the IoT gateway was closer to the base station. (overhead and delay used as metric)
- Effect of transmission range
  The delivery delay decreases as the transmission range increases. It is not proportional because once the transmission range is bigger than the contour range, the efficiency of spin phase does not increase significantly even if a bigger transmission range is used.
- Width of contour
  It is desirable to set the target contour small when the node density is high. In contrast, when the node density is low, it is desirable to set the target contour large. To denote efficiency, (delivery ratio/overhead ratio) × 100 used. Optimal contour width can be obtained by fixing a few of the other parameters.
- Buffer size
  The impact of the buffer size on performance is not substantial with reasonable buffer size.

1.8.2 Comparison with existing schemes

Proposed solution outperformed other existing schemes like Epidemic Routing, Spray and Wait and Prophet Routing based on the metric of efficiency.

1.8.3 Real mobility traces

Proposed solution once again outperformed the existing solutions when real traces were used (data on movement of nodes collected by tracing the movements of 20 students for 3 months).

1.8.4 Multi Cell environment extension

When an IoT device is in a different cell from the IoT gateway, the home cell information of the mobile nodes is used for the forwarding decision. That is, when a node that carries the data meets another node, they exchange their home cell information. Then, the proximity of the home cells of the two nodes to that of the cell where IoT gateway located is compared. The data is forwarded to the encountered node, if the new node's home cell is closer to the IoT gateway cell than the current node with data. Once a node with data arrives at the cell where IoT gateway locates, the basic version of the proposed scheme is executed.

1.8.5 RSSI measurement instability

Median of RSSIs used for calibration. The problems that may occur if this is not done include the ping pong effect at the target contour boundary and incorrect forwarding to a node that's actually farther away from the destination. The ping pong effect occurs when a node keeps moving in and out of the contour thus alternating between spin and approach phase routing.

2 Implementation

2.1 The ONE simulator

The ONE is a DTN simulation environment that is capable of

- Generating node movement using different movement models.
- Routing messages between nodes with various DTN routing algorithms and sender and receiver types.
- Visualizing both mobility and message passing in real time in its graphical user interface.
- ONE can import mobility data from real-world traces or other mobility generators. It can also produce a variety of reports from node movement to message passing and general statistics.

2.2 Description of Data and Functions Used

The code for the Contour Router can be found in the Appendix.

Data	
CONTOUR_SIZE: public static final String	Setting which can be manipulate in default_settings.txt. Controls contour width.
GATEWAY_LCN: public static final String	Setting which provides the gateway location to all nodes (value has to be hard coded in settings)
CROUTING_NS: public static final String	Name space for this router
contourSize: private int	Stores contour width. Set to a default value if not provided in settings.
gateway: private Coord	Stores location of gateway (obtained via settings)
baseStation: private Coord	Stores location of base station (value is hard coded)

Functions	
public ContourRouter(Settings s)	Constructor. Creates a new message router based on the settings in the given Settings object. Contour size is set. Gateway location is set. Location of base station is hard coded.
protected ContourRouter(ContourRouter r)	Copy constructor
public void update()	• Checks out all sending connections to finalize the ready ones and abort those whose connection went down. Also drops messages whose TTL <= 0 (checking every one simulated minute). • If no sending connections exist (i.e. no transfer is occurring), first checks if message(s) can be sent to final destination and sends those. • If no sending connections and if no deliverable messages (i.e. connected end

	hosts aren't final destinations for any message), based on presence in spin or approach phase, appropriate routing is done.
protected Connection tryAllMessagesToRelevantConnections()	All connections are checked and if the appropriate conditions are met so that a message can be sent to the other node of the connection, message transfer begins. Appropriate conditions refer to – • If node in spin phase, it sends messages to other connected nodes in spin phase. • If node in approach phase, it sends messages to other connected nodes that are closer to destination node.
protected double distanceFromGateway(DTNHost node)	Calculates and returns the approximate distance from node to gateway by subtracting distances from base station.
protected void transferDone(Connection con)	• If in approach phase, message on being transferred is deleted from sending host's buffer. • If in spin phase, no such deletion is performed.
protected boolean inSpin()	Returns true if host node is in spin phase.
public MessageRouter replicate()	Creates a replicate of this router. The replicate has the same settings as this router but empty buffers and routing tables.

2.3 Working

Two groups of nodes exist.

- Group 1 – This consists of 1 node and represents the IoT gateway (destination). It follows Stationary Movement model and thus, its location has to be hard coded.
- Group 2 – This represents the nodes in the system. They follow Random Waypoint Movement Model. A fraction of these nodes generate messages.

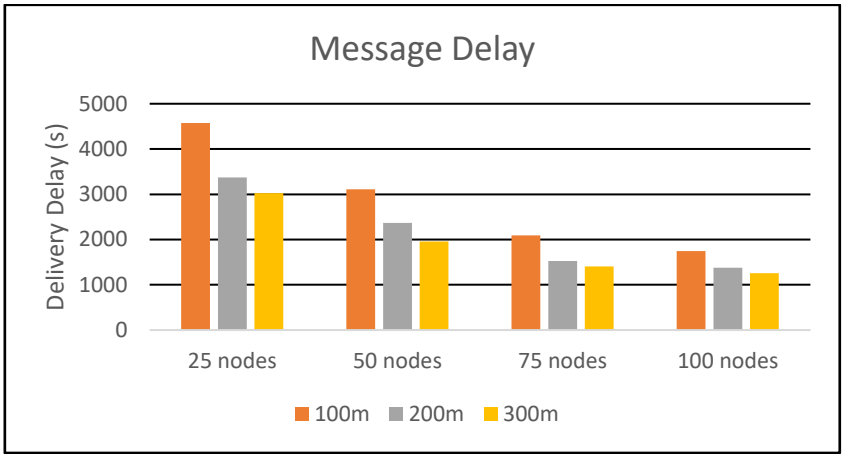
All groups have Bluetooth interface and implement Contour Router. A 2km x 2km cell is used with the base station located at the centre (coordinate (1000,1000)).
The Group 2 node speed is randomly selected from the range of 1.5 m/s to 2.0 m/s and the node pausing time is randomly selected from the range of 1 to 3 seconds. The transmission range of a node is set to 50m, if not specified otherwise. 25 nodes are responsible for generating traffic. Size of the message is randomly chosen from the range of 5 KB to 10 KB. Simulation time has been fixed to 3h and destination node's location has not been randomized (fixed to coordinates (500,500)).

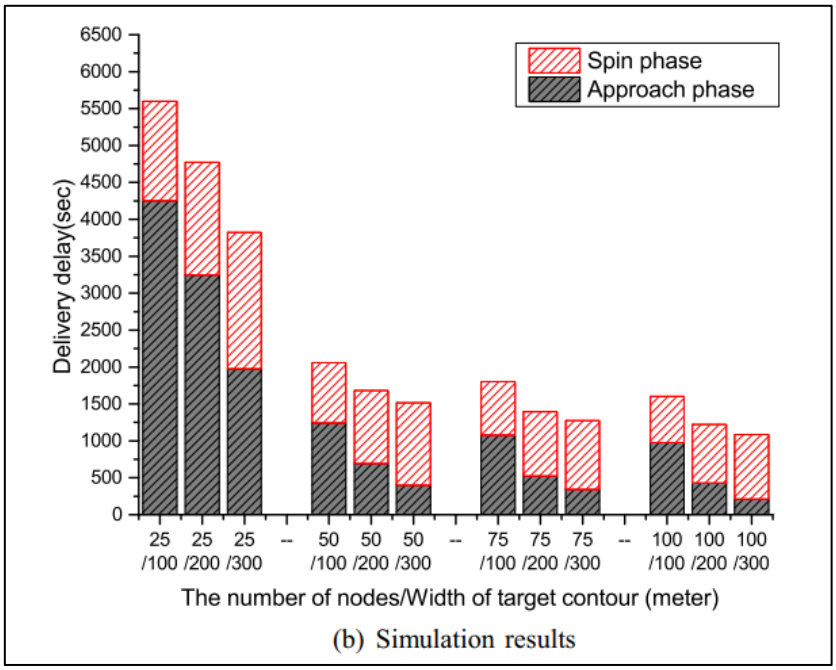
3 Results

Each subsection shows the graph obtained by simulations (using the code written by me) first and the graph showing the simulated results in the paper, second.

3.1 Variation of message delay with change in contour width and number of nodes

Separate time durations of the spin phase and approach phase were not recorded.

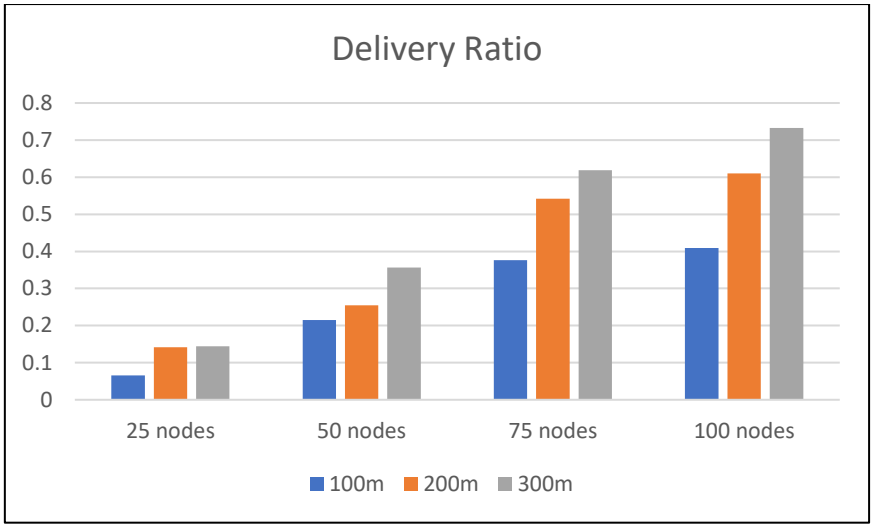


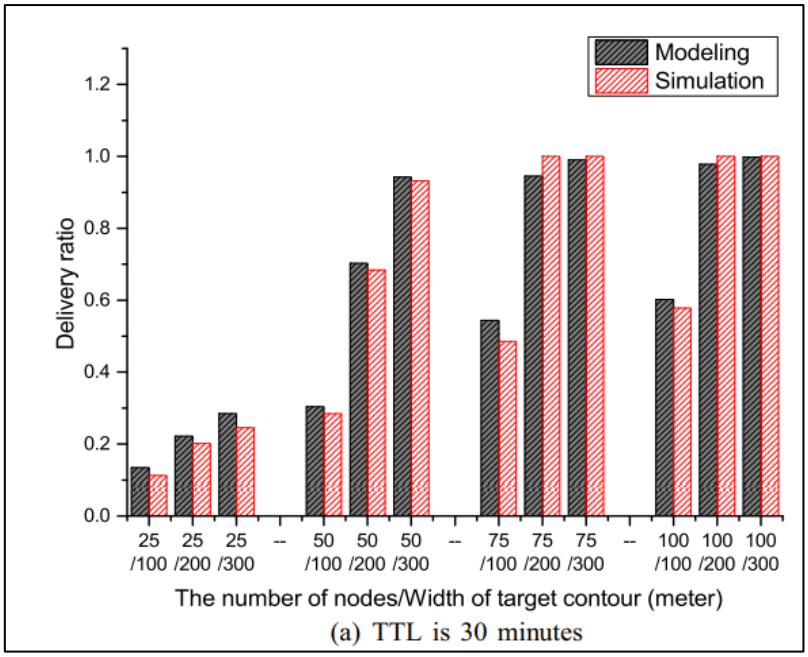


(b) Simulation results

3.2 Variation of delivery ratio with change in contour width and number of nodes

TTL has been set to 30 minutes.

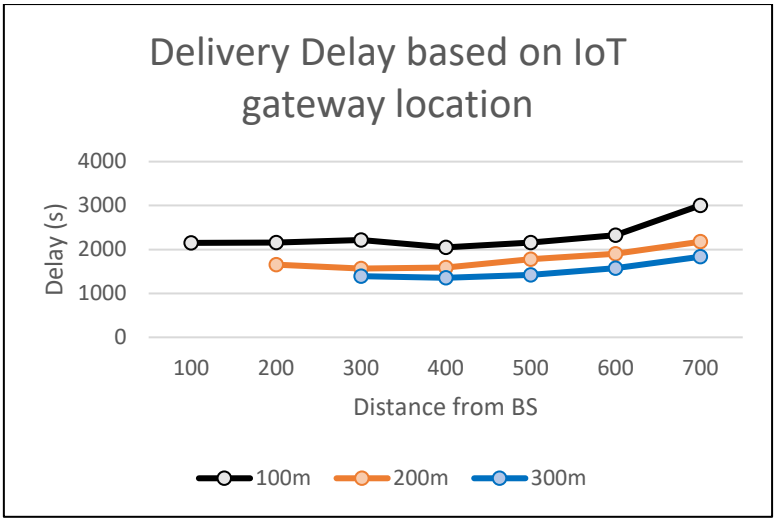


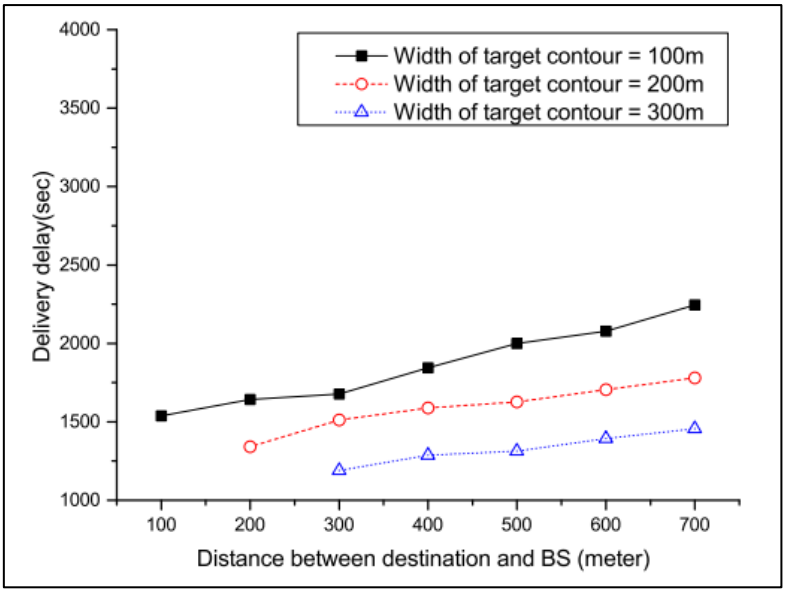


(a) TTL is 30 minutes

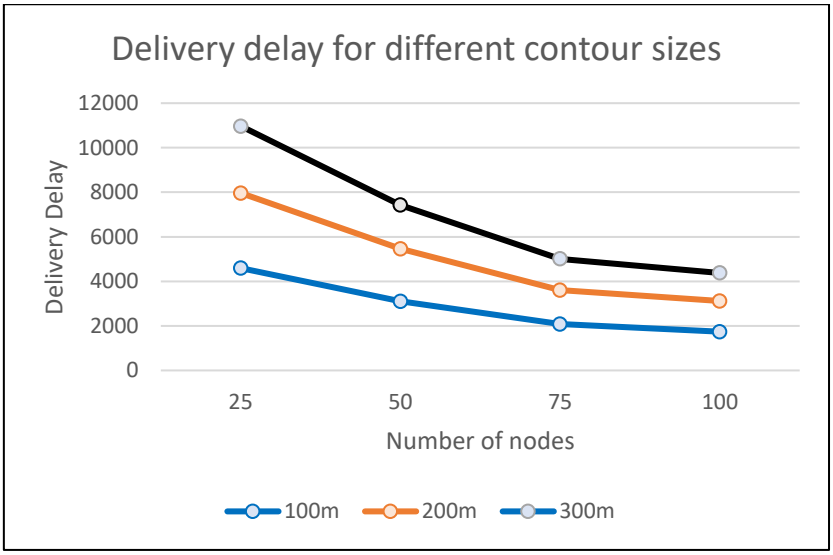
3.3 Delivery delay based on distance between IoT gateway and Base Station

There is a slight discrepancy when contour size is equal to the distance from base station when comparing the results I obtained with the results in the paper.





3.4 Delivery Delay for different contour sizes



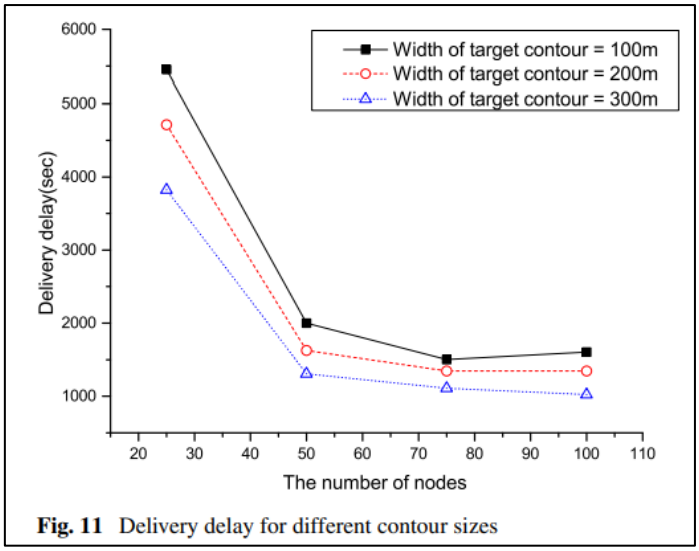
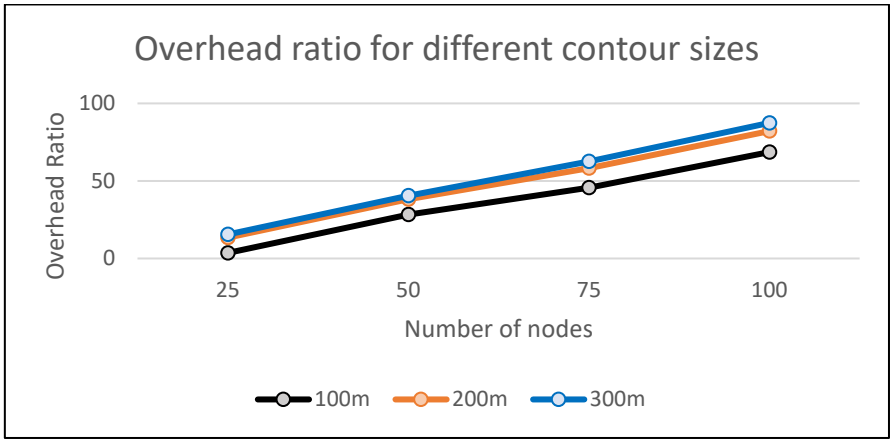


Fig. 11 Delivery delay for different contour sizes

3.5 Overhead Ratio for different contour sizes

A counter is incremented whenever a message delivery in spin phase occurs. This number is divided by the total number of created messages to obtain the overhead ratio. The overhead ratio as calculated in the MessageStatsReport in the ONE simulator is not used replicas are created only in the spin phase.



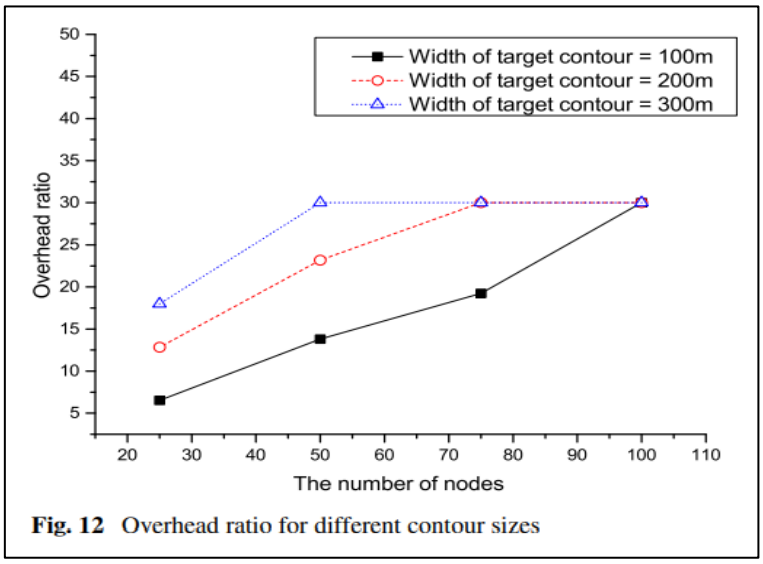
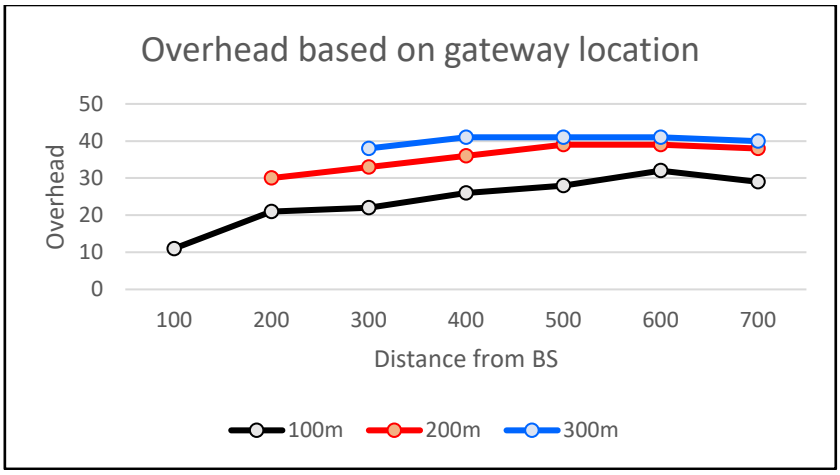
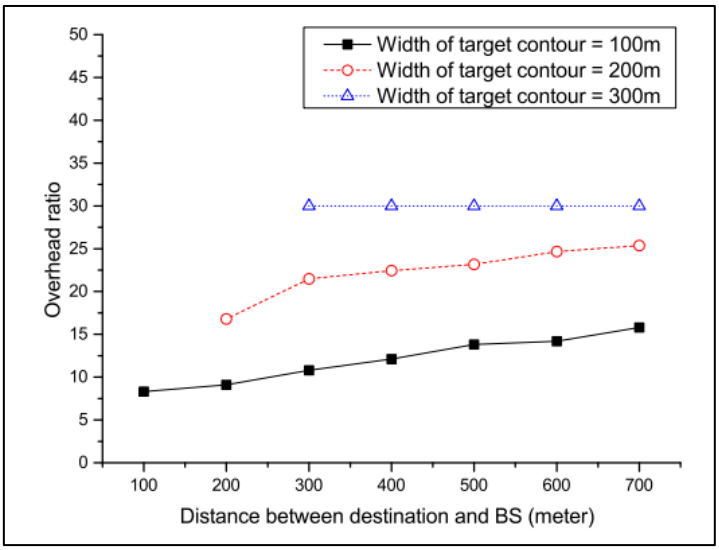


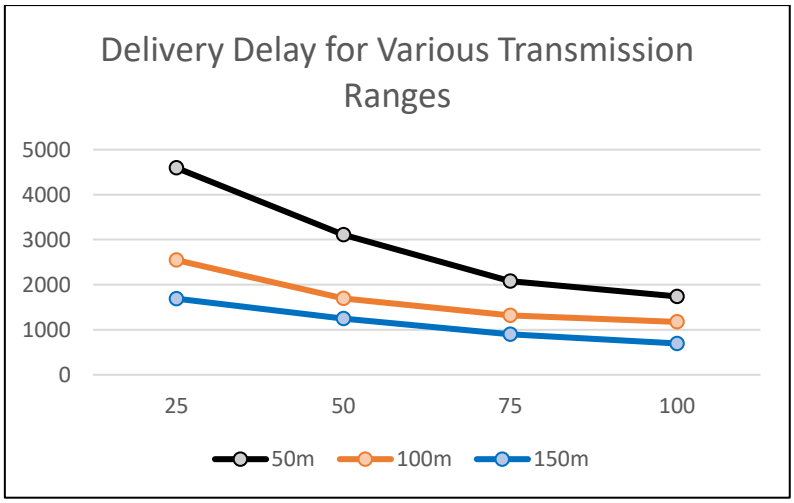
Fig. 12 Overhead ratio for different contour sizes

3.6 Overhead Ratio based on distance between IoT gateway and Base Station





3.7 Delivery Delay for various transmission ranges



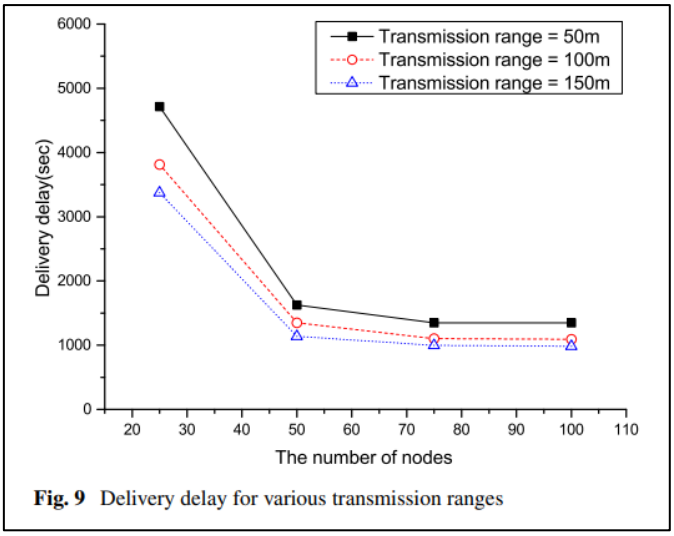
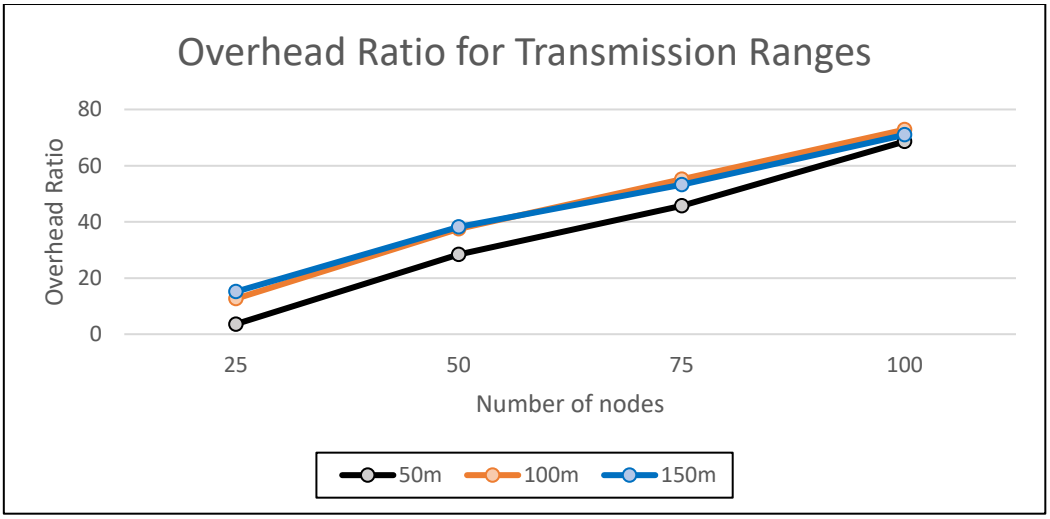


Fig. 9 Delivery delay for various transmission ranges

3.8 Overhead ratio for Various Transmission Ranges

A discrepancy with the trend shown in the research paper is observed.



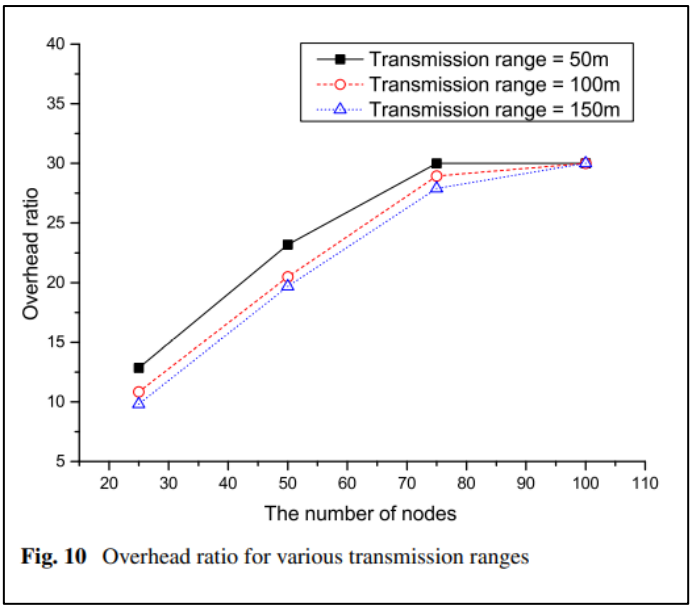
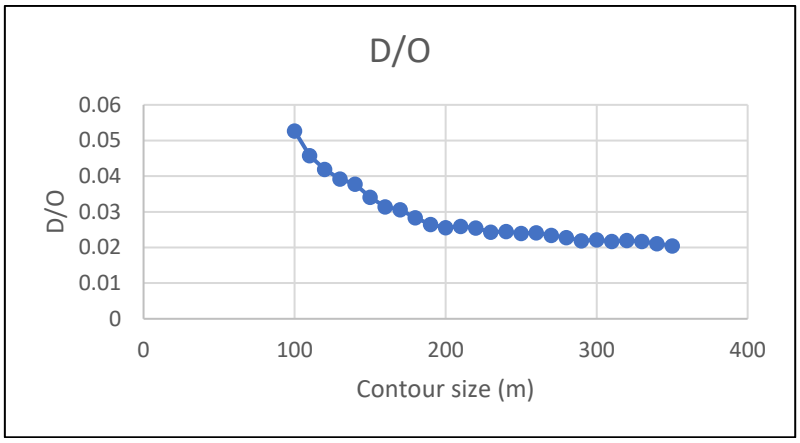


Fig. 10 Overhead ratio for various transmission ranges

3.9 Impact of target contour width

Factors like TTL, number of nodes and transmission range are kept constant while target contour width is varied while measuring the ratio of delivery ratio to overhead (denoted by D/O). It was found in the simulation that both overhead ratio and delivery ratio increase with increasing contour size. A discrepancy with the trend shown in the research paper is observed.



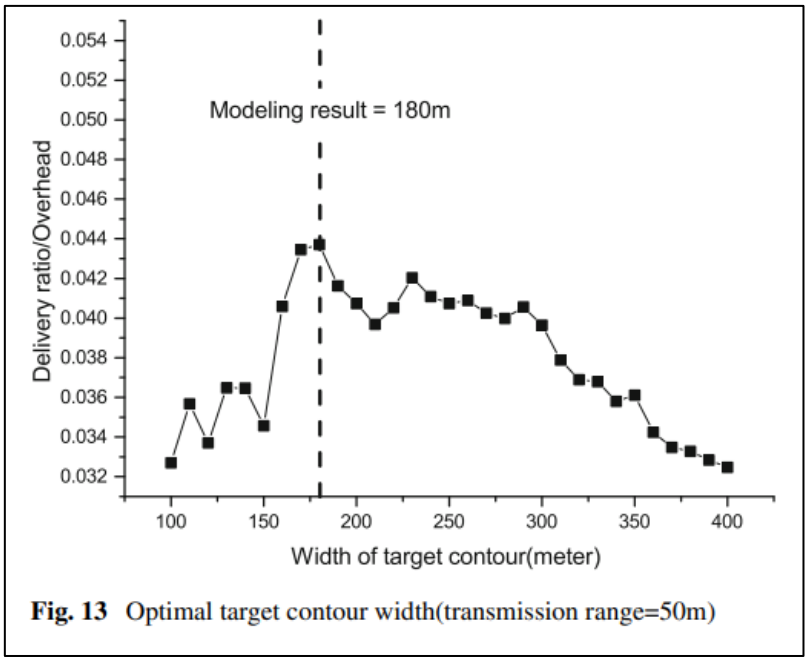


Fig. 13 Optimal target contour width(transmission range=50m)

4 Conclusion

4.1 Possible causes of discrepancies

Observations – When distance of IoT gateway was fixed (say 100m) from base station, varying its position while keeping radial distance from base station the same led to different delivery delay and overhead ratio values.

Thus, possible causes for the discrepancies include:

- Location of IoT gateway is not randomized for each simulation. It is fixed.
- A fixed set of same nodes (the first 25) generate traffic.
- Simulation time is too less.
- Incorrect calculation of overhead ratio due to some misunderstanding of the paper.

4.2 Possible scope for further work

- The reasoning behind why performance is better when the IoT gateway is closer to the base station has not been elaborated in the paper and thus further research in this area can be carried out.
- Currently, epidemic routing is used within the spin phase. Impacts on delivery ratio and overhead ratio can be examined when epidemic routing is replaced with spray and wait routing.

5 Appendix

5.1 Analytical model derivation

5.1.1 Delivery Delay

Let R be the radius of the total cell and w be the contour width. Area of the target contour is thus:

$$(R/2+w)^2\pi-(R/2-w)^2\pi=2Rw\pi$$

As a result, the probability S that a node is in target contour is:

$$S=\frac{2Rw\pi}{R^2\pi}=\frac{2w}{R}$$

1-S is the probability a node is not in the target contour.

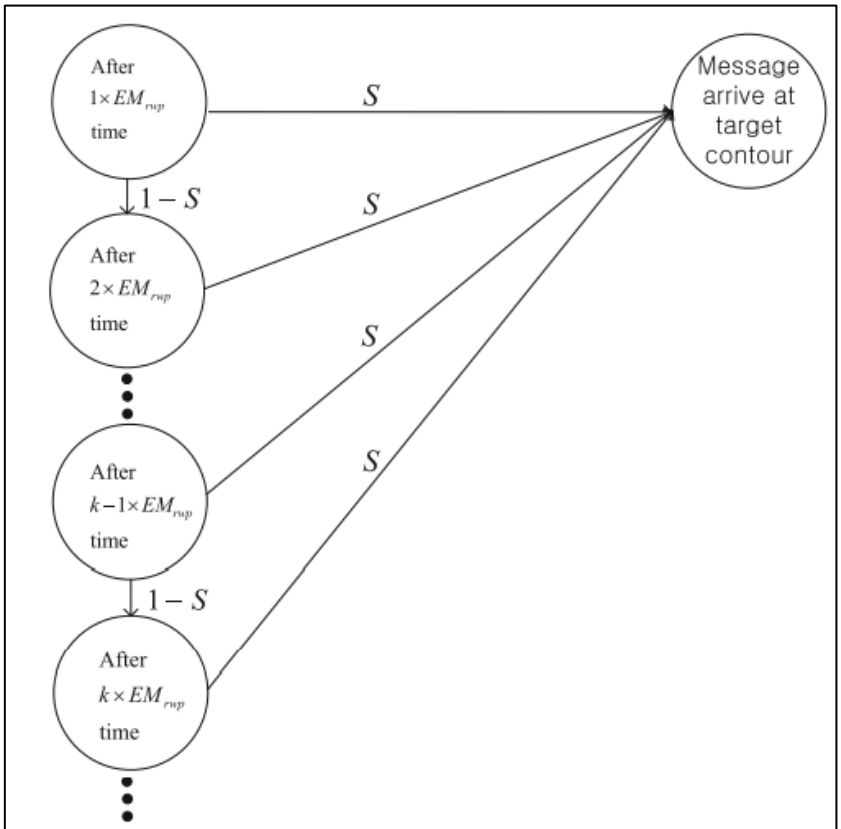


Fig. 3 The delay estimation model for the approach phase

Using the above probability graph, where EM_{rwp} is the expected meeting time between two nodes when the nodes follow a random waypoint movement model, expected delay of approach phase is:

$$\sum_{k=1}^{\infty} kS(1-S)^{(k-1)}\times EM_{rwp}$$

Following is the probability graph applied to the spin phase:

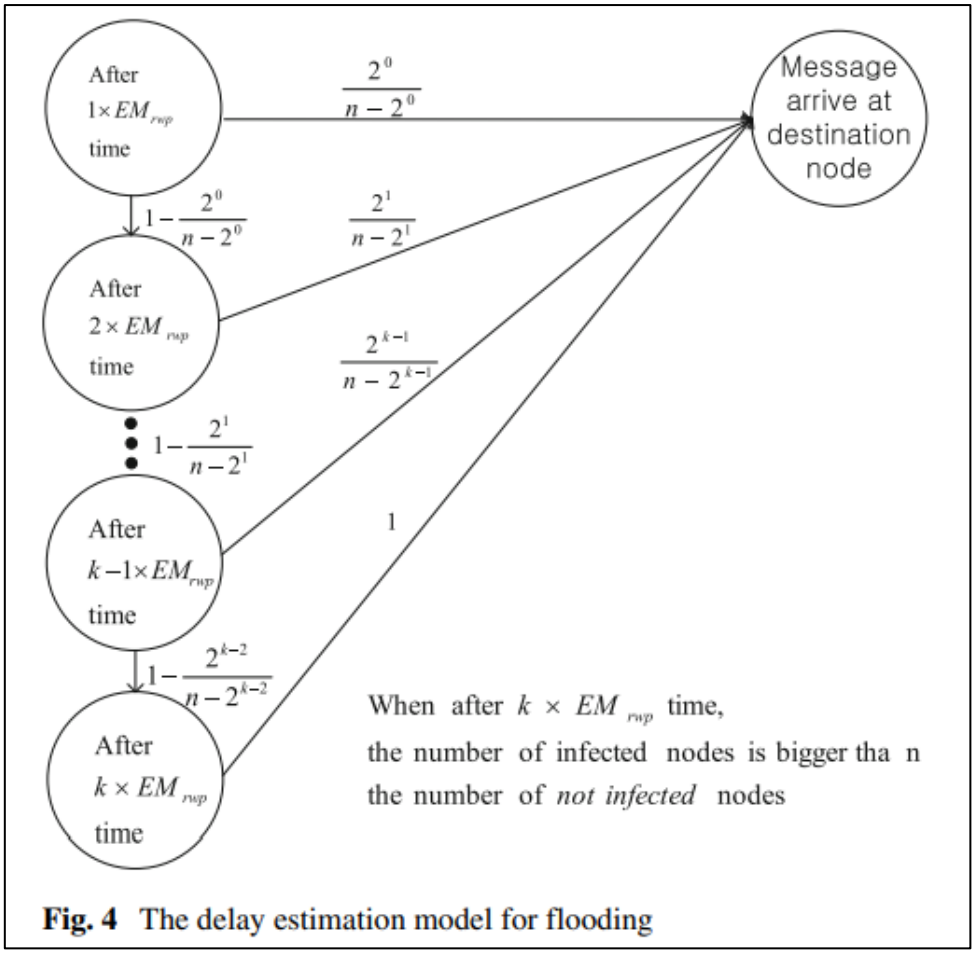


Fig. 4 The delay estimation model for flooding

The expected number of hops in spin phase, EH_{spin} is given by:

$$EH_{spin} = \sum_{k=1}^{k_{fc}} kP_k,$$

$$P_k = \begin{cases} \frac{2^0}{n_c-2^0} & \text{if } k=1 \\ \left(\prod_{i=1}^{k-1} \frac{2^{i-1}}{n_c-2^{i-1}}\right) \times \frac{2^{k-1}}{n_c-2^{k-1}} & \text{if } 2^{k-2} < n_c - 2^{k-2} \\ \prod_{i=1}^{k-1} \frac{2^{i-1}}{n_c-2^{i-1}} & \text{if } 2^{k-2} > n_c - 2^{k-2} \end{cases}$$

where k_{fc} is $min_k\{2^{k-2} > n_c - 2^{k-2}\}$.

The expected delay for spin phase is EH_{spin} * EM_{rwp}.

5.1.2 Delivery success ratio and delivery overhead

The probability of message delivery in k-hops, which is denoted by P_k, can be calculated as follows:

$$P_k = \begin{cases} \frac{2^0}{n_c - 2^0} & \text{if } k = 1 \\ \left(\prod_{i=1}^{k-1} \frac{2^{i-1}}{n_c - 2^{i-1}}\right) \times \frac{2^{k-1}}{n_c - 2^{k-1}} & \text{if } 2^{k-2} < n_c - 2^{k-2} \\ \prod_{i=1}^{k-1} \frac{2^{i-1}}{n_c - 2^{i-1}} & \text{if } 2^{k-2} > n_c - 2^{k-2} \end{cases}$$

By considering all possible cases that a message is delivered to the destination within TTL_h, the estimated delivery ratio can be calculated as follows:

$$\sum_{k=1}^{TTL_h} \left(S(1-S)^{k-1} \times \sum_{i=1}^{TTL_h - k} P_i \right)$$

The expected number of copies, or overhead ratio can be calculated as follows:

$$\sum_{k=1}^{TTL_h} \left(S(1-S)^{k-1} \times \sum_{i=1}^{TTL_h - k} P_i \cdot (2^i - 1) \right)$$

5.2 Code for Contour Router

Can also be found on https://github.com/samina-mulani/DTN-Project

```
package routing;

import java.util.ArrayList;
import java.util.List;

import core.Connection;
import core.Settings;
import core.Coord;
import core.DTNHost;
import core.Message;

public class ContourRouter extends ActiveRouter {

	public static final String CONTOUR_SIZE = "ctrSize";
	public static final String GATEWAY_LOCN = "gateway";
	public static final String CROUTING_NS = "ContourRouter"; //name
space

	private int contourSize;
	private Coord gateway;
	private Coord baseStation;
	/**
	 * Constructor. Creates a new message router based on the settings
in
	 * the given Settings object.
	 * @param s The settings object
	 */
	public ContourRouter(Settings s) {
		super(s);
```

```
                Settings contourRouterSettings = new Settings(CROUTING_NS);
                int defaultValue = 5;
                this.contourSize =
contourRouterSettings.getInt(CONTOUR_SIZE, defaultValue);
                double [] gateway_XY =
contourRouterSettings.getCsvDoubles(GATEWAY_LOCN,2);
                double x = gateway_XY[0];
                double y = gateway_XY[1];
                this.gateway = new Coord(x,y);
                this.baseStation = new Coord(1000,1000); //hard-coded for
now
        }

        /**
         * Copy constructor.
         * @param r The router prototype where setting values are copied
from
         */
        protected ContourRouter(ContourRouter r) {
                super(r);
                this.contourSize = r.contourSize;
                this.gateway = r.gateway;
                this.baseStation = r.baseStation;
        }

        @Override
        public void update() {
                super.update();
                if (isTransferring() || !canStartTransfer()) {
                        return;
                }

                if (exchangeDeliverableMessages() != null) {
                        return;
                }

                tryAllMessagesToRelevantConnections();

        }

        protected Connection tryAllMessagesToRelevantConnections()
        {
                //list of all connections obtained
                List<Connection> connections = getConnections();
                if (connections.size() == 0 || this.getNrofMessages() == 0)
{
                        return null;
                }

                //list of all messages obtained
                List<Message> messages =
                        new ArrayList<Message>(this.getMessageCollection());
                this.sortByQueueMode(messages);

                for (int i=0, n=connections.size(); i<n; i++) {
                        Connection con = connections.get(i);
                        DTNHost to = con.getOtherNode(this.getHost());
```

```
                        if(inSpin(this.getHost())) //node in spin phase
                        {
                                //rejects connections with hosts that are
NOT in spin phase
                                if(!inSpin(to)) continue;
                        }
                        else
                        {
                                //rejects connections with hosts that are
NOT closer to gateway/destination

        if(distanceFromGateway(to)>distanceFromGateway(this.getHost()))
                                        continue;
                        }

                        Message started = tryAllMessages(con, messages);
                        if (started != null) {
                                return con;
                        }
                }

                return null;

        }

        protected double distanceFromGateway(DTNHost node)
        {
                double rGateway = this.gateway.distance(this.baseStation);
                double rNode =
node.getLocation().distance(this.baseStation);
                return Math.abs(rGateway-rNode);
        }

        @Override
        protected void transferDone(Connection con) {

                if(!inSpin(this.getHost())) //if in approach phase
                        /* don't leave a copy for the sender */
                        this.deleteMessage(con.getMessage().getId(), false);
        }

        @Override
        public MessageRouter replicate() {
                return new ContourRouter(this);
        }

        public boolean inSpin(DTNHost node) {
                if(distanceFromGateway(node)<contourSize)
                return true;
                else return false;
        }
}
```