

# **The Impact of Rogue Nodes on the Dependability of Opportunistic Networks**

Faizan Agha

2014-2015

Erasmus Mundus MSc in Dependable Software Systems



Department of Computer Science

Maynooth University, Maynooth

Co. Kildare, Ireland

A dissertation submitted in partial fulfillment of the requirements for the  
Erasmus Mundus MSc Dependable Software Systems  
(Distributed Systems Stream)

Head of Department : Dr Adam Winstanley

Supervisor : Dr. Stephen Brown

09 June 2015

Word Count: 17,182

## Abstract

Opportunistic Networks (OppNets) are an extension to the classical Mobile Ad hoc Networks (MANETs) where the network is not dependent on any infrastructure (e.g. Access Points or centralized administrative nodes). OppNets can be more flexible than MANETs because an end to end path does not exist and much longer delays can be expected. Whereas a Rogue Access Point is typically immobile in the legacy infrastructure based networks and can have considerable impact on the overall connectivity, the research question in this project evaluates how the pattern and mobility of a rogue nodes impact the dependability and overall "Average Latency" in an Opportunistic Network Environment. We have simulated a subset of the mathematical modeling performed in a previous publication in this regard.

Ad hoc networks are very challenging to model due to their mobility and intricate routing schemes. We strategically started our research by exploring the evolution of Opportunistic networks, and then implemented the rogue behavior by utilizing The ONE (Opportunistic Network Environment, by Nokia Research Centre) simulator to carry out our research over rogue behavior. The ONE simulator is an open source simulator developed in Java, simulating the layer 3 of the OSI model. The Rogue behavior is implemented in the simulator to observe the effect of rogue nodes. Finally we extracted the desired dataset to measure the latency by carefully simulating the intended behavior, keeping rest of the parameters (e.g. Node Movement Models, Signal Range and Strength, Point of Interest (POI) etc) unchanged. Our results are encouraging, and coincide with the average latency deterioration patterns as modeled by the previous researchers, with a few exceptions. The practical implementation of plug-in in ONE simulator has shown that only a very high degree of rogue nodes impact the latency, making OppNets more resilient and less vulnerable to malicious attacks.

*I consider it an honour to work with Dr. Stephen Brown; without his knowledgeable guidance and kind supervision, it would not have been possible.*

*To my dearest and beloved parents; for their interminable love and care, that kept me motivated and genuinely helped me achieve this milestone.*

## Table of Contents

|  |    |
|--|----|
| Abstract.....  | 2  |
| 1 Introduction.....  | 6  |
| 2 Background and Motivation.....   | 9  |
| 2.1 Infrastructure-based communication networks.....                                   | 9  |
| 2.2 Infrastructure-less Networks.....  | 9  |
| 2.3 Evolution to Opportunistic Networks.....   | 10 |
| 3 Rogue Behavior in Opportunistic Networks.....  | 13 |
| 4 Epidemic Router [6].....   | 15 |
| 5 Related Work.....  | 17 |
| 5.1 Survey of DTN Protocols [2].....   | 17 |
| 5.1.1 Deterministic Routing.....   | 18 |
| 5.1.2 Stochastic Or Dynamic Routing.....   | 18 |
| 5.1.3 Model-Based Approach.....  | 19 |
| 5.1.4 Node Movement Control-Based Approaches.....                                      | 20 |
| 5.1.5 Coding Based Approaches.....   | 20 |
| 5.2 On effects of Cooperation in DTNs [8].....   | 21 |
| 5.3 Analytical Modeling to determine the impact of selfish or rogue behavior [10]..... | 24 |
| 5.3.1 Modeling Message Delivery Under Rogue Nodes.....                                 | 25 |
| 5.3.2 Message Delivery Under Unrestricted Relaying.....                                | 26 |
| 5.3.3 Message Delivery Under Two-Hop Relaying.....                                     | 26 |
| 5.3.4 Numerical Results.....   | 26 |
| 6 Why ONE Simulator?.....  | 29 |
| 6.1 Shortcomings in Network Simulators for OppNets before ONE.....                     | 29 |
| 6.2 Routing Specific and OppNet Specific simulators.....                               | 29 |
| 6.3 Issues with real world traces import for legacy simulators.....                    | 29 |
| 6.4 ONE Simulator with Mobility, Visibility and OppNet Support.....                    | 30 |

|       |   |    |
|-------|---|----|
| 7     | Methodology .....   | 32 |
| 7.1   | Higher Level Approach.....  | 32 |
| 7.2   | Plug-in for Mobile Rogue Nodes in ONE Simulator .....                     | 32 |
| 7.3   | Metric for Observation.....   | 33 |
| 7.4   | Rogue Plug-in Implementation .....  | 33 |
| 7.4.1 | Rogue Behavior in ONE: Implementation of subset of Analytical Model ..... | 34 |
| 7.4.2 | Routing implementation for Rogue Noes .....                               | 35 |
| 7.4.3 | Core Implementation for Rogue Nodes .....                                 | 36 |
| 7.5   | Experimental Setup.....   | 37 |
| 7.5.1 | Platform for Scenario Simulations .....                                   | 37 |
| 7.5.2 | Eclipse Settings .....  | 38 |
| 7.6   | Verification of Rogue Behavior in ONE .....                               | 38 |
| 7.6.1 | Verification of implementation and existing functionality .....           | 38 |
| 7.6.2 | Verification of implementation of Rogue Behavior .....                    | 39 |
| 7.7   | Experiments with Varying Rogue Degree .....                               | 41 |
| 7.7.1 | First Iteration.....  | 41 |
| 7.7.2 | Second Iteration .....  | 42 |
| 7.7.3 | Explanation of Metrics in Message Stats Report .....                      | 45 |
| 8     | Results.....  | 48 |
| 8.1   | Dataset and Explanation.....  | 48 |
| 8.2   | Deduction/Analysis.....   | 52 |
| 9     | Conclusions and Future Work.....  | 56 |
|       | Appendix A.....   | 58 |
| 10    | Appendix B .....  | 60 |
| 11    | Appendix C .....  | 61 |
| 11.1  | Input configuration file for Experiments.....                             | 61 |
| 12    | Bibliography .....  | 65 |

# 1 Introduction

Mobile nodes in Opportunistic Networks (OppNets) transmit and receive useful messages in a 'Store and Forward' architecture [1]. OppNets are an extension to classical Mobile Ad hoc Networks (MANETs) where the network is not dependent on any infrastructure (e.g. Access Points or centralized administrative nodes). The connectivity can also be intermittent and episodic, similar to the Delay Tolerant Networks (DTN). They are typically referred to as suitable for, and deployable in 'Challenged Networking Conditions' owing to their inherent resilience with respect to a path-less communication model, flexibility of internetworking between discrete technologies and critical dependence of nodes mobility and cooperation. An interesting as well as a highly challenging feature of these networks is, that an end to end path from source to destination may not necessarily exist at any point in time. Path management in MANETs is an expensive task to maintain over the period of time in a dynamic network topology. Nodes may enter or leave without any permanent association with the network, therefore the mobility is an inherent feature, which leads to creating or breaking of existing paths. OppNets offer their useful operating model to get rid of this constraint.

Each node in an OppNet is crucial for providing end to end delivery of useful messages. This leads to very important and critical concerns of reliability, security and privacy of the content: The data may or may not be encrypted and can be potentially seen by the intermediaries. Routing is already a big challenge for these networks [2], therefore the notion of altruism and cooperation is expected from the contributing nodes. This project will first investigate and explore OppNets with respect to their evolution, routing schemes and key research areas with the help of past publications and related work. This was also needed to build the required knowledge of domain with respect to key terminologies, protocols and tools available for the research and development.

The contributing nodes struggle to probe, and search for every available 'opportunity' (hence called 'Opportunistic' Networks) in order to deliver the messages, which can potentially bring them closer to the destination. However, the notion of 'rogue' (or selfish) behavior can also exist (as explained in Section 3 later) just like it can exist in the infrastructure based networks. In this project we will analyze, compare and contrast the impact of rogue nodes in the network against the normal networking conditions. Whereas a rogue Access Point is immobile, the research question in this project will address on *how the pattern of a rogue nodes impacts the overall "average latency" in an OppNet environment*. Although delays are expected in OppNets, but the existence of added malicious entities can further hamper the performance of network. Also it is important to mention here, that though the primary focus in OppNets is to maximize the transmission of data, very few papers have published the impact, that rogue mobile nodes can have on further latency in the network.

The impact of rogue nodes will be measured by first simulating communication scenarios under a specific mobility pattern assuming no malicious behavior exists. The simulation will again performed with the same parameters, except that this time a portion of nodes will act as rogue nodes.

We implement (and then simulate) the rogue behavior by utilizing The ONE simulator. The ONE (Opportunistic Network Environment) simulator is an open source simulator developed in Java simulating layer 3 of the OSI model. The rogue behavior will be customized in the simulator to observe the effect of malicious nodes. Finally we develop the desired dataset to measure the

performance and QoS metrics by rigorously simulating the intended behavior, keeping rest of the parameters (e.g. node movement models, signal range and strength, Point of Interest (POI) etc) unchanged. Our results are based on comprehensive simulation of rogue behavior, and averaging the delays introduced on the basis of 100 seeds for each run.

Although, ONE simulator is an open source platform developed by Nokia Research Centre (Finland), it has hundreds of interlinked APIs, import and export functionalities. It was built without any negative scenarios (e.g. routing via malicious nodes). It is comparatively a newer tool, research and development is primarily being done by peer and online collaboration (forums and emails) by self learning and experimenting. Simulations involving OppNets can be time consuming (being a 'delay' tolerant simulation), reason being (discussed in more detail in Section 2.3 in their evolution) since these networks do not suffer from the overhead of path management and connection establishment at TCP layer [3], they need to wait for the possible opportunities to find the optimal intermediary for the message delivery to the end destination. Finding the intermediate 'routers' (the term is used interchangeably with nodes/hosts in the literature) depends on the underlying routing protocol, mobility model and most importantly the altruism of the neighbors. Therefore simulation of scenarios requires careful analysis of input parameters and output dataset after every iteration. Sometimes it is also necessary to graphically monitor the movement and message logs even during the simulation, which motivated us that we chose ONE simulator for our project (Please refer to Section 6: Why ONE Simulator). The intent is to make the plug-in implemented as part of our implementation on Rogue Behavior as a potential addition to the research community of ONE. To the best of our knowledge, currently no plug-in or configurable parameter as an additional feature exists in the ONE simulator source files [4]. Although the researchers in [5] plot latency patterns on the basis of mathematical modeling and simulate in ONE, their source files or repositories are not available publically.

There is not a huge amount of work done previously on rogue behavior in OppNets, as compared to the efforts on routing and forwarding techniques. It makes sense because the prime intention of OppNets was maximizing communication for areas or networking conditions which are considered challenged for even slighter connectivity due to geographical areas, infrastructure deficiencies and poor coverage of wireless media [1]. However, the notion of altruism cannot be completely ignored since the mobility and portability of mobile devices can give birth to opportunism and selfishness.

A number of papers analyze the performance variations by mathematical modeling followed by simulation of key metrics like latency and delivery ratio. Majority of the researchers have used "Epidemic Routing [6] and Two hop Relay" [7] [5] routing protocols. We referred to two publications, one that analytically modeled the rogue behavior, and another performed the modeling as well as implemented the behavior in ONE simulator. While the Epidemic Routing Protocol is explained in much more detail in Section 4 (Epidemic Router). Two hop relay typically aims to reduce the number of transmissions during message forwarding. Both these protocols are multi-copy protocols but the essential difference between them is that in Two hop relay, only the source can infect the neighbors with the desired message. It also ensures that each time it encounters a node with no copy of the desired message, it forwards the message to it such that there are two copies of message available. The intermediaries are not allowed to infect other nodes with that very message, unless they are the intended destinations [8] [9]. Considering the scope of our project we only performed a subset of implementation and subsequent simulation as

(Section 7: Methodology for further details) performed analytically in [10], but we also referred to [5] for our some help with respect to simulation in ONE. There is a good correlation between the simulation results presented in our project and the analytical results from these two papers. Some dissimilarities in the impact are also identified and discussed (Section 8: Results).

## 2 Background and Motivation

### 2.1 Infrastructure-based communication networks

Two major classes of data and communication networks co-exist globally. In networking literature they are mostly referred to as: Infrastructure-based and Infrastructure-less (or ad hoc) networks. Communication Networks deployed on top of a fixed and proactively designed 'infrastructure' are commonly known as Infrastructure-based networks. These are designed keeping in view the Quality of Service (QoS), security, reliability and availability requirements. Typically the design involves performing geographical surveys prior to the design and implementation of physical infrastructure. Some examples include Ethernet wired LAN, 802.11 Access Point and GSM/UMTS/LTE based telecommunication networks. The network topology, capacity, bandwidth and related QoS parameter might differ in these distributed networks, but one thing is common: They are dependent on central nodes (or a set of pre-installed nodes) in order to carry out message transmission/Receiving. Network topology is pre-designed and fixed, and is not prone to any alterations during the routing and forwarding of messages. These networks have matured after being revolutionized and vivified by introducing newer protocols and underlying technologies. Security, though was an afterthought in infrastructure-based networks, modern cryptography schemes have made these networks less vulnerable to attacks (e.g. Fake Access Points, Packet Sniffing, Distributed Denial of Service (DDoS), Man in the Middle attacks etc).

### 2.2 Infrastructure-less Networks

With the inception of Wireless technologies, mobile devices started to gain more popularity worldwide than the desktop based end stations, the portability of these handheld devices leveraged by their multiple communication interfaces (e.g. Bluetooth IEEE 802.15, WLAN 802.11 family, NFC and Infrared etc) have helped mobile communication getting rid of underlying infrastructure for traffic management and data relaying [11] [12]. When the mobile nodes communicate in an ad hoc manner without any dependence on any central entity (Central Access Point, Router etc), they form an 'Infrastructure-less' network [13]. In the 802.11 family of WLANs, they are also said to be operating in an 'Ad hoc mode'. This led to the evolution of Mobile Ad hoc Networks (MANETs).

There is no fixed network topology in MANETs. Routes are built on the fly and need to be maintained over the period of time, at the expense of bandwidth. When a route is built, all the nodes in the current network need to be made aware by some handshaking mechanism. There are not pre-designated routing or relaying/gateway nodes in these networks [14]. Every node is a contributor to the overall performance and QoS parameters. Throughput, packet delivery ratio, packet losses, Round Trip Times, bandwidth and delivery delays keep fluctuating as the nodes move, leave or join the network in an ad hoc manner. The beauty of ad hoc network is, that the communication or message passing is no longer dependent on a single entity (e.g. an Access Point (AP) in an 802.11n configuration which acts a central unit for message passing, queue management, and bandwidth allocation etc). Even if a node goes down (i.e. disconnects from the network due to leaving, shutting down or connecting to another network) from an existing MANET, the routes are rebuilt dynamically via another feasible node. Although it results in more delays and bandwidth fluctuations, the connectivity is not hampered for all the valuable routes built in the past [13], unless one or more nodes involved in the route are altered.

There are several disadvantages with MANETs because dynamic routes management via contributing nodes is a memory expensive process. MANET protocols like AODV (Ad hoc on demand distance routing vector) demand that route information has to be maintained across the network at any given point in time, keeping all nodes notified of the topology changes. This is done by frequent and periodic 'Hello' messages [14]. This can cost a significant increase in the administrative responsibilities of nodes. However, from a network designer perspective MANETs should add value to the usability of the overall network. Thus their configurations are usually done as part of Mesh networks and other places on requirements basis. For example, consider a remote place (say, Amazon jungle) with no communication or connectivity to infrastructure based network is possible due to certain geographical and budgetary constraints. MANETs can be an excellent choice to create a local network and then choosing a gateway node to relay information to rest of the globally connected internet (e.g. via a BTS in a GSM/UMTS/4G network making use of BTS to GGSN/SGSN connectivity). Consider living in a status quo which does not offer connectivity to internet, or the infrastructure has collapsed due to a disaster or earthquake; situations like these make MANETs a promising research field for these emergency situations. Connectivity and its importance increases many folds under these crucial circumstances.

### 2.3 Evolution to Opportunistic Networks

MANETs can operate by making use of several underlying routing protocols (namely AODV, DSR, OLSR, DSDV etc, for more details please refer to [14]) which operate not only in a reactive manner but also proactively. Our intention was to give a background by briefly describing two major classes of communication networks. After building this crucial knowledge we now turn our attention to yet another evolution of MANETs which directly relates to our research question.

Common literature in ad hoc communication networks mistakenly refer to the terms "Opportunistic Networks" (OppNets) and "Delay Tolerant Networks" (DTN) quite interchangeably [1]. However, it should be noted that DTNs usually refer to those networks where the connectivity between any two nodes is "episodic" or intermittent. They are broadly divided into DTN Regions and DTN Gateways, where the former can be just like any other legacy internets having their own internet stacks and internal protocols. The DTN Gateways provide the connectivity between the two (or more) contributing Regions involved in sporadic messages transmission. An example can be seen in the following diagram where a helicopter acts as gateway to provide connectivity in between two separate internets. The aircraft carrier is responsible for sending useful messages to the soldiers in the battlefield via the mediator entity, which serves as a translator by configuring an overlay protocol layer, bridging the stack differences between the end regions. As can be seen, the topology of the overall network is known and not prone to dynamic changes in the routing tables.

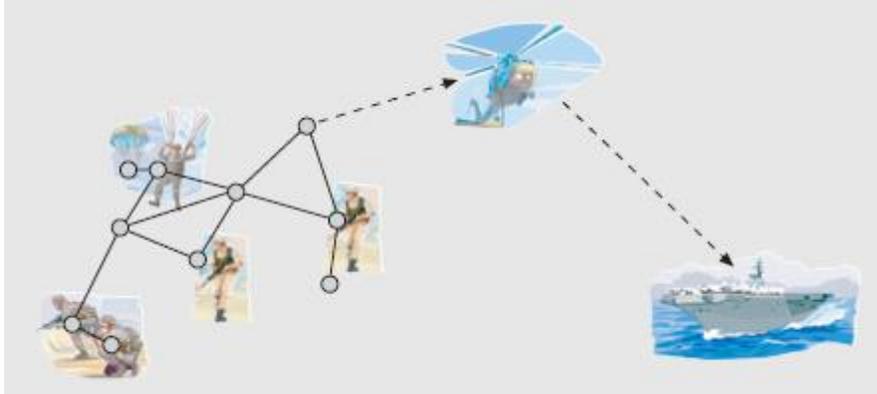


Figure 1 An example of DTN: Helicopter being a Gateway Node for two discrete internets

Opportunistic Networks, on the other hand are even a more generalized class of DTNs. There is absolutely no knowledge about the network topology in an OppNet. Every node acts as gateway, or more generally speaking, a router itself. Nodes make use of localization knowledge to find the potential 'opportunities' in order to relay messages towards the final destination. They are sometimes also referred to as 'Pocket Switched Networks' (contrasting the traditional packet switched networks). There are a large number of routing and message forwarding techniques (which are discussed in one of the related work section in this report).

As can be seen in Figure 2, the opportunistic networks primarily depend on encounters (known as contact times) between each other in order to optimally relay the messages. As discussed above, portability of human carried devices has been a great motivation for these networks. A modern device is usually designed with Bluetooth and 802.11 interfaces. Since wireless MAC is broadcast in nature, it proves handy in these networks.

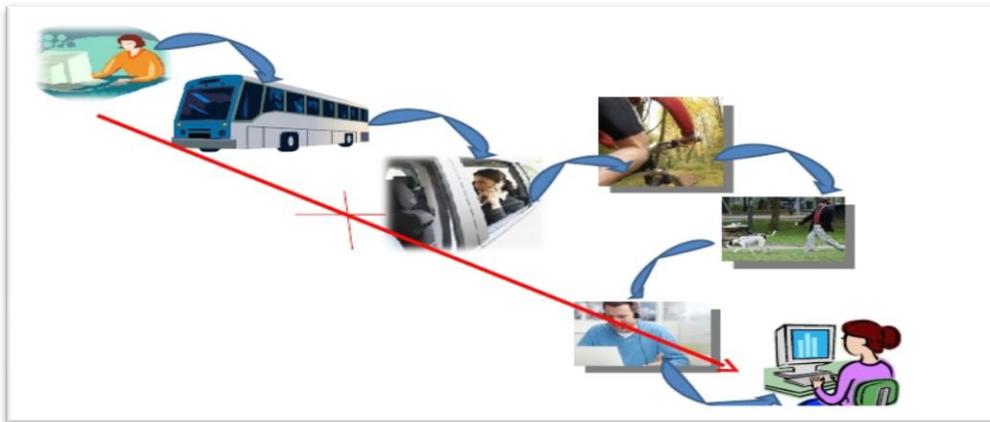


Figure 2 An Opportunistic Network Example: The red line indicates the possible delay i.e. latency introduced by each hop during message relay

One of the fundamental challenge in OppNets has been message forwarding right from their inception. What should be the most cost-effective forwarding protocol has largely been discussed in the initial research. Various routing schemes including but not limited to, multi copy, single copy paradigms have been proposed and discussed by researchers. However, the assumption of

unselfishness was always being made during the design and implementation of routing schemes. In actual, most of the humans carrying mobile devices are constrained by the limited energy available. Therefore, the notion of 'rogue' nodes (just like rogue Access Points (AP) in infrastructure based networks) is very much possible. In contrast to the fixed rogue APs, the rogue nodes in OppNets are mobile and can change their geographical locations quite dynamically as the network topology changes. This can have impact on the throughput, delivery ratio and latency of the network.

A network already challenged with the distinct and disconnected mobile nodes, and also vulnerable to rogue behavior demands much more effort to simulate its behavior in dynamic networking conditions. This challenge motivated and brought us to investigate and simulate *the impact of rogue nodes on the average latency in an epidemic routing protocol stack.*

### 3 Rogue Behavior in Opportunistic Networks

Altruism or Rogue Behavior is a widely displayed phenomenon in the human race beginning from the age of hunter gatherers to the current modern age. It has remained a subject of extensive research and one person or node can have different values with respect to altruism when compared to other nodes or people.

For the purpose of our research and simulation, it is assumed that altruism distribution is between 0% and 100% where 0 is completely rogue and 1 denotes completely altruistic node. However various dimensions of rogue or selfish behavior exist in the literature. In one of the studies on rogue and altruistic behavior in DTNs and Opportunistic Networks [15], various classes are defined:

**Percentage of Rogue Nodes:** The percentage of rogue nodes can be between 0% and 100% however in reality a node is not completely altruistic or selfish. This is the simplest altruism distribution.

**Uniform Distribution:** There is a uniform distribution of altruistic value in the entire population between 0 and 1. Such a distribution exists commonly in natural environment.

**Normal Distribution:** The pattern of altruistic values in the population has a normal distribution with values ranging between 0 and 1. 5% and 95% Distribution Function (CDF) has also been used in order to normalize the distribution.

**Geometric Distribution:** This approach calculates altruistic values on per pair of nodes and the probability has an inverse relationship with social hop distance ( $k$ ). The altruism values have been normalized against the parameter  $p$  (altruism value for the first hop). However in real life scenarios there is another factor to consider; which is decrease in altruism with decrease in kinship or distance from first hop.

**Degree-biased Distribution:** This approach attempts to show the relationship between altruism and node degree.

$$a_i = (k_i - k_{\min})^\alpha / (k_{\max} - k_{\min})^\alpha \text{ with } \alpha > 0$$

Where  $k_{\min}$  is the smallest while  $k_{\max}$  is the largest degree in the network. When  $k_i$  is equal to  $k_{\min}$  then value of  $a_i$  is zero and when  $k_i$  is equal to  $k_{\max}$  value of  $a_i$  is 1. When the value of  $\alpha$  is 1 then value of  $a_i$  grows linearly with degree. The model illustrates the scenario that when someone is more willing to help other people then that person becomes more popular on a social network. However because of resource limitations, the person may not be able to help all of them in reality. On the other hand if a person has only friend then there is a greater likelihood he/she will be willing to help that friends.

**Community-based Distribution:** According to this approach people will be more willing to help people from their own community rather than those from outside the community. Therefore this mode includes two variables;  $a$  to account for intra-community altruism and  $e$  to account for and inter-community altruism. It also assumes 0.7 probability for carrying data for intra-community and 0.1 probability for carrying data for intercommunity.

For our simulation and experiments, we will consider rogue behavior with respect to the percentage varying between 0 to 100%, and when the nodes acts rogue, it will deny forwarding the message from its neighbors, but will try to send its own messages always.

## 4 Epidemic Router [6]

In the experimental setup which is performed as part of this project, we have chosen Epidemic Router operating at layer 3 in the OSI model. The working of Epidemic router is thus essential to grasp to analyze the impacts on key metrics. Although the plug-in developed is extensible for any routing algorithm, in order to keep our evaluation closer to the analytical model in [10], we performed all the experiments and subsequent reports on Epidemic Router.

Existing protocols for ad hoc routing are based on the assumption that a connected path exist between source and destination which may not always be valid for example in the case of short range wireless networks. Epidemic routing uses pair wise message exchange as the mode for transmission. Maximizing the delivery rate of the message, minimizing latency, and minimizing resource consumption are the main aims of epidemic routing.

In epidemic routing approach the message is distributed to hosts called carriers. When carries come into contact with the connected portions of the network, the message gets transmitted and gets transferred from one island of nodes to another. Figure 3 Shows two scenario, where S,C and D are nodes and dotted circle shows the wireless communication range. Source transmits the message to the nodes that are in its communication range.

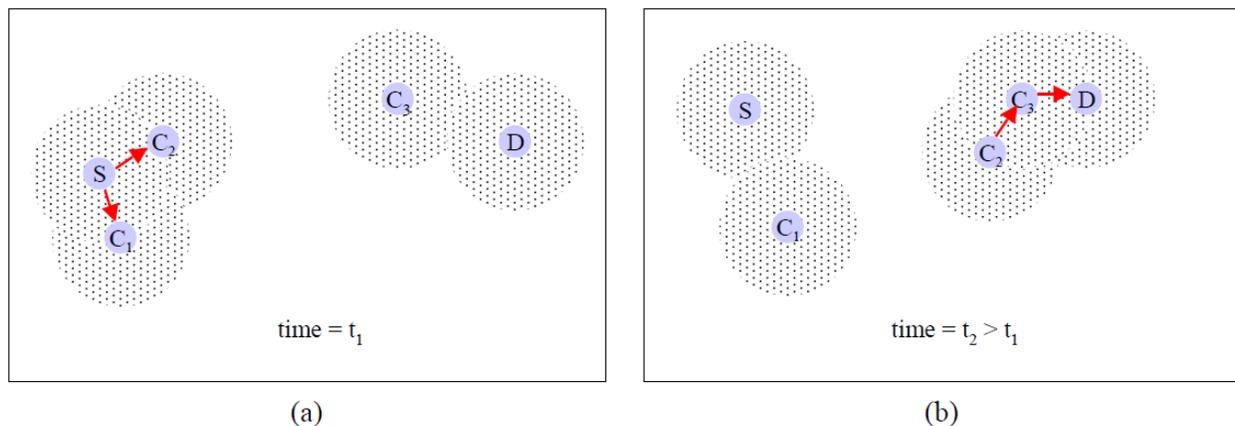


Figure 3 Epidemic Routing

The purpose of epidemic routing is to increase the probability of message delivery to a host and conserve resources by minimizing set of hosts that can transmit the message. These goals are achieved by imposing a limit on maximum number of hops and per node buffer space.

### Goals and Design Issues of Epidemic Router

For epidemic routing, the following design issues need to be considered.

- **Routing Under Uncertainty:** since the location of the receiver is not known, it is important to transmit the message when the host node comes into contact with the potential carries.
- **Resource Allocation:** Multiple copies of the message are transmitted and there should be a balance between resource consumption and message delivery.

- **Performance:** Performance can be measured in terms of bandwidth consumption, storage, latency and message delivery.
- **Reliability:** Some applications may ask for an acknowledgement of successful message delivery so that the transmitting nodes can free their resources if message delivery is successful.
- **Security:** A message may pass various routes before reaching its destination and hence it is important that sensitivity of information is taken care of and certain cryptographic techniques can be used to ensure this to some extent.

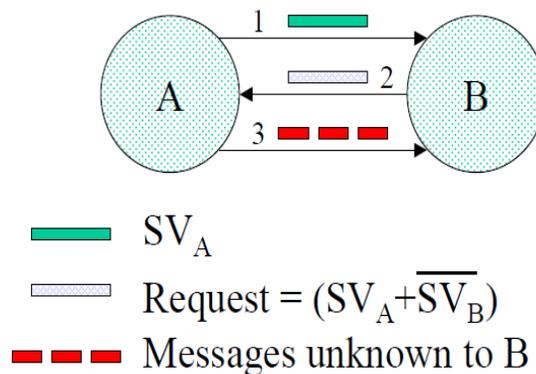


Figure 4 Message Transmission in Epidemic Router

Figure 4 shows how message transmission takes place in epidemic routing when two hosts come in transmission range. Each node maintains a buffer of the messages it has transmitted and some of the received ones as well. A unique identifier is attached with each message and listed in a hash table index. A bit vector called the summary vector is used to indicate which entries in the local hash tables are set. An anti-entropy session is triggered by a host that has a smaller identifier when two hosts come in contact. The exchange vector lists to compare which message is new to the host and such messages are exchanged while the receiving host is autonomous in making the decision to accept the message. In order to aid in this decision a maximum queue size is developed for each host and determines the maximum number of messages the host can hold.

In the initial study the researchers employed three features namely a unique message identifier (unique 32-bit number), hop count and an optional ack request. The message identifier is a concatenation of the ID of the host and ID of the message generated and hosts have been assigned IDs statically. Hop count is the number of maximum hops for the message. Larger hop counts help in delivering the messages quickly and important message can be assigned a higher hop count. The ack request is used in instances when the destination host is required to provide an acknowledgement of receipt. When a host reaches its buffer capacity then it drops the older messages for receiving new ones. The simple strategy of first in first out (FIFO) is used here but it compromises on fairness and quality of service. Another strategy called Weighted Fair Queuing can also be investigated as an alternative.

## 5 Related Work

### 5.1 Survey of DTN Protocols [2]

As discussed above, routing had been the first and foremost challenge in OppNets and even in DTNs. In this vast survey, the authors give a comprehensive classification of routing protocols which can be deployed in both environments.

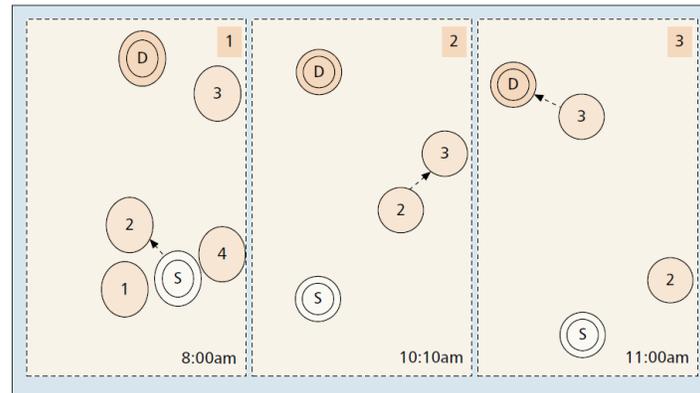


Figure 5 Time evolving behavior of ad-hoc networks

Figure 5 shows the lack of a direct path between nodes S and D and the message can be delivered only if the nodes in between are able to forward the message. Examples include inter-planet satellite communication networks, sensor networks and military and ad-hoc networks where the transmission takes place intermittently. DTNs are able to tolerate delays beyond conventional IP forwarding delays and have various applications. Inter-planetary network (IPN) for example involves intermittent connectivity and withstands long delays. Zebra population information is collected using Zebranet. An example of village network is where the rural busses are being used to provide connectivity to internet.

DTN research group has presented its findings and proposed architecture for DTNs that aims to address the associated communication issues in such networks. This architecture consists of regional networks and an overlay of transport layer on top. This architecture is able to provide retransmission, in-network data storage and forwarding. The aim of this architecture is to provide a solution that can deliver the message between dissimilar infrastructures. The basis of the DTN architecture is the store and forward mechanism and the issue of late binding addresses is covered using an addressing scheme. This results in a routing structure that is hierarchical in nature and because of this the implementation across networks becomes easier. DTNs can be deterministic or stochastic and based on the type different routing protocols are required.

Since the connectivity is intermittent, some destination nodes may not be available for certain duration. Store-carry-forward routing is used in order to insure that when a packet arrives but its destination is not available it is not dropped but stored instead till the next hop becomes available. However when a node buffers data it is also important to select the next hop properly. The DTN routing protocols can be categorized as deterministic or stochastic. Deterministic protocols are based on the assumption that future typology of the network is known while the stochastic approach assumes the opposite.

### 5.1.1 Deterministic Routing

Space and time routing, tree approach and modified shortest path approaches are deterministic routing models. According to the tree approach, the source host builds a tree where by children nodes are added and time is associated with nodes. Every node is able to record certain information including the node that the message is travelling to and the earliest possible time to do so. Through this tree, it is easy to identify a path which will take the shortest time and minimum number of hops to deliver the message to the destination. Building this tree requires that nodes exchange the profile information with their neighbors. Another essential requirement is that the nodes are able to record the sequence in which the message has travelled.

Based on the available knowledge regarding a network, various DTN routing algorithms have been proposed which define knowledge oracles

| Contacts Oracle | Summary               |  |
|-----------------|-----------------------|--|
|                 |                       | <ul style="list-style-type: none"><li>• Aggregate statistics of the contacts</li><li>• Time-invariant information</li></ul>      |
|                 | Contacts Oracle       | <ul style="list-style-type: none"><li>• Contact between two nodes at any point in time</li><li>• Time-varying networks</li></ul> |
|                 | Queuing Oracle        | <ul style="list-style-type: none"><li>• Instantaneous buffer occupancies</li></ul>   |
|                 | Traffic Demand Oracle | <ul style="list-style-type: none"><li>• Present or future traffic demand</li></ul>   |

Depending on the available information, the dynamics of the networks are modeled as a space-time graph and using the shortest path algorithm and dynamic programming, routing algorithms are constructed. Under these deterministic approaches, the entire path from start to end is determined even before the actual message transmission.

### 5.1.2 Stochastic Or Dynamic Routing

When the network behavior is random the decision to forward or drop a message is made using history and mobility data of nodes and uses various dynamic protocols. As also discussed in the previous section, in epidemic routing this decision to forward a message is made without using forwarding probability and the message is flooded into the network can be forwarded to any node except the one from which it was received. Another approach is where the source node holds the message till it comes into communication range with another node. This approach results in low overhead but long delays. 2-hop forwarding approach and have explored a

In the two hop framework the nodes have an infinite buffer and are able to move across the network independently and for a certain time slot two nodes can get close. The message from one node can be forwarded to any other receiver node that is one hop away. When this receiver node comes near the destination node it sends the message. In this method the message is delivered in two hops which may lead to delays but message delivery is ensured.

In Infostation model the infostations are distributed in the network coverage area and the users are able to connect to the network when they are near the Infostations which provide high quality radio signals however the connectivity is also intermittent as the destination node may be outside the coverage of the infestation. The Shared Wireless Infostation Model (SWIM) combines infostation and epidemic methods enabling many Infostations to serve as a destination nodes. However there is a tradeoff between capacity and delay.

The relay-based approach to makes use of the traditional protocols and mobility of the nodes is used to disseminate messages through the mobile relay protocol. In this protocol storage capabilities are integrated with routing when a destination node is unavailable the node relays to its neighbors who store it and enter relaying mode in which traditional routing protocols are also checked for a shorter hop. If no shorter path is found then it is stored instead.

In an opposite approach, the nodes estimate the probability of success of message reaching the destination and make the forwarding decision based on this knowledge. Per Contact Routing Based on Next Hop Information Only in which a bundle exchange takes place between two nodes along with their destinations and probability of delivery. Upon receiving this bundle the node removes its own messages and the messages that are more likely to be delivered by the other node and then categorizes these messages based on drop strategies.

Another approach is called PROPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity). The probability of delivery of each node is determined for each destination. The meeting nodes exchange summary vectors along with delivery predictability for destinations. In Context-Aware Routing (CAR) there is an integration of synchronous and asynchronous mechanisms. Under the synchronous mechanism when the packet arrives and the destination path can be found it is forwarded immediately using the existing protocol. In asynchronous method when the destination path for a packet is not there it is stored and then transmitted to a node which has the highest probability of delivering to destination.

In Interrogation-Based Relay Routing (IBRR) the nodes make decisions after they interrogate the other nodes regarding typology of the network and capacity of the nodes and forward message to the node that has the highest chances of delivering to destination.

Per Contact Routing Based on Average End-to-End Performance Metrics include protocols that base their decisions on the end to end performance including highest probability as well as the shortest path to destination and the delay in between. Meets and visits (MV) is based on a different mechanism to estimate forwarding probability by making use of the frequency at which nodes meet in certain regions and this historical data is used to determine chances of delivery for a specific path.

The shortest expected path routing (SEPR) protocol also uses historical data to determine probability of forwarding keeping in perspective the time period for which two nodes connect and then shortest route is determined. Similar approach is used in minimal estimated expected delay (MEED) routing however it is independent of time and uses history to calculate expected delay and tracks the connection and disconnection time of nodes which is relayed to other nodes if a change occurs.

### **5.1.3 Model-Based Approach**

All the research work discussed till this point estimates forwarding probability assuming that the mobile devices tend to move randomly and unknown trajectories. In practice mobile devices follow patterns that can be predicted to some extent for example while walking or driving. If the motion pattern is known then the intermediate nodes can more accurately predict forwarding probability. World models of mobile nodes are used in Model Based Routing (MBR) which helps in making a better selection in determining where the receiver node is located without flooding the network. Information such as road maps, building charts and user profiles are used to determine the motion pattern.

#### 5.1.4 Node Movement Control-Based Approaches

Certain approaches try to limit the delay by trying to control their trajectories. This involves altering the trajectories of the host so that communication can be facilitated in ad hoc networks. The mobile host is not waiting passively but actively modifying the trajectory. The message from one host to another is done by changing host trajectories. This is useful when a network partition comes up. Algorithms for trajectory modifications are based on two assumptions that all nodes are known but host movement is unknown. The algorithm helps in determining when location update should be shared by host, with whom it should be shared and how it should be sent. A minimum spanning tree (MST) contains full connectivity in the graph. And it's the responsibility of the host to update its location and inform the nodes connected in MST. In Virtual mobile nodes (VMN) it is assumed that an abstract node is moving in a predetermined and predictable path and during this movement, collects and delivers messages. A real node waits until the virtual node is in the vicinity and then transmits the message to the virtual node which collects it.

When there is a mismatch between available capacity and demand in a wireless networks the capacity can be increased by adding participants that can carry bundles and autonomous agents can be added to the network but it requires a control algorithm for coordinating the movement of agents. A control-based approach can be used to develop controllers for autonomous agents. Latency, bundle latency, unique bandwidth, and bandwidth are used as control mechanisms. A proactive and mobility assisted approach for message delivery in sparse networks in Message Ferrying (MF). Communication services are provided using special mobile nodes called message ferries which move in the deployment area and carry messages to and from nodes. A variation of message ferrying is Node-Initiated MF (NIMF) where the ferries follow specific routes. A service request is generated to the ferry when a node wants to send or receive a packet using long range radio. The ferry adjusts its trajectory to for the exchange using short-range radios. This minimizes message delay and increases system throughput and robustness. However designing the ferry route is a challenge and synchronization is needed when there are multiple ferries. There is a snake protocol as well that uses a snake-like carrier sequence in which the virtual nodes always remain adjacent, paired and move in a pre-determined way and cover a certain area in the network.

In the runners protocol the carriers randomly walk the network and sweep it for possibilities of message exchange. A three tiered architecture is used in DataMules where spare sensors are connected. Access points exist in top tier which can be used as repositories. DataMules are in middle tier which are mobile nodes with unknown mobility patterns. These have high capacity to store information and collect data from sensors which are in the bottom tier.

#### 5.1.5 Coding Based Approaches

Erasure and network coding are recent developments with a purpose to encode the original message into a large number of coding blocks. If the original message contains  $k$  blocks the message will be coded into  $n$  blocks where  $n > k$ . replication factor determines the redundancy ( $r = n/k$ ). The probability of successful transmission link ( $P_i$ ) is known. The blocks are sent over path  $i$  in order to maximize probability of reception. The coded blocks are equally distributed among the relays. It simply forwards the packets to the first "contact" the node encounters (all contacts are equally good relays).

Erasure coding and estimation-based forwarding can be combined for optimization. The erasure coding is used to code the original messages which are then forwarded to relays. A probabilistic forwarding approach based on network can be used in the case of DTNs where the intermediate nodes have the ability to combine packets that it has received so far and combine and send out as a new packet. This reduces the number of transmissions.

## 5.2 On effects of Cooperation in DTNs [8]

The purpose of this paper is to evaluate the effect of three major routing algorithms on node cooperation and examine the delay in message delivery and the resulting overhead incurred till the message reaches its destination. Most of the studies that have explored DTNs have focused more on the environment such as the size of the area the network covers. Behavior of nodes has rarely been studied and often based on the unrealistic assumption that there is full cooperation between the nodes. In reality the autonomous decision of the nodes is a major factor in determining the performance of the DTN. The purpose of the paper is to examine the behavior of the nodes in terms of cooperation when different routing algorithms are deployed, ways in which non-cooperation between nodes can be detected and strategies to overcome this. The study has been conducted under the framework of peer to peer and ad-hoc networks.

In order to improve cooperation, punishment mechanisms have been used which is a manifestation of game theoretic approach whereby depending on each node's behavior; different incentives are provided to increase cooperation. In addition, advanced punishment methods are used in the form of reputation mechanisms which monitor and record the actions of the nodes and refer to these actions to handle the node. The study is carried out assuming that the environment is non-cooperative.

Three routing mechanisms are studied. These mechanisms range from conservative scheme where message copies are spread in the network solely by the source node to a fully aggressive scheme in which the network is flooded with message copies. Transmission is measured until the point the actual message stops spreading in the network and cooperation is measured through the likelihood that a node will drop a message upon reception or forward it to another node.

The three multi copy algorithms used in the study are epidemic; two-hop and binary spray and wait.

| Routing Algorithm            | Description   |
|------------------------------|---|
| <b>Epidemic</b>              | <ul style="list-style-type: none"> <li>• Message copies are exchanged whenever two nodes encounter</li> <li>• Provides minimum delay in message delivery</li> <li>• Plagued by high utilization of bandwidth and buffer occupancy</li> </ul>  |
| <b>Two-Hop</b>               | <ul style="list-style-type: none"> <li>• Maximum limit on the number of copies the source and spread.</li> <li>• Whenever it encounters a node which does not have a copy, it transfers a copy up till the point it has only one left.</li> <li>• There is a restriction on intermediate nodes to transfer their copies to any other node except the destination node.</li> </ul> |
| <b>Binary Spray and Wait</b> | <ul style="list-style-type: none"> <li>• Each node transfers half of the copies it has to other nodes upon interaction till only one copy is left.</li> </ul>   |

- Faster than two hop algorithm.

A node that is aware of message delivery is termed as a notifier node and when this node encounters an unaware node that possesses a message copy, the unaware node will become a notifier and drop the message. There is always a tradeoff between the delay in message delivery and overhead incurred. It is misleading to equate overhead with the number of transmissions till the message is delivered and hence the following components are covered in this research:

|                            |  |
|----------------------------|--|
| <b>Overhead Components</b> | <b>Till Delivery:</b> Number of transmissions till the message is delivered to destination |
|                            | <b>Additional:</b> Number of transmissions till the message is delivered to destination    |
|                            | <b>Total:</b> Sum of till delivery and additional  |

The willingness of the node to spread a message is called cooperation and depends on various resource and buffer constraints. In some cases a node drops a message while in others it may keep the message without forwarding it. In Type I node behaviour, the message is dropped (probability  $P_{drop}$ ) or forwarded as per rules of the algorithm. The message is usually dropped because of buffer limitations. In Type II node behavior, although the message is forwarded, the probability is less than one (probability  $P_{forward}$ ). This usually occurs because the node is facing energy constraints. However the lack of cooperation may actually be a strategy that all the nodes have adapted because of the constraints.

In order to quantify how the performance of algorithm is effected by cooperation, two metrics have been used. The performance of the algorithm in a fully cooperative environment is compared to performance with a certain degree of cooperation. The more deviation between the performance in the two environments, the more sensitive the algorithm is. The other metric is used for Type I where the performance is compared for a certain degree of cooperation in the fully cooperative equivalent (FCE). The study was conducted in an 8km x 8km area with 100 uniformly distributed nodes.

The performance of the three algorithms is shown in Figure 1 and 2. Minimum delivery delay can be seen in epidemic routing however compromises on transmissions. In a fully cooperative environment the delivery delay is low in binary spray and wait. However below 0.75 cooperation degree, the delay performance of two hop is better than that of spray and wait. In less cooperative environments the total overhead included for binary spray and wait decreases while

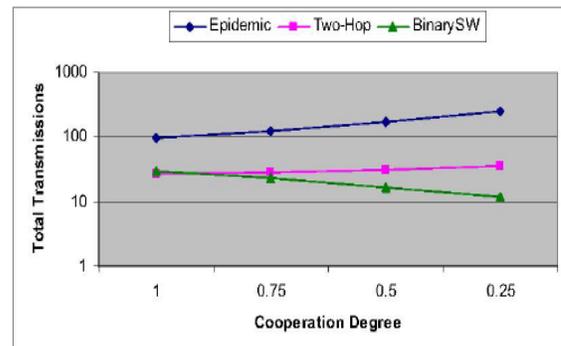
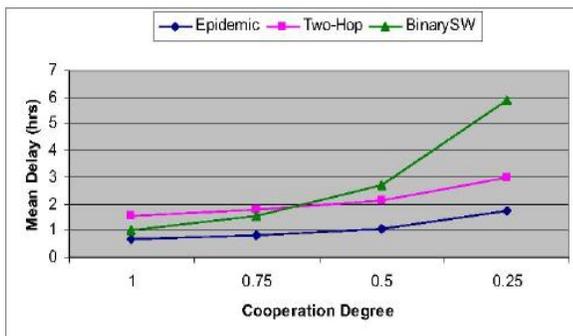


Figure 1. Mean delivery delay as a function of Type I cooperation degree. Figure 2. Total transmissions (in log scale) as a function of Type I cooperation

the exact opposite happens in two hop.

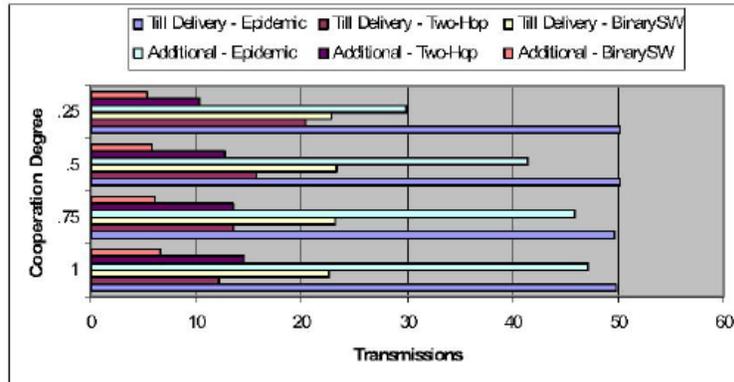


Figure 3. Distribution of the overhead into its components.

In figure 4, the graph shows the cumulative distribution function for all three algorithms with a cooperation degree of 0.5. Two hop algorithm has a smaller delivery delay while the binary spray and wait out performs it when the interval is less then 3 hours. The results reveal that the comparative performance of the two algorithms is different on the dimension of mean delay.

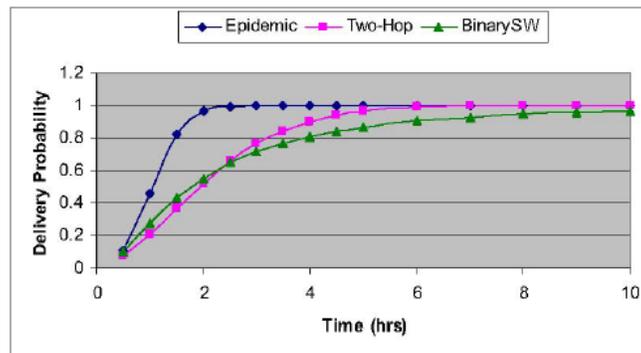


Figure 4. Cumulative distribution function (*cdf*).  
(Type I cooperation degree of 0.5).

Figures 5 and 6 represent the sensitivity of the algorithms to cooperation. As far as the total transmission is concerned, epidemic is the most sensitive algorithm. For mean delivery delay, spray and wait is the most sensitive. Two hop method was least sensitive in both metrics.

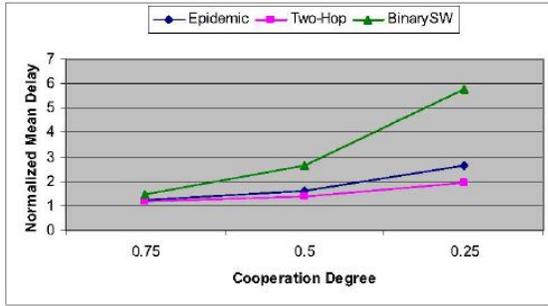


Figure 5. Normalized mean delay (wrt that in the case of a cooperation degree equal to 1) as a function of Type I cooperation degree.

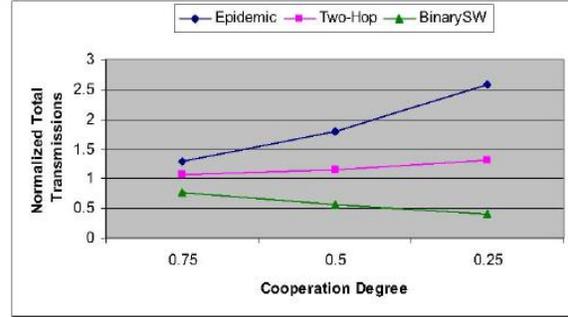
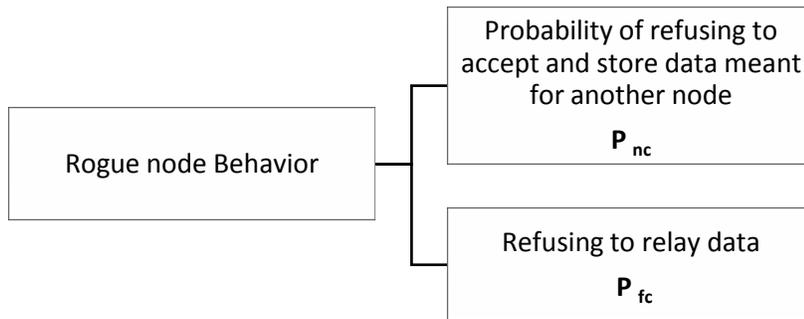


Figure 6. Normalized total number of transmissions (wrt that in the case of a cooperation degree equal to 1) as a function of Type I cooperation degree.

### 5.3 Analytical Modeling to determine the impact of selfish or rogue behavior [10]

This paper was primarily our reference and benchmark for the implementation and simulation performed. The purpose of explaining this paper is to study the impact of rogue node behavior on data transfer for two popular schemes namely “unrestricted” and “two-hop”. This study aims to reveal how vulnerable these schemes are when it comes down to node selfishness. Although we only worked on portion of the results drawn, our implementation is extensible for future work in the same direction.

Based on the node’s concern for storage space and energy consumption, selfishness has been defined as follows for the purpose of this paper:



Rogue Behavior may be demonstrated deterministically or probabilistically by all network nodes or a certain subset of the nodes.

Research design of the paper is such that assuming rogue nodes; the data transmission process has been modeled as absorbing two-dimensional Continuous Time Markov Chain (CTMC). This model is then used to study delay in message delivery. The performance level assuming full cooperation and that assuming rogue nodes is then compared to arrive at a metric for performance deterioration which has been termed as deceleration factor.



### 5.3.2 Message Delivery Under Unrestricted Relaying

According to the unrestricted relay method, message replication is not restricted. When the nodes are fully cooperative unrestricted relaying shows least delay but at the cost of high resource consumption. These transition states (non-zero) are as follows:

$$\left\{ \begin{array}{l} T_{ur}\{n+1, k|n, k\} = (n-k \cdot p_{nf})(N-n-K+k)\lambda \\ \quad \text{for } k \in [0, K], n \in [1, N-1], n-k \in [1, N-K-1] \\ T_{ur}\{n+1, k+1|n, k\} = (n-k \cdot p_{nf})(K-k) \cdot (1-p_{nc})\lambda \\ \quad \text{for } k \in [0, K-1], n \in [1, N-1], n-k \in [1, N-K] \\ T_{ur}\{n, k|n, k\} = -\sum_{\{p, q|p \neq n \parallel q \neq k\}} (T\{p, q|n, k\} + R\{p, q|n, k\}) \\ \quad \text{for } k \in [0, K], n \in [1, N], n-k \in [1, N-K] \end{array} \right.$$

Transition states for column vector are as follows:

$$R_{ur}\{D|n, k\} = (n-k \cdot p_{nf})\lambda \quad k \in [0, K], n \in [1, N], n-k \in [1, N-K]$$

### 5.3.3 Message Delivery Under Two-Hop Relaying

In the two-hop method a node can receive a message from a source and relay the message to a destination node only once. This method aims to optimize message delivery and resource consumption trade off. These transition states (non-zero) for two-hop method are as follows:

$$\left\{ \begin{array}{l} T_{2hr}\{n+1, k|n, k\} = (N-n-K+k)\lambda \\ \quad \text{for } k \in [0, K], n \in [1, N-1], n-k \in [1, N-K-1] \\ T_{2hr}\{n+1, k+1|n, k\} = (K-k)(1-p_{nc})\lambda \\ \quad \text{for } k \in [0, K), n \in [1, N-1], n-k \in [1, N-K] \\ T_{2hr}\{n, k|n, k\} = -\sum_{\{p, q|p \neq n \parallel q \neq k\}} (T\{p, q|n, k\} + R\{p, q|n, k\}) \\ \quad \text{for } k \in [0, K], n \in [1, N], n-k \in [1, N-K] \end{array} \right.$$

### 5.3.4 Numerical Results

The deceleration factor ( $F_D(N, K)$ ) is used to quantify the decrease in the performance of the DTN when nodes exhibit rogue behavior. This factor shows the ratio of expected delay in message delivery when  $K$  number of selfish nodes exists in the network as opposed to when the nodes are fully cooperative.

$$F_D(N, K) = \frac{\overline{D}(N, K)}{\overline{D}(N, 0)}.$$

Delay in message delivery for the two schemes is quantified as follows:

$$\overline{D}_{ur}(N, 0) = \frac{1}{\lambda N} \sum_{i=1}^N \frac{1}{i} \quad \overline{D}_{2hr}(N, 0) = \frac{1}{\lambda} \sum_{i=1}^N \frac{(N-1)!}{(N-i)N^i}.$$

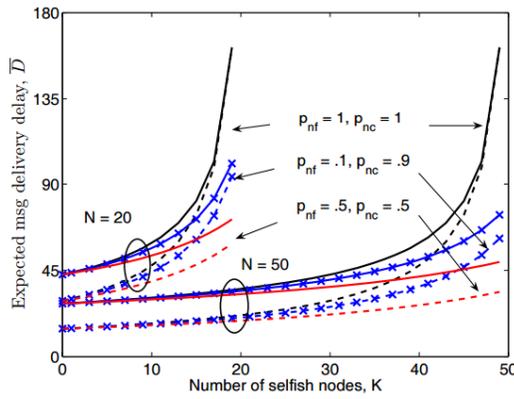


Figure2 (a): Expected message delivery delay

Figure 2(a) shows that the unrestricted relay method performs better than the two-hop because the data transmission scheme is quite aggressive. The gap between performance decreases as intensity and number of rogue nodes increases and disappears when all nodes are rogue.

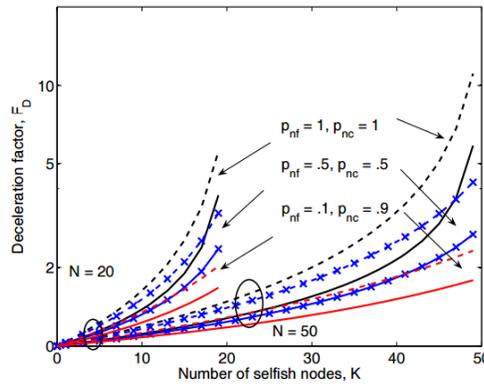


Figure2 (b): Deceleration factor

Figure 2(b) shows the degradation in performance by plotting the deceleration factor and reveals that unrestricted relay is more vulnerable to rogue nodes because performance degradation occurs faster. However the fact that both methods are quite tolerant to degradation cannot be ignored.

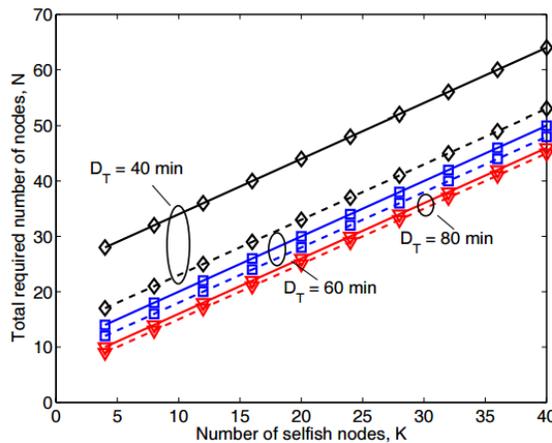


Figure3: Required number of network relay nodes vs. number of deterministically rogue nodes

The last part focuses on remediation actions for which this model can be used. The degradation in performance can be recompensed to some extent by increasing the range of transmission and velocity of the nodes. Another method can be to introduce robotic nodes which controllable mobility patterns. Figure 3 shows that even in the worst scenario where nodes are deterministically rogue, the performance can be preserved. When the limit for expected delay is made lenient, both methods require a comparable number of nodes.

Under normal conditions unrestricted method has a performance advantage when compared to two-hop however results reveal that this advantage decreases as the number of rogue nodes and the intensity of rogue behavior increases. In absolute terms, the unrestricted relay method outperforms two hop however in the presence of rogue behavior, its performance declines faster than that of two-hop. However both protocols displayed robustness against rogue behavior because the decline in performance was fairly slow.

## 6 Why ONE Simulator?

### 6.1 Shortcomings in Network Simulators for OppNets before ONE

After gaining the important background and crucial knowledge about Opportunistic Networks, fundamental to our research was the simulation of these networks. Opportunistic Networks are the most complicated ones to simulate owing to their inherent definition of geographical diversity, mode mobility and routing granularities. They are highly mobile and path formation in any OppNet is an uphill task. On top of that, simulating such a path or route requires considering several network parameters. Some of them include episodic connection, inter-contact time, message size, underlying protocol and fluctuating signal strengths. It is a known fact that we need to compromise on some of the parameters whenever we simulate such a network. Also, time delay in completion of overall simulation and visibility is quite challenging to produce. This can be frustrating and can lead to intuitional inferences if proper traces, GUI modeling and log reports are not readily available.

### 6.2 Routing Specific and OppNet Specific simulators

Since OppNets are an evolution of MANETs, initially ns2 [16], OMNeT++ [17], dtnsim [18] and dtnsim2 [19] provided functionalities such as routing, subsequent event simulation, mobility modeling and the required opportunistic networking support. One of the biggest challenge in OppNets has been the simulation of emerging protocols in order to research which protocol suits the best in these challenged networking conditions. These simulators either focus on routing or provide OppNet functionalities alone. A good balance between the required features of Event Generation, Modeling and Message passing has not been fully achieved using these tools.

Geographical movement of nodes for effective message traversal is at the heart of any OppNet. Those simulators which specifically consider random generation of interactions between nodes are impractical and unrealistic. Humans carrying devices or mobile Infrastructure equipped with radios have specific patterns in their movement. Be it a social gathering or normal day at work, the carriers follow a realistic path. The random generation of data for the optimal routing lacks this localization information.

A community resource, CRAWDAD [20] is utilized by the researchers to get the wireless traces which are based upon real experiments. In the recent past, their influx has increased ever since OppNets have proved their importance as the next generation networks. The versions of dtnsim and dtnsim2 provide a good support for OppNet routing protocols and support to import the traces. They somehow narrow down the gap which exists in ns2 for movement models, but the exact geography of contributing nodes, their interaction times (usually known as contact times) are still based on approximation. For example, the traces gathered via the interface of the mobile device can validate the Access Point (AP) being shared by the nodes (by the location/cell ID in GSM or 802.11 Base station identifiers). This can approximate somehow the locations of nodes but the exact X and Y coordinates up to the required granularities are still ambiguous.

### 6.3 Issues with real world traces import for legacy simulators

Another issue with import of real world traces into legacy simulators are their deficiency of length contact intervals. Because such a measurement requires physical involvement of skilled researchers or a trained group. The energy constraints tagged with the mobile devices only fulfill the contact instant during the scanning part of experiment, the exact duration is usually

constrained owing to the limited battery power. Now this has a stern impact on OppNet simulations because the ad hoc mobility of nodes always varies, and thus the actual realization of inter-contact time cannot be accurately calculated. Every nodes' interface has a specific range (e.g. 10m for a 802.15 Bluetooth, 10-100m for 802.11) and their precise contact times are needed to simulate the connection establishment issues, message start/stop times, message drop rate, success or failure and the required acknowledgment between the immediate nodes. Another limitation in the import of external traces is their scarce count of subjects (i.e. people doing experiments) involved in the measurement process. They are unable to depict or portray the real life behavior (most importantly for our research question, the rogue behavior) and thus do not represent actual communication and mobility paradigm. The researchers are usually proactive with pre-defined routes and artificial contacts which fail to expose the loopholes and further research challenges in OppNets.

#### 6.4 ONE Simulator with Mobility, Visibility and OppNet Support

ONE simulator has emerged as a promising research tool in recent past [21], in order to cater for the aforementioned tribulations in the existing MANET simulators. We can very briefly summarized the core functionalities of the simulator using the three terms: Mobility, Visibility and OppNet Support. They signify the tool's feasibility not only to the revolutionary OppNets and DTNs, but also address our research question. However, as we describe in the next section, there are difficulties with regards to rogue plug-in development in ONE, due to lack of extensibility and support for malicious behaviors. Nevertheless, to the best of our knowledge this tool is by far the most credible and widely used for research and analysis in the domain of DTNs and OppNets.

The ONE simulator addresses the challenges posed in the research and simulation of OppNets. The architecture and overview of main system components are described in Figure 6.

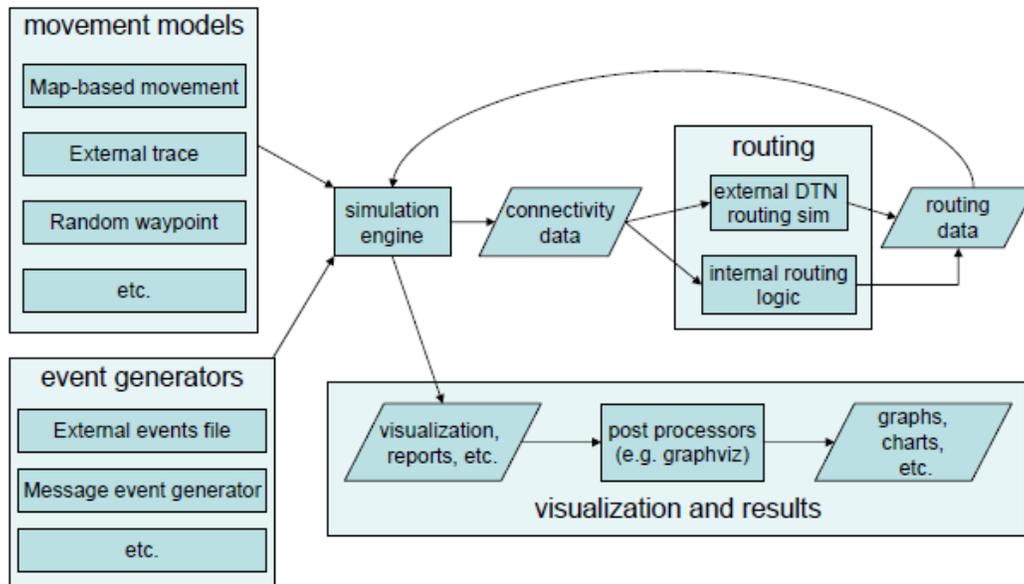


Figure 6 Architecture and Integration of ONE Core Components [21]

As can be seen, ONE simulator provides an integrated environment for a range of Movement Models. These movement models can be simulated based upon the coordinates of any underlying map (E.g. we developed our plug-in and performed simulations over the map of Helsinki, Finland). Apart of that, the mobility data can be imported from external traces and routing protocols can be applied on these movement models (Figure 6). Another configurable option in the simulator is to not only perform the routing algorithms internally for message forwarding, but by virtue of External Event Generators, the message passing information can be imported from traces. This flexible combination of Modeling and Event Generation can then be fed to the simulation engine, and visually observed in rich GUI available during the entire simulation period. At the end of simulation, a number of reports can be extracted and be post-processed for graphs or charts.

## 7 Methodology

### 7.1 Higher Level Approach

The following flowchart (Figure 7) describes, at a higher level, the approach we followed for developing the plug-in in order to simulate and analyze the Rogue Behavior. Our work started by exploring The ONE Simulator's technical workflows, process, tool and reporting functionalities. After considerable experience with the simulator we setup the environment on Windows/Eclipse IDE and then identified key packages that needed modifications for rogue behavior, along with the customizations needed for validating the analytical results on average latency in [10]. After the required implementation and verification of changes, we conducted set of experiments, each run with 100 seeds and the corresponding rogue degree of nodes. Finally, the dataset was analyzed and we deduced results and possible future work in this area.

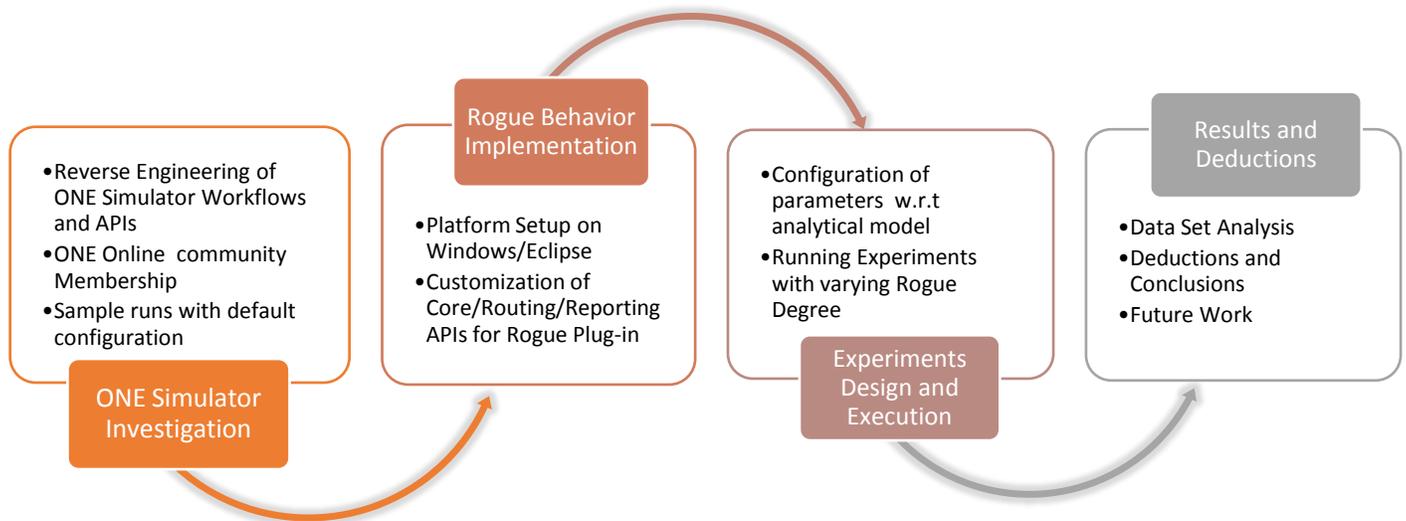


Figure 7 Higher Level Approach for the Rogue Plug-in Development

Our experiments were designed in order to evaluate the mathematical modeling done in [10]. Although not all parameters could be configured as per equations formulated by the researcher in that paper. We tried to bring our implementation and behavior as close to the pre-requisites, keeping theirs as the benchmark. Our implementation was only a subset of their model based on Poisson Distribution of nodes (as discussed in detail in Section 5.3). Also we did a few enhancements in the implementation w.r.t Map Based Mobility Model rather than the trivial Random Waypoint model which is hardly feasible in real world OppNets.

### 7.2 Plug-in for Mobile Rogue Nodes in ONE Simulator

We have used ONE simulator v1.5.1-RC2 [4] in order to develop the required plug-in for rogue nodes induction, and their varying rogue degree. The ONE simulators' original design architecture, class relationships, native APIs or interfaces do not provide any functionality with respect to the maliciousness which could be extended. Therefore it was crucial to reverse engineer and then develop the required plug-in to simulate the rogue behavior. This involved

investigating and debugging plenty of Java classes, configuration files, reports and method invocations and their respective executions, which was challenging mainly because of two reasons:

- ONE is an open source simulator developed in Java (1.6) and any support with respect to APIs, integration or extension by plug-ins is purely based on voluntary basis via an on-line mailing group
- Basic documentation is provided along-with the simulator, but the workflows and execution path supervised by the simulation engine, between the routing protocols, movement models and reporting artifacts can be realized by rigorous debugging.

Right from the inception of this project, we were strongly engaged with the on-line forum for online collaboration over the queries. The development of rogue behavior is rather a new research field since most of the researchers tend to work and investigate over routing and forwarding techniques in OppNets.

### **7.3 Metric for Observation**

The metric chosen in our research question is Average Latency. Latency is usually defined as the amount of time it takes a message to traverse from the source to the destination. Opportunistic Networks are disconnected networks. The delays occurring during the transmissions and routing of messages can be much more than the legacy networks, which is a known fact. However on top of that, how the rogue nodes further deteriorate the performance of the network is our main motive to investigate. The delays occurring may or may not be as drastic as opposed to the infrastructure based networks, reason being the inherent mobility factor in opportunistic networks.

### **7.4 Rogue Plug-in Implementation**

ONE simulator provides an extensible software architecture for new routing protocols, forwarding techniques and reporting functionalities. However, its main design aim was not to provide extensions with respect to development or research over any malevolent activities by the contributing nodes in an OppNet Environment. Figure 8 shows the package diagram and the interactions between the major Java packages in ONE simulator.

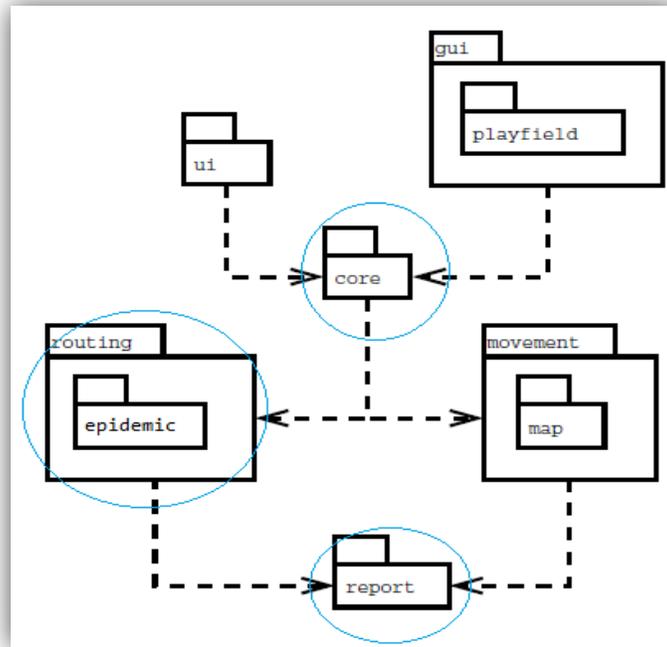
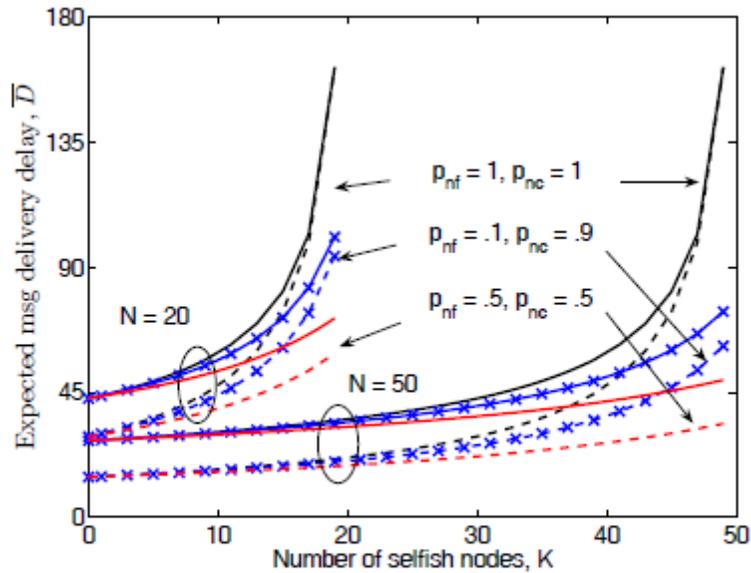


Figure 8 Rogue Plug-in impacted packages (Highlighted blue)

The packages highlighted with circles are the ones needed customizations with rogue logic and configurations of parameters for the APIs. We can discuss these one by one:

#### 7.4.1 Rogue Behavior in ONE: Implementation of subset of Analytical Model

In order to compliment the analytical modeling done in [10] into the ONE simulation, we kept the simulation parameters as close to the mathematical model as possible. As explained in Section 5.3, the below diagram shows the number of selfish nodes,  $K$ , versus the expected latency denoted by  $D$ . The authors in their modeling have made use of Epidemic as well as two hop relay protocol, varying the total nodes from 20 to 50. The degree of rogue nodes was simulated with two probability values namely  $P_{nc}$  and  $P_{nf}$ . Due to the scope and timelines of our project, we have simulated Epidemic protocol, with  $P_{nc}$  varying between 0 to 1. Therefore the dotted black line in the graph was intended behavior to be simulated and compared. But it should be noted that since the implementation of our plug-in is completely generic, the two hop relay protocol can be a small extension for future work and comparison between the two protocols.



(a) Expected message delivery delay.

#### 7.4.2 Routing implementation for Rogue Nodes

*MessageRouter* is the parent class for all kind of routers (e.g. Epidemic, Direct Delivery, Spray and wait etc) in ONE. This abstract super class provides all the basic methods needed in the routing functions of the simulator. Some of the important ones are to establish a connection between any nodes, send or receive message, provide buffer information, creating new messages, deleting messages and providing routing information. A complete list of methods provided by this class are provided in Appendix B. Another abstract class *ActiveRouter* provides all the utility functions needed for the current router active in the simulation. All kind of routers can override the methods from this class and must extend this router if they are to implement any routing scheme. The hierarchy between *MessageRouter* and *ActiveRouter* exists in the diagram give below. Passive Routers are usually utilized for external event traces and unless they are being invoked with relevant event traces they act as dummy throughout the simulation.

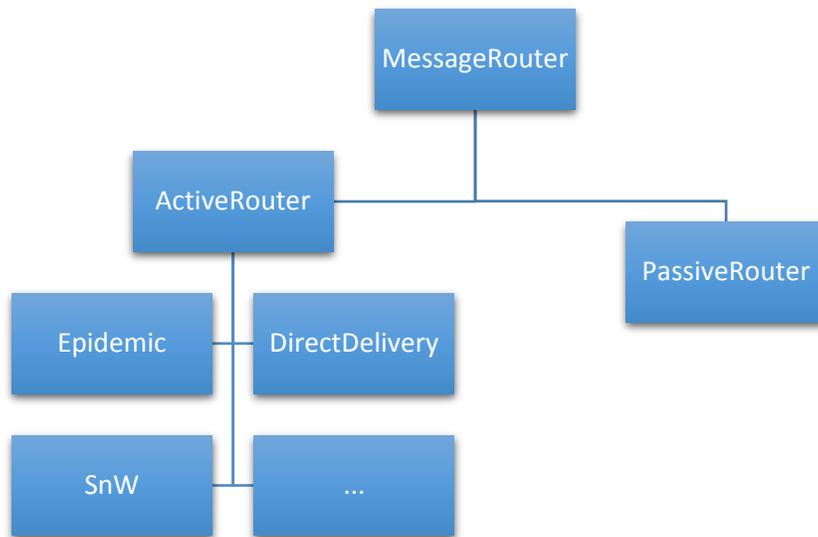


Figure 9 Hierarchy of Message Routers in ONE

In order to introduce the rogue behavior, a configurable and efficient algorithm was required so that all the routers can utilize this functionality. The *receiveMessage(Message m, Host from)* was overridden with the required logic and can be described with the following pseudo code:

*On every invocation of the message receipt:*

*Check if the rogue behavior is enabled in Input Configuration File,*

*If NO then*

*Put the message in the buffer and receive the message as per the protocol*

*If YES then*

*If the Router is willing to relay or store as per Pseudorandom Generator*

*if YES then continue receiving normally*

*If NO then check if the message was intended for the router itself*

*if YES then put the message in the incoming buffer*

*If NO then deny the message as rogue with probability P (0-100)*

### 7.4.3 Core Implementation for Rogue Nodes

#### 7.4.3.1 Rogue Flag and Rogue Degree

It was mandatory to change the core design of the simulator in order to create rogue nodes along with the required degree of selfishness. We achieved this by creating a two new parameter in the configuration file namely:

- Rogue Behavior Flag  
When the flag is true, ONE starts to take into consideration that rogue nodes will be considered during routing and reporting.
- Rogue Degree  
Rogue degree is defined as an integer parameter which varies from 0 - 100. This parameter is considered only when the flag is set to true. If the flag is true and rogue

degree is 0, there is no impact on the altruism in the OppNet. For the rogue behavior to come into operation the value should be  $> 0$  and  $\leq 100$ . For Rogue Degree = 100, the router (i.e. the rogue node) acts completely selfish and denies all the messages.

#### 7.4.3.2 Node Willingness decision

Once the rogue node flag is true, all the contributing nodes in the simulator behave rogue or altruistic based upon a pseudorandom number. In order to be consistent with the random number generator currently being used for Mobility and Movement Models, we used the same instance of the RNG value. And the decision if a particular node is willing or unwilling is described as follows:

*On every instance of a contributing Node in an Oppnet:*  
*Generate a random number (0-100) via the global generator;*  
*Compare the value with the input Rogue Degree*  
*If the random number is  $>$  Rogue Degree*  
*Node will relay the message as being altruistic*  
*else*  
*Node will deny the message as **ROGUE\_DENIED***

#### 7.4.3.3 Creating Rogue Hosts

The routing behavior and attributes of rogue nodes are different from the altruistic nodes. Therefore it was necessary to create a separate pool of node which will be exhibiting varying degree of altruism during the simulation. The class *SimScenario* is responsible in ONE for taking all the configuration parameters from the input file (The actual file is shown in Appendix B) and then creating the connection interfaces, listeners, simulation settings and then finally creating the hosts on the basis of inputs. This was customized this class to create rogue nodes, with the exact same settings as it would be for the altruistic nodes, except that a rogue node will now deny the message with a rogue degree, as modified in the *DTNHost* class based on the willingness decision.

*On every set up of the scenario for the simulation run:*  
*If the Rogue Behavior is disabled*  
*Perform simulation with altruistic nodes*  
*else*  
*Keep all the simulation settings and parameters same*  
*Create all the nodes being rogue, associating each with a Rogue Degree from the input file*

## 7.5 Experimental Setup

### 7.5.1 Platform for Scenario Simulations

ONE simulator can be deployed and customized in Linux, Unix and Windows platforms. All the experiments were run on a Windows machine with the following details:

**Hardware:** Intel (R) Core 2 (TM) i5-3230M (L2 cache 3MB @ 2.6GHz/Memory 8GB)

**Operating System:** Windows 8.1

## 7.5.2 Eclipse Settings

ONE Simulator version 1.5.1-RC2 comes as an executable with no customization option available. In order to perform the necessary modification for Rogue implementation, the project was setup on Eclipse IDE. Since it has now become a requirement for researchers, there are a number of steps are now available online [22].

## 7.6 Verification of Rogue Behavior in ONE

### 7.6.1 Verification of implementation and existing functionality

After the configuration of necessary parameters in the input configuration file, experiments were designed in order of varying rogue degree for all the nodes. As discussed, that gave a more realistic pattern that actual rogue nodes follow. However, before the experiments could start to observe the average latency impact, keeping the original logic and existing implementation of ONE simulator existing OppNet protocols, movement models, simulation settings and hundreds of other features intact was inevitably mandatory to verify and validate. One methodology we came up with was to cover the boundary conditions in the rogue degree and verifying the rogue enable/disable flag. This has also been recommended by some of the peer researchers via the Knowledge Base and online mailing digest for ONE simulator [23].

***Keeping the rogue flag enabled:***

***Configure the Rogue Degree  $\leq 0$  and keeping rest of the parameters same, measure the values of Average Hop Count and `rogue_denied` on Epidemic Rogue Router.***

***Keeping the rogue flag disabled:***

***Irrespective of the value of Rogue Degree, measure the same parameters, they should be equal.***

```
Message stats for scenario CS-645-0-1
sim_time: 43200.1000
created: 1458
started: 28328
relayed: 26732
aborted: 1594
dropped: 24089
rogue_denied: 0
removed: 0
delivered: 740
delivery_prob: 0.5075
response_prob: 0.0000
overhead_ratio: 35.1243
latency_avg: 5973.0820
latency_med: 4854.0000
hopcount_avg: 2.5676
hopcount_med: 2
buffertime_avg: 4818.4533
buffertime_med: 4386.4000
```

Figure 10 Message Stats Report for both Rogue Degree = 0 and Rogue Behavior set to false

As shown in Figure 10, there were no denial of messages in either case since all the routers were relaying the messages in an altruistic manner. Also the hop count average remained the same for a single run. This verified that the implementation of rogue behavior had no side effect in the existing routing, mobility and reporting functions of the simulator. A detailed explanation of the metrics mentioned in this report is given in Section 7.7.3.

## 7.6.2 Verification of implementation of Rogue Behavior

After verifying the existing APIs and workflows were intact, we wanted to confirm if the Rogue Behavior is correctly implemented as per the desired requirement. That is, each router should behave rogue with the required amount of degree. The previous verification also gave a hint that when there no rogue routers, system was working in a usual way.

For a second check, we performed the following experiment:

***Keeping the Rogue Degree  $\geq 100$ :***

***Keep rest of the Scenario Parameters intact and verify the Average Hop Count to be exactly equal to 1.***

When the rogue degree was configured greater or exactly equal to 100, all the routers were rogue in the simulation. No router relayed any message for the neighboring nodes and all messages were denied. In that case, although the underlying routing was based on Epidemic Router, it behaved like a Direct Delivery Router. Therefore the average hop count was exactly 1. All routers acting as sources had to find their destination individually for the final delivery of messages. This was validated by three experiments, since it was the most crucial factor to determine the correct implementation.

First we verified the report as shown in the Figure 11:

```
delivered: 702
delivery_prob: 0.4815
response_prob: 0.0000
overhead_ratio: 0.0000
latency_avg: 10604.9986
latency_med: 7786.7000
hopcount_avg: 1.0000
hopcount_med: 1
buffertime_avg: 43162.1000
buffertime_med: 43162.1000
```

Figure 11 Average Hop Count is 1 on setting Rogue Behavior 100%

Secondly we verified the individual stats of all the sources and destination by making use of GraphViz as a drawing tool to visually display the node interactions. Since the number of messages were huge, depending upon their creation every tens of seconds, we show a snapshot of some sources and destinations on their encounters in Figure 12.

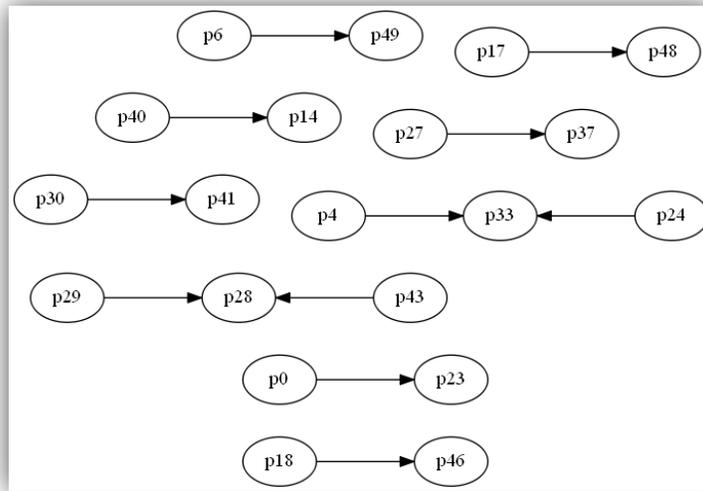


Figure 12: Individual Sources transmitting the messages directly to the destinations on Rogue Degree = 100

Thirdly, in order to verify that it was not the case generally we configured a nominal value of rogue degree and passed the Delivered Message Report to GraphViz again. In that case, the nodes showed a usual hop count. Again, only snapshot is attached here in Figure 13.

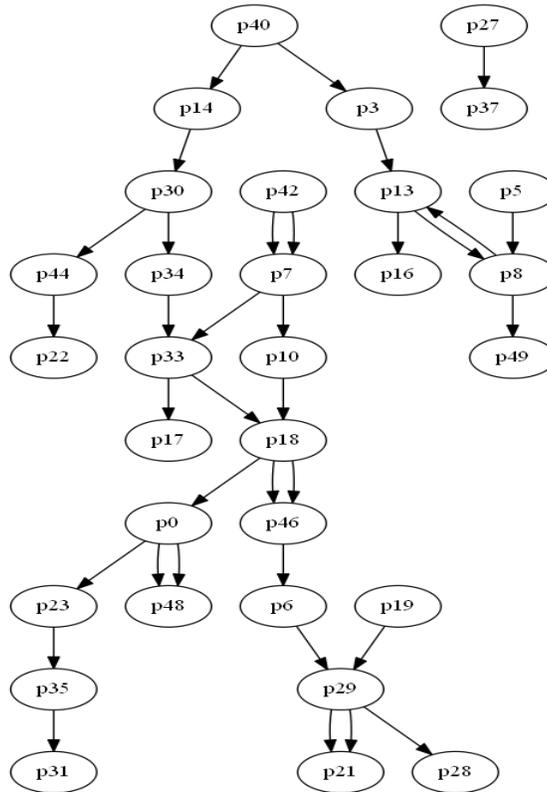


Figure 13: Rogue Degree other than 100%

As an example the node p19 delivers a message to p28 via p20 in the bottom right corner of the graph. The average hop count in that scenario was 2.56.

## 7.7 Experiments with Varying Rogue Degree

After the necessary verification and validation of intended behavior, rest of the project work involved in designing the experiments with varying rogue degree and in the presence of rogue behavior. We varied the rogue degree in the maximum feasible steps in order to provide sufficient data for analyzing the trend of change in the average latency. Rogue Degree 0, 20, 40, 60, 80, 90 and 100 was chosen for each simulation scenario. Each Scenario was executed with 100 different seeds of Movement Model so as the random generation of routing and message passing data can be averaged and any deviation could be compensated.

One of issues we faced was that ONE simulator does not allow execution of scenarios in parallel. Although many permutations can be designed for a set of experiments but still they are fed to the simulator in sequence. Our set of experiments had 2 concrete iterations over the period of time so as to learn not only the time taken for each scenario can be measured, and learn for the next iteration. So before running the final and second simulation we had learnt that running each scenario with 100 seeds with each simulation time of 12 hours (time on clock around 2hr, since ONE fast forwards the time slots) would not be feasible to run in a sequential manner. The estimated time would be around 14 hours for such complete execution of experiments. In order to accommodate this, we used a work around of creating multiple input configuration files with the specific rogue degree and ran each scenario with 100 seeds in parallel. The net simulation time for all the runs with every corresponding configuration settings file was then reduced to almost 8 hours in total.

### 7.7.1 First Iteration

The first iteration was performed immediately without any relevance with the values of count of nodes, speed, or range identical to the paper we wanted to compliment. The intent was to use the default parameters of ONE simulator and experience the execution time. This was an initial run with 5 seeds for each value of rogue degree:

| <b>Rogue Degree</b>  | <b>0</b>   | <b>30</b> | <b>60</b>  | <b>90</b>  | <b>100</b>  |
|----------------------|------------|-----------|------------|------------|-------------|
| <b>Latency (sec)</b> | 4761.07634 | 4862.1358 | 4803.75412 | 4779.77438 | 10321.99832 |

Figure 14 First iteration with default ONE settings and corresponding latency values

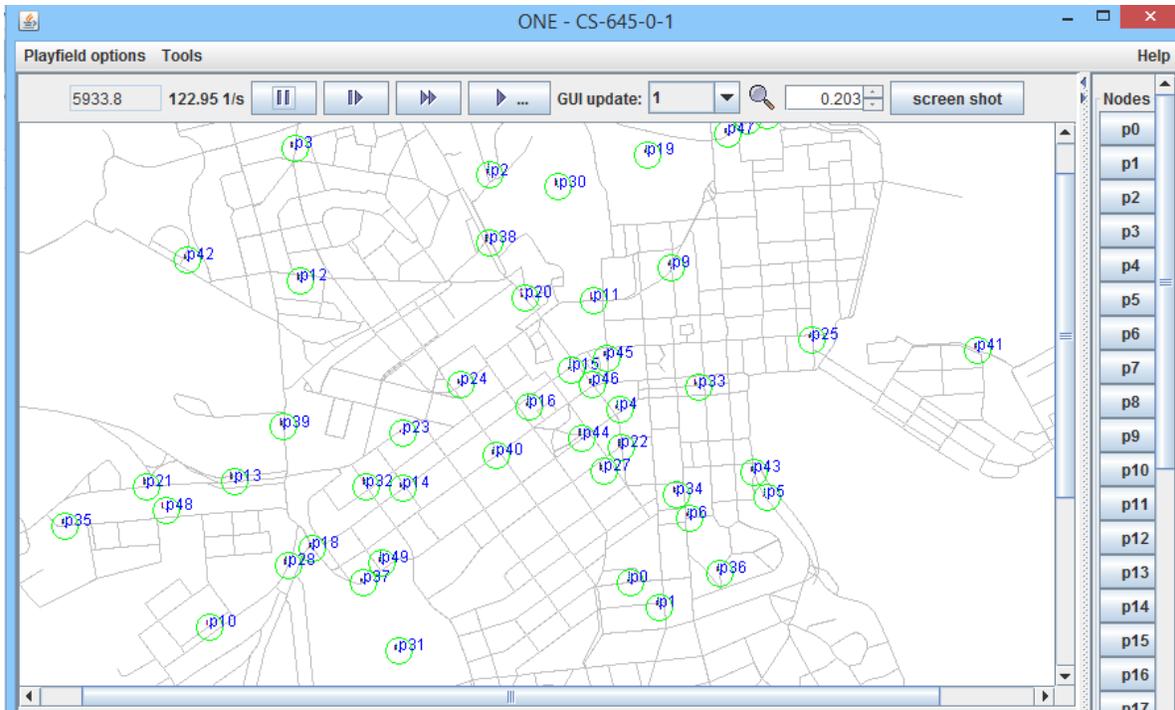


Figure 15 Snapshot of ONE simulator running in GUI mode. The green circles represent the range of the respective node, denoted by p<n> where n is the sequence number. Each scenario is listed as the Title, CS-645-0-1 in this case.

## 7.7.2 Second Iteration

The second iteration was performed after two lessons learnt from the first one:

- The ONE simulator can execute the configuration parameters of a particular scenario in parallel, if there are multiple configuration files available.
- In the first iteration, we used graphical mode in order to see the mobility of nodes and the event log (for message information). If we were to run with more discrete rogue degree values, 100 seeds/scenario, the analysis can be overwhelming considering the amount of information and data displayed on the GUI. So we decided to execute the scenarios in Batch Mode of the simulator (Shell Command: `-b <Number of Runs> <Input Settings>`)

```

Console Problems @ Javadoc Declaration Search
<terminated> DTNSim (3) [Java Application] C:\Program Files\Java\jdk1.7.0_45\
ONE Running in Batch Mode..
Run 1/1
Rogue behavior is ENABLED
Rogue Degree: 100
Running simulation 'CS-645-100-1'
60.0 32787: 546.45 1/s
90.7 43200: 339.03 1/s
Simulation done in 90.52s
---
All done in 92.30s

```

Figure 16: Terminal logs when running ONE in Batch Mode

A snapshot of one of such run with seed 1 and rogue degree 100 is shown in the Figure 16 above. As described above the Simulation Scenario is named as CS-645-100-1 where 100 represents the rogue degree and 1 is the seed value. Each simulation provides the logs if the Rogue behavior was enabled or not. The stats are also provided for the interim simulated and actual timestamps during the simulation.

In the second iteration we conducted 7 experiments in parallel with 7 different configuration files. In order to draw the graph between the rogue degree and the intended metric, average latency, we kept rest of the parameters intact. Those parameters are explained in the section below:

| Experiment No. | Rogue Degree | Rogue Flag |
|----------------|--------------|------------|
| 1              | 0            | True       |
| 2              | 20           | True       |
| 3              | 40           | True       |
| 4              | 60           | True       |
| 5              | 80           | True       |
| 6              | 90           | True       |
| 7              | 100          | True       |

Figure 17 Number of experiment with varying rogue degree

| Scenario | Host End Time (s) | Host Groups | Rogue Behavior | Transmitter Range (m) | TTL (s) | Seeds | Walking Speed (m/s) |
|----------|-------------------|-------------|----------------|-----------------------|---------|-------|---------------------|
| 43200    | 1                 | 1           | True           | 50                    | 43200   | 100   | 0.5 to 2.5          |

Figure 18 Some important input parameters

### 7.7.2.1 Configuration Parameters for all the Experiments

It is also important to explain the modified parameters, which were needed to bring the simulation conditions and configuration of network attributes in synchronization with the analytical model. There are plenty of configuration parameters but not all require our attention for the specific scenario needed for performing experiments on rogue behavior. The important parameters are discussed below.

#### 7.7.2.1.1 Simulation Time for Scenario

Simulation time was kept 43200 seconds or 12 hours. While the analytical model depicts the simulation scenario as an absorbing CTMC following a Poisson Distribution with  $t > 0$ , any OppNet simulation is time bound. We simulated the network with multiple runs to make sure that the results achieved are not impacted by the limitation of net simulation time. A scenario with a bound of 12 hrs is being used by many researchers and is considered sufficient to analyze the impact of desired attributes [5].

#### 7.7.2.1.2 Scenario Name Settings with Seeds

One of the major objective of this project was to comprehensively execute the simulations runs so that any deviations due to random seeds can be compensated by the handsome average statistics. As discussed in the Dataset section in detail, it lead to creation of hundreds of files containing network parameters. A Scenario convention followed is given below:

***CS-645-%%Scenario.rogueDegree%%-%%MovementModel.rngSeed%%***

Where the term *rogueDegree* is replaced by the input degree and *rngSeeds* denotes the seeds value for every run. It is useful for reporting and analysis purposes and was obviously, not required in the mathematical modeling.

#### 7.7.2.1.3 Number of Host Groups

ONE simulator by default creates 6 host groups ranging between Pedestrians, Cars and Trams. (Actually it is up to the researcher to define any number of groups). For the purpose of keeping similarity, it was required to stick to one kind of hosts. For all the simulation runs, Pedestrians were chosen as a single group involved in the simulation.

#### 7.7.2.1.4 Transmitter Range

Transmitter range was selected to be 50m as per the value chosen for the mathematical model. The type of interface was Bluetooth interfaces and every node had one such interface to broadcast the messages.

### 7.7.2.2 *Output Reports to extract the Data Set*

Every run of simulation scenario resulted in three reports. These reports are given below:

#### 7.7.2.2.1 Message Delay Report

This report provides the cumulative probability of delivery of messages sorted by the delay values of each message. It was useful to check the delay values and the probability of delivery at that very successful transmission of message.

#### 7.7.2.2.2 Message Delivery Report

This report provides the amount of messages delivered, delivered/created, every time a message is created or delivered.

#### 7.7.2.2.3 Delivered Message Report

A big number of messages and connections (up to hundreds and thousands) of messages are generated during the simulation of rogue behavior. Sometimes it is necessary to graphically view and verify if the message routing is as per the desired routing protocol and implementation performed. This report generates the message traversal expressions recognized by the famous GraphViz [24] tool. A sample report is shown in Appendix A. A few graphs generated by making use of this tool are shown in the previous section in Figure 12 and Figure 13.

Snapshots of some of the reports taken from the repository are shown in the Appendix A at the end of the report.

#### 7.7.2.2.4 Message Stats Report

This is the main report we were primarily interested in. It provides all the useful summaries of metrics e.g. Average Latency, Average Hop Count, Number of created messages, relayed messages and other important ones. Since the metrics in this report were crucial, we provide a brief explanation of the next section.

### 7.7.3 Explanation of Metrics in Message Stats Report

We have summarized the definitions of metrics as per the logic defined in the algorithms in the ONE simulator. These definitions although appear straightforward, but can infer different meanings, especially when some of the terms seem equivalent but they are actually not. We were mainly interested in Average Latency, which is highlighted blue in the below table. However a correlation between these metrics always exist, where an increase or decrease in one is directly or inversely proportional to the other, or has no impact in certain conditions. Therefore it is important to understand the difference between these metrics:

|  |  |
|--|--|
| <b>Created</b>                               | It denotes the total number of messages created during simulation. The count does not include replicated messages for the relaying or forwarding purpose. Every node in the simulation creates the messages as per a single configurable value (every x seconds).  |
| <b>Started (Relayed Aborted = Started)</b> + | It describes the total number of message transmissions started between network nodes. It is the sum of number of relayed and aborted values (explained below). The value is different because it gives the overall picture of message entities in the simulation.  |
| <b>Relayed</b>                               | Number of successful transmissions between nodes are denoted by Relayed count. It does not include the unsuccessful or aborted transmissions. This value increase many folds in dissemination or replication based protocols because the multiple-copy nature of those protocols implies more proliferation of messages as per the logic. Epidemic Router by far leads to great the greatest count of relayed messages since every node infects every other node on its encounter, and exchanges the missing copies of messages (Section 4: Epidemic Router) |
| <b>Aborted</b>                               | The count of aborted transmissions between nodes. It is different  |

|                       |  |
|-----------------------|--|
|                       | from dropped messages.   |
| <b>Dropped</b>        | Describes the total count of messages dropped from nodes' buffers. It can be due to a TTL timeout or Buffer space full reason.   |
| <b>Removed</b>        | <p>The pseudo-code for this metric is defined as follows in The ONE simulator:</p> <pre><i>if delivered</i> <i>then</i>   <i>true</i> <i>else</i>   <i>drop from buffer</i></pre> <p>If the message cannot be delivered due to any reason and is dropped from current node's buffer, a value in the removed count is increased</p>   |
| <b>Delivered</b>      | It just denotes the total number of successfully delivered messages in the overall simulation period. The delivered message has an acknowledgment status of RCV_OK as per the rules in ONE simulator.  |
| <b>Delivery_prob</b>  | This metric describes the message delivery probability at the end of the simulation. Sometimes it is also referred to as the delivery ratio since it is the ratio between the delivered messages and created messages. As described in the previous sections, one of the main objective of OppNets is to maximize this value   |
| <b>Response_prob</b>  | <p>We never used this metric since the message acknowledgments back to the initial source via a number of hops are not configured (and typically not required in OppNets) in The ONE simulator. But the logic is defined, if needed for future use, as follows:</p> <pre><i>if (protocol supports response)</i> <i>then</i>   <i>applicable</i> <i>else</i>   <i>0.000</i></pre> |
| <b>Overhead_ratio</b> | <p>It denoted an assessment of bandwidth efficiency during the simulation and is calculated as per the following formula:</p> $\frac{((\text{NumberOfRelayedMessages} - \text{NumberOfDeliveredMessages}))}{\text{NumberOfDeliveredMessages}}$   |
| <b>Latency_avg</b>    | It is the average message delay from creation of message from the source to its delivery to the destination. The term delay and latency are interchangeably used in OppNets. This is the main metric used in our evaluation criteria, in order to measure how the latency is impacted with the change in rogue intensity and behavior of nodes.                                  |
| <b>latency_med</b>    | It denoted the median of average message delay value at the end of   |

|                       |  |
|-----------------------|--|
|                       | the simulation.  |
| <b>hopcount_avg</b>   | This is an important metric to refer to after the end of simulation because it counts the average number of hops which were needed between source and destination nodes, in order to deliver the messages as per the underlying routing protocol. In OppNets, the routing protocol Direct Delivery always has the value 1. For the other protocols, the value can vary depending upon the altruism, the geographical positions of nodes, their intermeeting times and many other factors (Appendix C for all the parameters) |
| <b>hopcount_med</b>   | It describes the median of hop count values throughout the simulation time   |
| <b>buffertime_avg</b> | Average time that the messages stayed in the buffer at each node<br>The formula is: (creation time/receiving time) (next hop)  |
| <b>buffertime_med</b> | Median of buffer time values   |
| <b>rtt_avg</b>        | Round Trip Time Average – Average time from creation to confirmation of delivery   |

## 8 Results

All the set of experiments were executed without any error or exceptions. The batch mode of The ONE simulator ran parallel in 7 streams produced all the logs and reports as explained in Section 7.7.2 (Also refer to the git repository under 'reports' directory of ONE simulator). After every run of the scenario, the directory populated with the corresponding number of reports. The files can be post processed for further analysis and deductions. A total number of 2100 files (7 Experiments\*100 Seeds\*3 files per Experiment) were output after the simulation. In order to take the average of latency values for our evaluation against

We describe the detailed dataset extracted as a result of all simulation runs in the next section.

### 8.1 Dataset and Explanation

Each experiment resulted in three reports (output files showing required metrics) as explained before. Our main metric to observe was Average Latency which is dumped in Message Stats Report (Section 7.7.2.2.3). However for further investigation we often referred to Message Delivery Report, Delay Report (Figure 18) and Delivered Message Report. These reports contain interim statistics about messages as explained in the previous sections. We will primarily be focusing on the metrics obtained from Message Stats Report (Section 7.7.3). These contained the important metrics aggregated after each experiments we performed.

```
_ Seed_40_ :5634.3083
_ Seed_38_ :5501.2882
_ Seed_65_ :6055.6753
_ Seed_12_ :5807.1498
_ Seed_8_  :5815.9534
_ Seed_55_ :5774.3045
_ Seed_99_ :6028.6695
_ Seed_87_ :6039.7499
_ Seed_63_ :5751.6837
_ Seed_2_  :5885.8947
_ Seed_68_ :5597.7128
_ Seed_84_ :5644.4245
_ Seed_17_ :5801.3747
_ Seed_58_ :5992.4524
_ Seed_77_ :6210.7751
_ Seed_98_ :5410.1405
_ Seed_71_ :6036.2694
_ Seed_5_  :5940.2872
_ Seed_10_ :5799.6930
```

Figure 19 For Rogue Degree = 20, this file shows the average latency calculated after every run of experiment with a different seed. All these 100 values were again averaged to draw the graph with respect to the Rogue Degree

Figure 19 shows the average latency values obtained after 100 runs with different seeds with a single rogue degree. In this very case, the rogue degree was configured as 20. The values of average latency for the first six experiments did (up till rogue degree 90) did not show any major variations. In fact, the lowest average value was around 5860.299 seconds at a much higher

degree of rogue nodes. All the seven files each with 100 average latency values can be seen in the project repository. We also tried to use another mobility model and tried to vary the number of nodes involved in the simulation, to figure out why the latency value was not increasing but the pattern remained the same. In the Section 8.2 we discuss this in more detail. We also took help from the Message Stats Report to analyze the rest of the metrics.

```
Message stats for scenario CS-645-20-1
sim_time: 43200.1000
created: 1458
started: 28139
relayed: 26498
aborted: 1637
dropped: 23876
removed: 0
delivered: 722
delivery_prob: 0.4952
response_prob: 0.0000
overhead_ratio: 35.7008
latency_avg: 5634.3083
latency_med: 4508.4000
hopcount_avg: 2.5942
hopcount_med: 2
buffertime_avg: 4885.4179
buffertime_med: 4536.6000
rtt_avg: NaN
rtt_med: NaN
```

Figure 20 For Rogue Degree 20 the Message Stats Reports (All the metrics explained in Section 7.7.3)

Figure 20 shows the individual report of all the aggregated values of metrics obtained after every simulation run. We picked a random seeds value for the report as shown in the Figure. The total number of message created for this scenario were 1458. Since the router was Epidemic we expected a huge number of message transmissions considering the message was being created by

```
Message stats for scenario CS-645-80-1
sim_time: 43200.1000
created: 1458
started: 27673
relayed: 26110
aborted: 1562
dropped: 23457
removed: 0
delivered: 729
delivery_prob: 0.5000
response_prob: 0.0000
overhead_ratio: 34.8162
latency_avg: 6109.0502
latency_med: 4927.3000
hopcount_avg: 2.7394
hopcount_med: 2
buffertime_avg: 5024.3282
buffertime_med: 4767.3000
rtt_avg: NaN
rtt_med: NaN
```

Figure 21 For Rogue Degree 80 the Message Stats Reports. Notice the a subtle increase in delivered messages and almost 475 seconds increase in the delay. However, the delay was minimized by the average values later on

each node every 25 to 30 seconds. Excluding the warm up period (when the count of creation of messages is not taken into consideration), a total of 28139 messages were created. Out of these, 23876 were dropped due to connection loss, rogue denial and possibly the buffer limit of the node. We compared these stats to one of the report having the rogue degree of 80 (Figure 21). Notice that these are individual seed value not the overall average. If we compare the latency values between the two reports, there was an increase of 475 seconds in the delay. Also, there was a very subtle increase in the delivery ratio of the messages. Though we expected this value to decrease as the rogue degree is increased. This result was surprising, as we deduce and compare in the next section. Finally we compared these two seed values with the maximum rogue degree of 100, when all the nodes were denying every other message received from the neighbors.

```
Message stats for scenario CS-645-100-1
sim_time: 43200.1000
created: 1458
started: 702
relayed: 702
aborted: 0
dropped: 1
removed: 0
delivered: 702
delivery_prob: 0.4815
response_prob: 0.0000
overhead_ratio: 0.0000
latency_avg: 10604.9986
latency_med: 7786.7000
hopcount_avg: 1.0000
hopcount_med: 1
buffertime_avg: 43162.1000
buffertime_med: 43162.1000
rtt_avg: NaN
rtt_med: NaN
```

Figure 22 Rogue Degree of 100. The latency value was almost doubled with a hop count of 1

Figure 22 shows the snapshot of the report containing the message stats obtained after a running the scenario with rogue degree of 100. All the nodes were acting selfish all the times during this simulation. The total number of messages created were again the same i.e. 1458. The total of 702 transmissions started only when the source and destinations had the mutual contact. Therefore the equal number of 702 messages were relayed. We could not figure out the 1 dropped message in this case, which may have been created at the end of the simulation and before it could find the corresponding destination, it got dropped. No messages were aborted or removed considering the only immediate neighbor in contact were both source and destination.

The delivery ratio was decreased to 0.4815 in this case. But the critical deviation in this scenario was the doubling of average latency to 10604 (The average for all 100 seeds was 11039 seconds). The standard deviation in this case was 338 seconds. The median value shown as 7786.7 is only for one seed over one simulation. The average hop count came out to be exactly 1 since no intermediate nodes were involved for relaying. Since we configured the buffer time till the end of simulation, no messages were dropped due to time out of the buffer value.

Although not much useful for the overall results, we made use of Message Delay Report less frequently to monitor the delay value at the cumulative probability value over the period of simulation, as shown in Figure 23.

```
# messageDelay cumulativeProbability
2.3000 0.0007
2.6000 0.0014
3.0000 0.0021
3.7000 0.0027
4.9000 0.0034
6.2000 0.0041
13.8000 0.0048
24.0000 0.0055
35.7000 0.0062
39.2000 0.0069
43.3000 0.0075
45.9000 0.0082
46.3000 0.0089
65.4000 0.0096
72.9000 0.0103
97.2000 0.0110
123.2000 0.0117
149.2000 0.0123
171.4000 0.0130
178.3000 0.0137
219.8000 0.0144
```

Figure 23 Message Delay Report. Each line shows the delay of message and cumulative probability at that instant

Overall there were no considerable fluctuations in the our intended metric after varying the rogue degree from 0 to 90%. Average latency was apparently not impacted how much a node was behaving rogue during these configured values. It is only when all the nodes started to behave selfishly and denied others messages, average latency had a drastic increase to almost double the value during the entire previous simulation runs at varying rogue degrees less than 100.

## 8.2 Deduction/Analysis

As discussed in Section 7.7.1, we mainly performed two set of iterations. The first iteration gave us a high level picture of how the simulator was behaving after we implemented the rogue plugin ONE. The graph was not very credible since there were only 5 seeds per rogue value. The latency value started from 4761 depending upon the world size parameters, as can be seen in the repository.

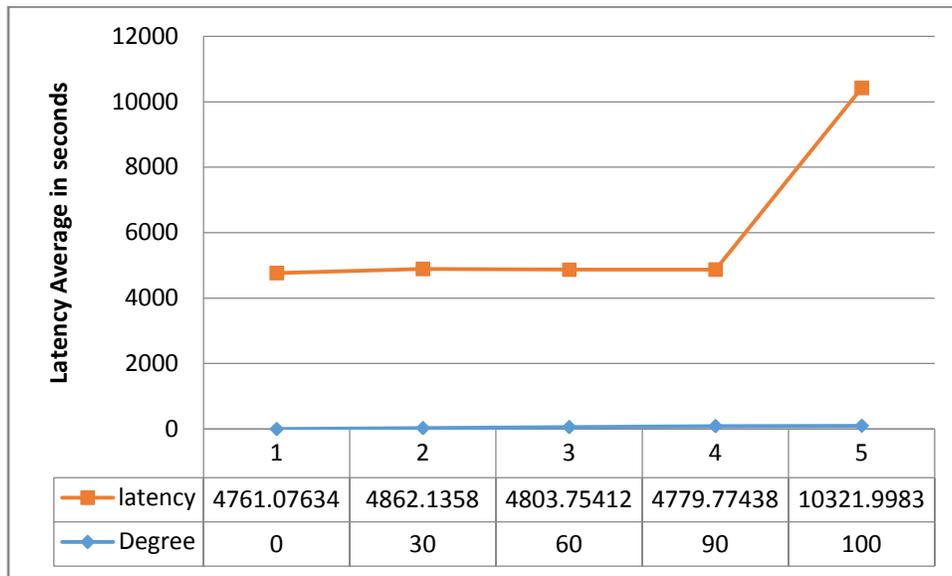
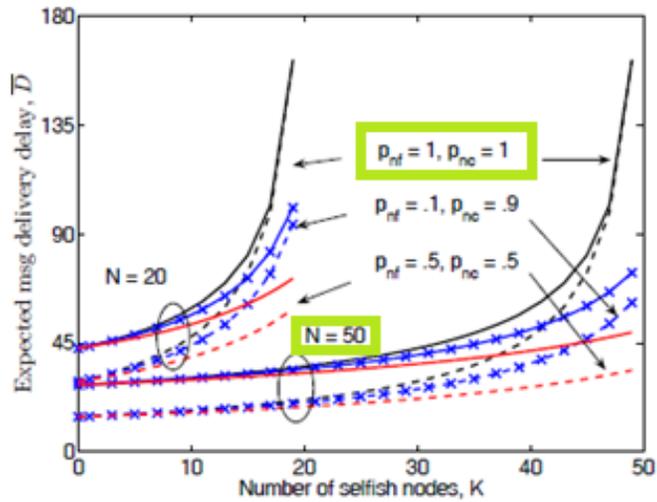


Figure 24 First Iteration of Latency against Rogue Values

In Figure 25 we again draw our benchmark for the comparison of trend of Average Latency against the rogue values. As described in the implementation section, our simulations and results were targeted to a portion of the analytical model. The value against  $P_{nc} = 1$  in the dotted line of the graph with 50 nodes represents the pattern of Average Latency against the rogue or selfish nodes. The graph drawn as a result of our implementation and experiments is shown in Figure 26, as a result of 2nd iteration of experiments during the project.

The graph shown in Figure 26 shows the relationship between Average Latency values and the increasing Rogue Degree from left to right, as a result of the experiments performed in the 2nd iteration of experiments. As opposed to the modeling based on Continuous Markov Chain and Poisson Distribution (Section 5.3) of nodes, the simulation in ONE was a discrete pair of values (Average Latency, Rogue Degree). Therefore the graph was expected to not be having as smooth transitions, especially after the results we got from the Rogue Degree 90 to 100. Also, the author in [10] made use of Random Way Point as a Mobility Model, which we have discussed is less realistic than the one we used (Shortest Map Based Model). Therefore the starting values of Average Latency at Rogue Degree = 0 can vary between our simulation and the modeling value done previously. It is the pattern that matters the most over the period of fluctuating altruism in the nodes but not the initial values.



(a) Expected message delivery delay.

Figure 25 Our evaluation was targeted against the black dotted line representing Epidemic Router with  $p_{nc} = 1$  and  $N = 50$ . As discussed in Section 5.3.

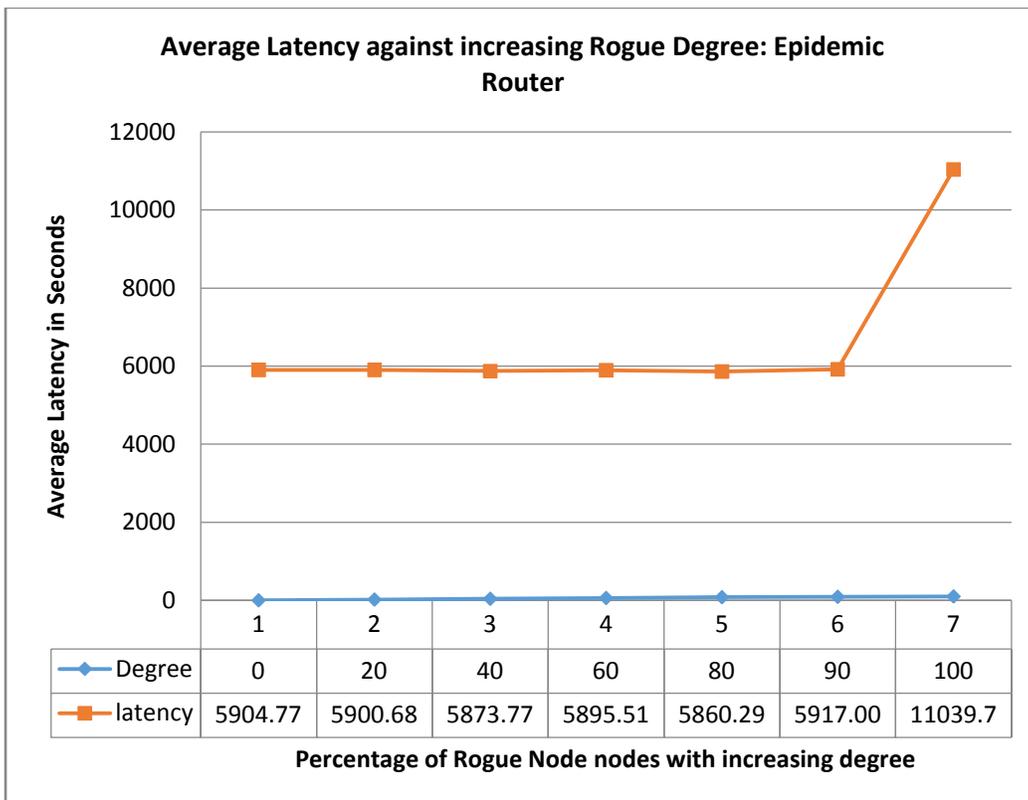


Figure 26 Iteration 2 of Average Latency versus Rogue Degree. Each experiment with the respective degree was run with 100 seed values

It also depends on the simulation area utilized by the mobile nodes involved in message transmission and receiving. As discussed in the configuration parameters, the map area used by the previous researchers was quite small (1000m x 1000m). We realized an area bigger than that (4500m x 3400m) to simulate a map based movement with WKTs (A new feature of ONE simulator). Larger area was specifically chosen in order to cover the default layout of Helsinki map and the default parameters of WKT files.

The difference resulted in the starting values of Average Latency, which in our case was more than twice (5905 seconds for our simulation and around 2200 seconds for the analytical model). However, it is the pattern of change which is important on varying the rogue degree. Moreover, there are a few parameters like contact duration ( $\lambda = 0.37$  contact/hr) which was not available in ONE. And parameters like buffer size and RTT were not mentioned in the paper. But to be on the safe side, we configured the parameters optimally (i.e. we configured the buffer big enough to last until the end of the simulation. TTL value was set so that no messages will be dropped on the basis of timing out, since the intent was to monitor the denial of messages from rogue nodes primarily) so as to bring the simulated scenario as close to the modeled graph.

As shown in the graph, there are very little fluctuations in the latency value from Rogue Degree 0 up till even 90. This coincides with the results achieved by earlier researchers. The noticeable increase in the delay happens after 90% of the nodes behave rogue. Whereas we could see a very smooth transition in the paper results, our discrete value of rogue degree of 100% had a sudden jump from an average latency of 5917 seconds to 11040 seconds. This is referred to as an exponential increase and leads to a very important deduction as explained below.

With the nodes moving at a constant speed of 0.5 to 2.5 m/s, the mobility offered by the altruistic nodes was a powerful aid which negated the rogue behavior instilled in the scenario at the times when nodes acted selfish as per the rogue degree configured for the scenario. The times when nodes were acting altruistic, they dominated successful message delivery over the rogue ones. The contacts and Shortest Path Map Based movement model was though randomly generated via the RNG seeds, but it offered very frequent interactions with the alternative nodes. And since we wanted to keep our scenario in sync with the modeling, we kept the buffer and TTL values big enough so that message dropping due to these metrics can be ignored. Only when the nodes were close to 100% selfish, then every node had to wait for its contact with its destination. This is the reason (as also discussed in the paper), the mobile OppNets are less vulnerable to the malicious attacks. And this also deducts that compared to those of legacy infrastructure based networks which are drastically impacted by the rogue access points, OppNets can utilize alternate means for message relay in these conditions.

Another important difference in our implementation and the mathematical model was the inherent definition of rogue behavior. As per our investigation and peer collaboration [23], the ONE simulator architecture, as discussed previously, is not designed for evaluating the security or scenarios which take malicious behavior into consideration. Therefore we could modify the core design by introducing the rogue degree as a random attribute to every node as per the RNG generator. Whereas, the it was straightforward for the author of the paper to model the selfish nodes in the formula:

$$\overline{D}_{ur}(N, 0) = \frac{1}{\lambda N} \sum_{i=1}^N \frac{1}{i}$$

Where 'ur' represents the unrestricted or epidemic router and  $N$  is the total number of nodes, with the presence of 0 rogue nodes. The authors model the increase in the rogue nodes ( $K$ ) until  $K = N - 1$ , which we simulated in our case as the 100% rogue degree. Although in real world, our implementation is more practical since a mobile node might not be deterministically acting rogue all the times. Therefore a random association of rogue degree was more suitable. However, it can lead to differences at higher level of rogue degrees and the associated contact time between the nodes. Whereas, a node acting rogue with a probability of  $X\%$  can vary its altruism level during the contact time with a set of nodes, a completely rogue node at that very instant will be rejecting all the messages relayed to this. This could be one of the reasons at higher level of rogue degrees, the mathematical model showed deterioration in the latency values.

The analytical model also draws the conclusion that both the two hop relay and epidemic router are *inherently resilient* by drawing the deceleration factor, on comparing the two protocols. We were not able to make such a comparison since we dealt with Epidemic Router only. However, as per their results, Epidemic Router's performance deteriorates more severely than the two hop relay. It is also shown in [5] and the numerical results in [8].

## 9 Conclusions and Future Work

Our research started with exploring the enhanced features and functionalities of Delay Tolerant Networks/Opportunistic Networks. This involved a comprehensive literature review on the subject area. Opportunistic Networks undoubtedly add the flexibility, reduce the inter networking technologies disparity and bridge these gaps by effectively bundling the important message communication at the Bundle Layer in the OSI stack. They have their pros especially making use of mobility and trust, but their cons also include indefinite delays and acknowledgements whatsoever. However, the emergency and disastrous conditions demand some mean to transfer useful information (Imagine the criticality and efficacy of an OppNet during recent earthquake in Nepal on April 25, 2015, where internet connectivity was severely degraded [25]). The notion of Rogue behavior can exist in these networks especially due to energy scarcity in the mobile devices mostly, and some intermediaries can act selfish by only taking part in the network for relaying their own messages.

As per our literature review, we could find 4 dedicated and critical publications by researchers in this area, and only a one of them considered simulating the OppNets in ONE simulator, taking Rogue Scenarios in consideration. We took one of the publication as a reference model for our simulation in The ONE simulator, as the modeling was done on the parameters we could see feasible for our project and scope of work. Our research question involved how the latency is impacted if the percentage of rogue degree in mobile nodes is increased. The results for the unrestricted relaying protocol (i.e. Epidemic Routing) showed correlation with the trend of deterioration of latency values as modeled in the chosen paper, with a few exceptions as detailed in Section 8.2. We provided a flexibility to make the scenario more realistic than the past paper by taking into account Map Based Movement rather than a hypothetical Random Way Point movement.

Our plug-in implementation for The ONE simulator, and the subsequent set of experiments showed that Map based mobility models are quite less vulnerable to rogue nodes. In particular, for areas of smaller size, even a rogue degree of 90% has negligible impact on the average latency of the network. Although the simulation was not based on real world mobility and connectivity of an OppNet, the synthetic node movements in Epidemic routing provide a good motivation to compare and contrast the results with real world traces, and intermeeting with the nodes. Two differences in our implementation was configurable fully rogue nodes and the absence of comparison with the two hop routing protocol, which we could not take into consideration due to the feasibility and scope of the project.

Having said that OppNets are less vulnerable to rogue node as compared to infrastructure based networks, their detection in an OppNet is more challenging and is currently a very active research area for OppNets. There are many efficient schemes for detecting malicious attacks for classical networks since their static topology leverages many methodologies to aid this objective [26]. After assessing the impacts on OppNets, the detection and penalization of rogue nodes can be investigated for future research areas in this regard. More importantly, most of the research for evaluating the rogue behavior is done on simulated scenarios based on random number generators (RNG). ONE simulator provides a flexibility to import wireless traces in the form of well known texts (commonly referred as WKTs), with respect to mobility as well as the connection establishment granularities between the corresponding nodes. To the best of our

knowledge, there has been no paper published on evaluating the rogue behavior on the basis of real world wireless traces. The plug-in developed in this project can readily aid in this regard.

One of the interesting future research area involve the incentives and penalties to encourage the cooperation in OppNets. There have been a few approaches like IRONMAN [27] which makes use of proliferation of useful information about the contacts who behave selfishly, and hence broadcast the repute of individual nodes for cooperation. Opportunistic Network is still a very immature field in terms of routing protocols and reliable communications, the research, implementation and simulation of rogue behavior has paved way for many challenging dimensions in OppNets, along with the ease it would provide for future research and development using The ONE Simulator.

## Appendix A

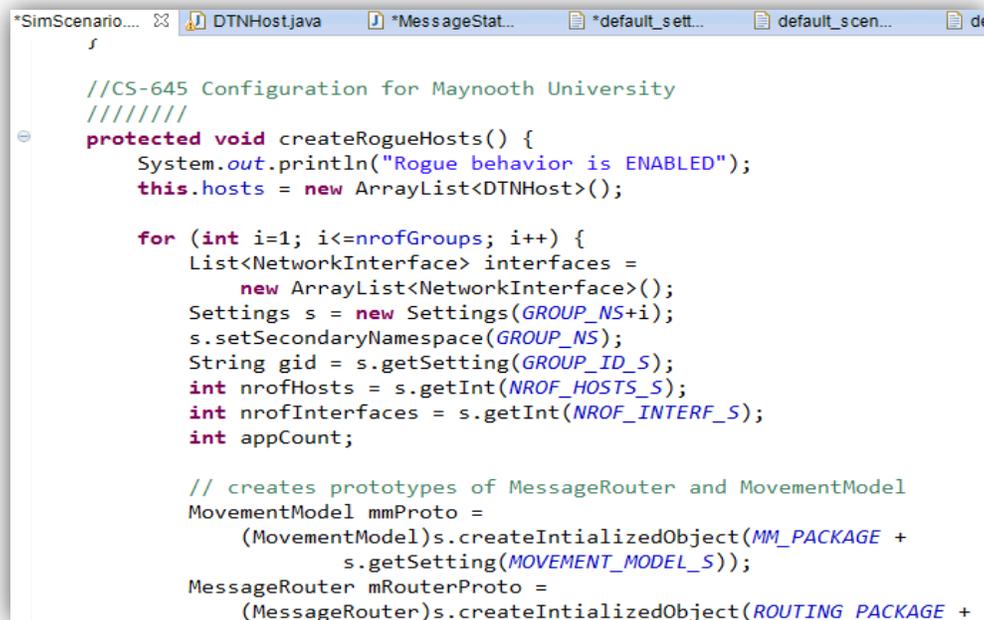
```
# Default settings for the simulation
# Plug-ins added for CS 645, Maynooth University (2014-2015)

## Scenario settings
Scenario.name = default_scenario
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
# 43200s == 12h
Scenario.endTime = 43200
# Define 6 different node groups
Scenario.nrofHostGroups = 6

# CS 645 Introduce Rogue Behavior and degree
Scenario.rogueBehavior = true
Scenario.rogueDegree = 50

## Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)
```

Figure 27 A ONE input simulation file. Rogue Behavior and Degree parameters are highlighted



```
*SimScenario...  DTNHost.java  *MessageStat...  *default_sett...  default_scen...  de
s

//CS-645 Configuration for Maynooth University
/////
protected void createRogueHosts() {
    System.out.println("Rogue behavior is ENABLED");
    this.hosts = new ArrayList<DTNHost>();

    for (int i=1; i<=nrofGroups; i++) {
        List<NetworkInterface> interfaces =
            new ArrayList<NetworkInterface>();
        Settings s = new Settings(GROUP_NS+i);
        s.setSecondaryNamespace(GROUP_NS);
        String gid = s.getSetting(GROUP_ID_S);
        int nrofHosts = s.getInt(NROF_HOSTS_S);
        int nrofInterfaces = s.getInt(NROF_INTERF_S);
        int appCount;

        // creates prototypes of MessageRouter and MovementModel
        MovementModel mmProto =
            (MovementModel)s.createIntializedObject(MM_PACKAGE +
                s.getSetting(MOVEMENT_MODEL_S));
        MessageRouter mRouterProto =
            (MessageRouter)s.createIntializedObject(ROUTING_PACKAGE +
```

Figure 28 Plug in for creating Rogue Nodes from the SimScenario Class

```

public int receiveMessage(Message m, DTNHost from) {
    Message newMessage = m.replicate();
    //CS-645 Configuration for Maynooth University
    boolean willingNess = getHost().wantToCooperate();
    System.out.println("Willingness of Destination: " + willingNess);
    if (getHost().getRogueBehaviorStatus()){
        if (m.getTo() != getHost()){
            if (!willingNess){
                /*System.out.println("From host: " + from + ", To host: " +
                    ", Dest host want to cooperate?: " + willingNess);*/
                this.rogueMsgCount++;
                return DENIED_ROGUE;
            }
        }
    }
}

```

Figure 29 The algorithm for denying the rogue messages. Customized in Message Router Class

```

/* scenario CS-645-0-1740 messages delivered at sim time 43200.09999965185 */
digraph msggraph {
    p40->p14;
    p40->p3->p13;
    p0->p23;
    p13->p16;
    p23->p35->p31;
    p29->p28;
    p27->p37;
    p19->p29->p21;
    p6->p29->p21;
    p5->p8->p13;
    p18->p46;
    p0->p48;
    p42->p7->p10->p18->p0->p48;
    p14->p30->p34->p33->p17;
    p13->p8->p49;
    p30->p44->p22;
    p42->p7->p33->p18->p46->p6;
    p33->p23->p0->p38->p37->p15;
    p37->p27->p0;
}

```

Figure 30 Data generated by Delivered Message Report in order to draw graph in GraphViz tool [24]. E.g. A message from node p37 was sent to p0 via p27 in the last line of the report.

## 10 Appendix B

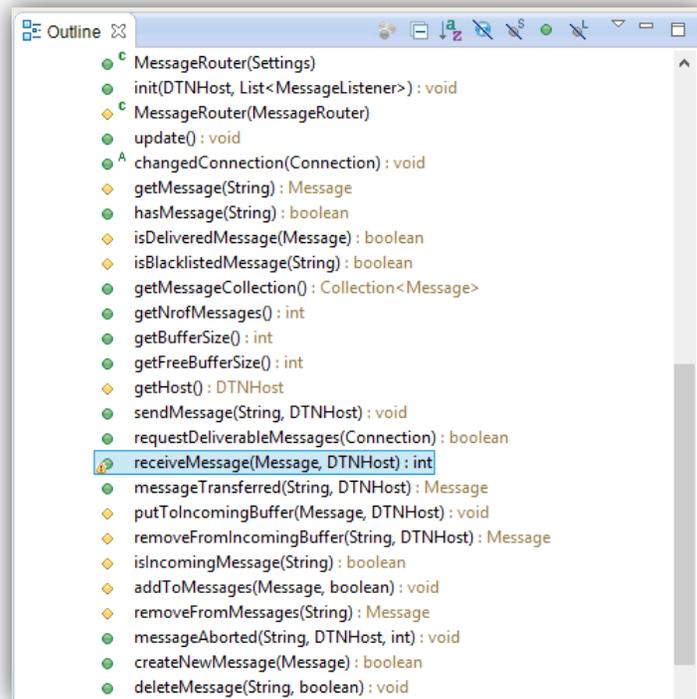


Figure 31 Methods provided by class `MessageRouter`

## 11 Appendix C

### 11.1 Input configuration file for Experiments.

The input settings and configuration file has been copied here for reference. Please note the parameters related to Rogue Degree were varied accordingly as described in the Results section.

```
#
# Default settings for the simulation
# Plug-ins added for CS 645, Maynooth University (2014-2015)
## Scenario settings
#####
#Scenario.name = default_scenario
Scenario.name = CS-645-%Scenario.rogueDegree%-%%MovementModel.rngSeed%%
#####3
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
# 43200s == 12h
Scenario.endTime = 43200
# Define 6 different node groups
Scenario.nrofHostGroups = 1
#Scenario.nrofHostGroups = 6
# CS 645 Introduce Rogue Behavior and degree
Scenario.rogueBehavior = true
Scenario.rogueDegree = 0
### Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)
# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
##### CS 645
#btInterface.transmitRange = 10
btInterface.transmitRange = 50
#####
# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 1000
### Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name from movement package)
# waitTime: minimum and maximum wait times (seconds) after reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from routing package)
# activeTimes: Time intervals when the nodes in the group are active (start1, end1, start2, end2, ...)
# msgTtl : TTL (minutes) of the messages created by this host group, default=infinite
### Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities (poiIndex1, poiProb1, poiIndex2, poiProb2, ... )
#   for ShortestPathMapBasedMovement
# okMaps : which map nodes are OK for the group (map file indexes), default=all
#   for all MapBasedMovement models
# routeFile: route's file path - for MapRouteMovement
```

```

# routeType: route's type - for MapRouteMovement
# Common settings for all groups
Group.movementModel = ShortestPathMapBasedMovement
Group.router = EpidemicRouter
#####
#Group.bufferSize = 5M
Group.bufferSize = 50M
#####
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
#####
#Group.speed = 0.5, 1.5
Group.speed = 0.5, 2.5
#####
# Message TTL of 300 minutes (5 hours)
#####
#Group.msgTtl = 300
Group.msgTtl = 720
#####
# 40 previously
Group.nrofHosts = 50
#####
# group1 (pedestrians) specific settings
Group1.groupID = p
#####
# group2 specific settings
#Group2.groupID = c
# cars can drive only on roads
# Group2.okMaps = 1
# 10-50 km/h
#Group2.speed = 2.7, 13.9
# another group of pedestrians
#Group3.groupID = w
# The Tram groups
#Group4.groupID = t
#Group4.bufferSize = 50M
#Group4.movementModel = MapRouteMovement
#Group4.routeFile = data/tram3.wkt
#Group4.routeType = 1
#Group4.waitTime = 10, 30
#Group4.speed = 7, 10
#Group4.nrofHosts = 2
#Group4.nrofInterfaces = 2
#Group4.interface1 = btInterface
#Group4.interface2 = highspeedInterface
#Group5.groupID = t
#Group5.bufferSize = 50M
#Group5.movementModel = MapRouteMovement
#Group5.routeFile = data/tram4.wkt
#Group5.routeType = 2
#Group5.waitTime = 10, 30
#Group5.speed = 7, 10
#Group5.nrofHosts = 2
#Group6.groupID = t
#Group6.bufferSize = 50M

```

```

#Group6.movementModel = MapRouteMovement
#Group6.routeFile = data/tram10.wkt
#Group6.routeType = 2
#Group6.waitTime = 10, 30
#Group6.speed = 7, 10
#Group6.nrofHosts = 2
## Message creation parameters
# How many event generators
Events.nrof = 1
# Class of the first event generator
Events1.class = MessageEventGenerator
# (following settings are specific for the MessageEventGenerator class)
# Creation interval in seconds (one new message every 25 to 35 seconds)
Events1.interval = 25,35
# Message sizes (500kB - 1MB)
Events1.size = 500k,1M
# range of message source/destination addresses
#Events1.hosts = 0,126
Events1.hosts = 0,50
# Message ID prefix
Events1.prefix = M
## Movement model settings
# seed for movement models' pseudo random number generator (default = 0)
# Model Seeds
MovementModel.rngSeed =
[1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;22;23;24;25;26;27;28;29;30;31;32;33;34;35;36;36;37;38;39;40;41;42;43
;44;45;46;47;48;49;50;51;52;53;54;55;56;57;58;59;60;61;62;63;64;65;66;67;68;69;70;71;72;73;74;75;76;77;78;79;80;81;82;82;
84;85;86;87;88;89;90;91;92;93;94;95;96;97;98;99;100]
# World's size for Movement Models without implicit size (width, height; meters)
#####
# MovementModel.worldSize = 4500, 3400
# 1401 for lemda 0.37
MovementModel.worldSize = 4500, 3400
#####
# How long time to move hosts in the world before real simulation
MovementModel.warmup = 1000
## Map based movement -movement model specific settings
MapBasedMovement.nrofMapFiles = 4
MapBasedMovement.mapFile1 = data/roads.wkt
MapBasedMovement.mapFile2 = data/main_roads.wkt
MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
MapBasedMovement.mapFile4 = data/shops.wkt
## Reports - all report names have to be valid report classes
# how many reports to load
Report.nrofReports = 3
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report with output setting)
Report.reportDir = reports/3rd_iteration/RogDeg=0
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = MessageDeliveryReport
Report.report3 = MessageDelayReport
## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true
## Optimization settings -- these affect the speed of the simulation

```

```
## see World class for details.
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true
## GUI settings
# GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
GUI.UnderlayImage.rotate = -0.015
# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 100
# Regular Expression log filter (see Pattern-class from the Java API for RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```

## 12 Bibliography

- [1] A. P. M. C. Luciana Pelusi, "Opportunistic networking: data forwarding in disconnected mobile ad hoc networks," 2006.
- [2] S. D. R. C. ZHENSHENG ZHANG, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges".
- [3] V. C. D. G. I. H. Stephen Farrell, "When TCP Breaks Delay- and Disruption-Tolerant Networking," pp. 72-78, 2006.
- [4] N. R. C. (. TKK, "The ONE," [Online]. Available: <http://www.netlab.tkk.fi/tutkimus/dtn/theone/>. [Accessed 03 01 2015].
- [5] S. M. I. G. S. D. O. W. S. M. I. Yong Li, "The Impact of Node Selfishness on Multicasting in Delay Tolerant Networks," in *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. 60, NO. 5, JUNE 2011*, 2011.
- [6] D. B. Amin Vahdat, "Epidemic Routing for Partially-Connected Ad Hoc Networks".
- [7] M. G. a. D. N. C. Tse, "Mobility increases the capacity of ad hoc Wireless Networks," 2002.
- [8] A. V. a. I. S. Antonis Panagakis, "On the Effects of Cooperation in DTNs".
- [9] C. Bettstetter, "Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects," *SIGMOBILE Mob. Computing*, pp. 55-66.
- [10] M. Karaliopoulos, "Assessing the Vulnerability of DTN Data Relaying Schemes to Node Selfishness," 2009.
- [11] M. Satyanarayanan, "Mobile computing: the next decade".
- [12] G. W. U. Z. J. Forman, "The challenges of mobile computing," pp. 38 - 47, 1994.
- [13] E. Royer, S. B. C. U. California Univ. and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," pp. 46 - 55, 1999.

- [14] C.-K. T. E.M. Royer, "A review of current routing protocols for ad hoc mobile wireless networks," 1999-4.
- [15] P. Hui, "Selfishness, Altruism and Message Spreading in Mobile Social Network".
- [16] "The Network Simulator NS-2," [Online]. Available: <http://www.isi.edu/nsnam/ns/>. [Accessed 06 05 2015].
- [17] A. Varga, "The OMNET++ discrete event simulation system," *Proceedings of the European Simulation Multiconference*, pp. 319-324, 2001.
- [18] K. F. a. R. P. Sushant Jain, "Routing in a delay tolerant," *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 145-158, 2004.
- [19] "dtnsim," [Online]. Available:  
[http://dtn.sourceforge.net/DTN2/doc/doxygen/html/classdtnsim\\_1\\_1SimConvergenceLayer.html](http://dtn.sourceforge.net/DTN2/doc/doxygen/html/classdtnsim_1_1SimConvergenceLayer.html).
- [20] D. College, "Community Resource for Archiving Wireless Data," [Online]. Available:  
<http://crawdad.cs.dartmouth.edu/>. [Accessed 04 06 2015].
- [21] J. O. T. K. Ari Keränen, "The ONE Simulator for DTN Protocol Evaluation," 2009.
- [22] "The ONE simulator tutorial for beginners," [Online]. Available: <http://one-simulator-for-beginners.blogspot.ie/2013/08/how-to-integrate-one-with-eclipse.html>.
- [23] "The ONE Knowledge Base," [Online]. Available: <http://theonekb-barunsaha.rhcloud.com/>.
- [24] "Graphviz - Graph Visualization Software," [Online]. Available: <http://graphviz.org/>. [Accessed 2015].
- [25] "Earthquake rocks Internet in Nepal," [Online]. Available:  
<http://research.dyn.com/2015/04/earthquake-rocks-internet-in-nepal/>. [Accessed 06 06 2015].
- [26] S. Shetty, "Rogue Access Point Detection by Analyzing Network Traffic Characteristics," 2007.
- [27] H. U. o. S. A. U. Bigwood, "IRONMAN: Using Social Networks to Add Incentives and Reputation to

Opportunistic Networks," *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on, 2011.*