A DISCRETE-EVENT SIMULATION TOOL FOR RESOURCE CONSTRAINED NETWORKS

By

Bolong Liang

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Examining Committee:

Boleslaw Szymanski, Thesis Adviser

Sibel Adali, Member

Christopher Carothers, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2015
(For Graduation December 2015)

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my advisor Prof. Boleslaw Szymanski for support my Master research, and for his encouragement, patience and valuable ideas. Without his inspiration and help, it would not be possible for me to complete my master thesis.

Besides my advisor, I would like to have special thanks to Thomas Babbitt for introducing me to the topic as well as giving me guidance and useful comments all the time.

Last but not the least, I would like to thank my family for supporting me throughout my study.

ABSTRACT

Resource Constrained Networks (RCN) have become a hot topic in recent years, they includes Wireless Sensor Networks (WSN), Mobile Ad-Hoc Networks (MANET) and Delay-Tolerant Networks (DTN). Security is one of the major concerns in these class of networks. Researchers are constantly devising new protocols and algorithms to secure different types of RCNs. There are a variety of different network simulation tools available. Each tool has its strengths and weaknesses. Some examples include Ns-3 and OMEet++. Both are quite robust, but they are complex and time consuming to learn. In order to test how a routing or security protocol works in an RCN, it requires significant user overhead in Ns-3. A researcher typically needs to do a lot of modifications to set up a properly functioning RCN test case. This is due to the robustness of the tool and the fact that each implements almost all network properties (ip, buffer, etc). While this is important it also is a reason why they run relatively slow. In this M.S. Thesis, we propose a network simulation tool that abstracts much of the networking details to allow for faster testing of proposed routing and security protocols. It implements a random way-point mobility model and is based on the principle of discrete-event simulation. Simplicity, speed and ease of implementation are the main goals of this tool. The time complexity is around $O(kn^2)$ where k is the number of iterations and n is the number of nodes in the network. We compared our output with those presented by *Babbitt and Szymanski* (Chapter 4), both are implemented with the same trust management scheme; the former in the simulator proposed here and the latter in Ns-3. This tool gets similar simulation outcomes and running time by two orders of magnitude better than the Ns-3 based simulation of the same system.

CHAPTER 1
INTRODUCTION

Many residents of Hong Kong took to the streets in September 2014 in a series of protests named "Umbrella Revolution." A mobile application named FireChat became popular during the protests [1]. In the first two weeks, the app was downloaded over 500,000 times. Why was FireChat so popular? It gained popularity because this app lets users communicate without an internet connection. It functions by creating a mesh network among its users, each mobile phone can be connected via Bluetooth or Wi-Fi,. Since it creates an Ad-Hoc network using these protocols it is considered off-grid. The more people use FireChat, the better the network gets for everybody [2]. But FireChat is not the ideal tool for protests, the problem with FireChat is that everyone can read the message even the police in this scenario. "We are working on adding private messaging with encryption. But this will take months, not weeks," said FireChat marketing chief Christophe Daligault. "This is much harder than for other communication apps as we need to make this work off the grid."

FireChat is an example of Mobile Ad-Hoc network (MANET) that is a class of Resource Constrained Networks (RCN). Security is one of the biggest problem for these type of networks. These Resource Constrained networks (RCN) can be found in other locations and for additional applications such as in a military operation or in a first response team after a natural disaster. These networks are characterized by limited resource (energy, bandwidth) and complex signal interaction due to the nature of mobility. There are different sub-classes of Resource Constrained networks: Delay-Tolerant Networks (DTN), Wireless Sensor Networks (WSN), Mobile Ad-Hoc Networks (MANET) and so on. As described in the example above, security is a major concern in these networks.

There are a number of security and routing algorithms used in RCNs. In order to provide information assurance (IA), a number of key services are required to include information availability, confidentiality, integrity, authentication, and non-

repudiation. Traditional routing and IA protocols such as TCP/IP and certificate revocation are not practical in a RCN. These protocols require end-to-end routing tables and in many cases client server architecture. Due to limitations in a RCN, modification to routing and security protocol are necessary.

One method used to assist in routing and IA is the use of distributed trust management; examples of such protocols in a DTN are *Denko et al.* [3] describes a Bayesian approach to compute trust values based on Bayes' theorem; *Ayday et. al* [4] develops an iterative malicious node detection mechanism for trust management; *Chen et al.* [5] propose a novel model-based methodology for the analysis their trust protocol; and *Babbitt and Szymanski* [6] presents fusion methods to integrate diverse clues into a composite trust values. *Wang et al.* [7] propose a number of trust principals for use in a DTN including the use of fully distributed entities to make decisions, trust should be determined in a highly customizable way, taking into account selfishness of node, and trust should be established in a self-organized reconfigurable way in order to not to be disrupted by the dynamics of a MANET environment.

There are network simulation tools such as ns-3, OMNet++, NetSim and openWNS with functionality adequate for simulating distributed trust management. They are great simulation tools but they are quite complicated to set up a network model and take a long time to perform a simulation. There is a need for a simulation environment that abstracts much of the network details and focuses on the performance of the trust management protocol. These details are important; however, a quick analysis will assist researchers in efficiently determining what approaches show merit. Then those can be ported to the aforementioned network simulator if they show results.

This thesis proposes a simulation tool that simulates mobility for the same type of Resource Constrained Networks. It is written in the C++ language, and is based on the principles of Discrete-Event Simulation [8]. This is a useful simulation tool for those people who just came up with a new trust system that secures the network and want to test if it works. It is not hard for people to implement the scheme because the tool provides the network's structure. All they need to do is

modify the trust module part to their trust scheme. Once the program is running, the pop up window asks for parameters such as the size of the network, number of network nodes and their maximum speed. This can also be scripted to run multiple simulations at once.

In summary, this thesis proposes a Discrete-event simulation tool for use in Resource Constrained Networks. It deals with trust management scheme in networks, its easy to use and it can get simulation results relatively fast. Chapter 2 talks about the background on simulation research and discrete-event simulators. Chapter 3 describe the details of our simulation tool from both user and system perspectives including initialization, execution and output, a part of calculation is shown in execution section. Chapter 4 shows simulation results. Chapter 5 concludes this thesis with a summary and a discussion of future work.

CHAPTER 2
BACKGROUND

2.1 Discrete-Event Simulation

In order to simulate Resource constrained networks, we use discrete-event simulation. Discrete-event simulation is used to do research on all layers of computer networks, including signal processing issues in the physical layers, medium access in the link layer, routing in the network layer, protocol issues in the transport layer, and finally design questions of the application layer [8]. The reason why it is popular is the simulation paradigm fits very well to the considered systems while on the other hand discrete-event based simulation is easily applied. Hence, discrete-event simulation provides a simple and flexible way to evaluate their approaches and study their behavior under different conditions.

2.1.1 The Principle of Discrete-event Simulation

The idea of a discrete-event simulator is to move from one key event in time to the next. The occurrence of an event may trigger change in the system state as well as the generation of new events, which are called future event notices. The event records all event notices as a list or a queue, which is an appropriate data structure to manage all the events in a discrete-event simulation. Figure 2.1 shows the evolution of a discrete-event simulation over time, during the simulation the system state changes only at discrete points t_i in time [8].

2.1.2 The Event-scheduling Time-Advance Algorithm

In this section we describe the core algorithm of a discrete-event simulator. During the simulation the system state evolves over time, thus there is a clock which gives the current time during the simulation. The event list or event queue contains all event notifications which are ordered by time, from smallest to largest. The following flow diagram Figure 2.2 shows how this algorithm work [8].

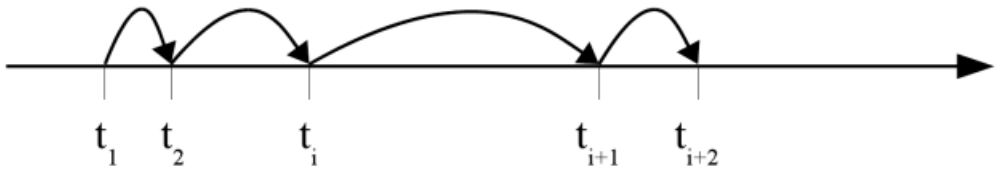


Figure 2.1: Principle of discrete-event simulation [8]

2.2 Tools for Discrete-event Simulation

2.2.1 Delay-tolerant networks

A Delay-Tolerant Networks (DTN) is a kind of network that is designed to work in such an environment which may lack continuous network connectivity due to nodes mobility. A DTN is usually used in an extreme environment for examples in military operation or rescue action after a nature disaster.

2.2.2 OMNet++ Network Simulator

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queueing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools. There are extensions for real-time simulation, network emulation, database integration, SystemC integration, and several other functions.

Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

OMNeT++ provides a component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation ker-

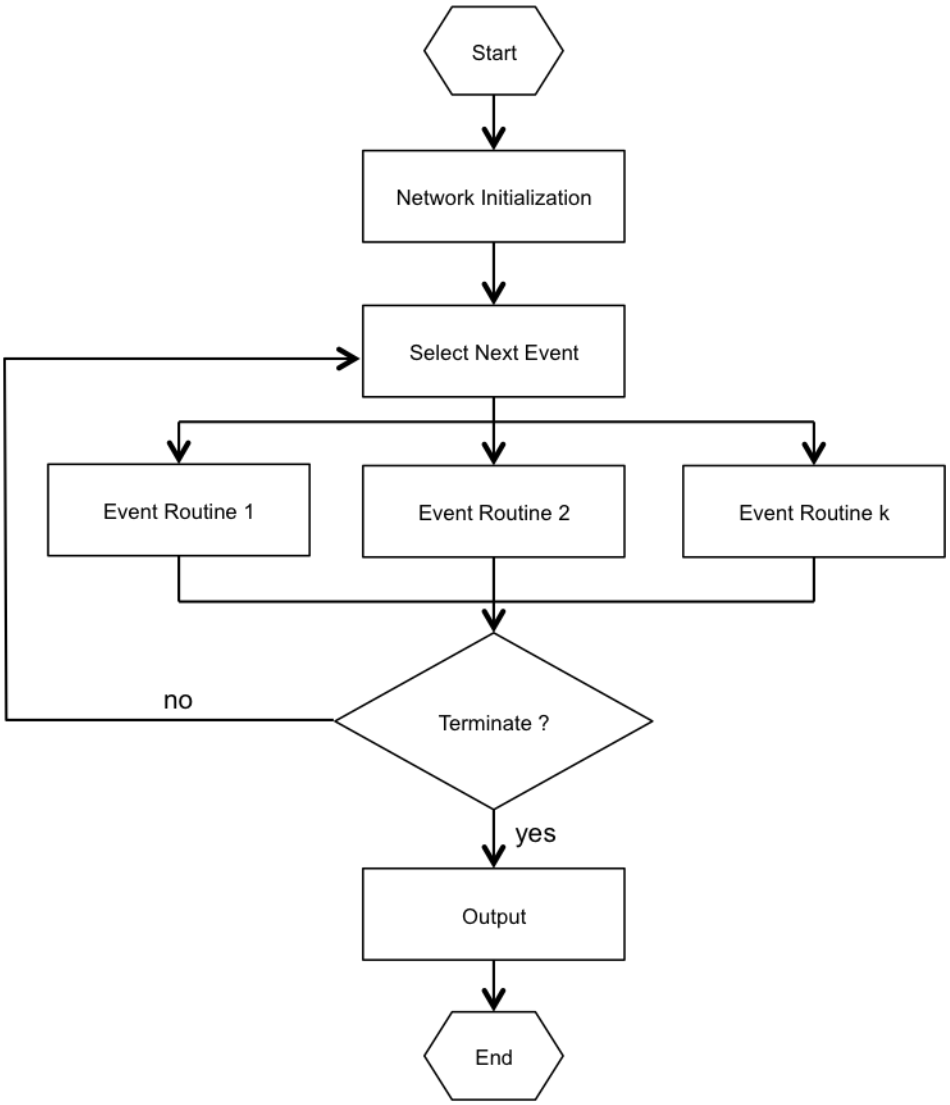


Figure 2.2: Flow diagram of the event-scheduling time-advance algorithm [8]

nel (and models) can be embedded easily into your applications. [9]

2.2.3 Ns-3 Network Simulator

Ns-3 is a free discrete-event simulator for network simulation. Ns-3 is built using C++ and Python with scripting capability. The Ns-3 library is wrapped to python thanks to the pybindgen library which delegates the parsing of the Ns-3 C++ headers to gccxml and pygccxml to generate automatically the corresponding C++ binding glue. These automatically-generated C++ files are finally compiled into the

Ns-3 python module to allow users to interact with the C++ Ns-3 models and core through python scripts. The Ns-3 simulator features an integrated attribute-based system to manage default and per-instance values for simulation parameters. [10]

Ns-3 is a new version of Ns-2, and it is not a commercially-supported tool. There are limited resources to perform long-term maintenance.

2.2.4 openWNS Network Simulator

OpenWNS is a wireless network simulator developed at the department of Communication networks at RWTH Aachen University and has been released as open source software. The simulation platform follows a modular design down to protocol building blocks, which makes it possible to rapidly modify the implemented protocol stacks. OpenWNS currently includes models from physical to application layer. OpenWNS has a dynamic event driven system level simulation platform, it is writtten in c++ and is heavily based on the Boost libraries. For the configuration part, the Python language is used, no need to compile and its easy to use. [8] [11]

CHAPTER 3
SIMULATOR

The Discrete-event simulation tool presented in this chapter fills a need to conduct fast and accurate approximate DTN network simulations. One of the main areas we think this will be beneficial is in determining what distributed trust management systems show promise. As previously stated, this abstracts much of the network details and focuses exclusively on the movement and meeting of events. Our Discrete-event simulation tool is built based on Microsoft Visual Studio 2013 in C++ language, it has been tested on Linux environments as well.

3.1 Initialization

3.1.1 User

We perceive a user wanting the flexibility to set the size and scope of the simulations. This includes the number of nodes, size of the simulation area, movement parameters, movement type, radius that a node can broadcast, routing protocol used, and the ability to modify the random seed for multiple simulation runs. To facilitate the user requirements, there a number of user defined inputs. These parameters can be entered at simulation execution either individually, by command line or scripted. These parameters are required as input for the program and include: the length and the width which defines an rectangle area where all nodes move in it; the number of nodes; the minimum and maximum speed you want them to move; the minimum and maximum stop time and move time. Currently only one mobility pattern is available (Random WayPoint model); however, additional mobility patterns can be added. This is left to future work.

3.1.2 System

In order to meet user requirements, from a system perspective, network initialization is key. In overview, this is done by initializing all nodes to include location, movement parameters (speed, direction, and movement time), messaging queues,

and any user defined structures such a trust management storage. Once the nodes are placed in the simulation grid, the initial movement events are added to the simulation queue. While mobility pattern specific, the next movement event is added for each node. Following that each node determines when, based on mobility pattern, it will next meet each of the $n-1$ remaining nodes. If that meeting occurs before either node's next movement event, then a node meeting event is added to the queue. We purposely limited the number of meeting to avoid having to go into the simulation queue later and remove events when node movement causes them to be voided. If the user defines any time driven even for each node such as updates at certain time periods, then those events are initialized randomly from 0.0 to the timer value. This ensures node timer events do not occur concurrently throughout the simulation. The following sections detail the simulation model, node class and how it initializes each node and the movement class and how it initializes the first movement events.

3.1.2.1 The conceptual framework (Domain model)

Figure 3.1 illustrates the conceptual model which describes the fundamental relationship among all layers. As you can see, there are five domain classes: nodes class, events class, movement class, event queue class/structure and trust class. In theory, a complete simulator requires a messaging class (shown as dashed line class in figure 3.1). In this program the message class is built inside of the trust for testing purpose.

There are two classes used to initialize the network: the node and the movement class. User defined classes such as trust might also require some initialization as well. Once the user inputs the number of nodes, size of the network grid, and any movement specific parameters, our simulator initializes the event queue as follows:

1. Create the user specified number of nodes
2. Based on mobility model, determine the first movement event and add to the event queue.
3. For each node pair (i, j), determine when they meet and if they meet prior to

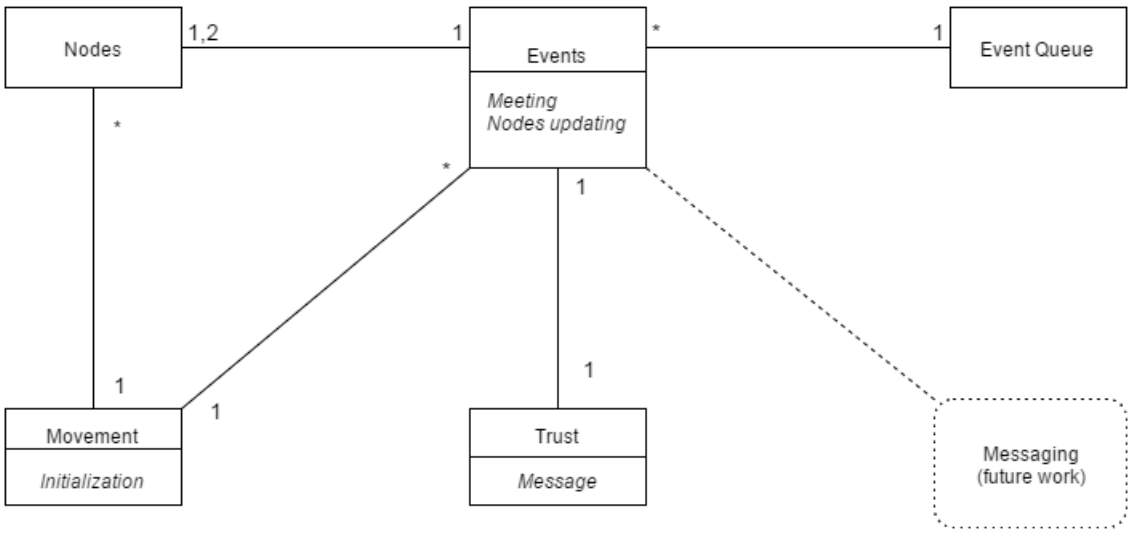


Figure 3.1: Diagram of the program's Domain model

either node i or node j's first movement event, add to the event queue

4. Sort the event queue for simulation execution.

The node class creates and internalizes the nodes (step 1). The movement class provides the functionality to determine the first movement event and determine when each node pair meets (step 2 and 3). The final step is done in the main program and is trivial.

3.1.2.2 Node class

he node class defines basic node properties to include node position (x and y coordinate), direction, speed, and remaining movement time. Other than the node ID all are defined as doubles in C++. These particular node attributes were selected to manage node movement and position. A small list versus a more robust was chosen to simplify execution and minimize processing time.

Included in the node class are a number of functions to set and get the key properties for use in other classes that are modified as needed. The node constructor is key during network initialization, it stores the node id (id), x and y coordinates (x, y), the angle alpha (a), the speed value (v) and the remaining time (t) which indicate how long a particular node keeps it's status (move, stop). Additionally, the

get and set functions are used to access and change data value. The following is the constructor of node class:

```
Node(int id, double x, double y, double a, double v, double t);
```

For the random waypoint mobility model, all values are randomly selected for each node. the only exceptions to that are the speed (v) and movement time (t). Those are randomly chosen in the user provided range. Other mobility models can allow for mode user input. An expansion could allow users to input a file that provides the exact starting parameters for all nodes. This is left to future work.

3.1.2.3 Movement Class(initialization)

The movement module contains all vital functions including initializing the movement, calculate nodes position, and nodes meeting time. During initialization this determines the first movement event for each node. The complete review of functions to support this are in Sections 3.2.4.1 and 3.2.4.2. The initialization for movement is shown below.

```
Movement(int nodeNum, double length, double width, double minS, double
    ↪ maxS, double minMT, double maxMT);
```

This allows for each node to make mobility decisions and ultimately create movement events. For each node the type and time of the next movement event is calculated and then added to the event queue (see addNode_update in Section 3.2.3). There are three types of movement events possible for each event. The first is a start event. Since all nodes start with a movement time and speed this will not be the first event for any node. The second is a stop event. This occurs when the current simulator time (ct) plus the time to move (t) is less then when that node would hit a boundary (hb) ($ct + t \leq hb$). The final occurs when a given node hits the boundary first. This will cause the node to reflect off the boundary (see Section 3.2.4.1).

Having each node initialized, and using the meet time function in the movement class (Section 3.2.4.2), determines the time at which a given pair of nodes will meet. This is executed for each node pair (i, j) and if the meeting time is prior to the next event for either it is added to the event queue (see addMeeting in Section 3.2.3).

3.2 Execution of the Simulation

3.2.1 User

The user provides the simulation execution time and the key parameters for executing the mobility model. Additionally the user can provide trust or any other object that is initialized and as the simulation executes updates key tracking variable. In order to test the utility of our approach a trust class is added; however, additional messaging, and mobility patterns can be added. The output of each event is added in chronological order to allow for quick run time inspection. Specific about the final log is in subsequent sections.

3.2.2 System

Each simulation will run for a user defined time period. That time is managed using the variable (ct) as the current time. Each event is taken from the event queue in calendar time order and executed. There are multiple types of event as listed below.

1. The first is a node meeting event. When that occurs the nodes execute the user defined trust object (Section 3.2.5) and executes and message exchanges. Additional movement information can be captured in an XML file for use in visualization. This is one avenue of future work.

2. The second type of event is a node movement event. There are three types (stop, start, and hit boundary). Once each is added to the queue, the node making the update checks to see if it can meet any other node in the network. If it can prior to both nodes next movement event a new meeting event is placed on the queue

   (a) If the event is a start, then the node updates direct, speed, and time of movement. Once the node is updated it check to see when it will hit a boundary. Like in the intalization, if it hit a boundary occurs prior to when it will stop the next event is a hit boundary else it is a stop.

   (b) If the event is a hit boundary, the direct changes based on the function in Section 3.2.4.1 and the time when that node will next hit a boundary is

checked. If that occur prior to the stop time then another hit boundary is added else a stop.

(c) If the node is a stop then the direct and speed are set to zero and the movement time is set to the user defined wait time; the next event is always a start.

The following sections outline the different classes that support execute of each event as they are popped off the queue.

3.2.3 Event Class

The event class is an important part of the program, it is the foundation of the event queue structure which is defined in main program. The event class obtains the basic information for two types of events, meeting events and node updating events 3.1. The main purpose of this module is making these two types of events. Besides that, it also crates a comparison operator that used for sorting the event queue.

The following two constructors are defined for meeting and node updating events:

```
void addMeeting(Node *x, Node *y, int id_x, int id_y, double time);
```

```
void addNode_update(Node *x, int id, double time, double a, double v,
    ↪ double t, double x_new, double y_new);
```

The overloaded operator function is defined as following:

```
bool operator< (Event other) { return (eventT < other.getEventT()); }
```

When events are generating, they are put into a event queue, then it leads us to the event queue class.

3.2.4 Event Queue Class

The event queue is not a real class but it works as other classes, essentially it's a function defined in main program and it manages the function calls to perform the calculations and determine which event occurs next. It is called event queue

because all events are sorted in this queue and we are always remove the first item on the event queue.

Here are a number of key functions to make sure the queue works properly.

3.2.4.1 Nodes Direction and Hit Boundary Time

Nodes direction and hit boundary time are used to make node update events. The direction of speed is important to a node's movement. In the program, we use the angle that we call alpha(a) between velocity and x axis (horizontal) to indicate the direction of speed.

The angle function is defined as :

```
double Movement::calculateA(int id, int boundaryNum){
  Node a = getNodeById(id);
  if (boundaryNum == 1 || boundaryNum == 3)
    return 2 * PI - a.getA();
  if (boundaryNum == 2 || boundaryNum == 4)
    return PI - a.getA();
}
```

which the id is the node id number and boundaryNum is a predefined number which is computed by the following function

```
int Movement::whichBoundary(int id, double hitTime);
```

In this function, we define four number from 1 to 4 that represent the four boundaries (rectangle box) a node hits.

$$B1 = \frac{length - a.getX()}{a.getV() \times \cos(a.getA())} \tag{3.1}$$

$$B2 = \frac{-a.getX()}{a.getV() \times \cos(a.getA())} \tag{3.2}$$

$$B3 = \frac{width - a.getY()}{a.getV() \times \sin(a.getA())} \tag{3.3}$$

$$B4 = \frac{-a.getY()}{a.getV() \times \sin(a.getA())} \tag{3.4}$$

For the return value, we compare hitTime with those hitBoundary time and chose the matched one as the boundary the node hits.

These equations 3.1 3.2 3.3 3.4 are reused to get hit boundary time. We put those number into a list and choose the smallest non-negative number by comparing them, that is the hit boundary for nodes.

3.2.4.2 Meet Time

When we have a meeting event, we need to know the meet time. In this situation, a quadratic function is used to determine the meet time between two nodes, here are the three coefficients:

$$\begin{aligned} coe_a &= a['v'] \times a['v'] + b['v'] \times b['v'] \\ &- 2 \times a['v'] \times b['v'] \times (sin(a['a']) \times sin(b['a']) \\ &+ (cos(a['a']) \times cos(b['a']))) \end{aligned} \tag{3.5}$$

$$\begin{aligned} coe_b &= 2 \times (a['y'] - b['y']) \times (sin(a['a']) \times a['v'] \\ &- sin(b['a']) \times b['v']) + 2 \times (a['x'] - b['x']) \\ &\times (cos(a['a']) \times a['v'] - cos(b['a']) \times b['v']) \end{aligned} \tag{3.6}$$

$$\begin{aligned} coe_c &= (a['y'] - b['y']) \times (a['y'] - b['y']) \\ &+ (a['x'] - b['x']) \times (a['x'] - b['x']) - r^2 \end{aligned} \tag{3.7}$$

and the determine term and root equations are

$$determ = coe_b^2 - 4 \times coe_a \times coe_c \tag{3.8}$$

$$root1 = \frac{-coe_b - \sqrt{determ}}{2 \times coe_a} \tag{3.9}$$

$$root2 = \frac{-coe_b + \sqrt{determ}}{2 \times coe_a} \tag{3.10}$$

In the last, we choose the right root under some conditions and we get the meet time value.

3.2.5 Trust Class

Trust class is an important part to validate the simulator. It contains messaging class for trading information and it compute the trust value for each event. The program calls trust function before the simulation, after every meeting events during the simulation, and the end of the simulation. Finally ,it will get the average trust value for a overall simulation performance. The follow function calls are when trust functions are called:

```
void trustInialization (int numNodes, double PG);
```

```
void updateTrust(int id1, int id2, double time);
```

```
void endSim (double time);
```

These functions are straightforward, the PG is the percentage of good nodes in the network.

3.3 Output for users

For simulation outputs, our program would display what is happening for each node if they stop, start, hit the boundary or meet with another node.

A sample output for a stop event is shown below:

```
std::cout << countTime << " event -> [update-stop] node " << id << std
    ↪ ::endl;
```

For the trust management part, we output the trust value for the nodes when their trust are get updated, and at the end of simulation, we have a summary of the system.

The following messages will printout when trust increased for any node:

```
std::cout << t << " node " << node << " increased trust for node " <<
    ↪ it2->lastHop << " from " <<  trustArray[node][it2->lastHop];
```

Additionally, the program can export an xml file for visualization purpose, basically it keeps track of nodes positions in different time period, we can see how nodes move around in a Linux environment for example inside of Ns-3.

(add example here)

3.4 Overview and Comparison

In summary, from nodes class to movement class, nodes information is carried into the movement class for initialization, then a serial of events are generated from events class, those events are put into a event queue structure which is defined in event queue class. Moreover, for testing purpose, a trust class is added and linked to the events, every time when a meeting event happens, the trust class would determine the trust value for the nodes involved.

The simulation tool follows the principles of Discrete-event simulation (we talked it in Chapter 2). Furthermore, the program shares the following components which are the common components of all discrete-event simulators.

- State: in this program, trust level for each nodes is used to describe the system state
- Clock: we use a global clock keep tracks of the current time during the simulation
- Event list: a event queue structure is constructed to manage the events
- Random number generator: we have a random number generator function to generate variables of various kinds (random speed, random direction, etc)
- Statistics: at the end of simulation, the trust class will compute a distributed and average trust value. Further variables could be added to contain statistical information

- Initialization and Ending condition: Initialization will start when the program gets all inputs, when the it reaches to the ending time, where the next event on the event queue is greater than the ending time, the simulation stops.

3.5 Performance and Design Constrains

The complexity of this program is $O(kn^2)$ where k is the number of iteration and n is the number of nodes in the network) which is not bad.

As for design constrains, this discrete event simulator is using random waypoint model so if a user want to test their scheme on other mobility model, they need to modify the movement class to make the simulation work. Besides, a separate messaging class should be added as future work.

CHAPTER 4
SIMULATION RESULTS

4.1 Erasure Coding From Trust Module

We compared results using our simulator and the Ns-3 simulator. The trust module was taken from [12] and implemented in the simulator designed and built according to the description in Chapter 3. The trust scheme proposed in [12] utilizes erasure coding and checksums to help to compute the trustworthiness of the nodes along the message paths. Erasure coding works by breaking a message into a set of message segments when enough of those segments arrive at the destination a message is recreated [13]. The authors in [12] also propose a number of utility functions to decide whether the destinations should keep waiting for additional message segments or request a new transmission from the destination. If the destination can recreate the message successfully, it will increase the trust level for nodes that receive additional segments, and vise versa. In this way, when the message segments are corrupt, overtime the bad actors (malicious node or not) can be identified by the network.

The scheme steps are listed below. [12]

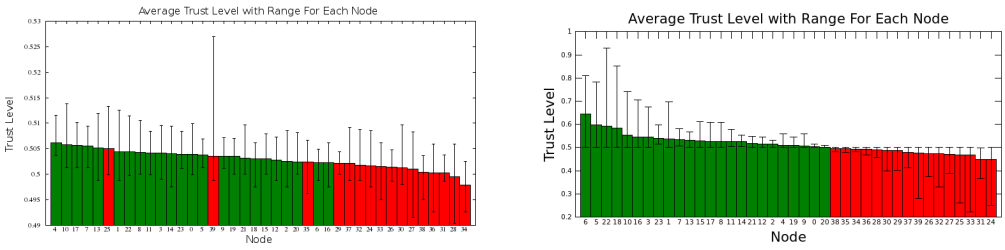
(1) Node x sends a message M to node y.

(2) Segment message M, with added checksum, using erasure coding such that k segments recreate M.

(3) When k unique segments arrive at node y, attempt to recreate M.

(4) If M has a valid checksum, y increases the trust for all nodes sending a valid segment and skips to 6, otherwise it continues to 5.

(5) Node y waits for each additional segment m until recreating M produces a valid checksum or based on cost it is better to request resending the message. If segment m successfully recreates M, then nodes sending valid segments receive a trust increase and all others a trust decrease; move to 6.



(a) Result in [12] **(b) Our Simulator**

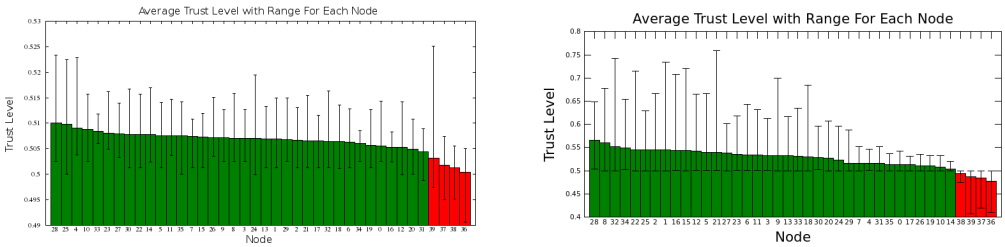
Figure 4.1: Trust values when 60% of the nodes act good

(6) The receiving node waits for time T and accepts any addition segments for M, validity of each is checked against k - *1* known good segments and trust along the relevant path is accordingly changed

4.2 Results

The results are two sets of graphs showing overtime that the average of probability across all nodes increase for good nodes and decrease for bad nodes. The first two graphs are where the percentage of good nodes is 90% and the last two graphs are where they are 60%.

1. 90% Path Trustworthiness: Figure 4.2a and 4.2b shows the average trust level for each of 40 nodes. Green bars are trustworthy nodes and red bars are untrustworthy nodes. As shown in the graph, the untrustworthy nodes are listed to the right with the lowest trust values. The error bars show the largest and smallest individual run and it gives an idea of the range of values. If we compare these two figures, there isn't much difference between them.



(a) Result in [12] **(b) Our Simulator**

Figure 4.2: Trust values when 90% of the nodes act good

2. 60% Path Trustworthiness: Figure 4.1a and 4.1b shows the average trust level for each of 40 nodes. Because of the number of malicious nodes are 16 in this case, both of the graphs don't have a steep slope. For the Ns-3 results, most of the malicious nodes have the lowest average trust level and there are only untrustworthy three nodes that have trust values higher than the trustworthy nodes. In our simulation, all red nodes get the lowest trust value, this is because our simulation doesn't take into account for transmission time, they all trade messages during the meeting.

Based on the simulation results, we can see the two simulator produce similar results. Additionally, the new simulator runs way faster than Ns-3 to get these results. It took only about 10 minutes to run the two complete sets of simulations on one machine essentially using one CPU. The Ns-3 simulation took approximately a day for one run. So we can say the program runs by two orders of magnitude better than the Ns-3 based simulation of the same system.

CHAPTER 5
CONCLUSION AND FUTURE WORKS

This paper proposes a simulation tool for use in a Resource Constrained Network which is different than traditional network, the tool is suited for researchers to test their network protocols. For instance in this thesis erasure coding for trust management runs well and provides a close approximation for the results presented in [12]. This program is a good tool because it is easy to set up and we provide a network structure. One of its main features is the significant reduction in running time for each simulation.

The simulator provides the basic mobile functions and allows user to add their own features. Based on the results in the last chapter, we can conclude the simulator can achieve similar results and run by two orders of magnitude quicker time than a common complex network simulator (Ns-3).

As for the future work, a dependent messaging class can be added at the top of event class in order to streamline the process of adding new routing protocols. For now, the message part is embedded in the trust class. To make the tool real, certain constraints should be set as flags. For example, the length of meetings; the limit number of nodes that can meet at the same time, that means if the number is 2 and there are two nodes meeting, a third node can't trade information with either of them until the the meeting ends. Besides using random waypoint model, other mobility models can be adapted from the existing mobility model. There is random direction model that forces nodes to travel to the edge of the simulation area before changing direction and speed, and there are column model and nomadic community model which are a set of nodes form a line and uniformly moving forward in a particular direction or a group of nodes move together from on location to another[14]. All works mentioned above are possible future work, and the changes should not be very hard based on the program.

REFERENCES

[1] P. Shadbolt. (Oct. 2014). Firechat in hong kong: How an app tapped its way into the protests, [Online]. Available: http://www.cnn.com/2014/10/16/tech/mobile/tomorrow-transformed-firechat/ (Date Last Accessed, Sep. 2, 2015).

[2] Firechat. (Sep. 2015). Chat live, no internet required, [Online]. Available: http://opengarden.com/how-to/ (Date Last Accessed, Sep. 5, 2015).

[3] M. K. Denko, T. Sun, and I. Woungang, "Trust management in ubiquitous computing: A bayesian approach," *Comput. Commun.*, vol. 34, no. 3, pp. 398–406, 2011, ISSN: 0140-3664. DOI: http://dx.doi.org/10.1016/j.comcom.2010.01.023. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366410000617.

[4] E. Ayday and F. Fekri, "An iterative algorithm for trust management and adversary detection for delay-tolerant networks," *IEEE Trans. Mobile Computing*, vol. 11, no. 9, pp. 1514–1531, 2012, ISSN: 1536-1233. DOI: 10.1109/TMC.2011.160.

[5] I.-R. Chen, F. Bao, M. Chang, and J.-H. Cho, "Dynamic trust management for delay tolerant networks and its application to secure routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1200–1210, 2014, ISSN: 1045-9219. DOI: 10.1109/TPDS.2013.116.

[6] T. A. Babbitt and B. Szymanski, "Trust metric integration in resource constrained networks via data fusion," in *18th Int. Conf. on Inform. Fusion (Fusion 2015)*, Washington, USA, Jul. 2015.

[7] S.-Y. Wang, "Distributed interplanetary delay/disruption tolerant network (dtn) monitor and control system," in *Aerospace Conf., 2012 IEEE*, 2012, pp. 1–9. DOI: 10.1109/AERO.2012.6187107.

[8] K. Wehrle, M. Günes, and J. Gross, *Modeling and tools for network simulation*. Heidelberg, Germany: Springer Science & Business Media, 2010.

[9] Omnet. (Oct. 2011). What is omnet++? [Online]. Available: https://omnetpp.org/intro/what-is-omnet/ (Date Last Accessed, Sep. 15, 2015).

[10] Ns3. (Dec. 2011). What is ns-3, [Online]. Available: https://www.nsnam.org/overview/what-is-ns-3/ (Date Last Accessed, Sep. 15, 2015).

[11] D. B. Maciej Mhleisen. (Jun. 2008). Openwns, open source wireless network simulator, [Online]. Available: http://www.ikr.uni-stuttgart.de/Content/itg/fg524/Meetings/2008-06-19-Stuttgart/07_ITG524_Stuttgart_Muehleisen.pdf (Date Last Accessed, Sep. 22, 2015).

[12] T. A. Babbitt and B. K. Szymanski, "Trust management in delay tolerant networks utilizing erasure coding," *IEEE ICC*, 2015.

[13] Y. Wang, S. Jain, M. Martonosi, and K. Fall, "Erasure-coding based routing for opportunistic networks," in *Proc. of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking*, ser. WDTN '05, Philadelphia, Pennsylvania, USA: ACM, Aug. 2005, pp. 229–236, ISBN: 1-59593-026-4. DOI: 10.1145/1080139.1080140. [Online]. Available: http://doi.acm.org/10.1145/1080139.1080140.

[14] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Commun. and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002, ISSN: 1530-8677. DOI: 10.1002/wcm.72. [Online]. Available: http://dx.doi.org/10.1002/wcm.72.