Zhiqi Lin

Augmenting Mobility Simulation by Public Transport: A Case Study for the ONE Simulator

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 19.11.2015

Thesis supervisor:

Prof. Jörg Ott

Thesis advisor:

Prof. Jörg Ott

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Zhiqi Lin

Title: Augmenting Mobility Simulation by Public Transport:
A Case Study for the ONE Simulator

Date: 19.11.2015 Language: English Number of pages: 6+65

Department of Communications and Networking

Professorship: Networking Technology Code: S3029

Supervisor: Prof. Jörg Ott

Advisor: Prof. Jörg Ott

The Opportunistic Network Environment (ONE) simulator is an extensible tool for evaluating Delay-Tolerent Networking (DTN) protocols and applications under different types of mobility patterns. To further increase the reality of the public transportation system modeling, we enable the modeling for metro system by extending the ONE simulator into a multi-plane structure as a first order approximation for a 3-dimensional (3D) world modeling.
As there are more and more public transit agencies open their timetable data to public, it is possible to utilize those open data and make the public transportation vehicles follow the real-world schedules. To achieve that, we developed tools for converting timetable data into the compatible format for the ONE simulator, and extended the movement model for public transport vehicles so that they can follow the provisioned schedule.
This master's thesis presents how the new features were designed, implemented and verified. In addition, we show sample simulations to demonstrate the impact of the new supported scenarios.

Keywords: Delay Tolerent Networking, simulation

Preface

I thank my supervisor, Professor Jörg Ott, for all his support and for giving me the opportunity to write my thesis on such an interesting topic.

I want to thank my wife for her patience during the most stressful times.

Maininkitie, 19.11.2015

Zhiqi Lin

Contents

Abbreviations

3D 3-dimensional. ii, 28, 59, 60

AODV Ad Hoc On Demand Distance Vector. 5

CCDF Complementary Cumulative Density Functions. 42, 44, 47

CRAWDAD Community Resource of Archiving Wireless Data At Dartmouth. 10

DSR Dynamic Source Routing. 5

DTN Delay-Tolerent Networking. ii, 1, 2

DTNs Delay-Tolerent Networks. 1, 2, 4, 5, 9, 13, 14

GMSF Generic Mobility Simulation Framework. 13

GTFS General Transit Feed Specification. 2, 28, 32, 35–37, 48, 49, 59, 60

MANET Mobile Ad hoc Network. 4, 9

MBM Map Based Mobility. 13

NS2 Network Simulator 2. 13, 15

NS3 Network Simulator 3. 13

OLSR Optimized Link State Routing. 5

ONE Opportunistic Network Environment. ii, 1, 2, 11–13, 15, 19, 28, 32, 33, 37, 38, 59, 61

OSPF Open Shortest Path First. 5

RIP Routing Information Protocol. 5

RWP Random Waypoint. 11–13

SPMBM Shortest Path Map Based Mobility. 11, 12, 41, 45, 47

TFL Transport for London. 32

TTL Time to Live. 41

WDM Working Day Movement. 2, 12, 13, 41, 45, 51, 57, 59, 60

WKT Well Known Text. 11, 60

1 Introduction

DTN is an approach to cope with extreme environments where may lack of continuous and instantaneous end-to-end network connectivity. This dis-connectivity can be caused by the mobility of the network nodes or nonsustained communication capability. The basic idea of DTN routing protocols is "store and forward", where packets are carried for a certain time before can be forwarded to other nodes closer to the destination.

To test and evaluate applications and protocols for DTN environments, simulation is often chosen as the first option. Contact traces are used to describe the network environment where DTN routing protocols are deployed. Contact traces describe when do two DTN nodes get in contact with each other and how long does the contact last. Contact traces can be derived from real-world traces or generated basing on some random process. The problem with real-world traces is the number of usable and publicly available traces is very limited. Therefore, they are not suitable for sensitivity analysis. On the other hand, random generated contact traces lacks validity behind them.

For Delay-Tolerent Networks (DTNs) whose connectivity is dominantly affected by node mobility, mobility simulation is another approach to generate contact traces. By simulating the movement of DTN nodes, the corresponding contact traces can be derived from that. As we have traces of the exact locations of the nodes and the radio signal range of them, we could know at any time if two points are in contact. The advantage of mobility simulation, compared with real-world traces, is the number of nodes and their behavior can be changed for different scenarios which is especially useful for sensitivity analysis. Compared to random distribution generated contact trace, the connectivity pattern derived from mobility simulation is more realistic.

The closer the movement models and scenarios are to the real world situation, the more realistic mobility traces can be generated, and the more convincing performance can be got from simulation for the DTN protocols and applications. The ONE simulator provides powerful mobility simulation capabilities and already includes an overground public transportation system model. A fact is that in most big cities, there are also underground public transportation systems. To add more reality to the transportation system model of the ONE simulator, we want to enable the underground public transportation system. However as the ONE simulator was designed as a 2D environment, situations that involves the third dimention of the real world can not be modeled. Therefore we chose to augment it with some extent of 3D capability and add the metro system basing on that. Another fact is that in all current existing public transportation modelings, the public transportation vehicles are moving on their routes with random pause times at stops and random movement speeds. There is an intension to reduce this uniformly distributed randomness in them. The reason is because in the real-world situation, their schedules do no have this uniformly distributed randomness but conform with the daily routine of citizens, which means at rush hours almost all vehicles are active for serving while less of them out of the rush hours. As there are more and more public transit agencies

open their timetable data to public, it is possible to realize this intention by utilizing those open data and making the public transportation vehicles follow the real-world schedules.

1.1 Aims and scope of this work

The thesis presents how we designed and implemented the new features on public transportation simulation, including adding metro system, enabling vehicles to follow specified schedule, and developing tool kits for converting real-world timetable data into compatible data to the ONE simulator. The objectives are:

1. Enable the modeling for metro system by adding 3D capability.
2. Change the relevant movement models so that public transportation vehicles(both overground and underground) can move according to provisioned schedule data.
3. Create tools for converting General Transit Feed Specification (GTFS) data into schedule data compatible to the ONE simulator.
4. Keep all implementation changes comply with the original design of ONE simulator as much as possible.
5. Study how much the metro system impacts the characteristics of mobility and network performance.
6. Study how much the real-world scheduled public transportation impacts the characteristics of mobility and network performance.

The limitation on the public transportation traveler keeps the same as the original design: travelers stick with the bus/tram/metro line configured for them. This also means they can not transfer between different routes nor switch between bus/tram and metro. Although the new features can work with many scenarios of city areas, considering a large amount of work has been done for developing new features, we limit the scope of the study described in objectives 5 and 6 to a scenario of a working week in Helsinki city which is mainly based on the Working Day Movement (WDM) model.

1.2 The structure of the thesis

The structure of the remaining thesis is as follows: Chapter 2 provides the background information of DTN, the field the simulator is created for. A couple of popular routing schemes are briefly discussed there because they are the core of DTN. Chapter 3 starts with the field of network simulation, and then turns to the more specific field of simulation on mobile DTNs. After that mobility modeling is discussed. Chapter 4 elaborates the existing design of the ONE simulator because it is the basis on which our decisions of changes are made. After the essential modules are discussed, it focuses on the public transportation system where most of

our changes have been made. Chapter 5 presents the design and implementation of our changes, especially on what and how the design choices were made. Chapter 6 describes the simulations conducted for objective 5 and 6. Chapter 7 summarizes the results of the work and draws the conclusions and insights. After that ideas about future work are presented.

2 Delay-tolerant networks and routing approaches

In this chapter, we describe DTNs in general, and discuss the role of simulation of routing protocols for DTNs.

2.1 Delay-tolerant networking and delay-tolerant networks

The terms “delay-tolerant networking” and “delay-tolerant network” were coined by Kevin Fall and published the first time in his paper proposing a networking architecture for interplanetary Internet[1]. This paper proposed a delay-tolerant networking architecture that interconnects normal networks that use TCP/IP with interplanetary network that with long signal propagation latencies and limited period of time for sending and receiving signals. The core entities of the architecture are DTN gateways whare are responsible for storing the inbound data when the outbound link is not available, and forwarding it when the outbound link becomes available. Two months later in his another paper [2], he applied a similar DTN architecture to interconnect many other special networks that lack of continuous connectivity. It is this paper that started the research motivation towards the DTN field.

While "DTNs" always refers to "delay-tolerant networks", the acronym “DTN” is used for both “delay-tolerant networking” and “delay-tolerant network”. One has to tell which it refers to from the context. In this thesis, we do not use the "DTN" acronym when it might result in ambiguity. Although there is no clear definition for DTNs in those two aforementioned paper. The RFC publication [5] describs them as networks that is occasionally-connected, subject to disruption and disconnection, and with frequent network partitioning and high-delay, while the delay-tolerant networking is referred to as the architectural approaches to address those issues in DTNs. Recently, the term “delay and disruption tolerant network” is also used to emphasize its characteristic of intermittent connectivity.

By examining networks in the perspective of long delays and intermittent connectivity, many types of networks fall into the category of delay-tolerant networks. Among them are Mobile Ad hoc Network (MANET)s, sensor networks, opportunistic networks, interplanetary networks, and even a kind of pigeon based network discussed by Jermiah Scholl and Anders Lindgren[6]. To be noted, these network types are not necessarily orthogonal to each other. For example most MANETs are also opportunistic networks. Some might confuse opportunistic networks with DTNs. We think opportunistic networks is a subset of DTNs. The only difference between them is that in opportunistic networks, the connectivity disruptions, or on the other side, the contacts, must be unpredictable while in delay-tolerant networks it does not have to be. Take interplanetary networks for example, the interruptions there are predictable because they are caused by orbital mechanism. So we consider interplanetary networks as delay-tolerant networks but not opportunistic networks.

In a delay-tolerant network the disconnection can be caused by the mobility of nodes. When nodes are close to and within radio range, they get in contact. When they move away, the contact ends. Disconnection can also caused by signal strength. When the signal of two nodes turns too weak, without any movement, the

established connection is torn down. This is quite normal for sensor networks where nodes only turn on communication function in intervals to save power consumption. Because of the disconnections, the network can not guarantee there is an end-to-end path between any two nodes. In worst case, between two nodes, there can be no end-to-end path existed ever.

2.2 Routing protocols in DTNs

DTNs are characterized by topology changes and intermittent connectivity. Routing in them can be quite different compared to traditional networks. Due to changing topology, proactive routing protocols, such as Open Shortest Path First (OSPF)[3] and Routing Information Protocol (RIP)[4] for Internet and Optimized Link State Routing (OLSR)[7] for Ad Hoc networks, operate poorly in delay-tolerant networks. The frequent changes of network topology result in floods of topology update messages, so that the network rarely arrives at convergence. Due to intermittent connectivity, reactive protocols popular in ad hoc networks such as Ad Hoc On Demand Distance Vector (AODV) routing[8], and Dynamic Source Routing (DSR)[9] protocol fail as well. Although they reduced the topology update messages by not trying to find a route until have the message to send, they assume an end-to-end path can be always found which is not true in DTNs.

DTN routing protocols solve this problem in an asynchronous approach. Instead of finding a route before transferring payload data, the payload data is directly forwarded based on local information hop by hop. Nodes carry packets until they get an opportunity to forward them. In such a "store-and-forward" fashion, packets are moved and stored throughout the network in hopes that it will succeed in reaching its destination[10][11][14].

The main considerations of DTN routing are delivery delay and resource consumption. Delivery delay is the time it takes to route a packet from the source node to the destination node. Resource consumption includes storage consumption, computing capacity consumption(reflected by energy consumption) and bandwidth consumption. Resource consumption can be briefly measured by the number of transmissions of an end to end delivery.

There are different DTN routing schemes and protocols proposed for opportunistic environment where contact opportunities are unpredictable. They all follow the "store-and-forward" fashion. They only differentiate on how do they make forwarding decision. The epidemic routing scheme[15] forwards messages to all nodes it encounters, while the direct-transmission scheme[16] only forwards a message to its destination node. To see their forwarding strategy from another point of view, they are actually different on the number of copies. For the epidemic scheme, it allows up to N copies of each message to be spreaded in a network of N nodes. For the direct-transmission scheme, at any time, there can be only one copy of each message in the network, either on the source node or on the destination node. That is why it is also referred to as single-copy scheme. As less copies of a message leads to smaller chances of successful delivery, the single-copy scheme normally suffers from low delivery ratio and long delivery delay. However with too many copies, it does

not always mean better performance. The more copies a network has, the more storage, computing capacity and transmission bandwidth is required. In real-world situation, the resources could easily become insufficient when the number of message copies are not limited. When storage is full, no new message can be received. When too many messages needed to be transferred between nodes, not all messages can be transmitted during the limited contact duration. Also the battery of nodes are drained faster for processing more message copies. When any of these three situation happens and causes delivery ratio drops and longer delays, we call it a network congestion. The Spray and Wait protocol[17] and Spray and Focus protocol[18] solve this problem by making a trade-off between the ideas behind the epidemic scheme and the direct-transmission scheme. They limit the number of the message copies that spreaded in the network to a specified number. So by tunning the number of copies, it can find the maxium number of copies that does not lead to a network congestion. Therefore a better or at least not worse routing effeciency can be reached than that by the epidemic scheme and the direct-transmission scheme. MaxProp[10] uses another strategy to solve the resource consumption problem confronted by the epidemic scheme. Rather than avoiding the situation of insuffient resources, it tries to achive an acceptable performance when resources are insufficient by giving up messages wisely. When two nodes encounter, it determines which messages should be transmitted first and which messages should be dropped first when the storage is full. PRoPHET protocol[19] tries to increase the routing efficiency from a different point of view. With PRoPHET, each node locally maintains an encounter history, and it only forwards a message to nodes that have a good enough chances of contacting the destination node again. The assumption behind that is for each node, the probabilities of encountering a certain node is different. And the more frequently two nodes have encountered before, the more likely they will encounter again in the future.

2.3 Summary

Since the ONE simulator is created for evaluating the DTN protocols and applications, knowing the main ideas in the DTN field is neccessary for a better understanding on the ONE simulator. In this chapter, we firstly presented the concept of delay-tolerant networking and delay-tolerant networks, and listed some subclasses of delay-tolerant networks as examples. Then we discussed the routing problem in delay tolerant networks and the basic idea of the solution. Finally we elaborated the differences between the ideas of some popular DTN routing schemes.

3 Simulation of delay-tolerant networks

This chapter starts with a brief description on general network simulation. Then it discusses the characteristics of simulation for delay-tolerent networks, especially on the mobility modeling.

3.1 Network simulation

To test and evaluate the performance of a new network, building up a real network environment for the experiment is both time consuming and costy. For a fixed network, it means to assemble multiple network devices, data links physically as a network and install all protocols and applications to make it work as a network. For mobile network, it is even harder because it needs to recruit a bunch of people with wireless devices with the protocols and applications installed, and keep the experiment going on for some days. There are situations that preparing a real network environment for testing and evaluation is even not possible, for instance evaluate a new routing protocol for a interplanetary network. That is why network simulation is normally the first choice for trying new ideas out. In network simulation, the network elements are modeled and assembled virtually in software. By introducing artificially generated traffic or traffic captured in real networks, the behavior of the network can be recorded and studied. Apart from testing prototypes of nonexistent protocols, simulation is also useful in controlled experimentation with scenarios hard to be configured in existed real networks.

Although the terms simulation and modeling are usually used interchangably, we prefer to consider modeling as the process of building a model which is a reduced representation of the considered system based on simplifications and assumptions, while simulation as the process of using the model to study the behavior and performance of the represented system under certain scenarios.

Modeling can be classified in two types: discrete-event model and continuous model. In discrete-event model, the state of the targeted system changes only at a countable number of points in time. These points in time are the ones at which the event occurs or state changes. In continuous model, the state changes in a continuous way, and not abruptly from one state to another, which means it has infinite number of states.

3.1.1 Basic process of a network simulator

Network simulators are frameworks in which the desired network configurations can be assembled virtually in software, virtual traffic loads can be introduced, measurements can be taken, and tools might be provided to facilitate analysis. In the field of network simulation, discrete-event model is used most. There are two types of time management for discrete-event model - event scan and periodic scan. With event scan, the simulator has a variable representing the time in the simulation environment, called the simulation time. It is initialized to an arbitrary value, usually zero, and advanced in a nondecreasing fashion as the state of the system progress.

The simulator also has a sorted list maintaining pending future events. The events can be loaded from provisioning data, generated as system progressing or triggered by previous events. The list is sorted ascending with time. With simulation time and event list initialized, the simulator can goes into its execution loop:

1. if there is no more pending future events, terminate.
2. Remove the earliest pending future event from the list.
3. Set the simulation time to the time stamp of the just removed event.
4. Process the state changes that occur caused by the event action. These state changes might further lead one or more additional pending future events to be created. For example, a movement event(two nodes meet each other within radio range) might cause a contact event in future(if link setup time is not ignored) and further cause a couple of transmission events(exchange the messages they have).

With periodic scan, or so called time slicing approach, the simulation time is advanced with a predetermined unit. The system is examined to determine whether any events occurred during that interval. If no event occurred, no action is taken; otherwise the event or events are processed. The events that occurred during a given interval are treated as if these events occurred at the end of that interval. If the time increment is too large, and multiple events occurred in the same interval, the order of the events will be lost. So the time increment must be chosen small enough so that the probability of multiple relevant events occurring during a single interval is small.

Apart from the main loop for triggering changes and proceeding simulation, a simulator has to provide building blocks, called state objects, for changes to apply to. State objects represent objects in a network such as nodes, network interface cards, links, queues, protocols, data packets, and applications. Objects in information form, such as applications, data packets, protocols, and queues, can be represented by state objects with a real implementation or a simplified implementation because they are already abstraction of states and behaviors. While physical objects, such as links, network interface cards, and nodes, can only be represented by state objects in a simplified way.

3.1.2 Fidelity range

Fidelity range is the level of detail characterized in the models. We always want the model to imitate the system under study as closely as possible. That is how the result can be accurate enough. However, a fully realistic simulation is not possible. We have to maintain the complexity at a tolerable level so that you don't have to wait, for instance a year, for the simulation result. Models always disregard some details of the real-world system by abstraction and only take limited number of characteristics and behaviors into consideration.

To determine which details to implement is a trade-off between accuracy and simplicity. Should real implementations stop at network layer, or data link layer, or physical layer, or electrons? This inevitable design choice is faced by every simulation designer. Good choices require good understanding about the system under study and well defined objectives of the simulation that the model is built for. For example, in wired networks, point-to-point links can be abstracted by bandwidth, delay and a queue, with framing, coding, and transmission errors ignored. But in wireless networks, this abstraction will lead to incorrect results because it doesn't consider concurrent transmissions for wireless communication channels [20]. In that case, lower layers need to be implemented. Sometimes properties can be ignored if they have too little affect on the metrics we want to analyze. For example in wired network, if a very large portion of end-to-end latency for an application depends on the network layer, it is safe to ignore the small latency effects introduced by data link layer. The discrete-event method falls into this case as well. It simply means we are only interested in the state of the system being simulated at discrete points in time and can safely ignore state at times in between.

3.2 Characteristics of mobile delay-tolerant networks simulations

Delay-tolerant networks with node mobility accounts for a large part of delay-tolerant networks. For example, in interplanetary networks, mobile ad hoc networks, mule networks[12], and some sensor networks[13], the nodes are all with mobility. The study on mobility in MANETs had started earlier than the emergence of the term delay-tolerant networking. The MANET researchers found that mobility models play a very important role on the protocol performance in MANETs, as stated in [21]. Hence, mobility simulation is an import part of simulation of DTNs which is quite different from general network simulation. The mobility causes the contacts between nodes. And it is during these contacts that messages get forwarded. To what extent of details does a mobility pattern needs to be modeled? The most straight forward answer is to have the locations of nodes at any time point. Another simplified approach is based on the assumption that the contacts resulted from the mobility can completely reflect the effect of mobility on the system. It takes contacts as the abstraction of mobility, and ignore the details of location and speed of nodes. The contact data is often referred to as contact traces. It describes when two nodes contact with each other, which means when the link between them is up and when it turns down.

Contact traces can be generated by some stochastic process, for instance generating the occurences of contacts as a Poisson process, and the durations of contacts follows a uniform distribution. However, the effects of mobility pattern are missing from this kind of abstraction. Considering two mobile opportunistic DTNs, one consists of nodes with high locality, where nodes tend to spend most of their time within a small area, while the other consists of nodes with low locality. Even though contact traces resulted from these two networks can follow the same stochastic metrics, their simulation results are totally different, as higher locality contributes to lower

delivery ratio. The reason for the fidelity losing is that, although contact trace reflects the effect of mobility, the statistics metrics of contact trace fails to. Therefore, according to the discussion in Section 3.1.2, stochastically generated contact traces is a bad abstraction of mobility, for the loss of some key properties of mobility.

Contact traces can be also derived from real-world movement traces. There are public available data sets of user traces obtained by real-world experiments. Besides contact traces, experiments with location awareness nodes can collect mobility traces which contain the locations of nodes at given timestamps. There are many such data sets at the websit of Community Resource of Archiving Wireless Data At Dartmouth (CRAWDAD), for example, mobility traces of taxi cabs in Rome, mobility traces of pedestrian in a part of downtown Stockholm, and contact traces collected from Android phones in a campus. The real-world traces do capture the most realistic movements, however the scenarios of the real-world experiments might not be what you want for your simulations, especially for sensitivity analysis where experiments need to be conducted on different conditions. For example, the real-world traces might not have enough nodes or the nodes are too sparsely located in the experiment area, compared to what your simulation requires.

Another approach for generating contact traces is to simulate the movement of the nodes, which is referred to as sythetic mobility models. With this approach the number of nodes and their behaviors can be easily varied for different scenarios. Even with simple synthetic mobility models, the derived contact traces are easily more realistic than the contact traces generated as stochastic process. However building a good and some kind of "realistic" syntheic mobility model is not always easy, which we will discuss in the next section.

3.3 Synthetic mobility models

Many synthetic mobility models have been proposed by researchers. Some random based mobility models were developed firstly, for instance the Random Walk model and Random Waypoint model. Because of their simplicity, they were widely used in simulations. However in the real world scenario, most nodes are not move randomly. To capture various mobility characteristics, more complex mobility models are developed.

3.3.1 Random based mobility models

The Random Walk model is the simplest mobility model. It was originally proposed to emulate the unpredictable movement of particles in physics, also referred to as the Brownian Motion. It is widely used in simulations in which the movements of mobile nodes are completely unpredictable. In the Random Walk, every node firstly randomly and uniformly chooses a direction and a speed from some configured range, and secondly moves in that direction for a constant time or a constant distance. It repeats these two steps for the length of the simulation. When the node reaches the boundary of the simulation field, it can wrap to the opposite boundary and continue its movement with the same direction and speed, which is called wrapping

approach. With reflection approach, when reaches any boundary with an angle of θ, the node is bounced back to the simulation field with the angle of π - θ. The Random Waypoint (RWP) model was first proposed by Johnson and Maltz[21]. It is often used in protocol evaluation because of its simplicity and wide availability. The mobility pattern is as follows: every node randomly chooses a destination location in the simulation field. It then moves towards the destination with a constant speed chosen uniformly and randomly within a range. Upon reaching the destination, the node pause for a fixed period before selecting another random location and speed. This behavior is repeated for the length of the simulation. Although the movement pattern in the RWP has strong randomness, the spatial node distribution is non-uniform. As observed by Bettstetter[22] and Blough[23] respectively, the node distribution is transformed from uniform distribution, the initial state, to non-uniform distribution as time elapses. When it reaches the steady state, the node density is maximum at the center region, whereas almost zero around the boundary area, which is called non-uniform spatial distribution. This is because, taking a circular area as an example, inside the circle, the longest straight line segment started from the current waypoint must go through the center of the circle. As destinations are uniformly distributed, longer line segment means bigger possibility. That is why nodes are more likely to either go towards or go through the center of the area than towards or through other points inside the area. This is mathematically explained in the work[24]. The Random Direction model is a variation of the RWP model. It is proposed by Royer, Melliar- Smith and Moser[25]. This model is able to overcome the non-uniform spatial distribution by, instead of selecting a random destination, choosing a direction randomly and uniformly to move. The node moves along the direction until it reaches the boundary, then it pauses for a while and chooses a direction again.

3.3.2 Map base movement model

Unlike the RWP model where nodes can move freely, in city scenario, mobile nodes are only allowed to move on the pathways. To integrate such geographic constraints into mobility model, a predefined map is needed, and nodes are restricted to the pathways in the map. The pathway graph can be randomly generated or carefully derived from a map of a real city.Tian, Hahner and Becker et al. [26] utilize a random graph to model the map of city. The ONE simulator provided a graph of of Helsinki city map in its first release of. The map graph is presented in the Well Known Text (WKT) format, consisting of map points connected by links. A node moves in the way that randomly chooses one of the directly connected map points to move to.

3.3.3 Shortest Path Map Based Mobility model

The Shortest Path Map Based Mobility (SPMBM) model works nearly the same way as RWP model except that all movements are restricted by map. Instead of randomly selecting a point as the next destination in simulation area in Random Waypoint model, in the SPMBM model the selected destination must be a map

point on the map. Also, unlike just move to the destination straight forward, nodes in the SPMBM model need to move along the shortest pathway to the destination calculated by using Dijkstra's algorithm. This model has a similar problem about spatial distribution as RWP has. Area with dense map points likely attracts more mobile nodes than area with less dense map points. So besides the distribution of pathways, the way of construction of the map also affects the spatial distribution of mobile nodes. For example, a street constructed with 20 connected map points will attract more mobile nodes than the same street constructed with only 10 connected map points.

3.3.4 Community Based mobility model

The Community Based model[27] takes account the effect of social relationship between nodes. It works like the RWP model that in each movement a node select a destination to move to. The difference is that, instead of randomly selecting the destination, nodes are more likely to go to places where most of their friends go. In this model, the friendships are described by an interaction matrix as an input. The interaction matrix indicates the relationships between any two nodes by a value between 0 and 1. The larger the relation value is, the closer relationship these two nodes have. The simulation area is divided into small areas called squares. The extend of social attractivity of a square to a node is represented by the averaged value of the relationship values between the nodes moving towards the square and the node in consideration. The larger social attractivity the square has to the node, the more likely the node selects a random destination inside the square.

3.3.5 Working Day Movement(WDM) model

The WDM model depicts the movements of working life in city in a very detailed way. It was first included in the second release of the ONE simulator. In contrast with most mobility models, heterogeneity is the most notable feature of WDM model. It is map based, and heterogeneous in both time and space by combining different mobility models. As people in city life move differently in different part of day, the WDM model is composed by three different sub-models: home activity model for evening and nights staying at home, office activity model for working hours, and evening activity model for evening activities. Nodes can be moving in different models at the same time, which embodies its spatial heterogeneity. There are nodes inside office with indoor movement meanwhile other nodes still on the way to office. The nodes on the way can be walking in or driving car or on bus or trams. And there are bus nodes themselves with wireless communication ability of long radio range. Some important real-world mobility characteristics are reflected by the model naturally. The community and social relationship based movements happen when a set of nodes are doing the same activity in the same location. For example nodes with the same office location are colleagues and nodes with the same home are family members. The group movement happens on the traveler nodes on the same bus node. The model is validated by comparing its resulted contact statistics characteristics with that from iMote trace which comes from real-world experiments.

3.4 Simulators for delay-tolerant networks

There are network simulators, mobility simulators and simulators supporting both networking simulation and mobility simulation. Network Simulator 2 (NS2) and Network Simulator 3 (NS3) are general purpose network simulators. They provide detailed protocol implementations for lower layers but only basic mobility models such as the Random Walk model and RWP model. There are simulators dedicated for DTNs developed within universities and other research institutes. Dtnsim and Dtnsim2 have been developed based on Java at University of Waterloo. They focus on the routing part of network simulation and leave the task of mobility simulation out of their scope by taking contact traces as input. As lack of location details, they can not be used for protocols or applications utilizing location information. Generic Mobility Simulation Framework (GMSF) is a moibility simulator, it contains the RWP model and the Map Based Mobility (MBM) model based on Swiss geographic information system. Due to copyright restrictions they are not allowed to release the road information from the Swiss GIS landscape model. However they plan to support the import of maps from OpenStreetMaps in a future release. MobiSim is a comprehensive mobility management tool. Besides providing many mobility models, it provides a graphical UI for map designing, and tools for analyzing the movement pattern from different perspectives. It also supports composing mulitple mobility models into one simulation area. In addition, it supports 3D map and running available mobility models in 3D environment. However, it is not easy to reuse existing map data. The ONE simulator supports both networking simulation and mobility simulation. It was developed for DTN simulation by Networking Laboratory at Helsinki University of Technology(Aalto University School of Electrical Engineering). The ONE simulator, as a network simulator, like Dtnsim and Dtnsim2, includes only the minimum possible details of network stacks. It abstracts the data link layer by bandwidth, delays and queues. This is fine when the nodes are sparsely distributed, which is the normal situation for DTNs, because the effect caused by concurrent transmissions can be safely ignored. When the simulation scenario with nodes are not sufficiently sparse and the effect of concurrent transmissions can not be ignored, lower layers of protocols need to be implemented. Researchers have solved this problem by combining the strength of the ONE simulator on mobility simulation and the advantages of NS2 and NS3 on detailed protocol stack implementations[28][29]. In this case, the ONE simulator is used as a mobility simulator while NS2 and NS3 as network simulators. As the ONE simulator provides the WDM model, it is especially good at simulating mobility at city areas.

3.5 Summary

In this chapter, we firstly introduced the background of network simulation. Then we showed how discrete- event simulator works in general. After that, we discussed the decision every simulation designer must face to – how to determine the level of details in modeling. Then we discussed the characteristics of delay-tolerant network simulation and presented popular mobility models. At last, we introduced some of

the available simulators for DTNs. This serves as a basis for our discussion on the design of the ONE simulator in the next chapter.

4 The design and implementation of th ONE simulator

This chapter elaborates the design of the ONE simulator, to lay the basis for discussing the changes for public transportation system in the next chapter. After the core modules are presented, it focuses on the public transportation system where most of our changes are made.

The core of the simulator is an discrete event simulator. To make it suitable and efficient enough for simultaneous movement and routing simulation, it uses a mixture of time slicing and event scan approach. Time slicing approach is used for location changes while event scan for message events. It provides movement simulation, routing simulation, visualization and reporting in one program. Because of its modular design, its modules can be easily reused. For example, the movement module can be used alone to produce movement traces that can be used as input for external routing simulators such as dtnsim or NS2. The routing module can be used alone as well by using external movement traces or contact traces. It supports multiple types of nodes in a simulation by configuring nodes in groups. A node group shares a set of common parameters such as mobility model, radio range, buffer size and some mobility model specific attributes. In this way, simulation with pedestrians, cars, bus and passengers on bus is possible.

The advancing of the simulation with the simulator is basically motivated by two types of changes, the message events and the location changes of nodes. In a macro perspective, the system runs in a way that message events trigger the data traffic, location changes change the networking topology which in turn influence the journey of the messages, meanwhile reporting module and visualization module are listening to those changes and note them down into to logs, reports and graphs.

4.1 State objects

As stated in Section 3.1.1, the building blocks of the system under simulation are state objects. They are where changes apply to. Figure 1 lists all the state objects in the ONE simulator. The DTNHost objects are created to represent DTN capable hosts. Location information is represented by the Coord objects containing fields x and y as the horizontal and vertical position. The movement of a host is described by the Path objects. A Path object consist of a list of coordinates describing the route to move along, and a list of speeds indicating the speed of the movement between each two successive coordinates. A host gets the paths to follow by asking its dedicated MovementModel object. A host may have one or more NetworkInterface objects representing the network interface cards. It abstracts the physical and data link layer by radio range and transmission speed. It also manages the established connections and does the connecting and disconnecting actions. A connection is represented by the Connection object. It is used for transferring messages and maintains the state of the transfer. It is the one that knows if a message is transferred already or not, and when a transfer completes. Each host has a MessageRouter object running as its DTN routing protocol. A MessageRouter manages a buffer

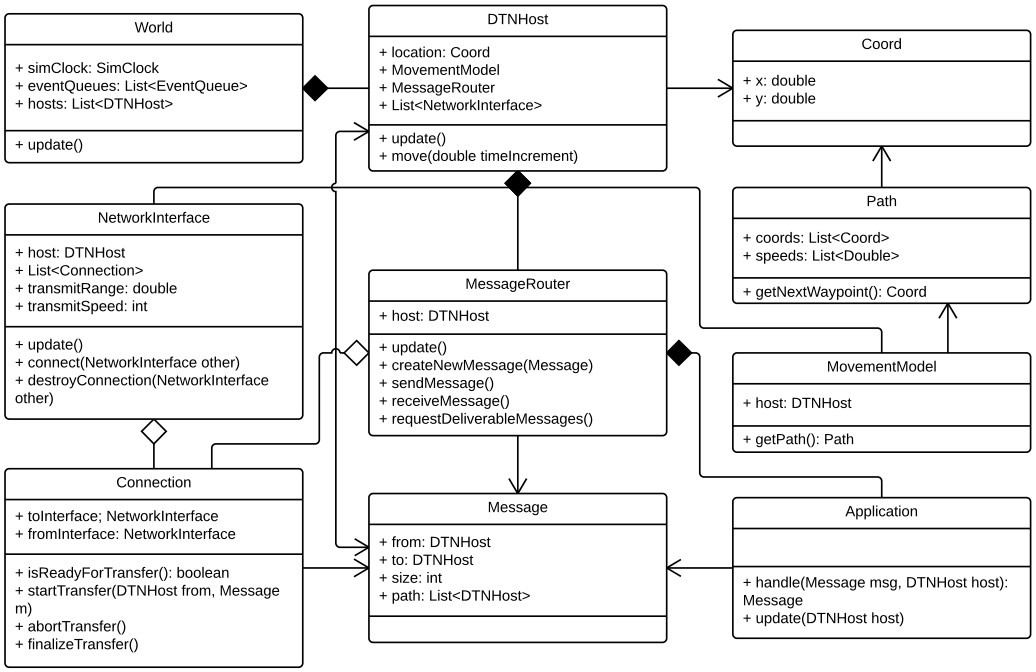

Figure 1: The state objects of the ONE simulator.

and the messages stored in the buffer. And it is responsible for initiating message transfers in a way based on the specific routing protocol it implements. The message transfer initiations can be triggered by external message events, new up connections, or communication intentions of applications which are represented by Application objects. When a message destined to this host is received, the MessageRouter object notifies the Application object by invoking its handler method. In handler method, the Application object defines the action for the received message. The messages transferred among hosts are represented by the Message objects. As in a routing simulation, the content of a message is not important, a Message object does not carry the data of a real message but just indicates the size of the message. The whole system is represented by the single World object. It contains all hosts, and maintains the simulation time and the coming events. By invoking the update behavior of the World object, the simulation time advances for a time interval. Then all the location changes during this time interval are calculated and applied to each host. Meanwile all the events that happen during the interval are triggered.

The flow chart in Figure 2 depicts the the main loop of a simulation which is also what the World object does in an update behavior. In step 1, the World object yields the end time point of this time step by adding current simulation time with the fixed time interval. Then, in step 2 it checks the external events queues to see if there are any events will happen during this time step. If there is not, it goes to step 5 to move the hosts by making each host updating their own locations to where they should be at the end time point of this time step. If there is any message creation events in step 2, it firstly processes the events, which means the MessageRouter object of the sending host will create and store a Message object there. While using contact

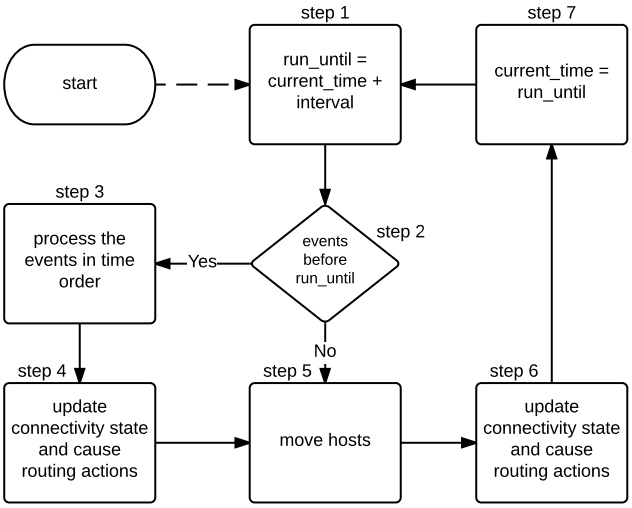

Figure 2: The main loop of a simulation.

traces instead of using ONE's own mobility simulation, the events can be connection events describing the changes on connectivity topology. That's why there is step 4 to actually apply those changes to the corresponding NetworkInterface objects and further cause routing actions on MessageRouter objects. After step 5 the location changes of hosts may lead to further connectivity topology changes, that is why in step 6, the World object needs to update the connectivity state again. After that, all the changes are applied to the state objects. The World object set the current time to the end of this time step and goes to next loop until the configured termination time.

4.2 Supporting modules

Besides the state objects, there are supporting modules for defining simulation scenario, inputting external events, recording the statistics of interested events and state changes.

4.2.1 Setting module

To define the scenario of a simulation means setting the properties of each type of state objects of the system in one or more setting files. The format of setting files is line delimited key-value pairs. The value can be a string, a number, or a class name. The key is composed by a property name prefixed with a name space. The idea of using name space is to indicate what state object this property is for. It is straight forward to think about having the class names of state objects as the name spaces, for example, a name space named "host" is for DTNHost state object. To enable different host groups to have different values for the same properties, the name space of a host group has to follow a pattern of "group" with the group number as the suffix. For example, if there are 4 groups, the name spaces of them have to

be “group1”, “group2”, “group3” and “group4”. Further, to make it convenient to set some common default properties for different groups, the setting module provides an option of secondary name space: if a property is not set in the primary name space, it is then looked up from the secondary name space. So the convention of setting host groups is to have a common name space containing the properties of default values, and set it as the secondary name space for each groups. In this way, each host group only needs to set its customized properties in its primary name space to override the default ones.

The simulator uses class Settings to map all these settings stored in files into memory. The class contains all the logic of loading properties from those setting files into memory, arranging them in coherent structures and offering them for creating state objects in a convenient way. It has to be initialized before use. In the initialization phase, all properties in the setting files are loaded in as a hash map. After that, instances of class Settings can be created with different name spaces. As a name space is a kind of identity of a state object, an instance of class Settings is the representation of the properties of a state object. By having all the properties of the state objects in memory, the simulator can instantiate the state objects. Since the specific class names of some of state objects, such as MessageRouter, NetworkInterface and MovementModel, are not known in application logic but defined in the settings files just like other normal properties, the simulator has to load those classes dynamically by using Java reflection mechanism.

4.2.2 External events

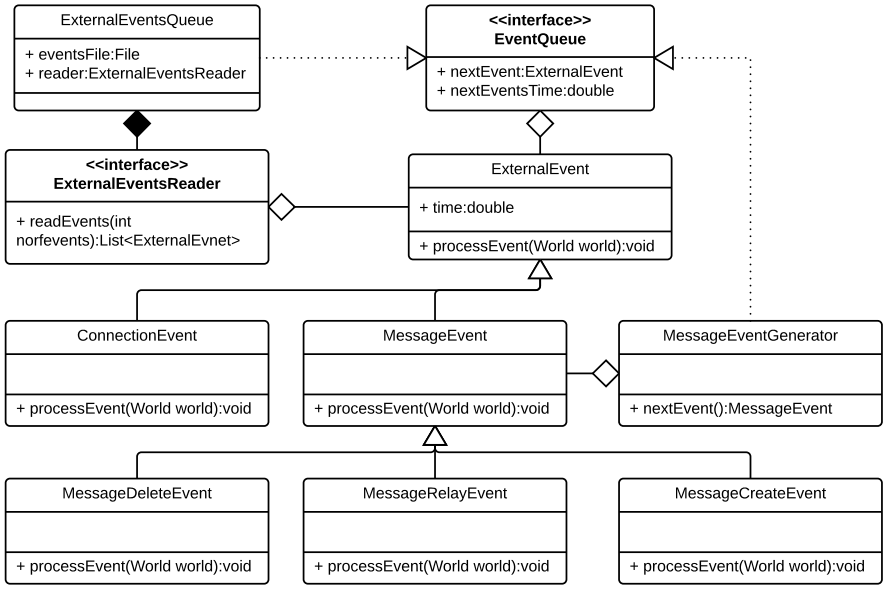

Figure 3: The class diagram of external events.

External events can be divided in two types, connection events and message events. Any event knows how itself should be processed by providing the behavior

processEvent. Events can imported to the simulated system by events generators or events readers which read in the events listed in trace files. In a trace file each event is described by a single line with a timestamp, a event type id, ids of the affected hosts and optional parameters for specific event types for example a MessageCreateEvent has to indicate the message size. Events are chronologically ordered in the trace file. By supporting trace file input, the simulator can utilize real experiment data or results from third party program for routing simulation. To further ease this use case, the simulator provides scripts to convert those third party data into its expected format. Event generators can dynamically create events with certain statistical characteristics. For example the class MessageEventsGenerator creates message creation event uniformly distributed on message size and inter-message intervals. The ranges of message size and intervals are configured under the namespace 'Events'. To be noted, generators normally creates only message creation events and let MessageRouter objects to decide when to relay and when to delete, while trace files can contain both connection and message events. This is because, in the ONE simulator, connection events are actually generated from the mobility of hosts. In principle, it can have generator producing random connection events basing on stochastic process. However, as discussed in Section 3.2, mobility driven connection events are more realistic and meaningful. External Events are provided to the World object through the interface EventQueue which guanrentees each time it returns the event that will happen in the nearest future. As the World object has a list of EventQueue instances, it means multiple simultaneous event sources are supported. By calling the method nextEventsTime on each queue and doing comparison, events from different sources are chronologically interleaved.

4.2.3 Reporting module

Reporting module is responsible for recording certain types of external events and state changes(internal events) that are interested for the purpose of the simulation. It is a bunch of different subclasses of the class Report for creating different types of reports. Each of them observes for specific events and produces a report based on its observation. It either logs information about the event to the report file or store the information in memory with some data structure and creates a summary when the simulation is done. The observer design pattern is used to notify report classes when something happens.

As shown in Figure 4, there are 5 listener interfaces defined corresponding to 5 types of events: events generated by application, message events, connection events, movement events and the fixed time step update event. The subclasses of the Report class implement the listener interfaces according to the event types it is interested at, and define what to do about the event. State objects hold the list of listener implementations, and they are responsible for invoking the related methods of listener implementations when an event occurs. The subclasses are configured in the setting files and loaded by the setting module and then added to the listener lists hold by state objects.

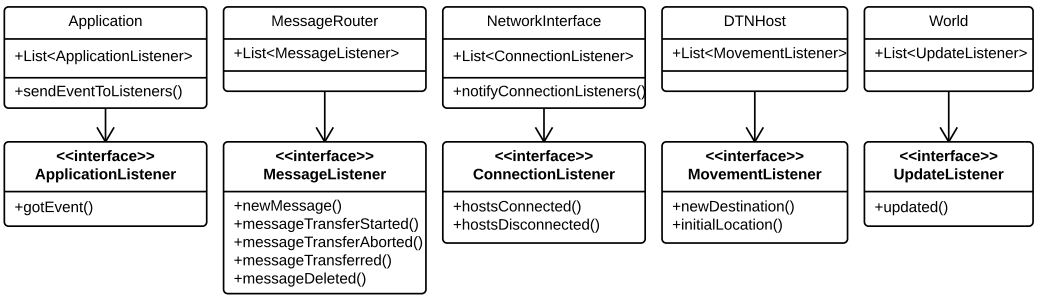

Figure 4: The class diagram of reporting module.

4.3 DTN routing module

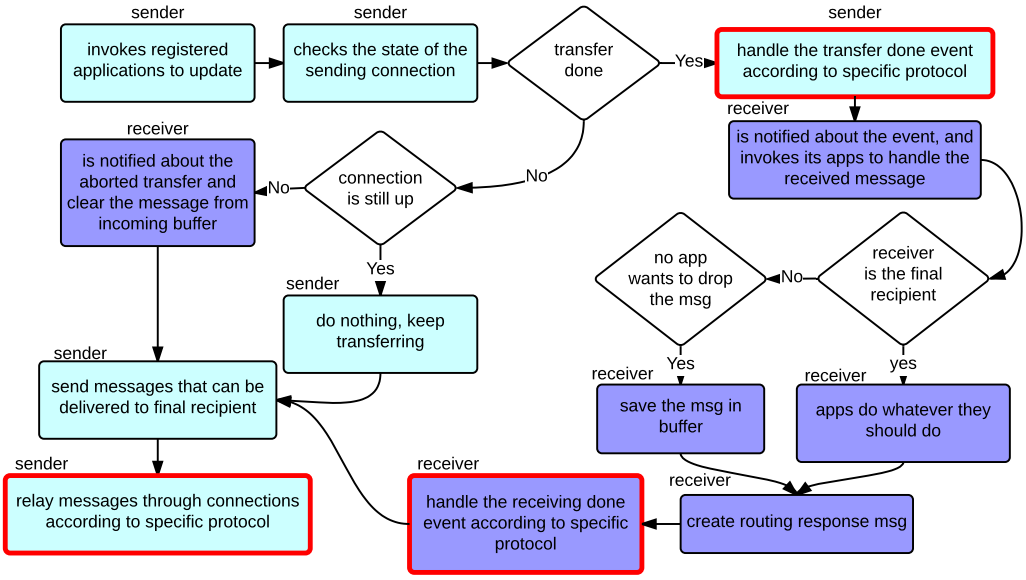

Figure 5: The flow chart of routing.

The routing module decides how messages are delivered to final recipients hop by hop. The simulator has 13 implementations of well known DTN routing protocols. Each implementation is a subtype of class MessageRouter. Although different subtypes act differently, they have common interfaces they providing to upper layer and the ones they consuming from lower layer.

MessageRouter objects are the service provider of message events and Application objects. When new message events occur or applications have communication intentions, they use MessageRouter objects to send messages, which means MessagRouter objects put the messages into the buffers, so that when two hosts meet and connect to each other, they have something to exchange. In addition, Application objects need MessagRouter objects to notify them when messages are received, so they can take actions like sending a response or not if it is the final recipient.

MessageRouter objects are the service consumer of Connection objects. Con-

nection objects provide the message transfer status - transfer done, transferring or transfer aborted - to MessageRouter objects so that they can act accordingly. MessageRouter objects also need to ask Connection object to do the actual message transfer between two directly connected hosts.

The flow chart in Figure 5 describes the sequence of an update on a MessageRouter object. After notifying Application objects to update, the MessageRouter object checks the status of the ongoing message transfer and acts accordingly. Then it tries to transfer messages to their final recipients if there are any directly connected to it. After that, it starts to relay messages in a protocol specific way.

Despite of the common interfaces and work flow, there are some points for protocols to act differently. The protocol specific steps are elements with red frame in the flow chart. One of them is when the MessageRouter object is dealing with message relaying. The reason is, for example, the direct-transmission protocol does not do relay messages, while the epidemic protocol relays all messages to all connected hosts and does not remove the transferred messages. The Spray and Wait protocol only relays messages that has at least 2 copies. Another two protocol specific steps are when the sending MessageRouter and the receiving MessageRouter are handling a successful message transfer respectively. For example in the Spray and Wait protocol, upon a successful transfer, the sender needs to reduce the number of copies of the message according to the mode it is running, while the receiver needs to decide the number of copies it can have according to the mode as well. There is another protocol specific point outside of the update work flow. While most protocols only care about having access to connections the network interface currently established and do not care when a new connection is turned up, some protocols are interested in these contact events. For example the PRoPHET protocol has to update its encounter history(discussed in Section 2.2) when two hosts connected. The simulator adds hooks in NetworkInterface object to invoke MessageRouter to deal with these contact events.

4.4 Movement module

The movement of hosts is achieved by the collaboration of DTNHost and MovementModel objects. The MovementModel object is responsible for providing initial location for host, and a Path object whenever the method getPath is called. As the path must start from where the previous path ends, the MovementModel object has to remember the end point of the just provided path. The DTNHost object is the one that deals with the actual location changes. It maintains its current location, and asks a new Path object from its MovementModel object. By having the current location, the next waypoint and the speed it should be at provided by the Path object, the DTNHost object can calculate where it will be after the given time interval. In many movement models, the hosts have to pause after a path is completed for some time. To achieve this, all MovementModel classes have to implement a nextPathAvaible() method that returns the simulation time of when the next path is available, while DTHHost objects, on the other hand, will not call method getPath() until the current time is later than the time returned from the

method nextPathAvailable().

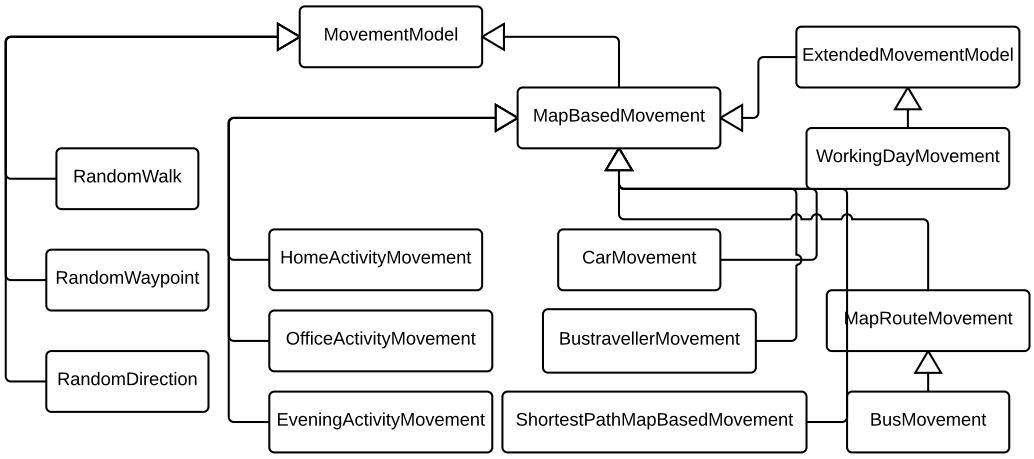

Figure 6: The class diagram of movement models.

The class diagram in Figure 6 shows the hierarchy of MovementModel classes the simulator provides. The basic models like RandomWalk, RandomWaypoint and RandomDirection can easily put their algorithm into its getPath() method. They just use the properties like maximum and minimum speeds and the size of the area, retrieved by the Setting module, to generate their random path. The MapBased-Movement models are more sophisticated.

4.4.1 MapbasedMovement models

The MapbasedMovement classes have to first construct their map structure from one or multiple route files before producing Path objects. The map structure is a collection of MapNode objects. Each MapNode object knows its own coordinates and the neighboring MapNode objects. Two neighbor MapNode objects means there is a straight line routes between them. So a collection of MapNode objects forms the routes that hosts can move along. The route files are in the format of Well-Known Text(WKT). The MapBasedMovement class utilizes a WKT reader for parsing WKT files while only POINT, LINESTRING and MULTILINESTRING objects are supported because that is enough for current needs. It is the route file provider's responsibility to make sure the points in WKT files are all connected, because isolated points and routes do not make sense. The MapBasedMovement class also helps to check the connectedness after all routes are added to the map structure. As it is necessary to make sure no pedestrians walk on free way and no cars or buses moving a pedestrian paths, the simulator needs a mechanism to mask some type of routes for certain types of movement models. This is achieved by having a node type property with a integer value for the MapNode class. In addition, it allows MapBasedMovement objects to specify the interested node types to indicate what types of routes are visible for the hosts with this movement model.

The MapBasedMovement class provides paths that the next waypoint is randomly selected among the coordinates of neighboring MapNode objects. The number of MapNode objects it will pass by is randomly generated as well. It prevents hosts to go back by excluding the previous MapNode objects from the random selection for the next waypoint. ShortestPathMapBasedMovement class each time produces a path to a randomly chosen point of interest. It uses the DijkstraPathFinder class to yield the shortest path. A point of interest is also represented by a MapNode object. The collection of points of interest is managed by the PointOfInterest class. On its instantiation, it imports MapNode objects from WKT file containing POINT objects with the help of the WKT reader. The MapRouteMovement class is used for fixed route movements like public transportation. It has a predefined route imported from a route file in WKT format. The route is consist of stops. And each path it produces is the path from one stop to the next stop calculated by the DijkstraPathFinder.

4.4.2 WorkingDayMovement model

The ExtendedMovement abstract class is the one that makes heterogeneous mobility possible. It does not generate paths by itself but delegates the task to other movement models. We refer to the movement models that do the actual work as worker models. The ExtendedMovemtn class has a variable refers to the current worker model it is using and has a mechanism to switch smoothly among different worker models.

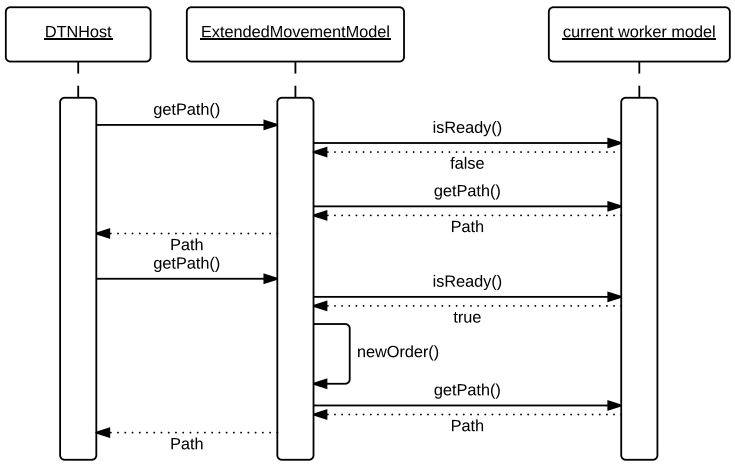

Figure 7: Sequence diagram of the invoking of getpath method on ExtendedMovementModel.

The UML sequence diagram in Figure 7 shows the interactions between ExtendedMovementModel object and its current worker model when the getPath method is invoked by a DTNHost object. The basic idea is it has to know when the current worker model is finished so that it can switch to the next worker model. So it requires the worker models to have a method telling if it is finished so that another

worker model can take over when it is done. The isReady() method is for that use. ExtendedMovementModel has a newOrder() method which is an abstract method whose task is to do the transition to the next worker model. It leaves its implementation class to decide the specific logic. When the next worker model starts to take over the job, it does not know where the host is currently located. So when the ExtendedMovementModel object is switching the current worker model to the next one, it first get the last location from the current one by calling its getLastLocation() method, and set it for the next one by calling setLocation() method. All these requirements on worker models are wrapped in the SwitchableMovement interface. Any movement model wants to be used as a worker model by ExtendedMovementModel class must implement it.

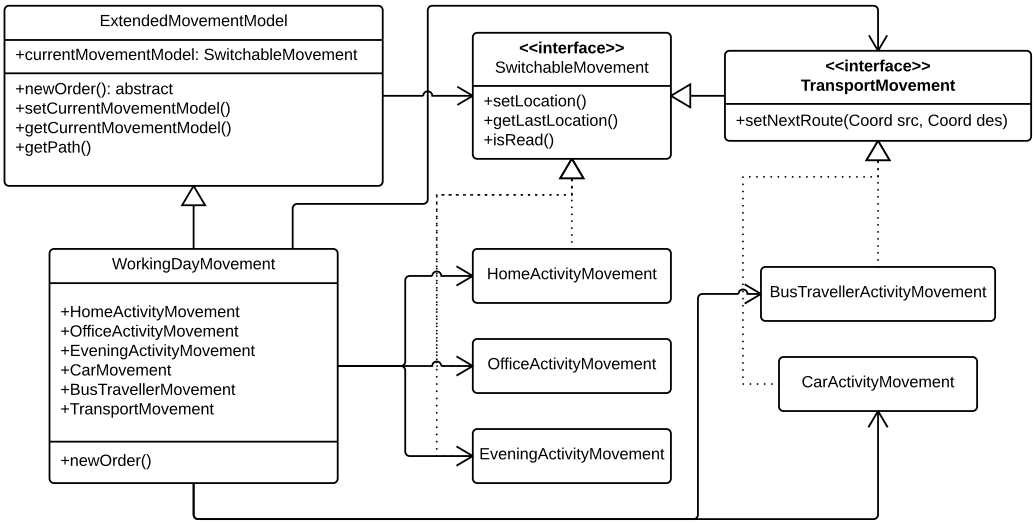

Figure 8: Class diagram of WorkingDayModel.

The WorkingDayMovement class is an implementation of the ExtendedMovement abstract class. It has three worker models to represent the movement patterns people normally do at different time during a work day – night and morning staying at home, limited movement in office during working hours and shopping or hanging out with friends at evening. As each of these three worker models actually describes the movement pattern of a daily life activity, we also call them activity models. When an activity model is finished, a TransportMovement model is switched on by the WorkingDayMovement object to move the host to the place where the next activity model happens. The WorkingDayMovement object tells the TransportMovement object where the host should move to by invoking the setNextRoute() method. A TransportMovement model can be either BusTravellerMovement model or CarMovement model depending on the setting of the host. When the TransportMovement model is finished, the next activity model is switched on. It is BustravellerMovement model's responsibility to walk the host to the starting bus stop. But the BustravellerMovement model only needs to move host to the bus stop closest to the destination, and it is the activity model that takes the responsibility of walking host

to the destination.

4.4.3 Bus transportation system

The bus transportation system consists of class BusTravellerActivityMovement, BusControllSystem and BusMovement as the class diagram shown in Figure 9.

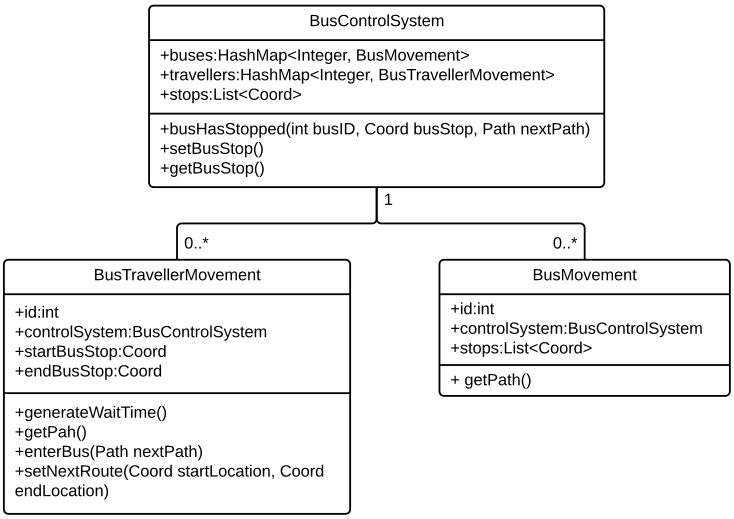

Figure 9: Class diagram of bus transportation system.

The BusMovement object represents a bus that serves a specific bus line while the BusControlSystem object represents the bus line. The BusTravellerMovement represents a traveler that can take buses of a specific bus line to its destination. BusControlSystem objects are lazy initialized singletons. Each of them controls a single distinct bus line. Whenever a BusTravellerMovement or BusMovement object is under instantiation, it registers itself to its BusControlSystem object. As it can only register to only one BusControlSystem, it means a bus traveler only knows a single bus line. A BusTravellerMovement object has three states. It can be walking to the nearest bus stop, waiting for a bus, or traveling on a bus. The path it produces depends on its current state. When it is walking to a bus stop, the path is calculated by the DijkstraPathFinder class. When it is waiting for a bus, it always returns null by its getPath() method, so the host will not move. When it is on a bus, it returns the path given by the bus it is traveling on. More specifically, the path of a BusMovement is a path started from the last passed stop and to the coming stop calculated by the DijkstraPathFinder class.

The Path object produced by BusMovement object is not handed directly to the BusTravellerMovement object, but through the help of BusControlSystem object, because in its design, a traveler does never know which bus it is on. It knows there is a bus it can enter just because the BusControlSystem object invoked its enterBus() method, and at that invocation, a Path object is passed to the traveler. So the

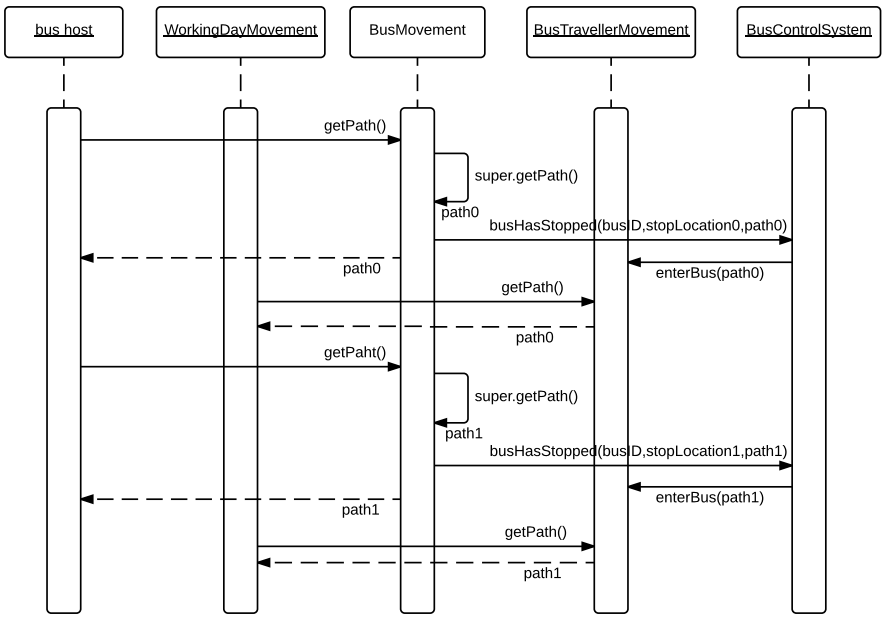

Figure 10: Sequence diagram of invoking getPath method on BusMovementModel.

traveler just moves as the Path object tells without knowing at the same time a bus is moving along the exact same path with it. In this way, it imitates the fact that a bunch of passengers are been carried by a bus.

The sequence diagram in Figure 10 shows how the Path object is handed from BusMovement object to BusTravellerMovement object. When the bus host has exhausted the previous path, which also means the bus has arrived at a new stop, it asks the BusMovement object for a new path. The BusMovement object gets the path to its next stop by asking its super class – MapRouteMovement class. Before returning the new path, the BusMovement object, by invoking the busHasStopped() method on its BusControlSystem object, tells the control system that which stop it is stopped right now, and also tells the new path it will move along. When the control system is notified with this information, it notifies all the traveler objects that belong to this route and is located at the stop. Then the travelers that are waiting for a bus will get on the bus, while those already on the bus will check if this is their destination stop and choose to get off or not.

Any MovementModel classes has a generateWaitTime() method for deciding how long the pause interval is until a new Path it can provide. Normally it just return some randomly generated floating value. To make sure bus travelers and the bus pause and start a new path at the same time, the BusTravellerMovement class overrides its generateWaitTime() method to always return 0. After a new Path object is got from the enterBus() method, the BusTravellerMovement object returns the Path object only once through its getPath() method. So when the traveler host has exhausted the current path but the bus is still paused, the traveler host will keep asking for new path from the traveler movement model. Because the current

path has already returned before, it will only return null. When the bus resumes from pause, it will generate the next path and hand it to the traveler movement model through the control system. Now the traveler movement model will return the next path to its host. In this way, the path will be started by both the bus and the traveler at the exact same simulation time. However this is only true with the constrain that, the move() methods of bus hosts are invoked before the ones of bus travelers.

4.5 Summary

In Section 4.1, we firstly introduced the state objects that make up the mobile opportunistic DTN network for simulation. Then we stepped through how a main loop of the simulation works. In Section 4.2, we introduced the supporting modules that facilitate a simulation. In Section 4.3, the work flow and mechanism of routing module were discussed generally. At last but not the least, we discussed the movement module in depth and with details, both the design and some subtle implementation, which lays the basis for discussing the changes for public transportation system in the next chapter.

5 Mobility augmentation on public transportation

The current version of the ONE simulator already has a public transportation system consisting of movement models for buses and bus travelers. The initial motivation of the augmentation is to add an underground public transportation system(metro system) to complement the current system. Basing on the fact that metro system works in the same way as the bus system except that underground nodes do not interfere with signals of the nodes overground, we found the task is not to build a new movement model, but to enable 3D capability for some extent on the simulator so that underground and overground nodes can be distinguished. Basing on that, the communication between these two planes can be blocked. Apart from that, a few behavior changes are needed for public transportation travelers because it requires a way to update which plane it is on, overground or underground.

The second goal is enabling the public transportation vehicles to follow real schedules. To achieve this we have to change the movement pattern of public transportation vehicles. Following schedules conflicts with the randomness property all the current movement models hold. So resolving the conflict is the main problem of this task. Apart from that, we have to decide how the schedule information should be stored. Another decision we have to make is whether to create our own format for simplicity or to use widely used format like GTFS for easier provisioning. After all, we chose both.

5.1 Adding underground system

5.1.1 Requirements and limitations

Besides distinguishing underground nodes from overground nodes and blocking their communication, we have to deal with nodes switching between underground and overground. For example metro travelers start at overground and switch to underground when arrive at the metro station, and switch back to overground when get off from the vehicle at the destination station. Also, this two-plane design should be easily reused to support multiple planes. For example it can be reused to make the OfficeActivity movement model support offices having more than one floors. The requirements and limitations for adding underground system is listed below.

1. NetworkInterface objects on different planes cannot connect to each other.
2. Public transportation vehicles are always on their own plane, buses and trams move overground while metro vehicles move underground.
3. Metro travelers switch to underground when it arrives at its starting metro station and switch back to ground when it gets off at destination station.
4. To keep the complexity within managable extent, we do not support transfer between metro system and bus/tram system.
5. The mechanism can be reused for supporting multistory office.

5.1.2 Design of multi-plane structure

We had two approaches for adding underground plane. One idea is using multiple World objects, each representing a unique plane. The plane a host is on is reflected by which World object it is registered to. As hosts in different World objects does not know each other, so by nature, their NetworkInterface objects don't connect to each other. A benefit of this approach is interfaces in different World objects do not have to check if they are near to each other. But it introduces overhead when hosts switching between planes. For example, when a metro traveler arrive the metro station, it has to unregister itself from the overground World object and register to the underground one. It also has to disconnect all established connections in the current world before the switch, because its NetworkInterface object has already got the references of the NetworkInterface objects on the other ends of the connections, and vise versa. To support multistory office, the program has to create a World object for each floor, which is not very straight forward to understand. Essentially, the approach described above is based on the understanding that network interfaces can not connect because they are on different planes. The other approach, on the other hand, is based on the understanding that being on different planes does not prevent network interfaces from connecting to each other, instead, the essential reason is the fact that radio signal can not pass through between planes. This understanding is consistent with the original design of the simulator that two NetworkInterface objects can connect to each other only if they are within the radio range of each other. As this logic is wrapped in the isWithinRange() method, we can reuse the original design by making the isWithinRange method return false if two hosts are on different planes. With this design, the plane information is represented by integer value called layer id. It can be thought as the z-axis of a three-dimensional coordinates, while the z-axis can only be integer value. The layer id is originally specified in setting file, then loaded in MovementModel objects when program starts, and then passed to DTNHost objects and used by NetworkInterface objects. Finally, we decided to go with the second approach because of its simplicity and the consistency with the original design.

5.1.3 Design of public transportation system

As metro system and bus system are very similar. The only difference is the plane they are moving on, so we kept the original design of the bus system which described in 4.4.3. We just added some fields and methods for plane handling, and rename the class name to PublicTransportMovement to reflect the fact that it can be used for both overground and underground. We also enable the BusTravellerMovment class update its layer id at points that plane switch is possible, and rename it as PublicTransportTravellerMovement. The class hierarchy for the new public transportation system is shown in Figure 11, and the changed places are marked in red.

As mentioned, the layer id is loaded in PublicTransportMovment from settings. Layer id -1 stands for underground plane and layer id of 0 for the overground plane. After the construction of PublicTransportMovment object, it passes its layer id to the control system by calling the setLayer() method of the control system, so later

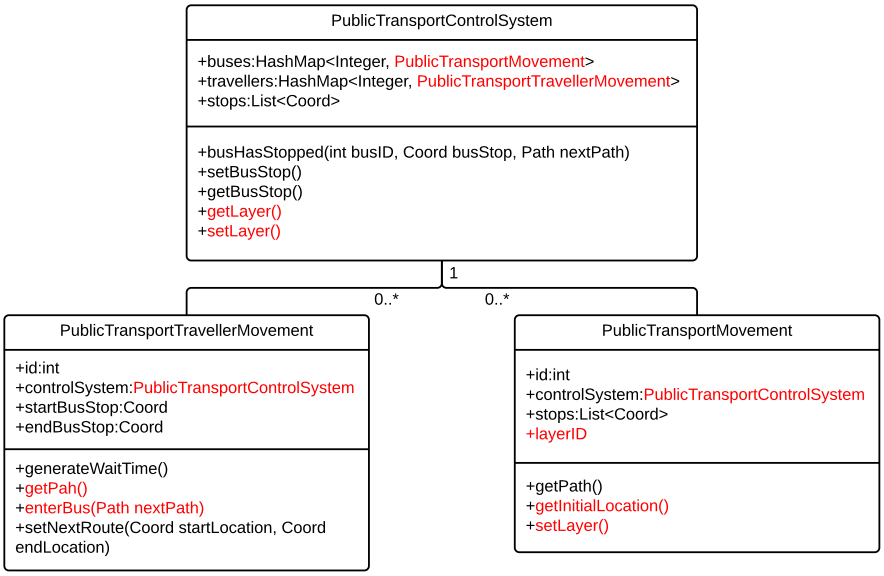

Figure 11: Class diagram of the new public transportation system.

the control system can pass this layer information to the travelers when they start to wait for public transportation vehicles.

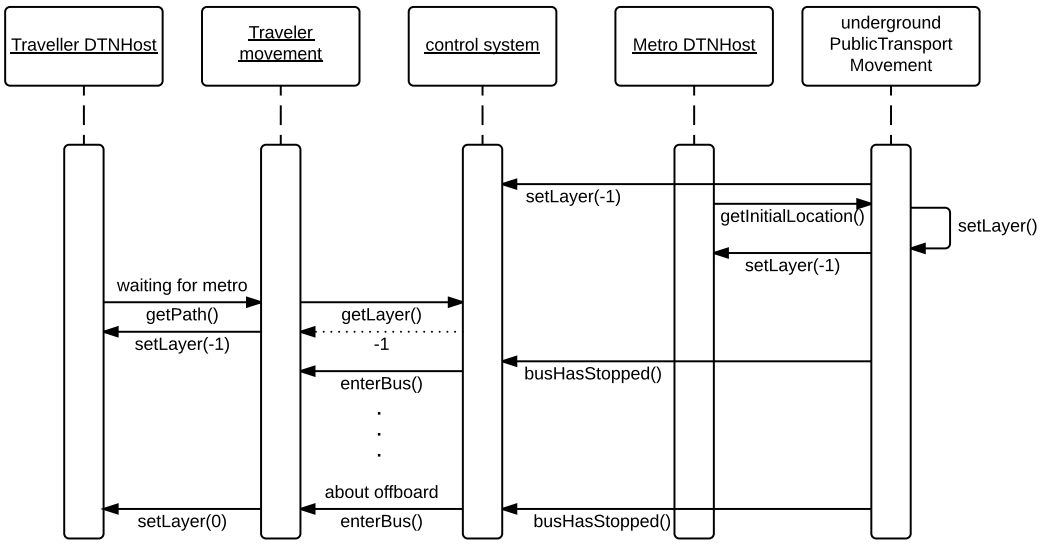

Figure 12: Sequence diagram of the spreading of plane information.

The sequence diagram in Figure 12 illustrates how plane information are passed around. The layer id of any DTNHost object is 0 by default. When a metro host asks for its initial location from its movement model, its layer id is updated to -1. The layer id of control system is set by the PublicTransportMovement after its instantiation. When a traveler host arrives at the starting metro station of its journey, it will call the getPath method of its movement model for the next path.

This then triggers its movement model to ask for the layer id from its control system and then update the host to -1 layer. After waiting at the station for some time, a vehicle comes and takes the traveler with it for a few stations. When the destination station is arrived and the enterBus method is invoked again, the traveler movement object will update its host back to layer id 0 which means the host has left the metro station and back on the ground.

5.1.4 Design of multistory office

Benefit from the multi-plane design, supporting multistory office for OfficeActivity-Movement can be achieved with few changes. With multistory, hosts in the same office building do not connect to hosts on different floor. Any offices by default has 1 floor. we allow user to configure the number of floors for each office. In the original design, there is a file containing the office locations. We add a file containing the numbers of floors of offices in the same order as the location file does. When constructing an OfficeActivityMovment object, it first reads in the number of floors specified for the office, then randomly selects a floor for itself. The floor number corresponds to the layer id. That means floor 0 is the first floor. If an OfficeActivityMovement object selects floor 2, when the host arrives at the office location, it will update the layer id of the host to 2. And when the host about leave the office, it will update the layer id back to 0.

5.1.5 Methodology for refactoring

When we implement the design for PublicTransportTravellerMovement class, we found it is not easy to find the point for layer switching because the state transition logic of a traveler is not easy to understand. To make sure we understand it right and insert the layer updating operations at the right point, we first created unit test cases for the state transition, and made sure they all pass with original implementation. Then we refactored the implementation so that the state transition is clear and responsibilities are placed at better place. After our refactoring is proven by the unit tests, we added our changes for layer switching with more confidence. The state transition after refactoring is shown in Figure 13.

Although there always are risks for refactoring legacy code without unit test, creating unit test before changes can result in clearer code, better specifications for future changes and some extent of confidence.

5.2 Integrate real-world timetable

5.2.1 Requirements and scope

To enable public transportation vehicles move according to schedule information, vehicles have to be initialized at certain starting stops on their routes, departure from and arrive at the right stops at the right time. To achieve this, when each vehicle movement object is created, its initial location and its schedule should be assigned.

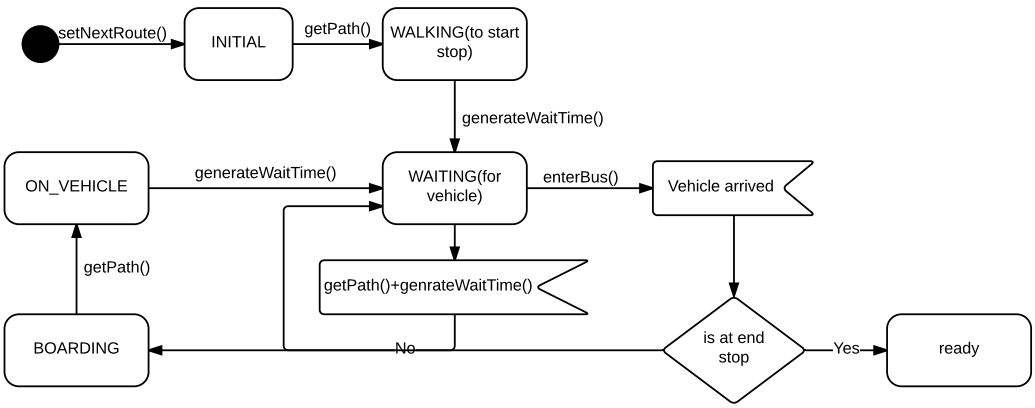

Figure 13: State diagram for a traveler.

We decided to support these new behaviors as a new mode of MapRouteMovement model so that the public transportation system classes - the PublicTransportTravellerMovement, PublicTransportMovement and PublicTransportControlSystem, can be reused without any change.

The difficult part is getting the compatible schedule data. Normally schedule data provided by public transit agencies are timetable which is trip specific rather than vehicle specific. This means, with trip specific schedule, we know that at time X there is a vehicle departs from stop A and at time Y it arrives at stop B, but we cannot tell which vehicle it is talking about. While the ONE simulator has to know how many vehicles it needs to create and the exact schedule for each of those vehicles. Apart from that, there are different timetable formats used by public transit agencies. For example, GTFS is the most popular format for storing transit schedule data. While the Transport for London (TFL) only provides data in TransXchange, a UK national XML based data standard for the interchange of bus route and timetable information. Considering these two facts,we decided to split the integration task into three smaller tasks. The first task is to define our own format for storing vehicle specific schedule data. The second task is making vehicles follows the vehicle specific schedule. The final task is converting real-world transit data into the format defined in the fist task. Since creating conversion tool for all data standards is obviously impossible by our own effort, we created tools for GTFS data as a demonstration. The requirements are listed below:

1. Define the format for vehicle specific schedule data.
2. Vehicles should be initialized at certain starting stops.
3. The paths produced by vehicle movement model should conform to the schedule.
4. changes on the current public transportation system should be as less as possible.
5. create conversion tool for timetable data in GTFS.

6. support one-week-long simulation.

5.2.2 Format of vehicle specific schedule data

There are certain knowlege the ONE simulator has to know so that it can create and move the scheduled public transportation vehicles accordingly. Firstly, it needs to know how many vehicle nodes to create. Secondly it has to know the sequence of stops each vehicle will stop at, as well the time stamp when it departs from and arrives at each stop. It also has to know the location of all the stops. In addition, it needs to know the one-to- many relation between routes and vehicles. To contain all these information, we defined the schedule data of a public transportation system into this logical structure: A system schedule is an array of route schedules. Each route schedule is an array of vehicle schedules. Each vehicle schedule consists of multiple trips. Each trip consists of a sequence of stop information. Each stop information includes the stop id, departure time and arrival time. We decided to use JSON string to represent the system schedule data because there are existing high quality JSON processing libraries for use. A snippet of the JSON string of the system schedule is shown in Listing 1 to demonstrate the data structure.

Listing 1: Sample data of vehicle specific schedule.

```
Schedule file:
[{
  "route_id" : 2,
  "layer_id" : 0,
  "stops" : [ "1040123", "1020134", "1040125"],
  "vehicles" : [{
    "vehicle_id" : 0,
    "trips" : [[{
      "stop_id" : "1040123",
      "arrT" : 25200.0,
      "depT" : 25200.0
      }, {
      "stop_id" : "1020134",
      "arrT" : 25320.0,
      "depT" : 25320.0
      }, {
      "stop_id" : "1040125",
      "arrT" : 25320.0,
      "depT" : 25320.0
    }]]
  }]
}]
Stop file:
{
  "1040123" : {
    "x" : 2549176.1194754364,
    "y" : 6677433.695592673
```

```
  },
  "1020134" : {
    "x" : 2549174.3506877944,
    "y" : 6677437.724481765
  },
  "1040125" : {
    "x" : 2551666.3816336305,
    "y" : 6677622.074734049
  },
}
```

The coordinates of stops are store in another file with an id-to-coordinates map structure. The reason why we split the location and the id of the stops is to reduce redundant data and save memory consumption. This is necessary especially for schedule data of big volumn.

At the initiation phase, the simulator parses and loads the system schedule and stop data from json files to memory. Then it constructs vehicle nodes and assign vehicle schedules to each of them. After simulation is started, the RouteMapMovement model goes through its own vehicle schedule and produce paths between each two stops when requested.

5.2.3 The scheduled mode of MapRouteMovement

MapRouteMovement class is for mobility pattern that moves along predetermined route. As a subclass of MapRouteMovement, PublicTransportMovement has additional logic for interacting with control system. The scheduled behavior is wrapped in MapRouteMovement rather than in PublicTransportMovement because it relates to how paths are produced from predetermined route and has nothing to do with control system.

When constructing a scheduled PublicTransportMovement object. The values of "route_id" and "stops" in the RotueSchedule object are needed to bind with its control system. The "layer_id" is required to specify the plane it moves on. The VehicleSchedule object and stop hash-map is used for constructing the MapRouteMovement part of the object.

We call the existing behavior of MapRouteMovement as random mode while the new mode as scheduled mode. In random mode, the starting stop, starting time, moving direction and waiting time at each stop are all randomly selected. In scheduled mode, all of these time points and map locations are predetermined in the VehicleSchedule object.

In scheduled mode, the initial location is set at the first stop of the first trip. When a new Path object is requested, firstly it uses Dijkstra algorithm to get the shortest Path object. At this stage, the Path object does not contain any speed information. Secondly it calculates the total distance of this path, and also the duration of the path which is the difference between the arrival time of next stop and the current time. Finally, it fields the speed according to the total distance and duration, and set it as the speed of the Path object. In this way, the produced path

conforms to the schedule precisely with deviation of only 1 or 2 seconds. We store stop infomation objects in a manner of a list of lists to preserve the trip concept. Each list of stop information objects represents a trip. Trip concept is used by most timetable data formats, and usually stands for a complete traversal through a route in one direction. Although the trip concept is not exploited in our current solution, it takes us nothing to preserve this additional information.

5.2.4 Convert GTFS to our format

GTFS data defines how routes, timetable and all relevant data is stored. The data is stored in muliple files with each file containing certain aspect of the data. The stops.txt file contains the location of stops. The routes.txt file describes the name and type of routes. There are 8 types for routes predefined by GTFS:

1. Tram, Streetcar, Light rail. Any light rail or street level system within a metropolitan area.
2. Subway, Metro. Any underground rail system within a metropolitan area.
3. Rail. Used for intercity or long-distance travel.
4. Bus. Used for short- and long-distance bus routes.
5. Ferry. Used for short- and long-distance boat service.
6. Cable car. Used for street-level cable cars where the cable runs beneath the car.
7. Gondola, Suspended cable car. Typically used for aerial cable cars where the car is suspended from the cable.
8. Funicular. Any rail system designed for steep inclines.

As we only consider the main transportation methods in Helsinki area, we ignore the types of ferry, cable car, gondola and funicular. A route normally but not necessarily has multiple trips each day. The trips.txt file specifies which trip belongs to which route. A trip is composed by multiple stops from the starting stop to the terminating stop of a certain route. The stop_times.txt file describes the stop sequence and time stamp on each of them for each trip. In addition, there is a calendar_dates.txt file specifying the service date range for each trip. We developed three tools to facilitate the conversion of GTFS data.

The schedule converter

In GTFS, the relationship direction between the elements is from bottom to top. For instance, a trip knows which route it belongs to, but a route does not know what trips it has. But in the simulator, the relationship direction must be from top to bottom. The simulator needs to know the vehicles a route has, the trips a vehicle has, and the sequence of stops a trip has. The UML Element Relationship diagram in Figure 14 shows the relationship changes done by the schedule converter.

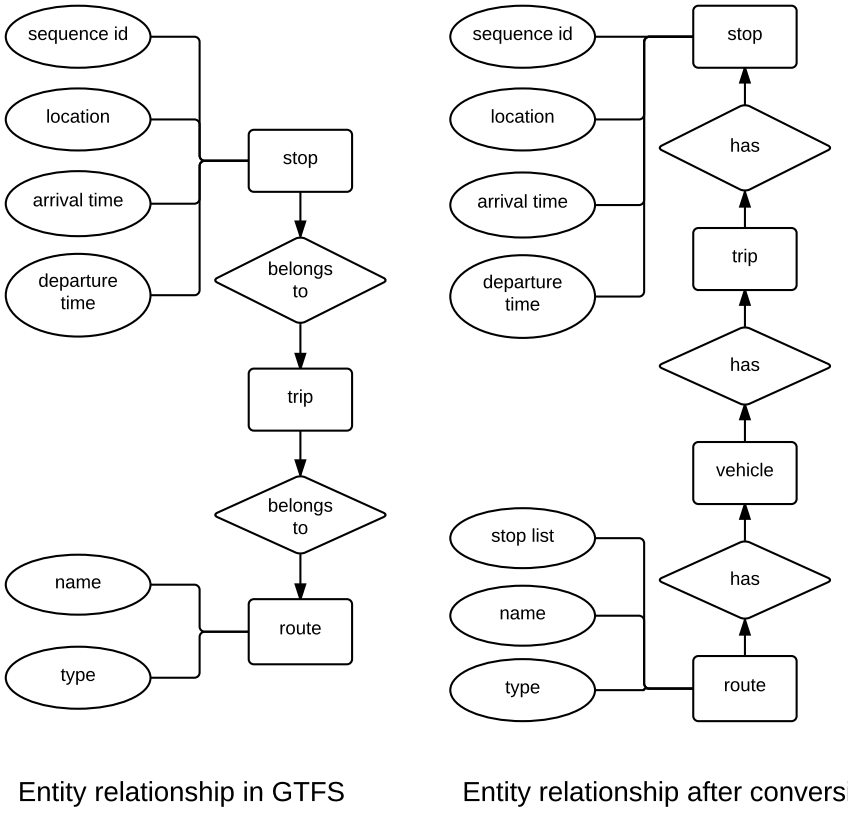

Figure 14: UML Element Relationship diagram before and after conversion

As GTFS is trip specific, the converter has to assign appropriate trips to vehicles. The simplest way is creating a dedicated vehicle for each trip. However, this can leads to more than a hundred thousands of vehicles for a week-long simulation for a big city like Helsinki region(including Helsinki, Espoo and Vantaa). This is not realistic nor desirable when considering the amount of memory to accommodate that many vehicle objects. To the opposite extreme, we could suppress the number of vehicles by reusing a vehicle for as many as possible trips. This means after a trip is finished, the vehicle will take the trip of the same route starting at the nearest future, regardless how far the starting location is away from its current location. This is not desirable as well because it leads to teleport behavior which against the philosophy of our simulator. A more appropriate reuse strategy can be achieved by just adding one condition. By setting the maximum speed of vehicles, a vehicle can only takes trips whose starting stop can be arrived by valid speed as candidates for the next trip. And it selects the nearest one as its next trip.

The time used in GTFS is specified by the time of the day and the day of the week. The converter converts this time format to the number of seconds past 00:00:00 of Monday so that it is compatible with the simulation time format.

In GTFS, there can be multiple version of trip schedules for the same week days but each targeting to different service date range. For the same route and the same

week day, the service date ranges of different version do not overlap on each other. For example there could be 50 trip schedules for route 1 on Monday valid from 1st January 2015 to 31st March 2015, and another 40 trip schedules for the same route on Monday but from 1st April 2015 to 20th May 2015. The converter always chooses the service date range that has the most trips.

A GTFS timetable is usually for an entire transit system which normally covers a large area. However our simulation is not necessarily covering the exact same area. Therefore the converter allows user to specify the boundaries of the interested area. The converter will exclude routes outside of the bounds. For routes partly within bounds, if more than half of its stops are within the boundaries, it will shorten the route so that it is completely within after shortening, otherwise it excludes the route.

In real simulations, a user can be only interested in a subset of the routes the GTFS data contains. So the converter allows a user specifying a list of interested route id in a plain text file, and provide the file name as a input parameter. When not specified, it assumes the user is interested in all the routes.

Points-to-roads merging tool

As required by the ONE simulator, there should be no isolated points on the map. This means each stop must be a point connected by the roads defined in WKT map files. To achieve that, we created a tool for merging stop points into WKT LineString and MultiLineString objects which represent the roads and paths on the map. The idea is inserting the point into its closest line segment, at the perpendicular intersection point, or just moving the point to the closest segment endpoint if the endpoint is closer than any line segment. If the distance to any segments or segment endpoints is farther than 50 meters, the tool does not change and warn the user to edit the map or stops further.

GTFS stops file Coordinate Reference System converter

The location of stops in GTFS is specified in latitude and longitude, while the aforementioned next trip candidates filtering and the simulator both depend on Cartesian coordinates. So we have to first convert the latitude/longitude location to Cartesian coordinates. As the well-know tool for this kind of task, Proj4, does not have a reliable released Java library, we decide to create a standalone web page tool powered by Proj4js library. To use the tool, a user firstly specifys the projection arguments needed by Proj4 to indicate the coordinate reference system to convert to. To get the best projection of the target area, go to http://spatialreference.org/ and search the suitable projection with the name of the area. For any selected projection, the website provides the corresponding Proj4 arguments for use. After this, user uploads the stops.txt file as input and clicks start parsing button. When the processing is completed, the converted file content is downloaded as a new file. The screen shot of the coordinate reference system converter is shown in Figure 15.

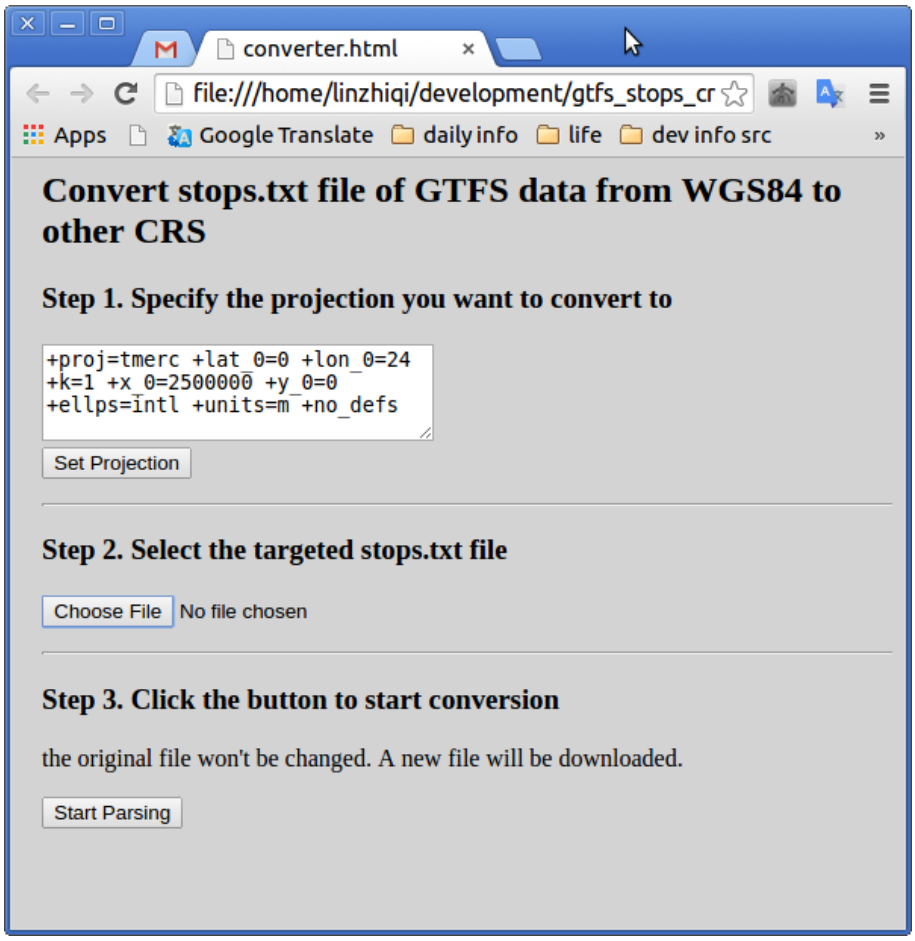

Figure 15: Screenshot of coordinate reference system converter

5.2.5 Creating scheduled vehicle hosts

With the convention of object construction in the ONE simulator, most state objects are constructed by themselves. Although the instantiation is invoked by the caller object, it is actually the targeted object itself that builds all of the dependencies it requires, with the help of Settings object. Encapsulating construction logic for specific type of objects into dedicated place is the right direction to good design. With the existing construction approach, we can create a MovementModel object by just passing a Settings object to its constructor. It looks neat and coherent, however it does not tell the whole story. Before invoking this simple constructor, the developer has to look through its inheritance chain, get the dependencies required by each of its ancestors by checking what do they request from the Setting object, and provision the setting file containing required properties and also required input data like map files. We call this arrangement is hiding dependencies. Although knowing dependencies is inevitable and this arrangement does not add much extra difficulty for knowing the dependencies, it makes unit testing difficult. Since the required dependencies of the object is hidden from the outside scope, the test logic outside cannot replace dependencies with mocked ones. This results in the situation that it

is not only the unit is being tested, but also the whole environment it depends on.

To eliminate this problem as much as possible but without changing the code base too much, we decided that the construction strategy for scheduled PublicTransportMovement object should not hide any new dependencies. The constructors for instantiating scheduled PublicTransportMovement class and MapRouteMovement class are show in Listing 2.

Listing 2: Constructor signature.

```
public PublicTransportMovement(
        Settings settings,
        PublicTransportControlSystem bcs,
        int layerID,
        VehicleSchedule schedule,
        List<String> stopIDList, HashMap<String, MapNode> stopMap,
        boolean isScheduled
)

public MapRouteMovement(
        Settings settings,
        boolean isScheduled,
        VehicleSchedule schedule,
        List<String> stopIDList,
        HashMap<String, MapNode> stopMapTranslated
)
```

The first parameter "settings" is kept for constructing their ancestors – MapBasedMovement and MovementModel class – in the old way. Because many classes depends on them, changing them will lead to much more changes. As our goal is not refactoring the project, we don't touch them. Here we prefer the constructor based dependency injection rather than setter based dependency injection. Although this leads to long constructor with too many parameters, the benefit over setter dependency injection is it is impossible missing any setter calls. You can see the dependencies introduced by scheduled movement are all declared in the constructors. As the object does not construct dependencies by itself, the dependencies can be faked in test code so that unit testing becomes possible.

5.3 Summary

In this chapter, we presented how and what the ONE simulator is augmented. Besides sufficient reasoning in design decision, we also presented our methodology from software engineer perspective on refactoring legecy code and on testable code.

We started at how underground transportation system is added on the basis of multi-plane structure. Then we elaborated the design of integrating real-world timetable data in three parts:

1. Defineing the format of vehicle specific schedule data
2. Enabling the ONE simulator to work with vehicle specific schedule data

3. Convert trip specific schedule data into vehicle specific schedule data

For the first two parts, we described the format of the vehicle specific schedule data and how these data elements are used by the simulator to instantiate vehicle nodes and make sure them follow the provided schedule. For the third part, we presented how we did the conversion from GTFS data as an example. Through that process, we dicussed the difficulties we had faced for GTFS conversion which also rerealed the general difficulties that needed to be solved when converting trip specific schedule data to vehicle specific schedule data.

6 Experiments and results

The goal of the experiments is to analyze the possible impact introduced by new supported scenarios including metro transportation, multistory office, and public transportation according to real-world schedule.

Because quite a lot implementation changes are added to previous version, we have to ensure the differences in experiment results are only caused by the new supported scenarios instead of implementation changes.

6.1 Default scenario

As all of the new supported features are developed for city area, the experiments are based on the city life in Helsinki downtown. Each node is with a network interface of 10-meter range and 100kBps speed. The storage size for carrying DTN messages is 500MB. The routing protocol is epidemic routing with a one-day long Time to Live (TTL) for messages. The size of messages are uniformly distributed from 10KB to 1MB. There are 6 new messages generated every hour. As each simulation simulates 7 days, 1008 messages will be created in total.

There are 1000 nodes with WDM model representing citizens with daily work. Half of them own cars, the other half use public transportation for commuting. The map of Helsinki center area is divided into 8 districts. To clarify this distric structure, we use the figure that used by paper "Working Day Movement Model"[30].

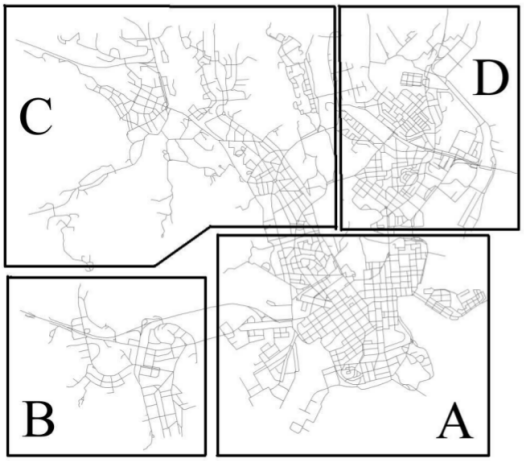

Figure 16: District structure

Figure 16 shows that the map is divided into 4 districts: A, B, C and D. Besides these 4 basic districts, there are 3 bigger districts each consists of two basic districts: E covers A and B; F covers A and C; G covers A and D. In addition, there is distric H that covers the entire map.

The 1000 citizens are further divided into 8 groups. Each group is constrained in their own district on the map, which means their homes, offices and evening event locations are located in the same area/district. There is a bus line constructed for each of the 8 groups. The route of a bus is constrained within the bounds of the district corresponding to the group. In addition, there are 10 nodes with SPMBM model representing tourists that do not work.

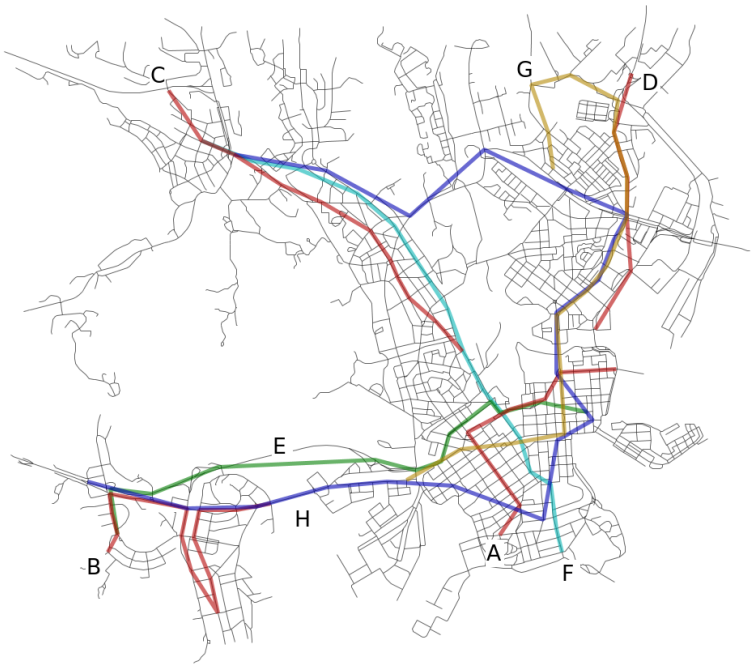

Figure 17: Topology of bus routes

We use delivery ratio, hop count and latency to to depict the impact of newly supported scenarios on the network performance. To directly evaluate the impact on mobility, we use contact duration, inter-contact time, hourly activity, unique encoutners and total encounters as metrics. Inter- contact time is the length of time between two sequential contacts of a pair of nodes. It indicates how often two nodes meet each other. Contact duration on the other hand indicates how long the contact lasts when two nodes meet. For these two metrics, we draw Complementary Cumulative Density Functions (CCDF) to show their statistical characteristics. Hourly activity corresponds to how many contacts in total happened during each hour. It is naturally to plot it with a run chart. It reflects the activity level variation along the course of time, which is useful for revealing time based movement pattern. The ratio of unique encounters to total encounters metric is used for revealing the locality of nodes. Unique encounters stand for how many different nodes a node meets. While total encounters stand for how many times a nodes meets other nodes. Both unique and total encounters are affected by the activity level of the node and density of nodes around it. In addition the unique encounters also are well affected by the locality of the node. For example, a node moves a lot but restricted within a small area can meets other nodes quite frequently, but they are the same group of nodes it meets again and again. In this case, this node is with a large number of total encounters and a small number of unique encounters. So the locality is reflected by the relation between these two instead of either of them alone. By calculating the ratio between the unique and total encounters, the impact of activity level and density is eliminated, and the one of locality is revealed. We use a scatter diagram to show this relation by having x-axis as the total encounters and y-axis as the unique encounters. Each node is a point in the diagram, and the ratio correlates to the

tilting angle of the line from coordinate origin to the point.

6.2 Implementation validation

Version 1.5.1rc2 is the direct ancestor to which our changes are added. During the composing of the thesis, the number for the new version is not determined yet. For simplicity, in this thesis, we call it version 1.5.2. However it could be different from its actual version number in the project repository.

To ensure the differences of experiment results are only caused by the changes of scenarios without the interference by possible implementation changes, we conducted experiments on both version 1.5.2 and version 1.5.1rc2 with the default scenario. We expected acceptable differences between their results. In software engineering, it is called system testing because it involves all components of the software. And we actually found a critical bug during this step which will be discussed later in this section.

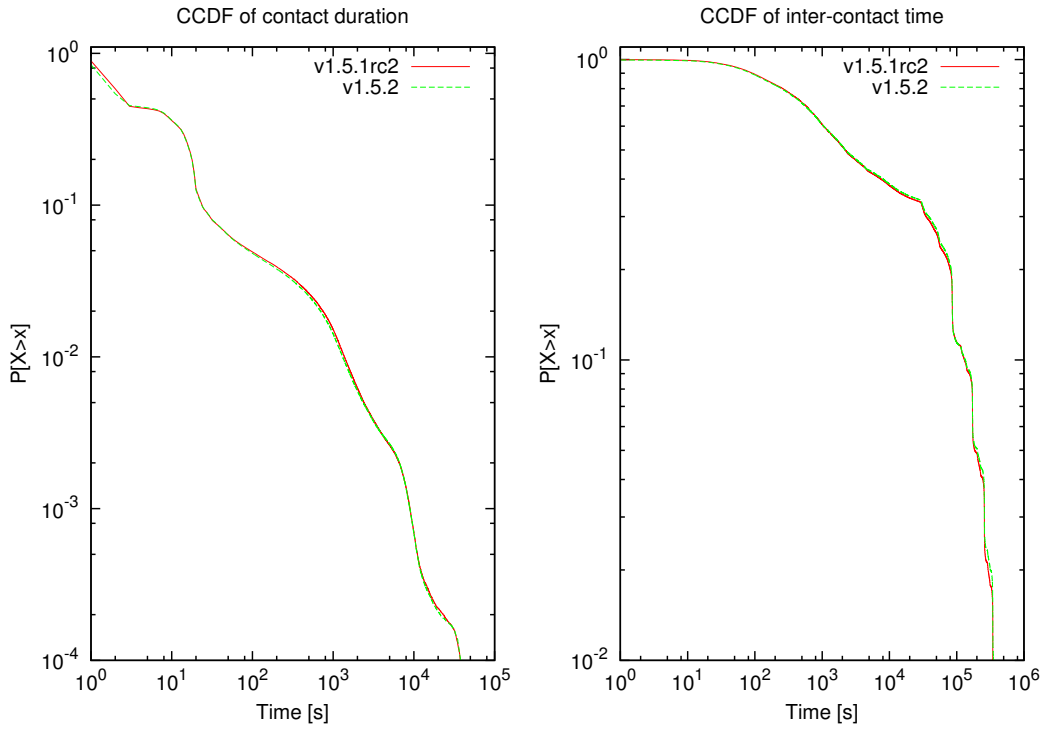

Figure 18: Comparing CCDF of contact duration and inter-contact time between version 1.5.1rc2 and 1.5.2

As one might think by using the same seeds for random number generating, the results should be exactly the same. Thus, there should be no need to plot graphs for those metrics. Using some file compare utility, for instance diff, on the resulted reports will do the work. However, in fact, the implementation changes are not required to have the same number of random number requests nor to keep the requests in the same order. For example, in OfficeActivityMovement class, to

support multistory feature, it has to request an extra random number for randomly determine the floor it belongs to even it only has one floor. As a result, the numbers in resulted reports are different. Figure 18 shows the CCDF of contact durations and inter-contact times for version 1.5.1rc2 and 1.5.2. We can see the lines are almost overlapping all the time.

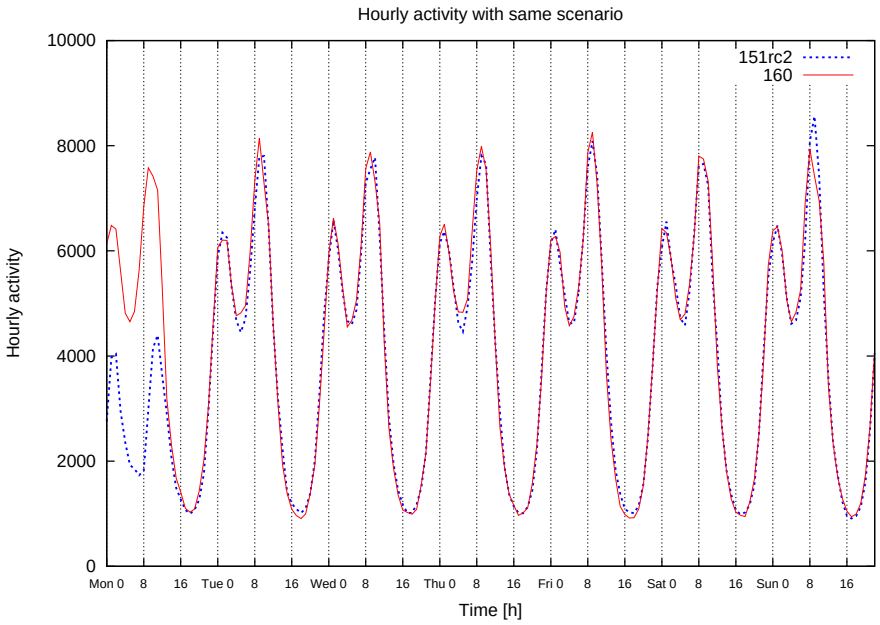

Figure 19: Comparing hourly activity between version 1.5.1rc2 and 1.5.2

The run chart in Figure 19 shows how the hourly activity level varies for a whole 7-day week. There are two peaks each day. The duration between two peaks is roughly 8 hours which correlates to the working hours set in the scenario. Except the first 16 hours on Monday, the timing of activity level variation of the two version match quite well. During the first 16 hours of the simulation, version 1.5.2 apparently has much more contacts happened than version 1.5.1rc2. This is because the bug we mentioned in the beginning of this section. With this bug, the movement warmup feature does not work.

The movement warmup feature lets nodes start to move before the zero simulation time, so that when simulation starts, the mobility model has reached a stationary state. This is done by setting the initial simulation time with a negative value, making location updates start from that negative value, and keeping network status updates and message events heppen only after the zero simulation time. The initial simulation time is used to generate the time of the first location update for each nodes. For example, if the update interval is 0.1 second, and the inital simulation time is -1000 second, then the time of the first location update will happen at -999.9 second. However, in version 1.5.1rc2, when generating the time of first location update, the initial simulation time is not set to the negative value yet, and still with the default zero value. Thus no node will update its location before the zero simulation time. Which further means the movement warmup feature does not

work at all. In version 1.5.2, we fixed this bug by setting the initial simulation time before generating the time of first location update.

As movement warmup feature does not work in version 1.5.1rc2, all the WDM nodes start to wake up at home only after the zero simulation time. While in version 1.5.2, at the zero simulation time, there are WDM nodes that are already woke up and on the bus to office. That explains why there are more contacts heppened in the first 16 hours.

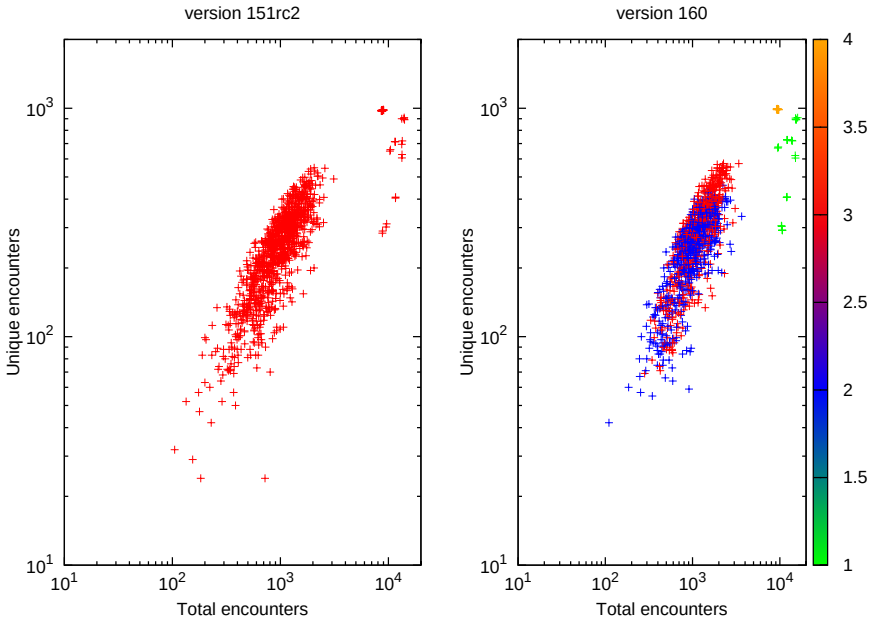

Figure 20: Total encounters vs unique encounters

In Figure 20, the two scatter diagrams depict the heterogeneity of the mobility patterns in the simulation. We can see the distribution pattern of the dots in version 1.5.2 is compliant with the pattern in version 1.5.1rc2. Addionally, for version 1.5.2, benefited from an improvement on reporting module, the dots can be painted in different colours to distinguish their movement model.

From the diagram of version 1.5.2, we can tell there are four types of nodes: WDM nodes with car(in blue), WDM nodes without car(in red), public transportation vehicles(in green) and SPMBM nodes(in orange). As WDM nodes with car move fast and always along the shortest path, they normally take less time on the road. That means their unique and total encounters are both smaller than those of WDM nodes without car. However because of the distribution of home and office location, and the different size of districts they live in, the differences between with car and without car are blurred. Public transportation vehicles are almost always moving, so they have larger total encounters than travelers, around 10000. However their locality are largely affected by their routes, like how long the route is and how many districts it across. SPMBM nodes for sure are the ones in the right top corner. That is because they are almost always moving and can move towards any where on the road network.

Figure 21 shows no significant difference on the network metrics between version 1.51rc and 1.5.2.

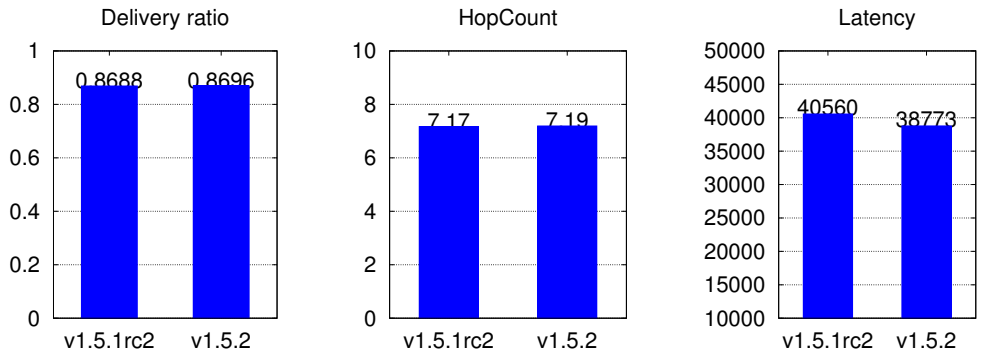

Figure 21: Message statistics

Based on the experiment results of the two versions of implementation, we consider the variations and differences are acceptable, and verified that the implementation changes introduced by version 1.5.2 works correctly one existing features.

6.3 Impact of metro system

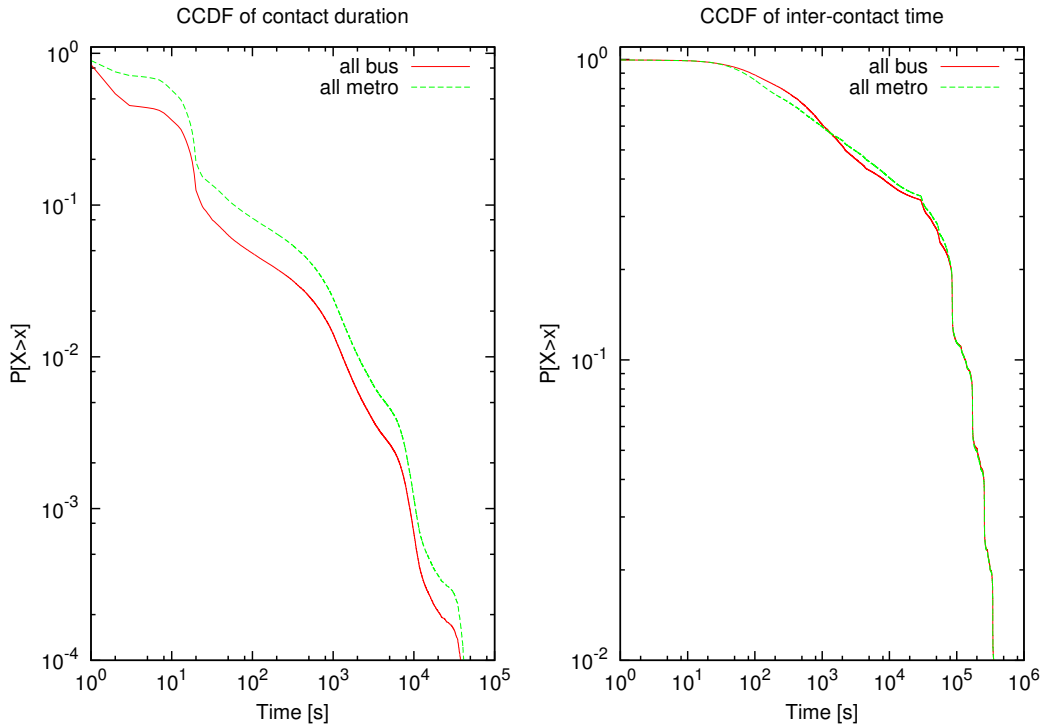

Figure 22: Comparing CCDF of contact duration and inter-contact time between all bus and all metro

To evaluate the impact of metro system, we did simulations on a scenario that all buses are replaced by metro vehicles. The metro vehicles use the same routes as of buses. So this scenario can be considered the same as the default scenario except that all the buses are running underground. The result is compared to the result of default scenario simulated with version 1.5.2.

From Figure 22 we can see, while inter-contact time does not have much difference, contact time CCDF shows that the proportion of longer contacts becomes larger. This is because of the fact that short contacts that were made between buses and cars, and between buses and pedestrians never happen in the all-metro scenario.

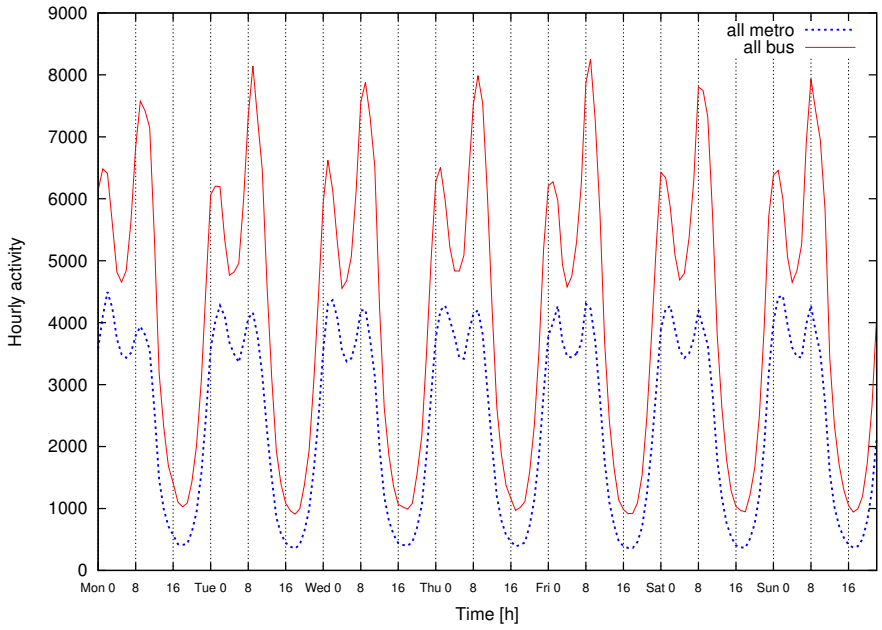

Figure 23: Comparing hourly activity between all bus and all metro

The activity level run chart in Figure 23 shows that the number of contacts is lessened, and the reduction is severe at peak hours. This can be explained by the fact that the contacts between bus, bus passengers, and cars, pedestrians account for a large proportion of the contacts in peak hours. By placing bus and bus passengers underground, the chances of contact are severely reduced.

From the scatter diagram in Figure 24, we can see the differences of impact corresponding to different movement types. For SPMBM nodes, there is no significant changes on neither total encounters nor unique encounters. The total and unique encounters reduce dramatically for public vehicles. This is because they move a lot but never contact overground nodes. For working citizens, they generally make less total and unique encounters, and the more active they are, the more to cut down.

The impact on network performance, as shown in the bar charts in Figure 25, is not as significant as we expected. This is because a large proportion of the contacts lost are very short contacts. Those short contacts like encounter with cars in different direction, do not provide enough up time for transferring big messages.

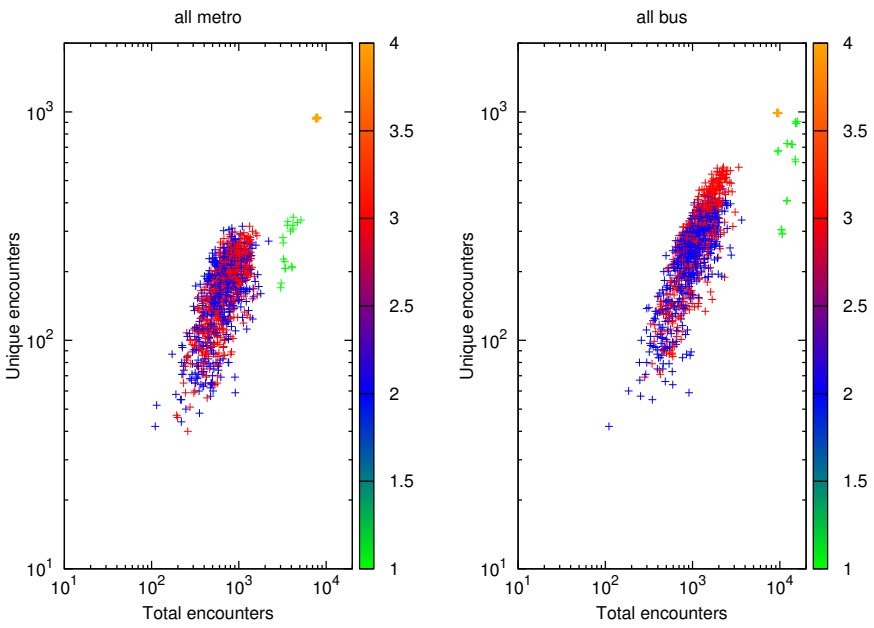

Figure 24: Comparing total encounters/unique encounters between all bus and all metro

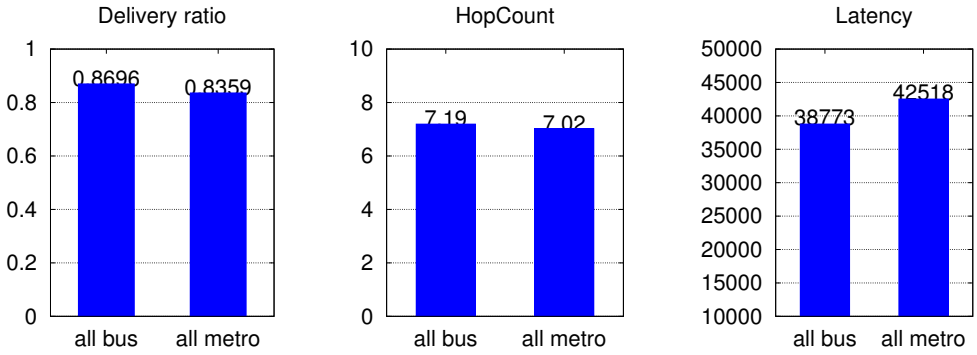

Figure 25: Comparing message statistics between all bus and all metro

6.4 Impact of real-world schedule

To evaluate how much difference it will make if the vehicles follow real-world schedules, we conducted an experiment with real-world schedules converted from GTFS data provided by HSL, the public transportation agency in Helsinki region.

Using real-world schedule means using real-world routes, stops and the time points arriving at and departing from stops. To avoid unnecessary scenario differences, we selected real-world public transportation lines that suitable for the district structure in the default scenario.

Table 1 shows the assignment of routes to the different districts. Figure 26 shows their topology.

District	sub district	Route name
A	/	17
B	/	20
C	/	4
D	/	6
E	A,B	20
F	A,C	4
G	A,D	6
H	A,B,C,D	65A

Table 1: Assignment of routes

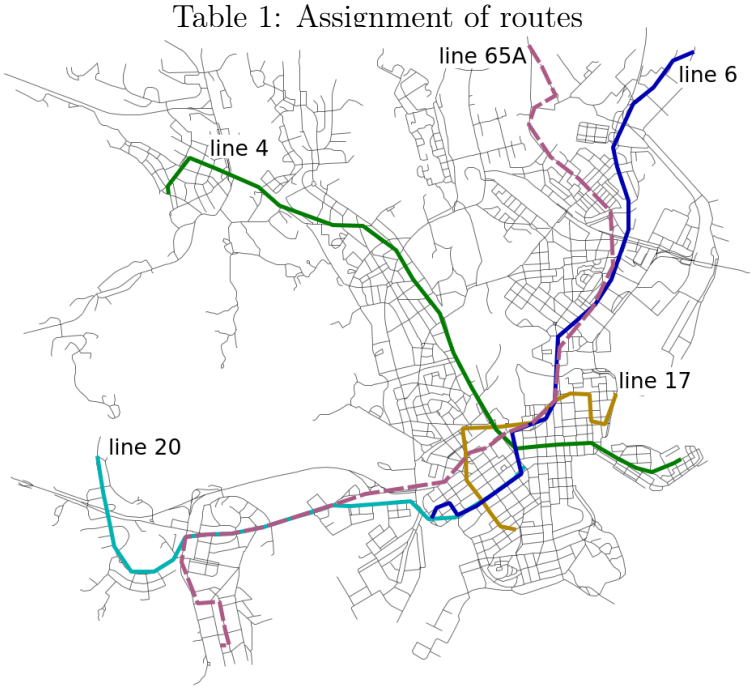

Figure 26: Topology of bus routes

6.4.1 Real-world schedule vs uniformly ditributed random schedule

We conducted two comparive experiments on real-world schedule. One compares uniformly distributed random schedule with real-world schedule to demonstrate the impact introduced by the un-uniform distribution of real-world schedule. As there are 73 vehicles generated from the GTFS data for the 5 routes, we used 18, 36 and 72 as the number of vehicles in random scenario.

From the CCDF of contact duration in Figure 27, we observe real-world schedule scenario has larger proportion of longer contacts. With real-world schedule, the number of active vehicles varies along the time. By investigating the schedule we use, we found from midnight to 6 am, there is rarely any moving bus. While in the random schedule scenario, the vehicles are always active and restless. Thereby travelers starting travel outside of rush hours have to wait longer, and this further results in more passengers on public transportation vehicles. Another factor that contributes to the increase of longer contacts is the inactive vehicles that stay at the starting or termination stops with its peers.

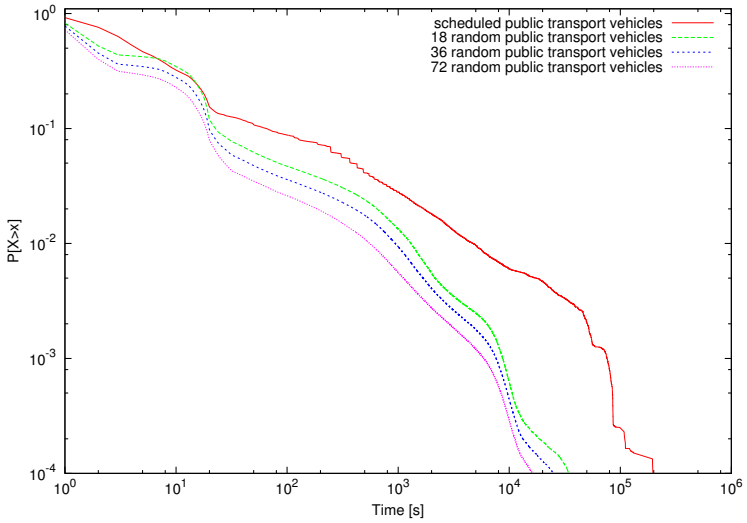

Figure 27: Comparing CCDF of contact duration between random schedule and real-world schedule

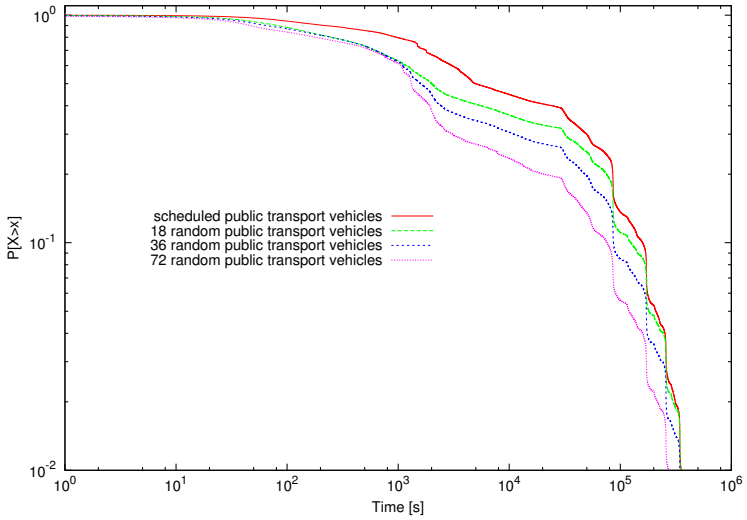

Figure 28: Comparing CCDF of inter-contact time between random schedule and real-world schedule

The CCDF of inter-contact times in Figure 28 shows real-world schedule scenario has larger proportion of longer intet-contact times. We believe this is caused by the less activeness of public transportation vehicles in real-world schedule, especially when out of rush hours.

From the hourly activity chart in Figure 29, we can tell the real-world schedule scenario has the same daily looped pattern for the first 5 days. Since the peak time of the activity level are matched, we can assure the wake up time (we set 8 am in this experiment) of citizen is matched with the schedule of HSL. We also observe the real-world scheduled scenario has the least contacts around midnight and relatively large number of contacts at peak hours, just less than the scenario of 72 random

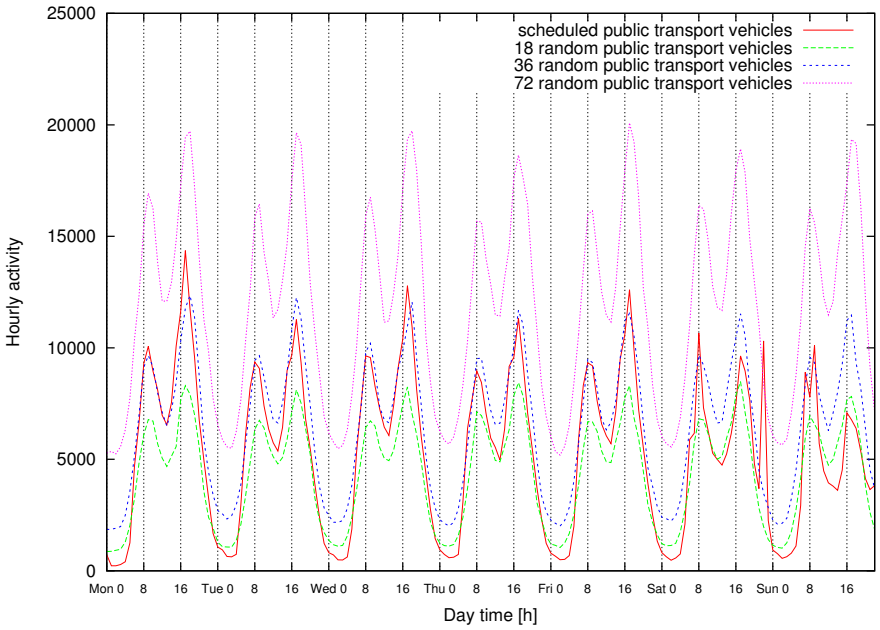

Figure 29: Comparing hourly activity between random schedule and real-world schedule

vehicles. In additon, the pattern becomes different on Saturday and Sunday, which can be explained by the fact that in Helsinki area, the public transportation schedule of weekends are different from that of work days, while our citizen node with WDM model spend every day as a work day.

From the scatter diagram in Figure 30 for unique and total encounters, we can see in the real-world schedule scenario, the public transportation vehicles(points in green) have quite different situation, which apparently comply with the real-world situation.

Shown by the network performance metrics in Figure 31, we can tell the performance of real-world schedule scenario is more similar to that of 18 random vehicles scenario. This is because even though there are 72 vehicles in real-world schedule scenario, they are not always serving all the time. Normally they wait on termination stops longer out of rush hours. It is also possible that some vehicles just serve for one day or just some certain trips.

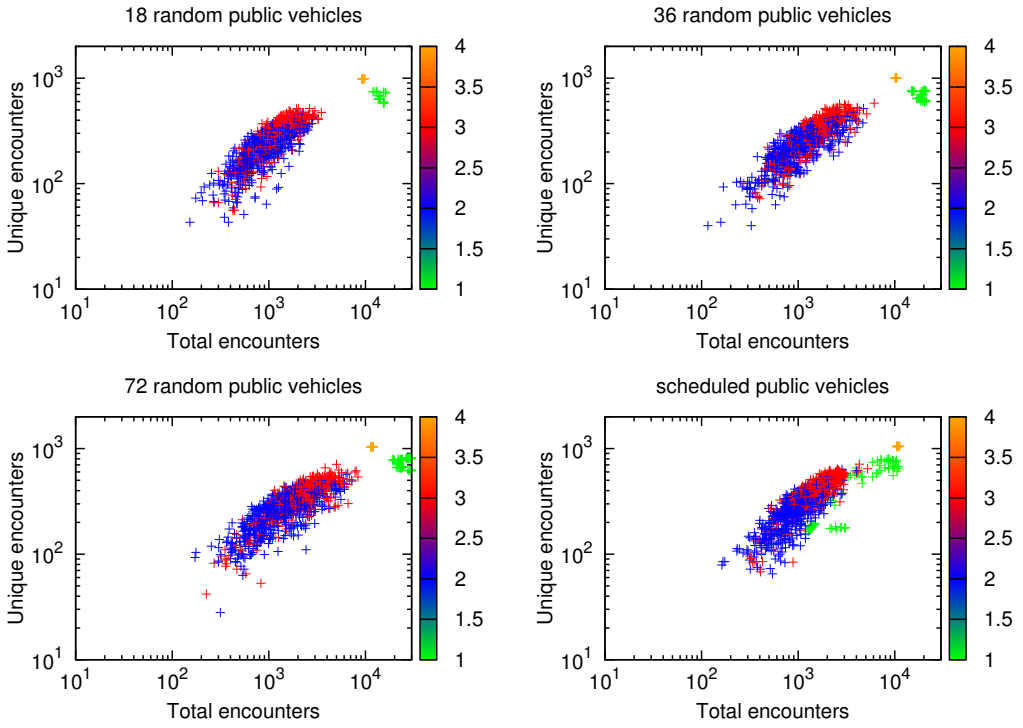

Figure 30: Comparing total encounters/unique encounters between random schedule and real-world schedule

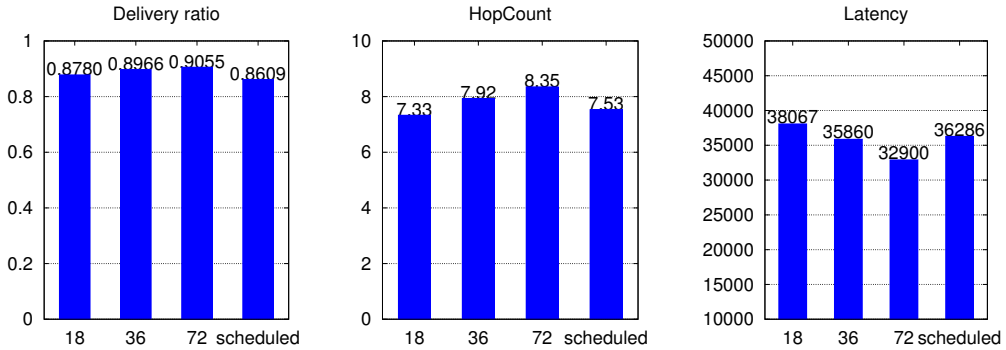

Figure 31: Comparing message statistics between random schedule and real-world schedule

6.4.2 Schedule match between citizen and public transportation

The other experiment compares the real-world schedule with citizens with different wake up time, to show the importance of the match between the schedules of citizen and public transportation system. We conducted simulation on three scenarios. Each of them only differentiates on the wake up time. We select 0 am, 4 am and 8 am as the wake up time. The wake up time of 8am had been prooved to be matching with the public transportation schedule in the the previous experiment.

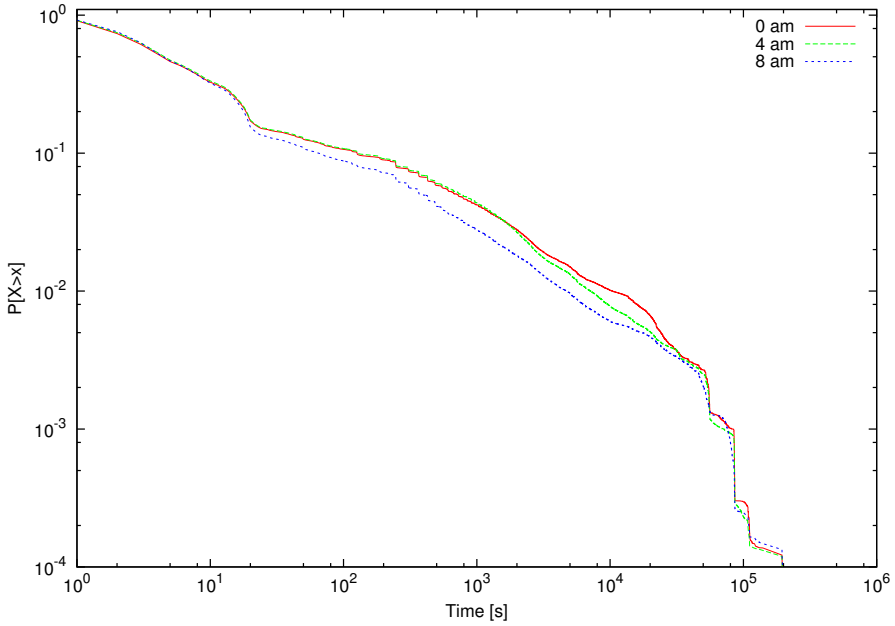

Figure 32: Comparing CCDF of contact duration between different wakeup time for real-world schedule

From the CCDF of contact duration in Figure 32 and inter-contact time in Figure 33, we observe the un-matched scenarios have longer contact duration and slightly longer inter-contact time. Both of the two changes can be explained by the longer time travelers have to wait at stops when the schedules are not matched. Longer contact duration is contributed by the longer time waiting together and more travelers on the same bus. While the longer inter-contact time is contributed by the longer time it takes to encounter a bus.

From the hourly activity chart in Figure 34, it shows the peak times are shifted and with dramastic burst in un-matched scenarios. While scenario of 4 am can still maintain a pattern of two peak times a day. The scenario of 0 am, with bigger mismatch with public transportation schedule, it develped more peaks during a day.

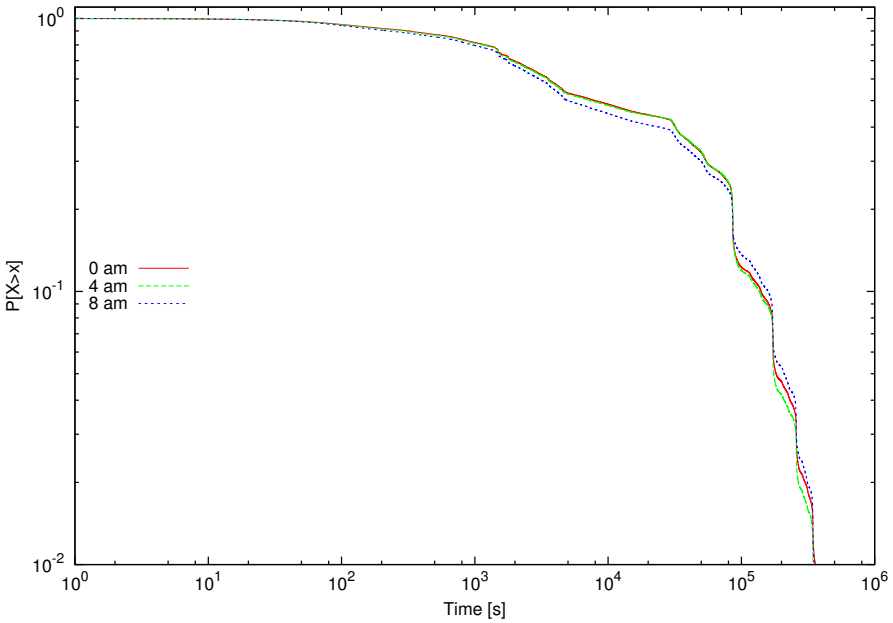

Figure 33: Comparing CCDF of inter-contact time between different wakeup time for real-world schedule

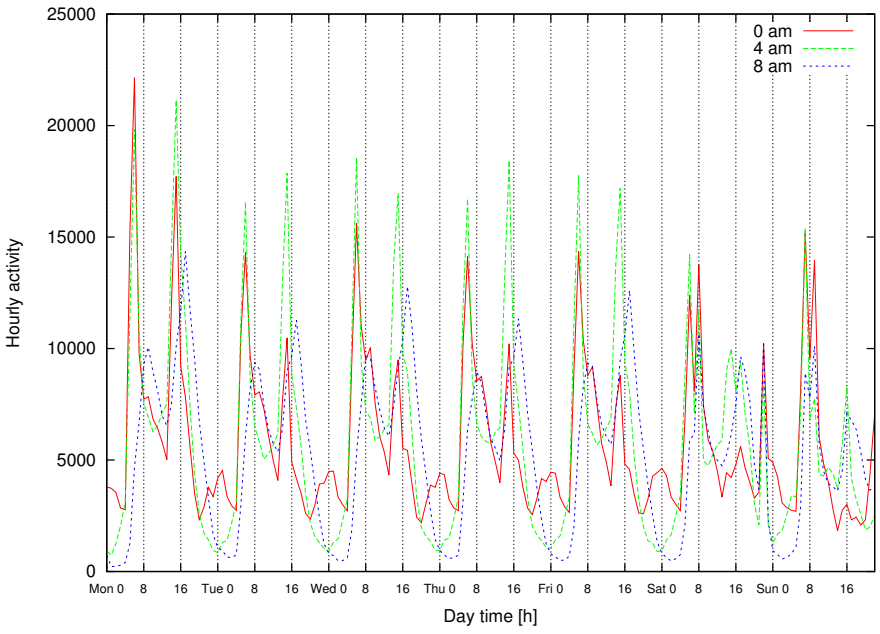

Figure 34: Comparing hourly activity between different wakeup time for real-world schedule

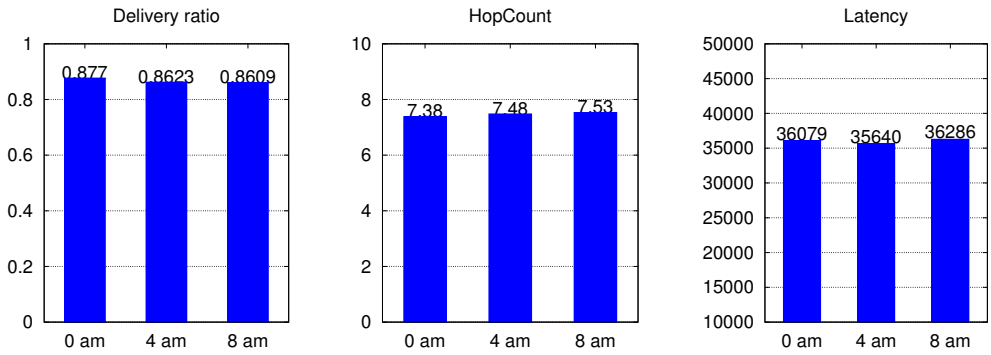

Figure 35: Comparing message statistics between different wakeup time for real-world schedule

6.5 Impact of multi-story office

We conducted simulations on scenarios with 1-story, 3-story and 5-story offices to see the impact of multi-story office.

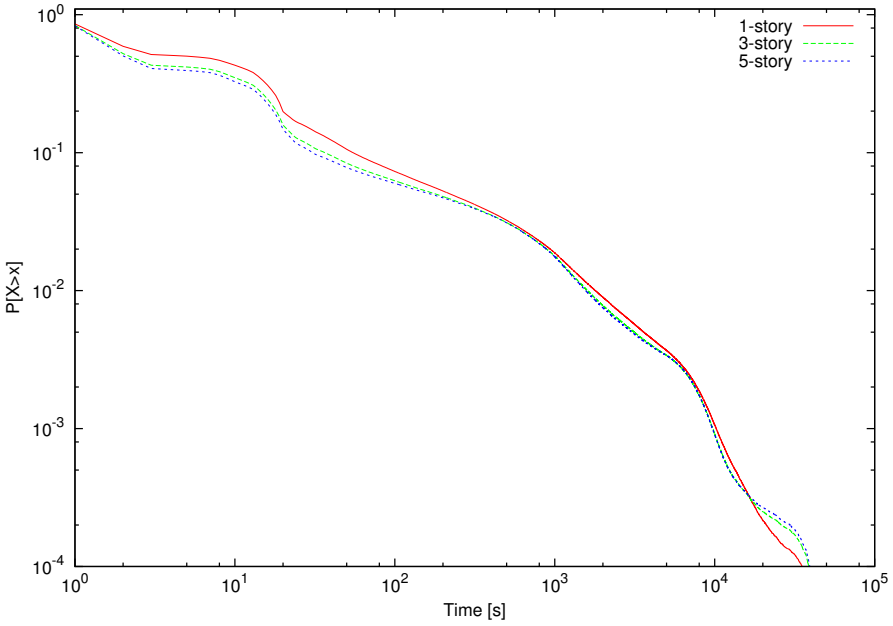

Figure 36: Comparing CCDF of contact duration between different stories

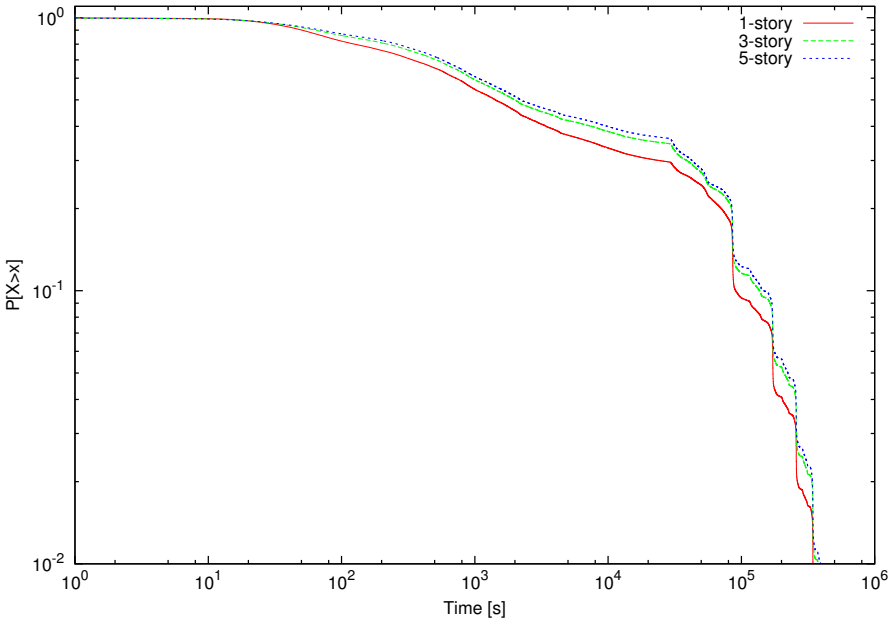

Figure 37: Comparing CCDF of inter-contact time between different stories

The CCDF of contact times in Figure 36 shows that in the scenario of 3-story and 5-story office, there are less contacts of moderate duration, roughly from 1 to

500 seconds. This correlates to the fact that in the 1-story scenario, contacts made in offices are normally within 500 seconds because in each 50*150 meters office there are only 5 people working there. And by randomly placing them into 3 or 5 stories, the chance to meet each other in this manner is smaller. The CCDF of inter-contact times in Figure 37 shows there are relatively more inter-contact times of long length as the result of less contacts made in offices.

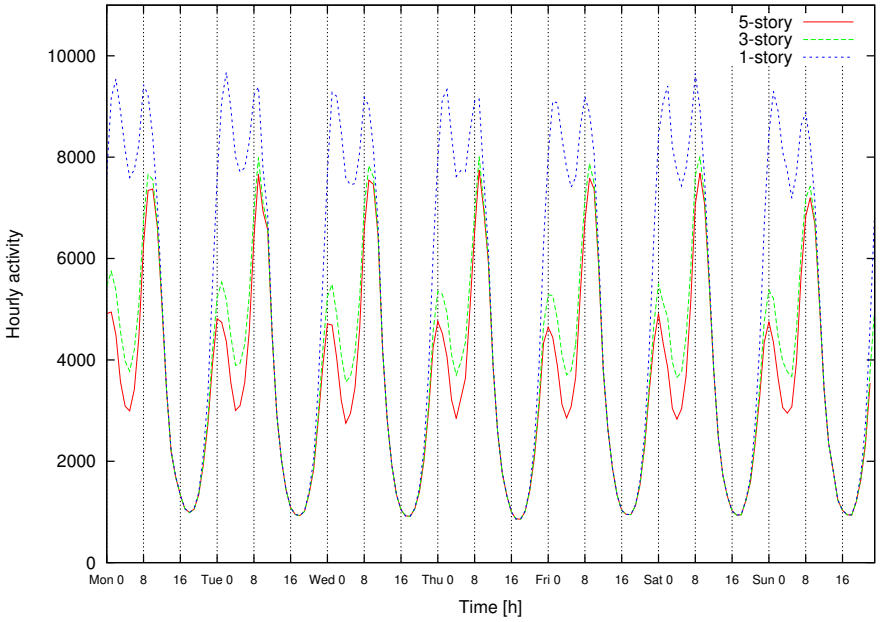

Figure 38: Comparing hourly activity between different stories

From the hourly activity chart in Figure 38, we observe some interesting difference caused by multi-story office. For scenarios of 3 and 5 stories, the first peak in a working day is much lower than the 1-story scenario, while the second peak does not have that much difference. This is because in multi-story scenario, the people moves from home to office have a much less chance to meet the ones already arrive at office. Thereby the line starts to go down earlier for the first peak than the line of 1-story scenario. Since the second peak is formed by the office to home or to evening events travelings, it is not affected that much by the multi-story office.

This scatter diagram in Figure 39 shows that as the number of stories increases, the the bunch of WDM nodes moves a little bit to the left and to the bottom, correlating to the fact that less contacts are made in office.

The network metrics charts in Figure 40 depict how much the performance of the epidemic routing protocol is affected by the number of stories of office. As the number of stories increases, the latency increases, and delivery ratio decreases because less contacts are made in office. The hop count always decreases because that messages relying on the relays that happen in office do not make it to the final receiver and are not counted in the hop count calculation.

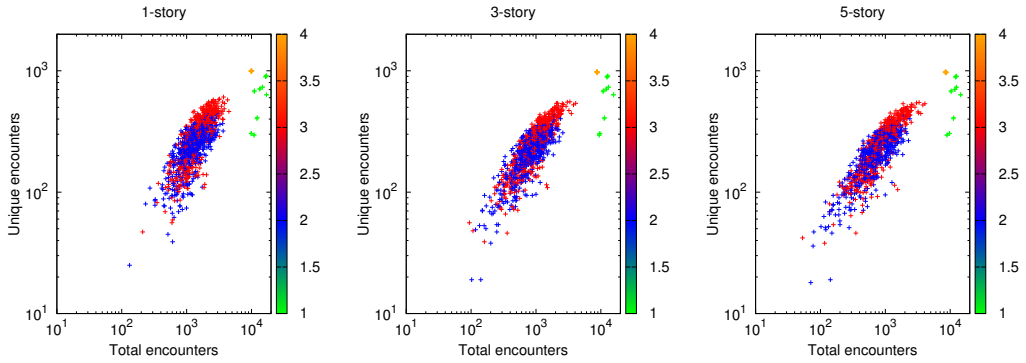

Figure 39: Comparing total encounters/unique encounters between different stories

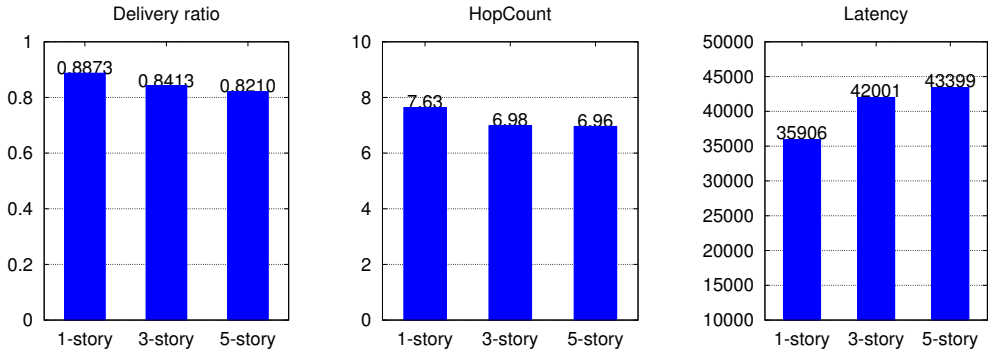

Figure 40: Comparing message statistics between different stories

6.6 Summary

In this chapter, we presented a series of comparitive experiments to reveal the impacts introduced by our implementation changes, the metro system, the real-world public transportation schedule, the match between citizen's wake up time and public transportation schedule, and the multistory offices. In the beginning part, we describe the default scenario for all of the experiments and the metrics we used for evaluating the experiment results. In the section of each comparitive experiment, we firstly descibed the varying condition to the default scenario. Then we gave the experiment result and the explanation and reasoning on the impacts revealed by the metrics.

7 Conclusions and future work

The goals of this thesis project can be grouped into two categories - software development and experiment. The output of software development phase serves as the basis of experiment phase. While the output of experiment phase serves as validation of software development, as well as the evaluation for the new supported scenarios. This chapter presents the the conclusions on these two categories respectively. After that the limitations of this project and the possible future work are discussed.

7.1 Software development achivements

We have changed the simulator into a multi-plane structure as a first order approximation of 3D model. Based on that, the scenarios of metro system and multistory office are supported. In addition, we have enhanced the models for public transportation system to support real-world schedule. Apart from that, we have developed a bunch of tools to facilitate converting GTFS data, the most popular timetable format used by public transit agencies in the world, into vehicle specific schedule data compatible with the ONE simulator.

7.2 Experiment findings

Metro sytem contributes to lower activity level which is easy to understand because the underground nodes can not contact to the overground nodes. We also found that the majority of the lost contacts are short contacts. Because short contacts can not support complete transfer on big messages, the impact has very limited effect on message delivery ratio, hop count and latency.

Our experiment on real-world schedule on public transportation system has proved that it has a significant impact on both mobility metrics and network performance. There is no general rule of how real-world schedules impact the simulation results. However there are two key factors of real-world schedule that makes a difference comparing to uniformly distributed random schedule. One factor is the interval of coming vehicles is not uniformly distributed. Normally the public transportation schedule of a city is compliant to the commuting schedule of citizens. There are more public vehicles running during on-peak hours while less running during off-peak hours. As the WDM traveler does not check the timetable before leaving home/office just like people normally do, longer intervals means more travelers waiting longer at the stops which further leads to more long duration contacts and better network performance. The other factor is the schedule match between citizens and public transportation system. Our experiment in 6.4.2 has shown that a 4-hour deviation results a totally different hourly activity level chart.

Multistory office has a signficant impact on network performance. With fixed number of nodes per office, the more stories the offices have, the lower message delivery ratio, longer latency and more hops there are. From this conclusion, we also know that office is an important place for message delivery.

7.3 Limitations

The multi-plane structure is a first order approximation of 3D model, which ignores quite a lot details a complete 3D model should have. Taking the behavior of going under into a metro station as an example, in real world, a people firstly enters the metro entrence overground, then takes the escalator to the underground platform. In our simulation, a metro station is just a poit on the 2D map, and when a metro traveler arrives at that point, it just teleports to the corresponding underground point without doing anything. The multistory office is built on the same simplicity. But there is another distortion for multistory office. As each story is a a different plane, there is no contact can be made cross stories. However, that is not true in real world where a strong wifi signal can be accessed from a couple of different stories.

There are two limitations for real-world schedule public transport. One is the public vehicle does not has the direction information about the current trip. Therefore, a traveler can get on a vehicle to the wrong direction. For random scheduled public transport, this is not a big problem. As vehicles will serve their routes until the simulation ends, the traveler in wrong direction can always be taken to the targeted stop in the next trip. Howver when comes to real-world schedule, vehicles are not necessarily serving forever. There could be vehicles serves only one day. In that case, if a travller gets on a bus on its last trip with the wrong direction, the traveler will be stuck in that vehicle until the simulation ends. To prevent that happens, we force travelers to get off when the vehicle finishes each trip. Although emptying a vehicl at the last stop of the trip is quite normal in real-world situation, considering WDM travelers can not tell the trip direction, it would introduce some extend of distortion to our simulation. The other limitation is our GTFS convertor can only produce a one week long schedule.

7.4 Future work

Solving the trip direction problem and the limitation of one week long schedule could be the possible next steps. Another improvement could be making the points-to-roads merging tool to automatically redress the displacement between road network and the points. It is likely that the stop locations from public transit agencies are not aligned with the road network data in WKT file. Currently, we have to manually redress this displacement by, for instance, showing stops and roads in OpenJump tool and figuring out the optimized offset vector to align them. This approach is time consuming and error-prone. Automation of this task is possible by moving the stops around to find the offset vector that has the smallest sum of the distances from stops to the nearest roads.

Following the direction of making public transportation travelers more intelligent, there could be a new feature that travelers can choose which route and which trip to take according to its current location, the destination, and the timetable of public transportation system. One option is to making travelers even more intelligent so that they know how to transfer between different routes.

Performance is a critical metric for any network simulator. For the current

ONE simulator, as all the simulation state updating logic is running in a single thread, running simulations in more powerful multi-core machine does not shorten the simulation time. One option is using multithreaded technique to take advantage of the power of multi-core CPU.

References

[1] Burleigh S., Hooke A., Torgerson L., Fall K., Cerf V., Durst B., Scott K., Weiss H. *Delay-Tolerant Networking: An Approach to Interplanetary Internet.* IEEE Communications Magazine, vol. 41, no. 6, pp. 128–136, June 2006.

[2] Kevin Fall. *A Delay-Tolerant Network Architecture for Challenged Internets.* Proc. of ACM SIGCOMM, August 2003.

[3] John Moy. *OSPF Version 2.* RFC 2328, April 1998.

[4] Gary Scott Malkin. *RIP Version 2.* RFC 2453, November 1998.

[5] Vinton Cerf and Scott Burleigh and Adrian Hooke and Leigh Torgerson and Robert Durst and Keith Scott and Kevin Fall and Howard Weiss. *Delay-Tolerant Networking Architecture.* RFC 4838, April 2007.

[6] Bardeen, J., Cooper, L. N. ja Schrieffer, J. R. Theory of Superconductivity. *Considering Pigeons for Carrying Delay-Tolerant Networking based Internet traffic in Developing Countries.* Electronic Journal of Information Systems in Developing Countries, vol. 54, 2012.

[7] Thomas Clausen and Philippe Jacquet. *Optimized Link State Routing Protocol (OLSR).* RFC 2326, October 2003.

[8] Charles E. Perkins and Elizabeth M. Belding-Royer and Samir R. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing* . Experimental RFC 3561, July 2003.

[9] David B. Johnson and David A. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks* . Mobile Computing, vol. 353, pp. 153–181, 1996.

[10] J. Burgess, B. Gallagher, D. Jensen, B. N. Levine. *MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks.* Proceedings of IEEE Infocom, Barcelona, April 2006.

[11] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh and Daniel Rubenstein. *Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet.* Proceedings of the ASPLOS-X conference, San Jose, CA, USA, October 2002.

[12] Alex (Sandy) Pentland, Richard Fletcher and Amir Hasson. *DakNet: Rethinking Connectivity in Developing Nations.* IEEE Computer, vol. 37, no. 1, pp. 78–83, January 2004.

[13] Shah R.C., Roy S., Jain S. and Brunette W.. *Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks.* Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, pp. 30–41, May 2003.

[14] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass and James Scott. *Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms.* Proceedings of INFOCOM 2006. 25th IEEE International Conference on Computer Communications, pp. 1–13, April 2006.

[15] A. Vahdat and D. Becker. *Epidemic routing for partially connected ad hoc networks.* Technical Report CS-200006, Duke University, April 2000.

[16] Thrasyvoulos Spyropoulos, Konstantinos Psounis and Cauligi S. Raghavendra. *Single-copy routing in intermittently connected mobile networks.* Proc. Sensor and Ad Hoc Communications and Networks SECON, pp. 235–244, October 2004.

[17] Thrasyvoulos Spyropoulos, Konstantinos Psounis and Cauligi S. Raghavendra. *Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks.* ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN), 2005.

[18] Thrasyvoulos Spyropoulos, Konstantinos Psounis and Cauligi S. Raghavendra. *Spray and Focus: Efficient Mobility-Assisted Routing for Heterogeneous and Correlated Mobility.* Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), 2007.

[19] Anders Lindgren and Avri Doria. *Probabilistic Routing Protocol for Intermittently Connected Networks.* Internet Draft draft- lindgren-dtnrg-prophet-02, Work in Progress, March 2006.

[20] John Heidemann, Nirupama Balusu, Nirupama Bulusu, Jeremy Elson, Chalermek Intanagonwiwat, Kun-chan Lan, Ya Xu, Wei Ye, Deborah Estrin and Ramesh Govindan. *Effects of Detail in Wireless Network Simulation.* SCS Communication Networks and Distributed Systems Modeling and Simulation Conference, January 2001.

[21] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva. *A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols.* Proceeding MobiCom '98 Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking, pp. 85–97, 1998.

[22] Christian Bettstetter. *Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects.* ACM SIGMOBILE Mobile Computing and Communications Review, vol. 5, no. 3, July 2001.

[23] Douglas M. Blough, Giovanni Resta and Paolo Santi. *A Statistical Analysis of the Long-Run Node Spatial Distribution in Mobile Ad Hoc Networks.* Proceeding MSWiM '02 Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems, pp. 30–37, 2002.

[24] Christian Bettstetter, Hannes Hartenstein and Xavier Pérez-Costa. *Stochastic Properties of the Random Waypoint Mobility Model.* Wireless Networks, vol. 10, no. 5, pp. 555–567, September 2004.

[25] Royer E.M., Melliar-Smith P.M. and Moser L.E.. *An Analysis of the Optimum Node Density of Ad hoc Mobile Networks.* Communications, 2001. ICC 2001. IEEE International Conference, vol. 3, pp. 857–861, June 2001.

[26] Tian J., Hähner, J., Becker C., Stepanov I.. *Graph-based Mobility Model for Mobile Ad Hoc Network Simulation.* Proceedings of Simulation Symposium, 35th Annual, pp. 14–18, April 2002.

[27] Mirco Musolesi and Cecilia Mascolo. *A community based mobility model forad hoc network research.* REALMAN '06 Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality, pp. 31–38, May 2006.

[28] Jani Lakkakorpi, Mikko Pitkänen and Jörg Ott. *Adaptive Routing in Mobile Opportunistic Networks.* MSWIM '10 Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems, October 2010.

[29] Jani Lakkakorpi and Philip Ginzboorg. *ns-3 Module for Routing and Congestion ControlStudies in Mobile Opportunistic DTNs.* International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS '13), pp. 46–50, July 2013.

[30] Frans Ekman, Ari Keränen, Jouni Karvo and Jörg Ott. *Working Day Movement Model.* MobilityModels'08, Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models, pp. 33–40, May 26, 2008.