Routing in Networks with Stochastic Waiting and Transit Times

Claudio Salmin

25th February, 2010

Contents

Chapter 1

Introduction

Intermittently Connected Networks (ICNs), also known as Delay Tolerant Networks (DTNs), are mobile wireless networks where most of the time there is no complete path from a source to a destination (because the network is sparse or because of nodes mobility and vagaries of the wireless channel). In such a scenario information delivery is then based on the store-carry-forward paradigm: a mobile node first stores the routing message from the source, carries it from a physical location to another and then forwards it to another intermediate node or to the destination. Most of the research on routing in ICNs has focused on two extreme cases: 1) when contacts among nodes are deterministic and known in advance or 2) when they cannot be predicted and are supposed to obey to some generic random mobility model. Many interesting scenarios do not fall in any of these two cases, because the underlying node mobility is known in advance, but it can be modified by random effects. Maestro team at INRIA is currently working on such networks, called *"quasi-deterministic" ICNs*. In particular they want to address the routing and scheduling problem in a public bus transportation of a major industrial city. My contribution to this research project is to develop a smulator which, given the real data traces or the scheduling of the buses in the public transportation network, determines all the possible transmission opportunities between both bus to bus and bus to stop. Given the problem of where to place the wireless device to transmit the data, on the buses, on the stops or on both buses and stops, the results of this simulator will help to decide which one can be the best strategy under the conditions of flooding algorithm for the dissamination of the data.

My research project was organized into two major parts: modelization of the algorithm to determine the transmission opportunities and simulator implementation.

1.1 Scientific Background of ICNs

Some of the recent applications using wireless communications (wildlife monitoring, inter-vehicles communication, battlefield communication,...) are characterized by challenging network scenarios. Most of the time there is not a complete path from a source to a destination (because the network is sparse), or such a path is highly unstable and may change or break while being discovered (because of nodes mobility and time-variations of the wireless channel). Networks under these conditions are usually referred to as Intermittently Connected Networks (ICNs) or Delay Tolerant Networks (DTNs). In these challenging environments, popular ad hoc routing protocols such as AODV [2] and OLSR [1] fail to establish routes. This is due to these protocols trying to first establish a complete route and then, after the route has been established, forward the actual data. Such protocols are therefore not suited for these new scenarios. Information delivery is then based on the store-carry-forward paradigm: a mobile node first stores the routing message from the source, carries it from a physical location to another and then forwards it to an intermediate node or to the destination. It is important to notice that the performance of ICNs are directly dictated by the inter-contact times between the nodes (which, in turn, are the result of nodes mobility pattern in the network). In particular, packet delivery delays become comparable with these inter-contact times, implying that application running on such networks should be tolerant to delays of order of minutes/hours (from which the name of delay tolerant networks originates). At the core of this research line are *routing and scheduling algorithms*: at any given time, each node should find when and where to forward the data stored in its buffer so that it reaches the destination in a timely manner. Moreover routing for ICNs is not only limited to *forwarding schemes*, where a single copy of each packet is present in the network [7], but it also include *replication schemes*, which send many copies of the same data packet across the network. A prime example of replication schemes are *epidemic* routing algorithms (a.k.a *flooding algorithms*) in which each node sends each packet to all its neighbors. Replication improves performance in terms of delivery probability and delivery delay when contacts cannot be predicted or when transmissions are unreliable, but at the same time it implies higher costs in terms of required bandwidth, transmission energy and buffer requirements (see [8]).

1.2 General Research Objectives

Most of the research on routing in ICNs has focused on two extreme cases: 1) when contacts among nodes are deterministic and known in advance (e.g. in the case of space communications among satellites, probes and earth or space stations [9]) or 2) when they cannot be predicted (e.g. for human and animal mobility [3, 4]) and are supposed to obey to some generic random mobility model, like random way-point, random direction or Brownian models. Many interesting scenarios do not fall in any of these two cases: even complex mobility patterns often exhibit some form of periodicity or in other cases the underlying node mobility is known in advance, but it can be modified by random effects. A clear example is that of a vehicular network carrying data over public transportation (e.g., buses): the predictions of the contact times are derived from the schedule and routes of the buses; on the other hand, delays in bus operations clearly change the contact times or even prevent contact to occur, implying the predictions are not necessarily accurate.

A preliminary investigation done by *Maestro team (INRIA)* in collaboration with *Polythecnic of Turin* and *Columbia University* suggests that there is currently no framework to study comprehensively all the range of possible scenarios between deterministic contacts and unpredictable random contacts. For this reason, during the project, they plan to focus only on a specific class of networks characterized by small deviations from the deterministic contact model ("quasi-deterministic" ICNs). In particular, their assumption is that predictions of the contact times between nodes are known in advance, however these predictions are not necessarily accurate: contacts can occur either before or after their prediction times (e.g. the bus my arrive earlier or later to the stop).

1.3 Bus Switched Network

The model is applied to a public transportation network, where a node can be either a bus or a bus stop. It is considered an opportunistic data network formed by (some) buses and bus stops equipped with wireless devices, e.g. based on WiFi technologies, like in DieselNet [15]. Most of the stops act as disconnected relay nodes, and a few of them are also connected to the Internet. Data are delivered across town following the store-carry-forward network paradigm, based on multi-hop communication in which two nodes may exchange data messages whenever they are within transmission range of each other. A Bus-based network is a convenient solution as wireless backbone for delay tolerant applications in a urban scenario. In fact, the public transportation system provides access to a large set of users (e.g. the passengers themselves) and is already designed to guarantee a coverage of the urban area, taking into account human mobility patterns.

1.4 The Specific Task

The aim of my research project is to build a simulator that, given the real data traces or the schedules of a public bus transportation, detects all the possible *transmission opportunities* between both bus to bus and bus to stop. I consider as *transmission opportunities* a moment in time when a bus enters in the wireless device range of either an other bus or a stop, at this moment, for both the nodes, is possible to establish a link and exchange some data. With the simulator I can determine both the start and the finish time of the transmission opportunities.

Thanks to the results of the simulation I can define how data is spread over the network (how the bus stops are infected). I have considered a public transportation system in a major industrial city (Turin), with more than one million people living in the main urban area and more then two millions inhabitants in the whole metropolitan area. My starting point is a set of schedules of this public transportation system serving an area of about 200 km^2 through about 7500 stops and 1500 different vehicles distributed among 250 different lines. These traces include the complete schedule for a working day.

Given the high number of stops compared to the number of buses, with the results provided by the simulator is also possible to understand whether investing money in infrastructure equipment to install wireless device at the stops (at least half of them should be equipped) is a good strategy or not.

The traces use the format specified by the *Google Transit Data Feed*. All the transmission opportunity are determined using geometrical models described in sections 2.3.1 and 2.3.2.

Chapter 2

The Simulator

2.1 Challenges and Design Choices

I suppose that each node (bus or bus stop) is equipped with a wireless device, this means that each node is able to establish a link and deliver data whenever an other node enter its wireless range. The aim of this simulator is to find all the possible transmission opportunities between both bus to bus and bus to stop. From the collection of traces I derived 7500 stops and 1500 different vehicles distributed among 250 different lines which create a quite big graph if I want to determine all the possible transmission opportunities. Moreover the buses are changing their position with the time and I did not want to sample the system because it is difficult to determine the good sampling frequency: if it is too high your system state might not change between two sampling periods, on the other hand, if it is too low you might loose some states. In order to simplify this complex scenario I decided to divide the problem into two easier subproblems:

- Given two consecutive stops, I create an imaginary line segment the connects them. This provides me a large set of segments where each pair of segments represent a potential transmission opportunity, let me call it intersection. Of course the main condition is that the two segment do not have to belong to the same bus line. The problem is that each node has a transmission range, this means that I can not simply apply a segment to segment intersection mechanism. Hence, the final solution is to determine the minimum distance between two line segments, if this distance is lower than the wireless device range, these two segments are stored as *potential intersection*.

- At this point I have all the *potential intersections* between segments. To determine the bus to stop it is really easy because the stop is never moving. For the transmission opportunities of a bus with the stops of its line this is really trivial, I just need to consider the stop as a circle,

where the radius of the circle is the range of the wireless device, the bus as line segment that intersect the circle with a given speed, and determine the instant time the transmission opportunity start and finishes, this transmission opportunities always happen. Not more complicate than the previous case is the case of bus transmission opportunities with the other stops of all the other lines, in fact in this case the procedure is more or less the same as before, I just need to make sure that the distance between the segment I am considering and the bus stop is lower than the wireless device range, also in this case I am able to determine the instant time the transmission opportunity start and finishes. More complex is the case of the transmission opportunity between two segments. In fact, it might happen that two segments are physically close enough that there is a transmission opportunity but the schedule of the buses is such that they can not meet. In this case the solution is to consider one of the two buses as not moving circle, where the radius of the circle is the range of the wireless device, and the other bus moving with a relative speed to the first bus. If the second bus intersects the circle before one of the two buses change segment, I am sure that the two buses intersect and I am able to determine both the start and the finish transmission opportunity time.

Technical Details

The simulator is written in *C++*, the development environment is eclipse with the C/C++ Development Toolkit (CDT). The CDT is an open source project (licensed under the Common Public License) implemented purely in the Java programming language as a set of plug-ins for the Eclipse SDK Platform. These plug-ins add a C/C++ Perspective to the Eclipse Workbench that can now support C/C++ development with a number of views and wizards, along with advanced editing and debugging support, for more details about CDT consult [14]. The choice of C++ language is due to the following reason:

- efficient execution;
- explicit memory management;
- using references and pointers;

2.2 High Level Description

Hereafter there is a high level description of the steps followed to create the data structure and to determine the transmission opportunities:

1. Read from the file *route.txt* all the available routes. Read the file *calendar_dates.txt* and store the service_id associated to a specific date. Read the *trips.txt* file and associate each trip to the its corresponding route, the trip is considered only if its service_id is in the ones previously stored. Read the *stop_times_filtered.txt* file and associate to each trip the list of its bus stops with the arrival time and the sequence number. Read the file *stops.txt* which provides the position (in Latitude and Longitude) of each bus stop. At this point the *data structure* is correctly created.

2. We now have to create the *segments*. To do this I first need to create a *map* where the KEY is the trip ID of a trip with a specific directions and the $VALUE$ is the list of bus stops associated to that trip, it is important to highlight that different trips might have same direction and indeed the same list of bus stops, with my *map* I consider this trips only once. Once the *map of stops* is created I can create the segments, a segment is a line segment that joins two consecutive bus stops, in one segment I can have one and only one bus. It is clear that a segment is actually an approximation of the possible path that a bus can follow, anyway from a theoretical point of view looking at only one trajectory I have no ways to understand if the path is really oblique or if you are skipping corner.

3. After the creation of the segment I can start my first algorithm, the one that calculates the minimum distance between to segment and, if this distance is lower than the wireless device range, these two segments are stored in a *map* that has a pair of bus stops IDs as KEY (this pair actually represent the segment) and a list of all that segments that it might intersect with.

4. As last step, I apply the circle segment intersection algorithm to all the segments contained in the potential intersection map. We call this mechanism circle segment because, give the two segments that might intersect, I consider one bus as not moving (which identifies the circle) and the other bus moving (which identifies the segment) with a relative speed the other bus. With this mechanism, depending from the schedule of each bus, I determine whether an transmission opportunity is really happening or not. In this last step I also determine all the transmission opportunity between bus and stops.

In sections 2.3.1 and 2.3.2 I describe in more details the *segment to segment* and *circle to segment* mechanisms.

2.3 Main Algorithms

2.3.1 Segment to Segment Algorithm

When I apply this mechanism I have two possible cases:

1. The two segments intersect: in this case I simply add the two segment in my potential intersection map;

2. The two segments do not intersect: I need to calculate the minimum distance between them and, if this distance is smaller than a given threshold (wireless device range), I can add the two segment to the potential intersection map.

Segment to Segment Intersection

We have two line segments l_1 and l_2 and let l_i is defined in the interval $[(x_1^i, y_1^i), (x_2^i, y_2^i)]$. If these lines were infinite and pass through these points, I could say that these lines intersect at a unique point if they are not parallel. l_i (assuming infinity) is defined by the following formula:

$$y - y_1^i = \frac{y_2^i - y_1^i}{x_2^i - x_1^i}(x - x_1^i)$$

So, intersection point is the solution of the following linear equation.

$$Au = B$$

where $u = [x \quad y]^T$, $A = \begin{bmatrix} -(y_2^1 - y_1^1) & (x_2^1 - x_1^1) \\ -(y_2^2 - y_1^2) & (x_2^2 - x_1^2) \end{bmatrix}$

and $B = \begin{bmatrix} y_1^1(x_2^1 - x_1^1) - x_1^1(y_2^1 - y_1^1) \\ y_1^2(x_2^2 - x_1^2) - x_1^2(y_2^2 - y_1^2) \end{bmatrix}$.

The solution is $u = A^{-1}B$. There is unique u if A is non-singular, otherwise the lines are parallel and they don't intersect. If the solution lies on both the segments, i.e. $x_1^i \leq x \leq x_2^i$ (assuming $x_2^i > x_1^1$) and $y_1^i \leq y \leq y_2^i$ for both $i = 1$ and $i = 2$, I can conclude that the line segments intersect.

Segment to Segment Minimum Distance

Here is a simple mechanism to see if the minimum distance between two line segment is above a given threshold R. If this is true, there is no possibility for a bus on one of these segments to create a link with another bus on the other segment. Note that I have to repeat this for all segments, if there are n segments, the number of comparisons is $n(n-1)/2$.

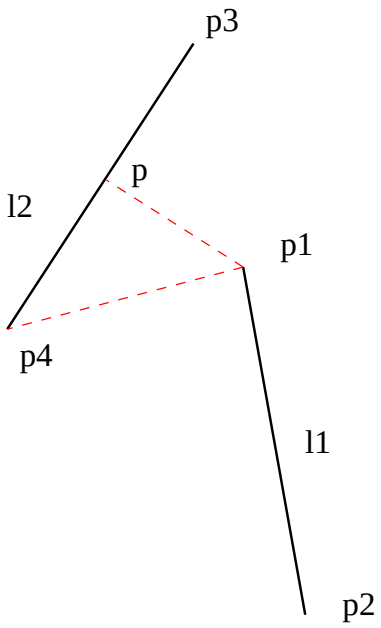

Figure 2.1: Minimum distance between $l1$ and $l2$

First consider the case in Fig. 2.1 that two line segments $l1$ and $l2$ don't intersect. The minimum distance between two line segments is the minimum of the four following values $D(p1, l2), D(p2, l2), D(p3, l1), D(p4, l1)$ where $D(a, b)$ is the distance operator, a is a point and b is either a point or a line segment. If b is a line segment, $D(a, b)$ gives the minimum distance between a and any point on b.

So, I start with the distance between a point and a line segment. Let's consider $D(p1, l2)$. We have to find the distance $D(p1, p)$, the projection of the vector $\mathbf{v1}$, that connects $p4$ to $p1$, onto the vector that connects $p4$ to $p3$, $\mathbf{v2}$. Note that using $p3$ instead of $p4$ would give us the same projection and the same point p.

$$D(p4, p) = \frac{\mathbf{v1} \cdot \mathbf{v2}}{|\mathbf{v2}|}$$

Note that if the dot product is negative, $\mathbf{v1} \cdot \mathbf{v2} \leq 0$, point p is not located on $l2$ and $D(p1, l2) = \min(D(p1, p3), D(p1, p4))$.

Using $D(p4, p)$, it easy to calculate $D(p1, p)$.

$$D(p1, p) = \sqrt{|\mathbf{v1}|^2 - (D(p4, p))^2}$$

If p is on the line segment $D(p1, l2) = D(p1, p)$. However, it is likely that p is still out of the line segment. In this case, $D(p1, l2) = \min(D(p1, p3), D(p1, p4))$.

Similarly, I can calculate $D(p2, l2), D(p3, l1), D(p4, l1)$ as well. The minimum of these values is the minimum distance between $l1$ and $l2$.

Pseudo-Code

Following is a the pseudo-code to find the minimum distance between $l1$ and $l2$. We should perform this procedure for each $l1$, $l2$ pair.

`Find the minimum distance between` $l1$ `and` $l2$

1: **if** $l1$ intersects $l2$ (found using the mechanism in Section 2.3.1) **then**
2: Return 0
3: **end if**
4: $Dmin = \infty$
5: $Dmin = \min(Dmin, dist(p1, l2))$
6: $Dmin = \min(Dmin, dist(p2, l2))$
7: $Dmin = \min(Dmin, dist(p3, l1))$
8: $Dmin = \min(Dmin, dist(p4, l1))$
9: Return $Dmin$

`dist(p,l)`

1: Calculate **v1** and **v2**
//$e1(l)$ is the first endpoint of segment l
//**v1** connects $e1(l)$ to p
//**v2** connects $e1(l)$ to $e2(l)$
2: **if** $\mathbf{v1} \cdot \mathbf{v2} < 0$ **then**
3: Return $\min(D(p, e1(l)), D(p, e2(l)))$
4: **end if**
5: $d = \sqrt{|\mathbf{v1}|^2 - \left(\frac{\mathbf{v1} \cdot \mathbf{v2}}{|\mathbf{v2}|}\right)^2}$
6: Return $\min(d, D(p, e1(l)), D(p, e2(l)))$

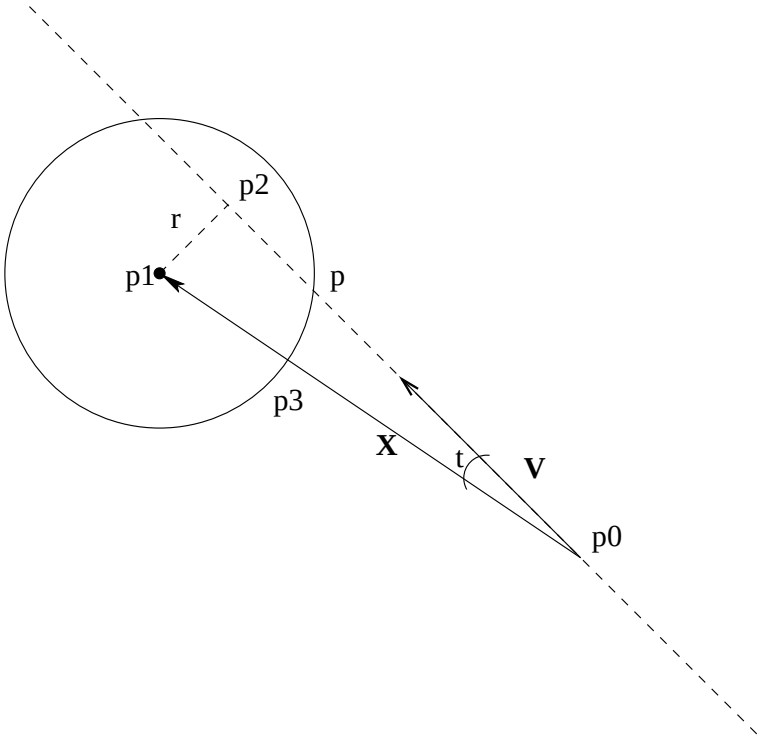

Figure 2.2: Intersection between a line and a circle

2.3.2 Circle to Segment Algorithm

The major idea of this mechanism is that given two buses (two segments) that might intersect, I consider one of them as not moving and the other as moving with relative speed to the other bus. The difficulty is that the schedule of the buses has the precision of one minute, this implies that in the provided data it might happen that, if the distance between two stops is small enough (around 200 meters), one bus has the same arrival time for two different stops.

Let us consider first the case when the arrival time at the two stops of a segment is different for the both buses. In this case, given the two buses $b1$ and $b2$, their arrival time at the two stops of the segment t_1 t_2 for $b1$ and $t_1^{'}$ $t_2^{'}$ for $b2$, I have to determine $max(t_1, t_1^{'}) = t_{1max}$ and $min(t_2, t_2^{'}) = t_{2min}$ and calculate the position of the two buses in these two instant of time. After this I can consider that two buses are located in points $p0$ and $p1$ at time t_{1max}. We would like to see if one bus gets in the communication range of the other. We would like to find if the point p exists. In Figure 2.2, bold notations in capital letters, i.e. $\mathbf{V}$ and $\mathbf{X}$, denote vectors. $\mathbf{V}$ denotes the relative speed of the located at point $p0$ as seen by the other node and obtained by vector difference. Let $\mathbf{V}$ hold until t_{2min}, (i.e. until either of

the buses arrive at another stop which also means change of line segment). $\mathbf{X}$ is the vector that connects point $p0$ to $p1$ and $D(a, b)$ be the distance operator between two points a, b. Note that $|\mathbf{X}| = D(p0, p1)$.

We start the evaluation by calculating the dot product, $\mathbf{X} \cdot \mathbf{V}$. If this value is negative, then the buses are going away from each other so there is no such point p. Otherwise, in order to see if there is an intersection I have to calculate r. If $r > R$, R is the radius of the circle, there is no intersection. We have

$$r = \sqrt{|\mathbf{X}|^2 - D(p0, p2)^2}, \tag{2.1}$$

where $D(p0, p2)$ is the projection of $\mathbf{X}$ onto $\mathbf{V}$ and can be calculated as

$$D(p0, p2) = \frac{\mathbf{X} \cdot \mathbf{V}}{|\mathbf{V}|}. \tag{2.2}$$

Dot product also gives us the angle t: $\cos(t) = \frac{\mathbf{X} \cdot \mathbf{V}}{|\mathbf{X}||\mathbf{V}|}$

If $r \leq R$, there is an intersection and I need to know when this happens. First I have to find $D(p0, p)$. Let $d = D(p0, p)$. From law of cosines

$$R^2 = d^2 + |\mathbf{X}|^2 - 2d|\mathbf{X}|\cos(t). \tag{2.3}$$

Note that $D(p, p1) = R$. Using this relationship, I derive d. The time that the intersection occurs is $T = t_{1max} + \frac{d}{|\mathbf{V}|}$. If $T \leq t_{2min}$, then two nodes contact at time T.

The pseudo-code to evaluate if there is intersection is following:

`FindIntersectionTime` $(p0, p1, \mathbf{V}, t_{1max}, t_{2min})$

1: Calculate $\mathbf{X}$, (the vector that connects $p0$ to $p1$
2: $DP \leftarrow \mathbf{X} \cdot \mathbf{V}$
3: **if** $DP < 0$ **then**
4: Return NULL
5: **end if**
6: Calculate $D(p0, p2)$ using (2.2)
7: Calculate r using (2.1)
8: **if** $r > R$ **then**
9: Return NULL
10: **end if**
11: Derive d from (2.3)
12: $T \leftarrow t_{1max} + \frac{d}{|\mathbf{V}|}$
13: **if** $T > t_{2min}$ **then**
14: Return NULL
15: **end if**
16: Return T

If I now consider the case where at least one of the two buses has the same arrival time at the two stops of one segment, I am not in the condition

of evaluate the speed. After a long analysis I found out that there are five particular cases that need to be considered:

1. $\boldsymbol{t_1 = t_2}$ ***and*** $\boldsymbol{t_1 = t_1^{'}}$ ***and*** $\boldsymbol{t_1^{'} < t_2^{'}}$: in this case I have to determine the distance between a line segment (the one of $\boldsymbol{b1}$) and a point (the position of $\boldsymbol{b2}$ at time $\boldsymbol{t_1^{'}}$), if this distance is below a give threshold it means that the transmission opportunity is going to happen at time $\boldsymbol{t_1 = t_2 = t_1^{'}}$.

2. $\boldsymbol{t_1 < t_2}$ ***and*** $\boldsymbol{t_2 = t_1^{'}}$ ***and*** $\boldsymbol{t_1^{'} = t_2^{'}}$: this case the same as the previous one except that now I consider the segment of the bus $\boldsymbol{b2}$ and the position of $\boldsymbol{b1}$ at time $\boldsymbol{t_2}$, if this distance is below a give threshold it means that the transmission opportunity is going to happen at time $\boldsymbol{t_2 = t_1^{'} = t_2^{'}}$.

3. $\boldsymbol{t_1 = t_2}$ ***and*** $\boldsymbol{t_1^{'} = t_2^{'}}$: in this case I have that all the four arrival time are the same and I can infer that the transmission opportunity is going to happen at time $\boldsymbol{t_1 = t_2 = t_1^{'} = t_2^{'}}$.

4. $\boldsymbol{t_1 = t_1^{'}}$ ***and*** $\boldsymbol{t_1 = t_2^{'}}$: in this case I have to determine the distance between a line segment (the one of $\boldsymbol{b2}$) and a point (the position of $\boldsymbol{b1}$ at time $\boldsymbol{t_1}$), if this distance is below a give threshold it means that the transmission opportunity is going to happen at time $\boldsymbol{t_1 = t_1^{'} = t_2^{'}}$.

5. $\boldsymbol{t_1 < t_2}$ ***and*** $\boldsymbol{t_1^{'} = t_2^{'}}$: in this case I need to evaluate the position of the bus $\boldsymbol{b1}$ in the instant time $\boldsymbol{t_1^{'} = t_2^{'}}$ and then evaluate the minimum distance between the new generated segment and the one of $\boldsymbol{b2}$, if this distance is below a give threshold it means that the transmission opportunity is going to happen at time $\boldsymbol{t_1^{'} = t_2^{'}}$.

Chapter 3

Results

As said before, the starting point is a set of schedules of a this public transportation system serving an area of about 200 km^2 through about 7500 stops and 1500 different vehicles distributed among 250 different lines. These traces include the complete schedule for a working day and the corresponding GPS traces with the positions of all the vehicles during the morning rush hour period (6 AM– 10 AM). With the simulator I was able to find around 115000 bus to bus transmission opportunities and 233000 bus to stop transmission opportunities. In figure 3.1 are plotted three curves which represent the cumulative number of infected stops over the time when are considered three different ways of routing data:

- using bus to bus and bus to stop routing;
- using bus to stop and stop to bus routing;
- using all to all.

To have this result I examined a scenario where a data packet is injected into the network starting from *Porta Susa*, which is a train station and indeed a place with a lot of bus connections, the packet is disseminate in the network using a flooding algorithm. When the packet reaches a bus stop (potential destination) we increase the number of infected stops.

As can be seen from the figure, the curves *All & All* and *Bus2Bus & Bus2Stop* are really close to each other, therefore a first conclusion is that, using flooding algorithm, the installation of wireless devices at the bus stops (half of them is enough) and consequently use the *all to all* routing algorithm does not gives really better performance then using the *bus to bus and bus to stop* only.

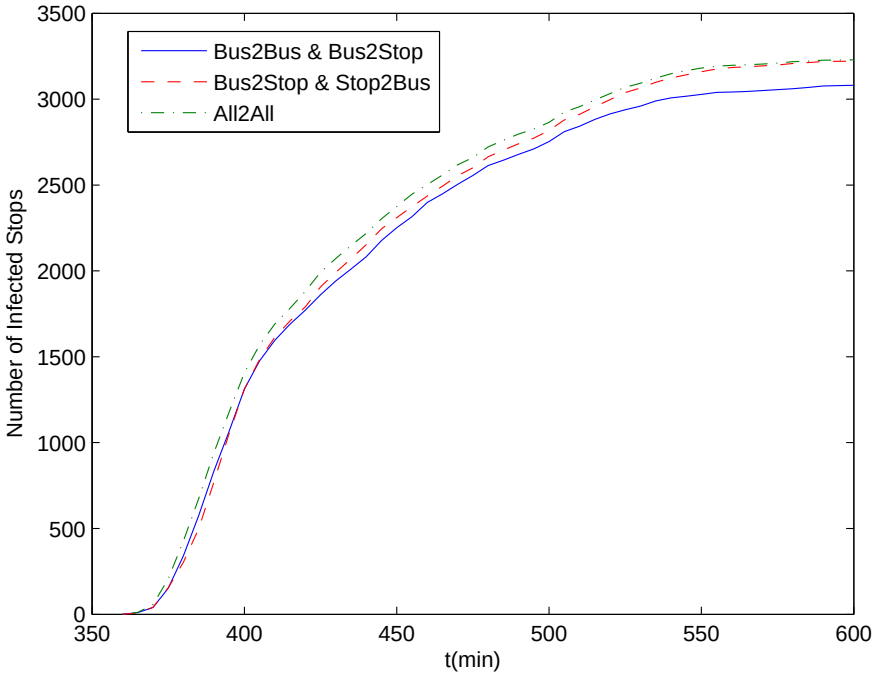

Figure 3.1: Number of Infected Stops

Chapter 4

Conclusion and Future Work

During this project I developed a simulator that, given the real traces or the schedule of a public transportation network, is able to determine all the transmission opportunities between bus to bus and bus to stop. Thanks to this transmission opportunities I am able to determine how data is going to be spread in the network and the number of infected stops over the time. Moreover, from the results I saw that the bus to stop and stop two bus routing does not give an important added value to the us to bus and bus to stop routing in terms of number of infected stops.

Future work related to this project is the design of routing algorithms in a public bus transportation network, where Wi-Fi enabled buses and stops may be used for data delivery as well as for the transportation of passengers. There are different aspects to work on for this problem:

- characterization of bus mobility and bus network topology starting from real-world traces,
- study of existing routing algorithms for networks with stochastic waiting and transit times,
- proposal of off-line and on-line routing algorithms,
- implementation of the new algorithms and of other standard ones for DTNs in order to compare their performance.

Bibliography

[1] T. Clausen, P. J. (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot, Optimized link state routing protocol (OLSR), RFC 3626, pages 1-75, pp. 1–75, October 2003, network Working Group. [Online]. Available: http://ietf.org/rfc/rfc3626.txt.

[2] C. Perkins, E. Royer, and S. Das, "RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing," 2003. [Online]. Available: http://tools.ietf.org/html/rfc3561

[3] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks," in IEEE INFOCOM, 2006.

[4] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in zebranet," in SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems. New York, NY, USA: ACM, 2004, pp. 227–238.

[5] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in ACM SIGCOMM, 2004, pp. 145–158.

[6] "Delay tolerant networking research group." [Online]. Available: http://www.dtnrg.org.

[7] A. Balasubramanian, B. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem," SIGCOMM Comput. Commun. Rev., vol. 37, no. 4, pp. 373–384, 2007.

[8] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance modeling of epidemic routing," Comput. Netw., vol. 51, no. 10, pp. 2867–2891, 2007.

[9] L. Wood, W. Ivancic, W. Eddy, D. Stewart, J. Northam, C. Jackson, and A. da Silva Curiel, "Use of the delay-tolerant networking bundle protocol from space," in Proc. of the 59th International Astronautical Congress, September 2008.

[10] Google Transit Feed Specifications, Available: http://code.google.com/transit/spec/transit_feed_specification.html.

[11] Universal Transverse Mercator, Available: http://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system.

[12] Universal Transverse Mercator conversion formulas, Available: http://www.uwgb.edu/dutchs/UsefulData/UTMFormulas.HTM.

[13] Johon P. Snyder, "Map Projection - A Working Manual", Chapter 7.

[14] C++ Development Toolkit, Available: http://www.ibm.com/developerworks/opensource/library/os-ecc/.

[15] UMass DieselNet Research. http://prisms.cs.umass.edu/dome/umassdieselnet.

Appendix A

Google Transit Feed

The Google Transit Data Feed Open Source Software project is an effort to offer tools for reading, writing, and converting to and from the Google Transit Feed Specification (GTFS) format, to help make public transit information projects more successful for agencies and other interested parties. The project currently offers code for working with transit data in the Java and Python languages. The GTFS defines a common format for public transportation schedules and associated geographic information. The GTFS document [10] explains the types of files that comprise a GTFS transit feed and defines the fields used in all of those files. In this report I define only those files that are used for my purpose.

- **route.txt:** this file contains information about a transit organization's routes. A route is a group of trips that are displayed to riders as a single service.
- **calendar.txt:** this file defines dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
- **calendar_dates.txt:** this file lists exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
- **trips.txt:** this file lists all trips and their routes. A trip is a sequence of two or more stops that occurs at specific time.
- **stop_times_filtered.txt:** this file lists the times that a vehicle arrives at and departs from individual stops for each trip.
- **stops.txt:** this file contains information about individual locations where vehicles pick up or drop off passengers.

Appendix B

Converting Latitude/Longitude to Universal Transverse Mercator (UTM)

The position of the bus stops provided in the GTDF use Latitude and Longitude as unit of measurement. For my project, in order to calculate the transmission opportunities between two buses, I need to evaluate the distance between two bus stops and the speed of the buses. To be able to do that I need to convert the *Latitude* and *Longitude* coordinates into *meters*. The method that I adopted for such conversion is the Universal Transverse Mercantor (UTM). As described in [11] the Universal Transverse Mercantor coordinate system is a grid-based method of specifying locations on the surface of the Earth that is a practical application of a 2-dimensional Cartesian coordinate system. It is used to identify locations on the earth, but differs from the traditional method of latitude and longitude in several respects. The UTM system is not a single map projection. The system instead employs a series of sixty zones, each of which is based on a specifically defined secant transverse Mercator projection. The UTM system divides the surface of Earth between 80°S and 84°N latitude into 60 zones, each 6° of longitude in width and centered over a meridian of longitude. Zones are numbered from 1 to 60. Zone 1 is bounded by longitude 180° to 174° W and is centered on the 177th West meridian. Zone numbering increases in an easterly direction. Each of the 60 longitude zones in the UTM system is based on a transverse Mercator projection, which is capable of mapping a region of large north-south extent with a low amount of distortion. By using narrow zones of 6° (up to 800 km) in width, and reducing the scale factor along the central meridian by only 0.0004 (to 0.9996, a reduction of 1:2500) the amount of distortion is held below 1 part in 1,000 inside each zone. Dis-

tortion of scale increases to 1.0010 at the outer zone boundaries along the equator. In this project I just exploit this conversion methodology hence I am not going into details about the formulas I use, good references to find details on both formulas and description of UTM are [12] and [13].

Hereafter you can find the code that I use to make this conversion.

```
void Tools::transformLonLat2UTM(float iLon, float iLat, float* ox, float* oy)
{
  float a = 6378137; // equatorial radius
  float b = 6356752.31; // polar radius
  float e = sqrt(1 - ((b/a)*(b/a))); // eccentricity
  float e1sq = 0.0067395;
  float k0 = 0.9996; //scale factor

  int aLonZone = 31 + int(iLon/6);
  int aLonZoneCM = 6 * aLonZone -183;
  float aDeltaLon_rad = (iLon - aLonZoneCM) * M_PI / 180;
  float aLat_rad = iLat * M_PI /180;
  float abase = pow(e*sin(aLat_rad), 2);
  abase = pow(e*sin(aLat_rad),2);
  float ar_curv2 = a / pow(1-abase, 1/2);
  float aMeridionalArc = 6367449.15 * aLat_rad - 16038.4296 * sin(2*aLat_rad) +
                         16.8326133 * sin(4*aLat_rad) - 0.0219844 * sin(6*aLat_rad) +
                         0.00031271 * sin(8*aLat_rad);
  float aki = aMeridionalArc * k0;
  float akii = ar_curv2 * sin(aLat_rad)*cos(aLat_rad)/2;
  float aPart1 = (ar_curv2*sin(aLat_rad)*pow(cos(aLat_rad),3))/24;
  float aPart2 = 5 - pow(tan(aLat_rad),2) + 9 * e1sq * pow(cos(aLat_rad),2) +
                 4 * pow(e1sq,2) * pow(cos(aLat_rad),4);
  float akiii = aPart1 * aPart2 * k0;
  float akiv = ar_curv2*cos(aLat_rad)*k0;
  float akv = pow(cos(aLat_rad),3)*(ar_curv2/6)*(1-pow(tan(aLat_rad),2)+
              e1sq*pow(cos(aLat_rad),2)) * k0;
  float aRawNorthing = (aki+akii*aDeltaLon_rad*aDeltaLon_rad+
                        akiii*pow(aDeltaLon_rad,4));
  float aNorthing;
  if (aRawNorthing < 0)
  {
    aNorthing = 10000000 + aRawNorthing;
  }
  else
  {
  aNorthing = aRawNorthing;
  }
  float aEsting = 500000+(akiv*aDeltaLon_rad+akv*pow(aDeltaLon_rad,3));
  *ox = aEsting;
  *oy = aNorthing;
}
```