Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik und Kommunikationsnetze

ulm university universität ulm

Master Thesis

at the department of Information Technology

and Electrical Engineering

Experimental Performance Evaluation of Routing In Delay Tolerant Networks

Abdullah Alhussainy

Supervisors: Prof. Dr. Bernhard Plattner,

Prof. Dr. H.P.Grossmann

Advisors: Theus Hossmann, Dr. Franck Legendre, Dr. Andreas Schmeiser, Bernhard Wiegel

Zürich

03.08.2011

Acknowledgments

Foremost, I would like to express my sincere gratitude to Prof. Bernhard Plattner and Dr. Franck Legendre for leading this thesis and giving me the opportunity of working with the Communication Systems Group at ETHZ. My deep and special thanks to my advisor Theus Hossmann for his continuous support of my thesis work, for his patience, motivation, enthusiasm, and helpful advice. My sincere thanks also to Prof. H.P. Grossmann, Dr. Andreas Schmeiser, and Benhard Wiegel for their support and for making everything easy for me to get this work done.

I would really like to thank the friendly members of the CSG who were very helpful during the work, they did not hesitate to cooperate, and we had great discussions together. My special thanks to my friends Tamas Veres and Paolo Carta for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last 7 months. Thanks to all my friends in Zürich and Ulm especially Khalid Ashmawy, Ahmed M. Saad, Amr NourEldin, Ahmed Saad, Ahmed Khedr, Karim Elsharabasy and Ahmed El-Maghraby for being always there when I needed help.

My deep and sincere thanks to my fiancé Noha Wagih for her love, support and patience. I would also like to express my gratitude to my great family, my parents, my sisters and Alloushy by telling them: Without your support, I could not have done this work ever.

Abstract

Delay Tolerant Networks (DTNs) are the class of networks designed to support communications in a distributed infrastructure-less environment between a source and a destination where no end-to-end connectivity might exist. Such networks mainly rely on the mobility of nodes for routing where data is transferred in a store-carry-forward fashion between a source and a destination using relays. Typical applications are sensor networks, interplanetary networks, communication between rural areas, and user-to-user communication in harsh conditions where traditional protocols can not perform. Whenever two nodes are in proximity (within radio range of one another), they exchange information based on local knowledge to make forwarding decisions. In this context, several routing protocols have been proposed to choose the best relays that would bring data closer to the destination. In particular, *Complex Network Analysis (CNA)* based approaches propose metrics that can be leveraged for such forwarding decisions. To date, DTN researchers have relied heavily on simulations for the evaluation of such protocols, but no real-world deployment were performed in practice. Therefore, we develop the first framework to implement, test and deploy *CNA* based DTN routing on Android phones. In experiments we validate the framework and compare *CNA* routing to other state-of-the-art social routing protocols.

Contents

List of Figures

Chapter 1

Introduction

1.1 Communication Without Infrastructure

Wired and wireless networks have enabled a wide range of devices to be inter-connected over very large distances. For example, it is possible today to connect from a cell phone to millions of powerful servers around the world. As successful as these networks have been, they still cannot reach everywhere. In fact, these networks rely on a set of assumptions that are not true in all environments. One major assumption is the existence of end-to-end connectivity from source to destination. This assumption can be easily violated due to mobility, power saving, or unreliable networks. To this end, *Delay Tolerant Networks (DTN)* are the class of networks designed to support communications in distributed infrastructure-less environments where no end- to-end connectivity might exist.

Since the number of new and cheap wireless networking solutions has increased, opportunities for networking in new situations have been created, and new applications that use the network have emerged. With the existence of techniques such as WiFi ad-hoc, Bluetooth, and other radio solutions , it has become possible to equip almost any device with wireless networking capabilities [2]. Typical DTN applications are :

- Communication between villages or living in locations where no fixed infrastructure is available like in rural villages in India and other poor regions [5].
- Inter-planetary networks [6] which are designed to cope with the noise and delays incurred by communication across astronomical distances.
- Accounting for occlusion of satellites and planned communication windows
- Military and disaster recovery operations, sensor networking and monitoring.
- Communication Censorship:
  During the recent political uprising in Egypt, thousands of border gateway protocol routers were shut down along with mobile services, which resulted in an almost total blackout of communication and Internet access for citizens. *The disruption of the country's mainstream Internet and cellular networks has inspired Egyptians to seek out alternative communication methods [1].* We believe DTN networking is a very promising candidate in similar situations.

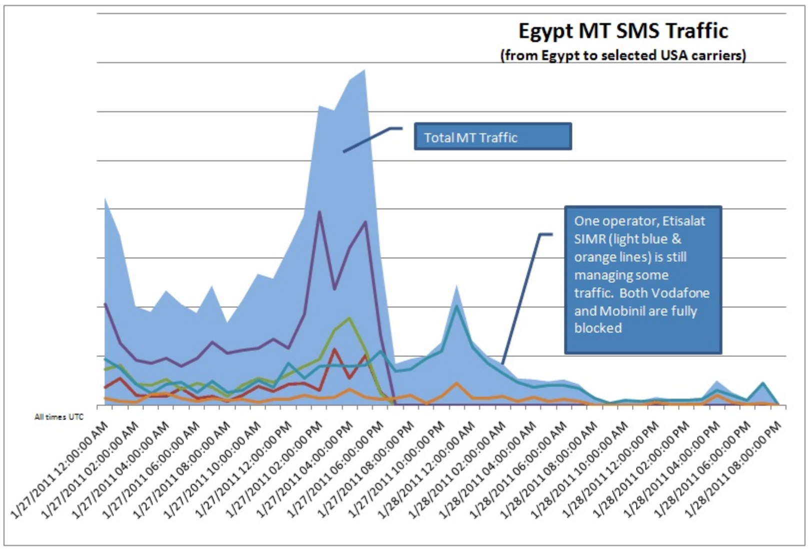


Figure 1.1: Egypt's SMS traffic in January 2011[1]

In figure 2.1 a snapshot of Egypt's SMS traffic clearly shows a dramatic slowdown in the evening of Jan. 27 that continued until at least 8 pm on Jan. 28 (as measured in universal or *Greenwich Mean Time*). Two of Egypt's major operators, *Mobinil* and *Vodafone*, were fully blocked while *Etisalat* seemed to be carrying a small amount of text message traffic [1].

In all the scenarios exemplified above for a possible communication, messages have to be buffered for a long time by intermediate nodes, and due to node mobility messages can be brought closer to their destinations by exchanging data between them as nodes meet, in a store-carry-forward fashion.

1.2 Routing Challenges

An important example of DTNs are the *Pocket Switched Networks (PSN)*, networks constituted by mobile devices carried around by human beings. People will be carrying around more and more feature-rich mobile devices, giving more and more scope for networking applications. Although mobility can help to provide some sort of connected paths between nodes when they come to communication range, still this mobility is uncertain to happen. Therefore, It has also been envisioned that knowledge of human movement patterns and proximity networks can be used to solve different problems that were already solved for connected networks. One of the very important problems is routing a message from source to destination in such environment.

To overcome the uncertainty of future contact opportunities, many protocols are proposed to forward parallel multiple copies of the same content across the whole network. From observations and studies, node mobility and contact opportunities are not entirely random. Instead, human mobility is driven by intentions, social ties (e.g. *friendship*) that guide a node to a certain destination, where the location dictates the path followed [3]. All these features lead to the existence of weak or strong patterns. From this observation, several *utility-based routing* schemes tend to differentiate nodes that are more likely to deliver content or bring it closer to the destination. *PRoPHET* is one of the widely accepted DTN routing protocols in this aspect. It makes routing decisions based on delivery probability. Its calculations are based on a vector that has timestamps of last encounters, that is updated on every node contact. Nodes not only update their contact history, they also exchange it in order to have some transitive information. Basically, the idea behind *PRoPHET* is that in a network people/nodes that met only a short while ago have a higher probability of meeting again sooner than two nodes that haven‘t met for a long time. Following this approach, *PRoPHET* indirectly relies on social properties of human mobility, albeit not relying on any explicit social metrics such as node‘s degree or its closeness to the destination in the network.

1.2.1 CNA Based Routing

Other proposed routing protocols rely on social metrics more explicitly. In particular, *Complex Network Analyisis (CNA)* based protocols. Such protocols try to predict future contact opportunities, by using a complex network representation of who meets whom. In other words, past observed contacts are aggregated to a social contact graph where strong or weak ties between nodes are better reflected. Based on this new contact graph, several recently proposed social routing protocols, *SimBet* [7] and *BubbleRap* [8], make explicit use of social metrics and algorithms in order to assess the utility of a node to act as an intermediate relay/carrier for messages destined to other destinations. The idea of both algorithms is that more "central" or "more connected" nodes in the graph are chosen as carriers to relay content over different communities, until a node that shares many neighbors with the destination is reached [7].

It has been observed that social routing protocols can outperform other DTN routing schemes which are not explicitly "social". However, to the best of our knowledge, all studies in this scope heavily depended only on simulations to evaluate the performance of such protocols, by utilizing contact traces such as **ETH** [9] and **MIT** [10], and relying on synthetic mobility models. Nevertheless, experimental testing allows for more realistic scenarios, uncovers practical software or hardware issues and, in general constitutes an environment where behavior of participating entities is unpredictable, unscheduled and involves a wide spectrum of parameters. To date, there is no real framework deployed for evaluating the performance of DTN social routing protocols.

1.3 Objectives and Contributions

Due to the lack of real testbeds for evaluating the proposed social routing protocols, and from our belief that experimental testing of these protocols is crucial before deployment, the aim of this thesis is to design, deploy and implement a testbed that provides researchers with a reliable and flexible platform for testing and evaluating *CNA* based protocols in *DTNs*.

We choose *Android* as a platform for our implementation due to the development facilities and the fast spread of *Android OS* worldwide. Then, we port the code to *Nexus One* devices which communicate with WiFi Ad-hoc. Neighbor discovery, reliability, and synchronization mechanisms are implemented to end up with a reliable social-aware *Android* testbed that is capable of evaluating the performance of social routing protocols.
The starting point is based on the message flow logic of *PRoPHET* protocol. Whenever two nodes encounter each other, they exchange different control messages (*Hello, Dictionary, BundleOffer, and BundleRequest*) to eventually come to a forwarding decision of the data carried. As our aim is to provide a social-aware platform, the contents and the functionalities of these control messages are re-designed to perform social-based forwarding. We implement two methods for a node to aggregate past observed contacts to a social contact graph, that reflects the social structure of the network. Based on this graph, *SimBet* algorithm is implemented to assess the utility of a node to carry a message based on the centrality and the similarity metrics. They are defined as the number of direct edges connected to a node, and the number of common neighbors with the destination, respectively.

We run simulations to validate the design choices, and to make sure that nodes in contact base their decision on the same network picture after aggregation, which is crucial for correct forwarding decisions. Eventually we run several experiment to validate the testbed by comparing the performance of three routing protocols with respect to different performance measures.
This testbed is expected to benefit those designing and evaluating social routing protocols for use in human contact networks, aiming to maximize delivery ratio.

1.4 Thesis Outline

As the first step is understanding the characteristics of DTNs, chapter 2 is completely dedicated towards describing the evolution of DTNs, the routing problem and prior work in the context of testing and evaluating DTN protocols. Chapter 3 deals with the *CNA*-based routing including two major techniques: Contacts *aggregation*, as well as the *SimBet* routing algorithm. In chapter 4, all implementation details about designing the testbed, messages exchange, aggregation and routing algorithm are explained. The fifth chapter looks at simulations we run to validate design choices. Additionally several experiments results are presented for the validation of the testbed. This is achieved by evaluating the performance of *SimBet* relative to *Epidemic routing* and *direct transmission*. The sixth chapter provides the conclusions from the work done along with possible future work proposals.

Chapter 2

Literature Review

2.1 Mobile Ad-hoc Networks

MANETs are made of many mobile nodes, which moves independently of each other in the network. Each node is equipped with short-range wireless communication capabilities through which communication with all other nodes is possible. A much smaller radio range than the total area covered by the nodes forces them often to act as relays in data dissemination. These situations introduce challenges in establishing efficient routes, with minimal overhead, with the existence of frequent topology change caused by nodes' mobility.

MANETs have been well examined and several routing protocols were proposed for use in these networks. There are *pro-active* protocols such as *OLSR* [11] and *DSDV* [12] which routinely discover and maintain routes to all destinations, in addition to *reactive* protocols such as AODV [13] and DSR [14], where routes are established on demand. These protocols however, are often designed to work under the assumption that the overall network topology remains connected: if a path to a destination is not existing or cannot be found at the time of message forwarding, the message cannot be delivered and is dropped. Considering contacts between humans, the contact graphs at any instant in time are sparse and may remain disconnected for long periods. Therefore the end-to-end connected path assumption of MANET protocols no longer holds. Under these conditions, the contact graphs form Intermittently Connected Networks [15].

2.2 Intermittently Connected Networks

One of the most basic requirements for traditional networking, which also holds for MANETs, is that a fully connected path between entities must exist for communication to be possible. However many scenarios are present where connectivity is intermittent, but where the possibility of communication still is desirable. Delivery with traditional MANET routing would fail completely with the existence of frequent network partitioning and long disconnection periods. In intermittently connected networks to overcome this problem, nodes may store messages during disconnections and opportunistically pass the message closer to destination whenever there is a possibility. This is called Store-Carry-and-Forward routing [16] that shows the role of node mobility to bridge network partitions/communities.

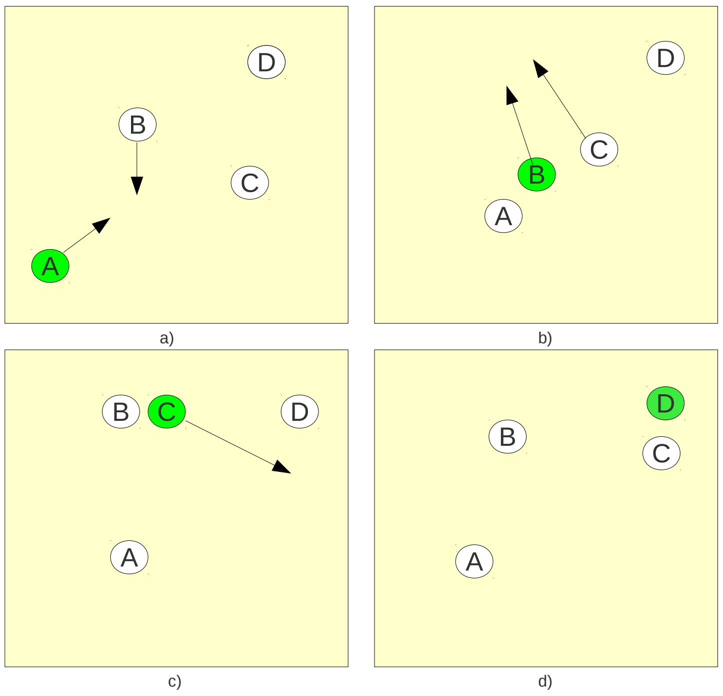


Figure 2.1: Store and Forward : A message is passed from node A to node D through intermediate nodes (Green node is the carrier) [2] .

In figure 2.1, node A has a message (indicated by the node being green) destined to node D, but a path between nodes A and D does not exist. As shown in subfigures a)-d), the mobility of the nodes allow the message to first be transferred to node B, then to node C, and finally node C moves within the proximity of node D and can deliver the message to its final destination.

The challenge in this case is to find the appropriate routing and storage strategies for temporarily unreachable nodes. According to the behavior, the contact patterns that may occur when several mobile nodes come into proximity, frequency and duration of node interactions, messages may experience different delay that reach hours before being delivered to their destinations. Consequently, quality of service in this network is not possible to guarantee and this environment is clearly not suited to the delivery of real-time information such as voice or video. Instead, it can be appropriate to information that is tolerant of a certain delay [15]. Delay Tolerant Networks [17] are intermittently connected networks which carry such type of traffic.

2.3 Routing In Delay Tolerant Networks

A fundamental issue in DTNs is how to effectively and efficiently route information and to achieve the natural objective which is to maximize the delivery ratio. In this section, several major classes of DTN routing protocols are described.

2.3.1 Schedule-based Routing

This class deals with routing for a particular group of delay tolerant networking applications where the future motion and contacts of nodes is well known in advance. If all nodes in the network move along a predefined paths, the scheduled meetings can allow nodes to exchange information. These schedule meeting times can be also exchanged by some signaling methods. An example studied by [18] is the trajectories of buses through San Francisco and communication between a remote village and a city. In the remote village scenario, communication may happen via a scheduled data courier, telephone line (at limited times only during the night), or satellites passing on known trajectories overhead for a short and calculable time. Knowing the times, duration, and bandwidth of these links allows scheduling of messages for delivery between the village and the city. [19] presents *Minimum Estimated Expected Delay (MEED)* protocol as an example of this class. It requires no knowledge of the network structure or node movements in advance and instead uses previous contacts as the basis to construct a graph considering all the scheduled contacts and assigns a weight to every link. The link weight is a function of schedule time, queuing time and propagation time. *MEED* maintains a full network state knowledge at every node in the network and thus propagates each nodes local estimates to all others in the network and this must happen each time the network state changes. With this knowledge of the full network state we can then choose which messages to forward at each contact based on running a shortest path algorithm over the current network state [15].

2.3.2 Probabilistic Routing

This class deals with routing strategies that forward messages probabilistically whenever the nodes are in contact with each other. Here, there are no routing table maintained at any node and next hop is not pre defined but randomly chosen. One of the earliest proposals for routing in disconnected networks is *Epidemic routing* [19]. *Epidemic routing* supports the eventual delivery of messages to arbitrary destinations with minimal assumptions regarding the underlying topology and connectivity of the underlying network. In fact, only recurring pair-wise connectivity is required to ensure eventual message delivery. It works as follows: Each message is disseminated or flooded across the entire network. When a host is carrying a message for some other host, it only transfers a copy of that message to every other hosts it encounters. Each host maintains a buffer consisting of messages that it has originated as well as messages that it is buffering on behalf of other hosts. Hosts usually exchange this buffer information to decide which messages are not yet existing at both of them. Then, eventually, both hosts carry the same messages. This protocol is well-suited to networks where the contact pattern between nodes is unpredictable. But it is very expensive in terms of the number of transmissions and buffer space. In particular, this approach can not easily scale as the number of messages in the network grows.

To overcome the efficiency problems, different optimizations have been proposed to improve the performance of *Epidemic routing*. On one hand, the number of replicas of a given message can be restricted to a certain amount to reduce the resource usage costs. On the other hand, randomized flooding can be used to only transfer copies of messages with some probability $p < 1$ [20].

Another proposed scheme is the *Spray-and-Wait* routing protocol . Spraying in general is the controlled replication of messages, usually in a random manner in a localized area of the network around the source. By controlling the amount of replication we can then reduce the high cost of epidemic flooding. Control is achieved via a static parameter to the algorithms which indicates a trade-off between lower cost (less replication) and higher delivery ratio/lower latency (more replication) [15]. Two approaches deduced from this scheme as described in [21]

- Source Spray-and-Wait
- Binary Spray-and-Wait

In the first approach, there are two phases: Spray phase in which for every message originating at a source node, L message copies are initially spread and forwarded by the source to L distinct "relays". The second phase is the wait phase. In this phase, if the destination is not found in the spraying phase, each of the L nodes carrying a message copy performs direct transmission (i.e. will forward the message only to its destination).

In Binary Spray-and-Wait a finite number of forwarding L tokens are initially created for each message to represent the amount of replication. In the spraying phase, when we come into contact with another node which does not already have a copy of the message, we replicate the message and pass it on together with half of our forwarding tokens L/2. Untill we have reached the point with only a single token. Then, when $L = 1$ a node can transfer the message only to the destination. The number of tokens has been addressed as an interesting point of decision. If L is very small in a very large network, the whole network will not be covered with only a small number of copies. As a result, messages may not reach destinations far away from the source. If the tokens are with large number in a small network, the performance is almost similar to the epidemic routing behavior. Therefore, to overcome this problem, it is essential to have a good estimation of the number of nodes in the network by means of network distributed estimation.

2.3.3 Utility-based Routing

Utility-based routing schemes takes advantages of various contact properties such as time of last encounter between two nodes , frequency of past encounters, and mobility patterns to make better forwarding decisions in comparison to scheduled or epidemic routing strategies. These fetched properties are maintained and analyzed to assess the probability of a given node to get closer to the destination.
Probabilistic Routing Protocol using History of Encounters and Transitivity *PRoPHET* protocol [2] is a widely accepted DTN Utility-based routing scheme. It uses an algorithm that attempts to exploit the non-randomness of real-world encounters by maintaining a set of probabilities for successful delivery to known destinations in the DTN (delivery predictabilities) and replicating messages during opportunistic encounters only if the node that does not have the message appears to have a better chance of delivering it. *PRoPHET* basically operates like epidemic schemes at the part when nodes in contact exchange a summary vector that contains all the messages information carried by those nodes. Additionally, each node maintains a probability metric (delivery

predictability) P $<$ 1 for every known destination. Upon receiving such summary vector this information is used to update the internal delivery predictability vector (probability values for all destinations) as described below, and then the information in the summary vector is used to decide the better carrier. If this node is a better carrier it will request the message from the other node.

$$P = P_{old} + (1 - P_{old}) \times P_{init}$$

where P_{init} is an initialization constant [0,1]

This metric is updated whenever two nodes encounter each other. This gives a meaning to a node having a high delivery probability that this node is often encountered more than other nodes. It may happen that a pair of nodes experience no meetings for a while. Then the delivery probability must age, indicating they are less likely to carry messages for each other.

$$P = P_{pld} \times \gamma^k$$

where γ is aging constant [0, 1] and k is number of time units elapsed since the last encounter

The delivery predictability also has a *transitive* property, that is based on the observation that if node A frequently encounters node B, and node B frequently encounters node C, then node C probably is a good node to forward messages destined for node A [2].

$$P_{(a,c)} = P_{(a,c)old} + 1 - P_{(a,c)old} \times P_{(a,b)old} \times P_{(b,c)} \times \beta$$

where ,β is scaling constant [0, 1] that decides impact of transitivity .

Basically the idea of Prophet is that in a social network people/nodes that met only a short while ago have a higher probability of meeting again sooner than two nodes that haven't met for a long time. Considering this approach Prophet is indirectly a social protocol, albeit not relying on any explicit social metrics.
By neglecting social metrics, Prophet is not exploiting the social information it collects, and therefore not able to make real "smart" decisions. There are some other schemes that assess the strength of "social" ties between nodes. For example [22] uses contact frequency as an indication of the similarity of mobility patterns , and [23] uses the time of the last encounter as a forwarding metric. More recently, *Complex Network Analysis (CNA)* [24] has been proposed as a more powerful tool to solve the problem of future contact prediction in DTNs. This is basically the scope of our work and will be discussed in more details.

2.3.4 CNA-based Routing

CNA (Complex Network Analysis) is both old and new field of research. It is old, as it brought together researchers from many areas including mathematics, physics, biology, computer science, sociology, epidemiology, and others who study network structures since the early years of the 20th century. It is new because it was found recently that many kinds of large networks from the world wide web (WWW) network of web-pages to the network of co-authors of papers to networks of molecules or genes have common behavior. The analysis consists of modeling and analysis of various kinds of networks of entities which are somehow linked. For example biological (gene activating each other), technical (inter-linked web pages, communicating routers), social (friendship or advice

giving people, people writing joint papers), organizational (buying and selling relations), animals (hunting each other), physical (interacting particles), etc [25].
The most notable behaviors of complex networks are the "small world", "clustering" and "scale-free". Small world means that the average number of hops between any two nodes is very small. The term appeared when Milgram conducted his famous experiment "*Six-degree of separation*" in 1976 which suggested that human society is a small world network type characterized by short path lengths.

2.3.4.1 CNA metrics

CNA based routing protocols tend to aggregate past observed contacts to a social contact graph where social ties that drive node mobility is well reflected. This provides a predictive capacity for future contact opportunities, which is beneficial for forwarding decisions. Accordingly, social routing protocols are proposed that make use of *CNA* metrics to asses the utility of a node to act as a relay and bring the message closer to the destination. There are several *CNA* metrics that have been used in forwarding by social routing schemes. Here we will give insight to two metrics Similarity and Centrality.

Similarity :

One major observation in social networks is that people demonstrate periodic reappearances at certain locations, which in turn brings connection among similar instances . Thus, people with similar behavioral principle tie together . Therefore It is useful to exploit similarity in DTN and utilitze this feature for efficient message dissemination. Similarity is a property of social networks that measures the how close two nodes are. For example if nodes A & B and B & C are neighbors, then with high probability nodes A & C are also neighbors. In other words, it is more likely that two nodes are friends if they have one or more friends in common. [26] addressees issues related to mobile user similarity, its definition, analysis and modeling. Similarity of a node u to an encountered node v can be denoted as :

$$sim(u,v) = |N(u) \cap N(v)| \tag{2.1}$$

where N(u) is the set of neighbors of node u.

Centrality :

Centrality is an important notion in network analysis and is used to measure the degree to which network structure contributes to the importance of a node in a network [27]. In terms of social networks it determines the relative importance of a person within a social network. Or how a road is well used in an urban network. This measure is generally dependent on the network structure. The simplest centrality metric, degree centrality, measures the number of edges that connect a node to other nodes in a network. A node having higher degree centrality holds more important position in the network regarding the information exchange. On the other hands, nodes having smaller degree centrality hold fewer neighbors and less involved in information dissemination in the network. Degree centrality of a give node $p_i$is calculated as :

$$C_D(p_i) = \sum_{k=1}^{N} a(p_i, p_k) \tag{2.2}$$

where $a(p_i, p_k) = 1$if a direct link exists between p_i and p_k and $i \neq k$.
Over the years many more complex centrality metrics have been proposed and studied, including α-centrality [28] , and betweenness centrality which measures the extent to which a node lies on the path of information exchange linking other nodes. In other words it measure the extent to which a node can influence, or control how much information flows to other nodes in a network. There are other several variants based on random walk [29], the most famous of which is Google Page Rank [30].
Based on the described routing metrics two major social routing protocols have been proposed by researchers. *SimBet* [7] and *BubbleRap* [8]. *SimBet* protocol derives the routing decision making use of similarity and betweenness whereas *BubbleRap* uses the centrality metric to transfer messages to bridging nodes then apply community detection algorithms to drop the message to its destination. In 3.2, more details are presented.

2.4 Prior Work

Here we briefly describe a few types of DTN applications and highlight prior work done in testing DTN algorithms on various platforms, such as simulations, emulations and testbeds.

The most common approaches to research and evaluation of new DTN algorithms or the performance of DTN routing in diverse networking applications are:

- Simulation and emulation
- Fixed Infrastructure testbeds
- Choreographed mobile tests

There are many studies that rely on simulations to evaluate the performance of DTN routing schemes. We will focus on the one in [3] that provides promising tools to increase the performance of social routing algorithms in DTN networks and on which this work is based. More details are presented in chapter 3.
In [31] Emulab was used to test the DTN2 Reference Implementation under a variety of scenarios. In that paper the DTN2 implementation was compared to *Sendmail* and *FTP* in its ability to transfer data. Emulab provides a flexible platform to conduct repeatable experiments. Fixed infrastructure testbeds such as DieselNet and [32]provide realism by using an actual DTN implementation and real hardware. The DTN2 Reference Implementation, freely available through the DTNRG, provides a common platform for these types of experiments. The last approach is the choreographed mobile tests which also provide real mobile and wireless network conditions and allow for the use of the DTN2 reference implementation and real hardware. A drawback of choreographed mobile field tests is that they cannot be re-run arbitrarily, and even when they are re-run, it is impossible to replicate the exact set of conditions each time. Some wireless testbeds have been considered for DTN experimentation such as the Roomba MADNeT testbed [33].

In [34] Haggle provides a search-based data dissemination framework for mobile opportunistic communication environments, to easily share content directly between intermittently connected mobile devices. MobiClique [35] is a social networking middleware implemented on Windows Mobile smart phones, that form ad-hoc communications for social networking and social graph based opportunistic communications. To the best of our knowledge, there are no testbeds deployed for social routing protocols.

Chapter 3

CNA-Based Routing

In chapter 2 we reviewed the evolution and the importance of DTN applications, along with the routing protocols proposed to solve the challenging problem of efficiently delivering messages to their destinations in disconnected networks. As mentioned in the previous chapter, social-aware forwarding schemes have been proposed as powerful techniques to improve the delivery performance of routing in DTNs. It has been observed that these schemes perform better if they operate over a contact graph that better reflects the underlying social structure which drives nodes mobility. Therefore, in this work as we provide a *testbed* for evaluating DTN social-aware routing protocols, we implement the aggregation techniques proposed by [3]. In this chapter we specify the aggregation details, then two major schemes *SimBet* and *BubbleRap* that make use of CNA metrics for forwarding decisions, are explained.

3.1 Aggregation To a Social Contact Graph

Finding a better carrier that is more likely to deliver the message to destination (future contact) is the key for better routing performance in DTN. One promising way of predicting future contact opportunities is to *aggregate* contacts seen in the past to a social graph and use metrics from complex network analysis (e.g., *centrality and similarity*) to assess the utility of a node to carry the message. This aggregation presents a compromise between the information about past observed contacts lost during this mapping and the ability which complex network analysis obtains for prediction in this context [3]. There are two different ways to perform aggregation, time-based and density-based.

3.1.1 Time Window Based Aggregation

Proposed algorithms such as [7] and [8] likely aggregate contacts using a time window. Two approaches exist :

- **Growing Time Window :** In the original SimBet [7] for example, CNA metrics are calculated over a social graph, where an edge indicates at least one contact between two nodes at any time in the past.

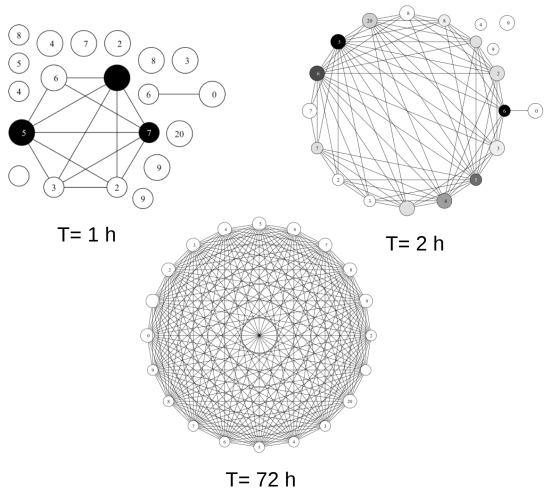


Figure 3.1: Aggregated contacts for the ETH trace at different time instants [3]

- **Sliding Time Window :** A fixed time window of a limited duration is used to aggregate contacts and only contacts happened in this limited time duration are represented as edges in the social graph. E.g. contacts in the last 6 hrs time window.

But time window aggregation is problematic since the graph gets more meshed over time. As a result, heterogeneity of the nodes, with respect to the social network metrics is not reflected after long network lifetime [3]. For a very small time window (all the nodes have similar centrality and similarity values since no enough contacts have occurred, and hence such metrics are not properly defined. Similarly, for large time windows or over a long period of time, all the nodes have encountered almost all other nodes. Consequently, they have again similar values of centrality and similarity. In this case, the graph is very highly dense. To this end, it can be seen that time window based aggregation can lead to forwarding decisions that deteriorate to random, which affects the performance of social-aware routing [3]. Figure 3.1 illustrates this, using a real trace of contacts collected at ETH. See [9] for more details about the contact trace.

3.1.2 Density Based Aggregation

The choice of how many contacts and which ones to include in the graph determines the quality of the aggregated social graph. For the graph to be beneficial in prediction, all edges included must indicate “regular“ contacts (whose past occurrence is predictive for a future occurrence) but only few “random” incidental ones. In this sense, there is an optimal *density* for the graph that results in mostly regular nodes. If we order the contacts from time 0 to n as $C_{0,n}$, then an aggregation mapping f can be defined at time n as :

$$f = C_{0,n} \rightarrow G_n(V, E_n).$$

where, G_n is the output social graph at time n, V is the set of all network nodes/vertices, E_n subset of edges included in the graph at time n among the complete edge set E.

It has been proposed that a more useful and robust approach than time-based aggregation is to choose the aggregation function such that the resulting social graph has a given density. This density d of the aggregated graph G_n can be defined as the fraction of aggregated edges, $|E_n|$, over all possible edges (i.e., all combinations) $|E| = \frac{V.(V-1)}{2}$

$$d(G_n) = \frac{|E_n|}{E}$$

The social graph can for example operate at a certain density, say $d(G_n) = 0.4$, this means the "best" edges, according to some criterion (see below), such that $E_n$will have the desired cardinality [3]. There are two methods of selecting the edges to fill the social graph.

- Most Recent Contacts (MR) : Each edge in the graph is assigned a timestamp according to the occurrence of the last contact. Here, we keep all the recent contacts which happened after a pre-determined timestamp $t_{threshold}$.
- Most Frequent Contacts (MF) : Each occurrence of an edge is counted, i.e., whenever two nodes encounter each other a counter is incremented to keep track of the frequency of contact. Only those edges are included in the final graph G_n with a contact frequency at least equal to $F_{threshold}$.

The density at which a node operates its social graph can be chosen as a fixed value, or can be assessed online (i.e., as new contacts arrive).

3.1.3 Online Aggregation Algorithm

[3] describes an online aggregation algorithm, that observes the aggregated social graph online as new contacts arrive, and tries to select the density at which this graph has a structure that best reflects the social pattens that drive node mobility. It is regarded as a *unsupervised learning* problem to distinguish regular neighbors from random neighbors, by their similarity values. *Each node will see a set of nodes to which it is high similar (i.e., many shared regular neighbors in the group) and another set of nodes to which it is less similar (i.e., random neighbors).* The algorithm works as follows :

When a node encounters another, it logs its similarity value to this node. Out of the contacts observed over time, it creates a histogram of similarity values. It was observed that using a 2-means algorithm to produce two distinguishable clusters, one for similar regular nodes and one for similar random nodes, and relying only on cluster center distance, might draw deceiving conclusions. This is due to the fact that two clear cluster centers at low and high densities are difficult to observe, since in both cases nodes have *similar* similarities close to 0 (no similarity) and 1 (all nodes similar), respectively.

Therefore, one approach is used to assess how distinguishable the two clusters are. It performs *spectral analysis* of a pre-processed similarity matrix by measuring the study of the matrix' *algebraic connectivity* [36].

3.1.3.1 Spectral Analysis :

Supposing that node u has collected a set of n contacts c_i (i = 1....n) during a time period (according to MR or MF). Node u uses these contacts to build its view of the social graph. Each contact c_i observed is assigned a real number s_i to measure the *normalized similarity* between u and the encountered node v :

$$s_i = \frac{|N(u) \cap N(v)}{min\{|N(u)|, |N(v)|\}}$$

$$(s_i \in [0,1])$$

Where $N(x)$ is the set of neighbors of node x in the aggregated social graph.

As a result, each node obtains a vector of n real-valued elements in $[0,1]$ indicating the similarity values collected so far. The values in this vector cluster in high and small values, as the right number of contacts has been observed. To measure the *algebraic connectivity*, the vector s is converted into an $n \times n$ affinity matrix W.

$$W = \{w_{ij}\}$$

$$w_{ij} = exp(-\tfrac{||s_i - s_j||^2}{2\sigma^2}) \ ,$$

if $i \neq j$, and $w_{ij} = 1$, with $\sigma \in [0,1]$ (threshold value).

Then the *Laplacian* of W is calculated as

$$L = I - D^{-\frac{1}{2}} \times W \times D^{-\frac{1}{2}},$$

Where I is the identity matrix and D is the diagonal matrix whose (i,i)-element $d_{ii} = \sum_j w_{ij}$ (i.e., is the degree of vertex i on the matrix W). The spectral clustering then performs eigenvalue decomposition of the Laplacian L to identify k strongly connected components in W with few weak links between them, by projecting the n points into the eigenspace of L consisting of L's first K eigenvectors. Matrix perturbation theory [37] suggests that if the clusters are compact or identifiable, the eigenvalues corresponding to these clusters will still be small. Thus, the algorithm used seeks to locally minimize the second eigenvalue λ_2 of the Laplacian (known as *Algebraic Connectivity*) of the similarity vector s observed over time [3].

To summarize the online aggregation algorithm :

- The idea is to find the density at which the *Algebraic Connectivity* of observed similarity values is minimal
- These similarity values are calculated, using the aggregated social graph at different densities.

- Use the vector of similarities to decide the gradient of the *Algebraic Connectivity*, hence adapt the density of operation accordingly towards the optimum

3.2 SimBet Routing

In this section we present *SimBet* routing [7] as the first routing protocol directly relying on social metrics and the one implemented in this work. It uses a forwarding algorithm based on betweenness centrality and similarity as described in the last chapter (2.3.4.1). The algorithm is not based on assumptions of global knowledge or pre-defined node mobility and forwarding decisions are based solely on local calculations. And this is how it works :

On every contact, a contact history vector is mutually updated with the history vector of the encountered node. Then, *SimBet* nodes calculate two CNA/social-based metrics, similarity and betweenness to formulate the routing decision.
Similarity is individually calculated by considering the number of common nodes between the node and the destinations encountered in the past. This makes a reasonable estimation of how the two nodes are socially close.

$$sim_t(d) = |N_t \cap N_x|$$

with N_x =contact history vector of node x.

Betweenness is calculated the same way as described before as the number of nodes the specified node has already met. See equation 2.2.

For the comparison, both values are normalized as follows.

$$SimUtil_t(d) = \frac{Sim_t(d)}{Sim_t(d) + Sim_e(d)} \tag{3.1}$$

with $Sim_x(y)$ = similarity of node x to node y and t=This node ; e = Encountered node.

$$BetUtil_t(d) = \frac{Bet_t(d)}{Bet_t(d) + Bet_e(d)} \tag{3.2}$$

with Bet_x =Betweenness of node x.

Then the $SimBetUtil_t(d)$ is given by combining the normalized relative weights of the attributes given by:

$$SimBetUtil_t(d) = \alpha \times SimUtil_t(d) + \beta \times BetUtil_t(d) \tag{3.3}$$

where α and β are tuning parameters. Those parameters allow for the adjustment of the relative importance of the two utility values and $\alpha + \beta = 1$. In our implementation, we set $\alpha = 0.5$ and $\beta = 0.5$ as in [3].

Forwarding decision are made based on the comparison of the two SimBetUtil values, which indicates which is a better carrier of each message.

$$if(SimBetUtil_t(d) < SimBetUtil_e(d) \tag{3.4}$$

Then the message is forwarded to node e (the encountered node).

Chapter 4

Implementation

Unlike a simulation tool, a testbed allows for more realistic scenarios, uncovers practical software or hardware problems and, in general constitutes an environment where behavior of participating entities is unpredictable, unscheduled and involves a wide spectrum of parameters. Therefore, the scope of this chapter is to provide a detailed insight in the development and implementation of a Delay Tolerant Networking (DTN) testbed specifically designed to evaluate the performance of social-aware routing protocols. Here we provide detailed information about its architecture and the implementation details.

4.1 Android OS

Building an extensible design and choosing the appropriate platform for the testbed is essential. We choose Android OS as the main platform for our testbed. This is because our aim is to provide a simple application on mobile devices that can be easily carried by individuals during experiments.

Android is an Operating System (OS) created by *Google* to run on any small electronic devices such as cellphones, e-books, Media Internet Devices (MID), netbooks, Internet tablets, and many others devices in the future. Any phone manufacturer can use Android without expensive license fee from *Google*. Because it is Open, manufacturers can modify Android without restriction, allowing it to fit the device they are making - total freedom. This creates a big incentive for any device manufacturers to adopt Android. The ability to run tens of thousands of apps is another big incentive [4].

4.1.1 Development on Android

There are many advantages to developing applications for Google's Android Mobile Operating System. The most prominent of these, is Android's open-source nature. With the right software development tools (see Appendix), Android developers can do whatever they want with the OS. This is not common to other smartphone platforms.
It is expected that Android Market will at some point offer more applications for download and/or purchase than Apple's App Store, as the latter's growth has been slowing down the late, while the Android application store's growth rate has is steadily increasing. Figure 4.1 shows the growth difference between Android Apps and Apple Apps.

Android app developers use the classic open source Linux OS. This means all of its source code is transparent and available to any developer who wants to modify it or see how it works. Anyone who installs Linux on a machine can change any of the files that control the way the operating system works. It provides the possibility to modify these files in order to use system resources for a new application. Thus, the future of the Android platform looks bright for Android application developers who love to have access to everything.

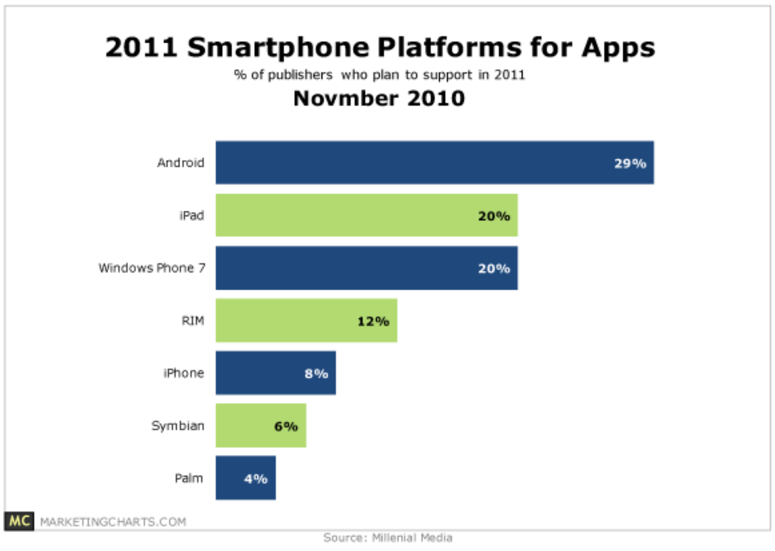


Figure 4.1: 2011 Smarphone Platforms for Apps [4]

Our testbed application is implemented in Qt/C++ [38] on Nexus One phones, and we use Java Native Interface (JNI) [39] to get it running on Android. See appendix for more details.

Figure 4.2: Nexus One Phones used

4.2 Testbed Architecture Overview

In order for our DTN testbed to accurately test and evaluate DTN social routing protocols, different requirements and functionalities are provided:

- Concurrent ad hoc communication between nodes
- Easy configuration of nodes parameters
- Flexibility to incorporate different social routing protocols and forwarding mechanisms

Fig 4.3 illustrates the main DTN testbed components and their interconnections. *Graphical User Interface (GUI)* and *Node Configuration (NC)* compose the administrative part of the testbed. *Listener and Broadcast Sender along* with *Ad hoc shell script* compose the connection part. The most important part is the *Social Routing Agent* where messages exchange, *aggregation* and *social metrics* calculation take place and which make our testbed social-aware. The last component is the *Performance Statistics* that captures the performance measures of the routing protocol being tested. All implementation details of these components are described in the following sections.

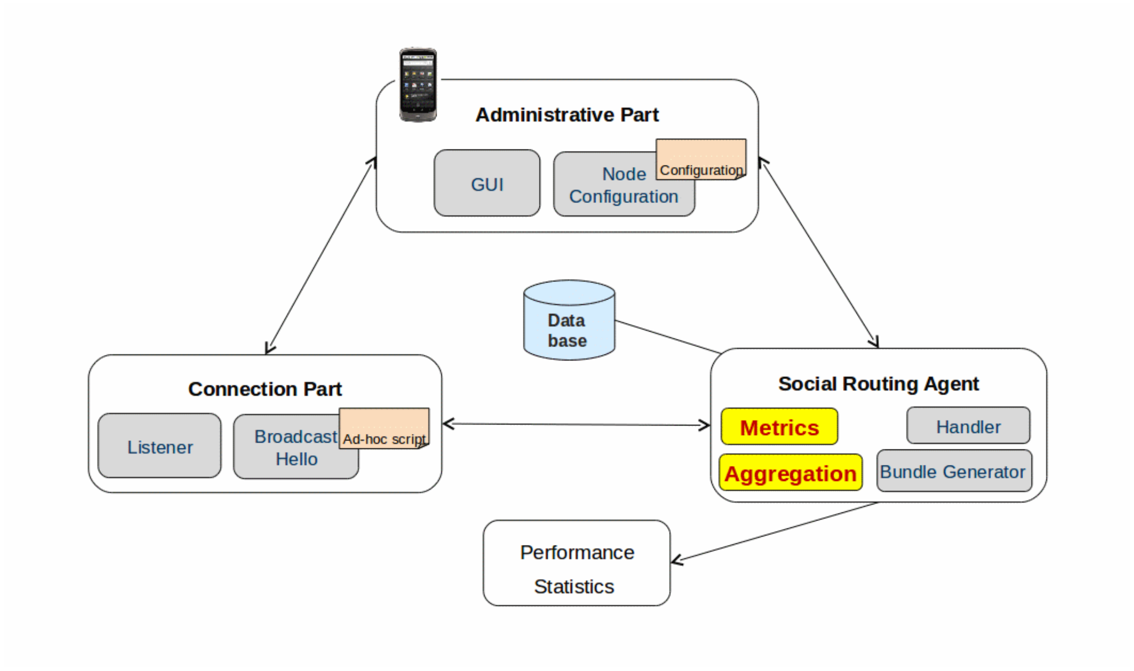


Figure 4.3: Social Routing Testbed Architecture

4.3 Data Structures

Each node is uniquely identified by a 32-bit integer and an IP address. This identity is assigned at the time of node creation, and can be set from the configuration files and the network shell script (see appendix). A node maintains a database where it stores all data of routing and communication relevant information. Figure 4.4 shows some data structures used in the implementation (See Appendix for more details).
A list of neighbors is used, that contains structures of type *nodeinformation*. The structure is

```
struct nodeInformation
{
   int nodeid;
   int contactFlag; // 0 == directly connected; 1 = indirectly connected; 2 = 3 hops away
   QDateTime timeStamp; // time of meeting
   QMap<QDateTime,
   QMap <int,int> Encounters;  // All edges (nodeid is one vertix) with their timestamps
}

Struct Bundle
{
    int msgID;
    int similarity;              // similarity value
    int betweenness;             // centrality value
    QString Data;                // Data of the message
    int sourceID;                // bundle source ID
    int destinationID;           // bundle destination ID
    QString creationtime;        // bundle generation time string
    int routing;                 // to specifiy the routing scheme
    int hopcount;                // to track of the hopcount of a bundle
}
.................................................................
// Identifiers for different message types
#define MSG_HELLO                1
#define MSG_DICTIONARY           2
#define MSG_BUNDLEOFFER          3
#define MSG_BUNDLEOFFERREPLY     4
#define MSG_BUNDLES              5
#define DICT_ACK                 6
#define Bundle_Ack;              7
.................................................................
QList <nodeInformation> neighborlist;
QMap <int,QList <nodeInformation> > m_strongest; // (Graph) list of neighbors after aggregation
QMap <int,QMap <int,int> > SimilarityMap;
QList <int> ActiveNeibors;
```

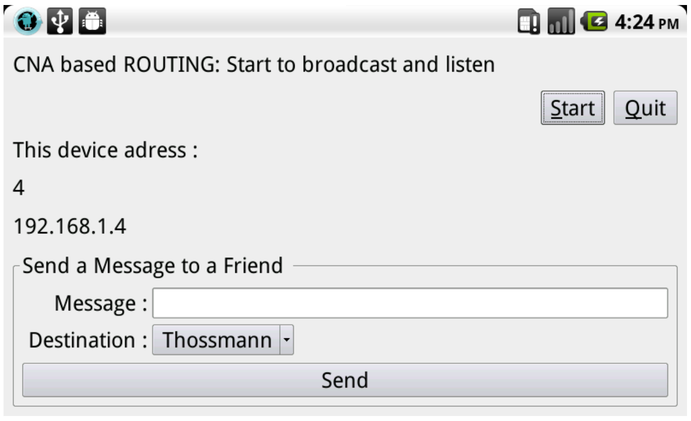
Figure 4.4: Some Data structures obtained by a Node

composed of members of different types: *NodeId, ContactFlag, TimeofContact*, and an *EncountersMap* that has all links this neighbor created in previous encounters. Moreover, each node has a list of bundles (messages) carried, as structures of type *Bundle*. A bundle is defined as single protocol data unit containing the control data and application payload across the nodes in the DTN network (see RFC 5050 [40]). The fields of a *Bundle* structure in our implementation are: *Bundle_Id, Data, Similarity_value, Centrality_ Value, Source_Id, Destination_Id, Creationtime*, Hopcount and some routing flags. And they are created for marking each bundle‘s identity, data, social metrics values, its source and destination nodes, its hop count and the time of its creation. Additionally, there are lists of active nodes that are currently in contact, most recent and most frequent neighbors that are mainly maintained for neighbor discovery and routing decisions. Other lists of timers are also used for reliability mechanisms.

For communication, each node sends/receives different messages as we will describe later in subsection 4.5.3. Messages are assigned unique identifiers, along with different bundle headers that we use, in order to efficiently perform coding and decoding of information exchanged between nodes. A common Header is used for all types of messages to identify the source and destination nodes currently at contact.

4.4 Administrative Part

A Graphical User Interface (GUI) is designed to display all information exchanged between nodes, debug and log messages. Additionally, configuration files are designed for the testbed application to allow flexible configurations for each mobile device. On initialization, each node reads its Ip address, all routing protocol coefficients, and different reliability and timers specifications. The format of the the configuration files is shown in the appendix. Figure 4.5 shows a snapshot of the application GUI.


4:24 PM

CNA based ROUTING: Start to broadcast and listen

This device adress :

4

192.168.1.4

Send a Message to a Friend

Message :

Destination : Thossmann

Figure 4.5: Social-aware DTN testbed Graphical User Interface

4.5 Connection Part

4.5.1 Ad hoc Communication

There are three options to enable ad hoc communication between devices: Bluetooth, Infrastructure mode and WiFi ad-hoc. Bluetooth is not the ideal choice for ad-hoc networking due to various limitations including its slow operation and lack of broadcasting function. Moreover, the limitations in discoverability which at maximum stays for five minutes (in Android), thus devices need peering before they can communicate with each other. Another problem is the short communication range and the limited number of peers that are allowed to communicate in parallel.
The second option is the (Opportunistic WiFi) Infrastructure mode [41], where some nodes act as servers and the clients can connect to them. But this leads to scenarios where nodes are in communication range with each other but they are unable to communicate, since they are connected to different servers.

Due to the mentioned problems , we chose the third option which is WiFi ad hoc. Although it does not have any of the mentioned problems with respect to peering, communication range or the number of parallel communication nodes within proximity, the challenge is that WiFi ad hoc is not enabled in Android stock phones.

Therefore we had to root the devices and install specific network drivers to get it enabled. A shell script is implemented to create the ad hoc network or connect to it automatically when the application starts at each node. Details about rooting Android phones, enabling WiFi ad hoc and the WiFi ad hoc sell script are provided in the Appendix.

4.5.2 Neighbor Discovery Mechanism

A neighbor discovery mechanism is implemented for each node to detect all other nodes within its proximity. This is done by continuously broadcasting beacons (Hello messages) every specific interval. Each Hello message contains identity, address and beacon interval used by the sender node. Upon reception of a Hello message, the following can happen :

- The source node is in the list maintained for the active neighbors. In this case a node updates time and beacon interval.
- Source node is not in the list of active neighbors. Here a node adds the sender node to the list of active neighbors and record the time and beacon interval.

If a Hello message is not received from an active node within a time period of *Num_ accepted_ losses * Beacon_ Interval* (used by source node), a node directly assumes that the source node is no longer active then its entry is removed from the active list. At the time a peer disconnects, the current node records the disconnection time to keep track of encounters and contacts duration. Beacon intervals and the number of accepted losses can be defined from the configuration files.

Based on many trials, it should be considered not to choose a very short interval for the time period after which a node is regarded as disconnected, because if the WiFi link is being lost and restored, many of the recorded encounters are actually re-connections in this case. This exaggerates the number of human contacts and distorts the real mobility behavior. This issue has also been reported by *PRoPHET [2]* designers and developers. More details of the content of *Hello messages* are presented later.

4.5.3 Messages Exchange

Since *PRoPHET* is a widely accepted DTN routing protocol and conforms to DTN bundle protocol specification, it has been used as the base of the way we implement the flow of messages exchange between nodes. Whenever two nodes encounter each other, they exchange a series of control messages before the actual message transfer. These control messages are used to determine which bundles to forward to the encountered node based on the different routing schemes used by the testbed. Figure 4.6 explains the interaction and messages exchanged between two nodes in contact. It consists of three main control phases *Neighbor Discovery Phase, Dictionary Phase and Bundle Offer.* At each of these stages there are different messages that are essential for exchanging information about past encounters and converging to a routing decision that eventually leads to a message transfer. Here we describe with each message.

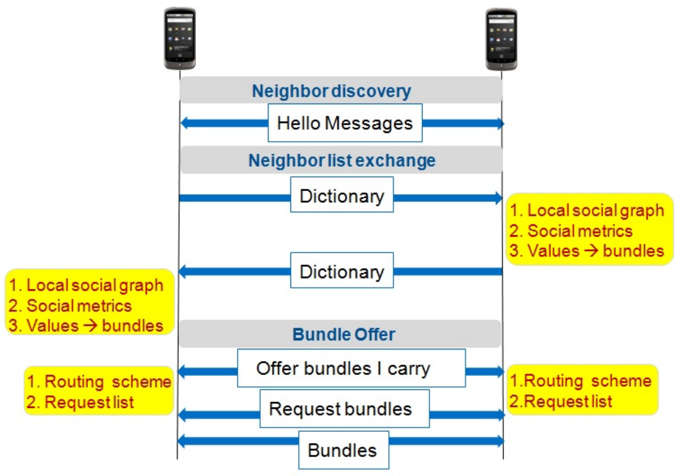


Figure 4.6: Interaction between two nodes in proximity

4.5.3.1 *Hello*

The whole procedure of *Hello* is clarified in figure 4.7. When two nodes encounter each other, the first message they exchange is a *Hello message* indicating their existence, as explained previously in the neighbor discovery. On getting a Hello message, a node decodes the message to extract the sender node Id an IP address. Each node then adds that peer node to its neighbor list and updates the time and frequency of their contact. The peer node is marked as a direct contact. Additionally, each node maintains a list of links as shown previously, where it stores all past observed links with their timestamps.

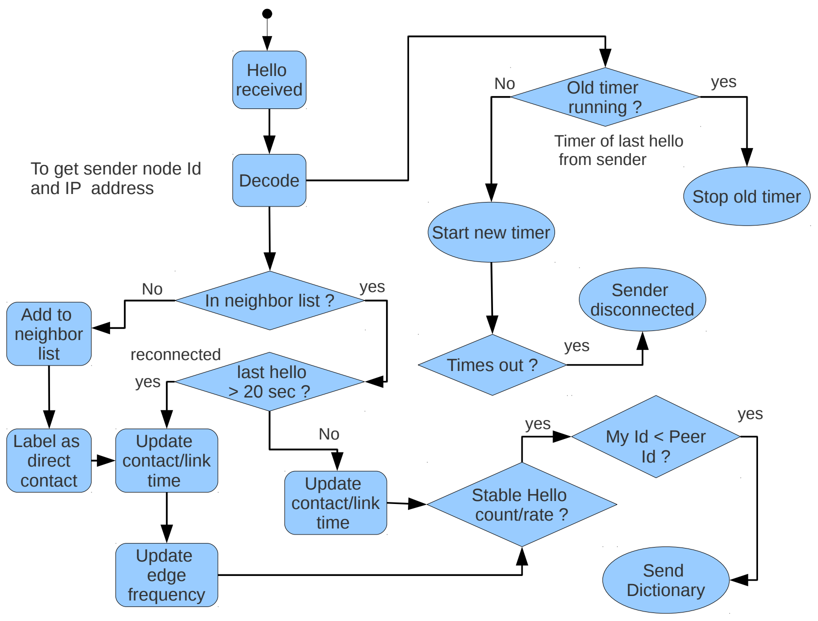


Figure 4.7: Hello flowchart

For example when A receives a *Hello message* from B, A stores A-B in the list of links. During the contact duration, the link time stamp is updated with the periodic Hello messages received, but the frequency is only updated on encounter basis. If the node has already encountered the peer then each node will just update the time and the frequency of contact and link. As long as two nodes make sure of the stability of the contact, in this implementation by receiving 5 hello messages with a certain rate, the node with the smaller Id initiates a *Dictionary* message. This choice is considered for synchronization as we will show later in 4.5.4.

4.5.3.2 *Dictionary*

After a node detects all active peers in its proximity, nodes exchange information about past observed contacts to perform different steps before applying the social routing algorithm. Each node sends a message known as *Dictionary* to its neighbors. A Dictionary message contains information of all past observed encounters which a node has seen. It includes a block of structures that has a list of encountered nodes, time stamps of the links created and their frequency of occurrence. On receiving a Dictionary message a node goes through all the nodes in the message and add them in its own neighbor list but as Indirect contact which means that the node has not met them yet directly. After updating the neighbor list a node performs two basic functions.
The first one is the *Aggregation* to a social contact graph with a certain density (See chapter 3). This means that only the top links are included by this node to form its network picture. Each node maintains a map of all the past observed *links,* then the links are sorted and different lists are maintained based on the applied filters to fill the contact graph (Most Recent) or (Most Frequent). If the density is chosen to be 10 % , then the number of included links from the chosen lists is :

$$Num = 0.1 \times \frac{N \times (N-1)}{2}$$

Where N is the total number of neighbors this node has, and the term multiplied by 0.1 indicates all possible link combinations between nodes in the neighbor list. After this step, a node now has a contact graph of the best (MR or MF) observed links in the past. Then, based on this new graph, a node estimates its importance in the aggregated social graph by calculating the social metrics. If *SimBet* is used as a routing algorithm *Similarity* and *Centrality* metrics are calculated. For calculating the centrality we take the number of direct contacts which a node is having. For similarity, we check how close the node is to all other nodes in the contact graph, by calculating the number of common neighbors. The results are stored in a map of similarity values with the form <node id, similarity value> . All bundles carried by this node have a destination node id. If this id exists in similarity values map, the corresponding similarity value is assigned to the similarity flag of the bundle. To summarize in the following steps are performed upon the reception of a *Dictionary*:

a. Update the neighbor list with the information received from the encountered node
b. Mark them as indirect contacts
c. Perform Aggregation and apply MR of MF filters
d. Calculate social metrics and assign values to the bundles carried

More details about the Dictionary stage is illustrated in 4.5.4.

4.5.3.3 *Bundle Offer*

After both nodes in contact make sure they are done with the Dictionary stage. Each node tells the peer about the bundles it carries along with the calculated similarity and centrality coefficients. We send the information in a different control message known as *BundleOffer* message. Upon receiving information from the peer node about the bundles it carries, a node has to apply the social routing algorithm to decide upon the best carrier for the bundles offered. In case SimBet algorithm is used, utilities are calculated for each offered bundle as described in equations 3.1, 3.2,3.3, and3.4. Then a node checks who is having higher value of *SimBet* coefficient to be the eventual carrier of the bundle. If it is found that a bundle offered by the peer is best carried by this node, the node will append that bundle to a list called *RequestList* which is sent to the peer in the next control message.

4.5.3.4 *Bundles Request*

A node will send the RequestList to the peer with the bundles it thinks it is a better carrier for them. The message is called *Bundles*Request. On receiving the requestList from a neighbor, node will transfer those bundles to the neighbor in the next control message. A node can remove the bundles after transfer, if it is sure that they are successfully received by the peer. A reliability mechanism is implemented for this purpose. Each node waits for a bundle acknowledgement after sending the bundles to the peer. If not received for 3 seconds (configuration files), a node retransmits the bundles. We operate our testbed to have 3 retransmissions, then the peer is considered inactive. Multiple copies of a bundle can stay in the network if the goal is to perform information flooding. This can be configured differently for each bundle form the configuration files.

4.5.3.5 *Bundles*

The final transfer between peers in a contact duration is when they exchange the bundles. Upon reception of a bundle, a node can either delete the bundle or keep a copy to forward to other potential carriers.

4.5.4 *Dictionary* Issues

The *Dictionary* stage is a key part of our work, where different operations are performed that makes the testbed social-aware. As briefly mentioned in subsection 4.5.3.2, nodes at this stage exchange information about their past encounters to perform two (social) functions before converging to a forwarding decision. In order to have a correct decision, and due to the local view maintained by each node about the contact history, not only both nodes should have the same network picture/graph before arguing about the best carrier of any bundle, but they also should not process any further communication step before confirming the reliability of the information exchange in this dictionary phase. When we talk about the same network picture, it means the same number of links/edges with the exact same time stamps at both peers. Scenarios where two nodes are in contact, perform the forwarding decision on a different network picture may lead to situations in which both nodes think they are better carriers for a bundle at the same time, and pass the message back and forth.

In the following, three important issues are considered in this context :

4.5.4.1 Information Included in the Dictionary Message

Two crucial decisions were taken at the very first design steps: *How/which locally collected information is stored at node's memory* about the past encounters, and *what kind of information a node should include in the Dictionary message.*

To build the network view at each node, it is considered that a node should have a clear view of its direct and indirect node connections, besides how other indirect nodes are connected to each other. This is useful in calculating social metrics like *Similarity* that needs more than one hop information about indirect neighbors. Thus; It is flexible to change the number of hops included in the Dictionary message in our implementation.

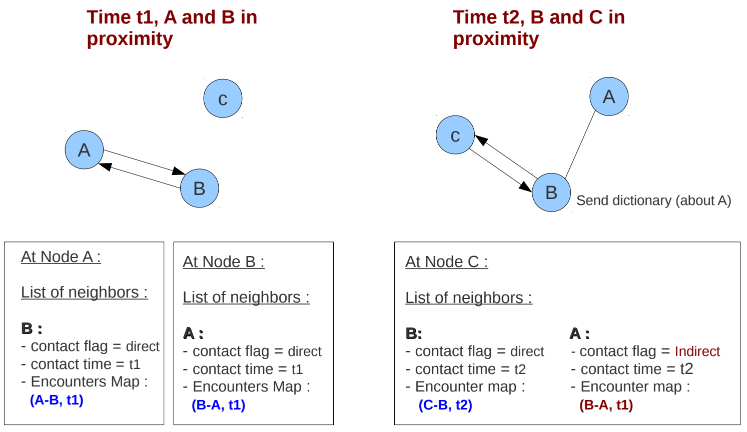


Figure 4.8: Scenario describing the exchange of indirect neighbors information

Therefore, in the Dictionary message sent, not only information about the nodes encountered in the past are included, but also all edges created between these vertices with their time stamps. Figure 4.8 shows how edges are stored and exchanged between nodes by describing a scenario : nodes A and B encounter each other, then each of them stores the edge created (A-B, t1). Later nodes C and B meet, then B sends information about node A in the Dictionary message to C. In case a node sends only direct neighbors in the dictionary to the peer, both of them will have different network picture. See figure 4.9.

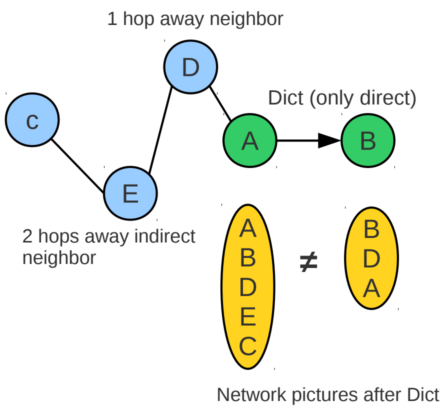


Figure 4.9: Different network pictures at peers when only sending direct links

4.5.4.2 Time Synchronization of Links

Another challenging issue regarding building the same network picture is the time synchronization of links between peers. It is observed that two nodes in contact does not have the same exact time stamp of the current link between them, due to two main observations. The first one is caused by the different times at which both nodes receive *Hello* messages from each other. As previously mentioned the contact link is created at a node‘s neighbor list, upon the reception of a *Hello message* from the peer. The second one is due to the fact that *Hello messages* are being received periodically each *Beacon_Interval* from the peer. This frequently updates the time stamp of the link at both nodes differently. To understand this clearly, we present the following scenario :

For example, Nodes A and B are in communication range, they exchange Hello messages. The information stored about each other is as follows:

Node A :
neighbor list : B (Link A–B, t^{A}_{A-B})

Node B :
neighbor list : A (Link B–A, t^{B}_{A-B}) ;
where t^{z}_{x-y} is the time stamp of link x-y , stored at node z, and $t^{B}_{A-B} = t^{A}_{A-B} \pm \triangle t$; and Δt is time difference between the link time stamps.

Eventually, after nodes exchange the dictionary message, link A–B appears with two different occurrences at each node, although only one link has been created. In turn, this leads to wrong network/graph picture at both peers. Therefore, a mechanism is implemented to synchronize the time stamp of this link A-B in their lists, before starting sorting, aggregation or any routing calculations. This mechanism works as follows :

The node with the smaller Id (node A), is the *Dictionary* message initiator. It sends the neighbor list (B : (Link A–B, t^{A}_{A-B}) to node B, and stores this time stamp t^{A}_{A-B} as the time stamp of the link A–B in a list called "*Synchronized links*". This storing aims at accurately capturing the time sample at which a node synchronizes its link with the peer, in spite of the periodic update/change of this time stamp by every Hello message received. Both nodes will eventually agree on this time as a time stamp for this link.

Node B then receives the *Dictionary message* from node A, and it extracts the time of the link A—B included, which is t^A_{A-B} and assigns it to the link stored in its own neighbor list. Here the *"Synchronized links"* list at node B is also updated with the new synchronized time.

Node B:
neighbor list : A (Link B–A, t^A_{A-B}) , instead of t^B_{A-B} that is probably time shifted by $\triangle t$.

Whereas node B received the *Dictionary message* and synchronized its link with node A, node B can now start sorting, aggregation and calculating the routing metrics. Then it sends a *Dictionary* back to node A. When node A receives the Dictionary message, it invokes the synchronized time stamp stored before for the link A–B and update its neighbor list, to perform the aggregation and metrics calculation.

Node A:
neighbor list : B (Link A–B, t^A_{A-B}) , instead of any time that was assigned to this link due to the periodic reception of *Hello messages*.

Another synchronization problem appears when more than two nodes communicate concurrently. We now investigate the following scenario, when nodes 1, 2 and 3 are in communication range with each other. Since all nodes initially broadcast *Hello messages*, the neighbor list information stored by the nodes (before sending *Dictionary*) is as shown in figure 4.10.

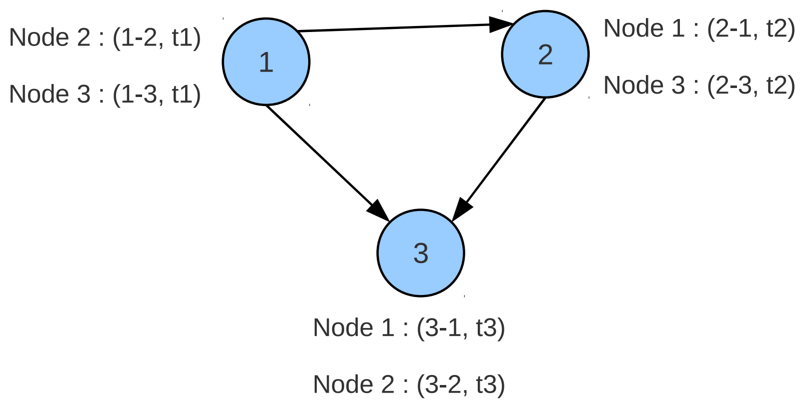


Figure 4.10: Arrows show *Dictionary* initiators

After neighbor discovery, *dictionary* initiators (Smaller Id) send their neighbor lists to the peers as shown in the figure.
Node 1 sends dictionary to node 3 including the following links (1—2 , t1) and (1—3 , t1) . Node 3 applies the mechanism we described above to synchronize the time stamp of link (1—3). In parallel, node 3 receives *dictionary* message from node 2 including link (2—1 , t2) . Consequently, node 3 nows obtains information about the same link with different timestamps (1—2 , t1) and (2—1 , t2) .

From this, it is observed that links (1—2 , t1) from node 1 and (2—1 , t2) from node 2, are sent to node 3, before nodes 1 and 2 do apply the synchronization agreement of the link (1—2). As a result, in this network topology, different occurrences of the same link circulate and lead to wrong network pictures at all nodes. Therefore, in addition to the synchronization mechanism illustrated above, a node sending a *dictionary* to one peer, includes only its *synchronized* links with other active peers in the network.

4.5.4.3 *Dictionary* Reliability Mechanism

Successful data delivery in this challenging environment can be exceptionally difficult or impossible. Transforming end-to-end reliability into hop-by-hop reliable communication is crucial to guarantee the synchronization, hence a correct similar network picture at communicating peers. Therefore, a reliability mechanism (see figure 4.11) is implemented as follows :

A node maintains a list of *reliability timers* for all communicating peers in the *Dictionary* level. Whenever a *dictionary* is sent, a timer is activated and in case the acknowledgement does not arrive during a specific time period (3 secs), the timer expires and triggers another *dictionary* retransmission. The maximum number of allowed retransmissions in our implementation is 3, after which a node is considered inactive and the contact is ignored, unless it reconnects again. It is flexible to modify the retransmissions and the timer period from the configuration files.
Once peers in contact make sure they obtain the same network picture, they start performing aggregation and metrics calculations according to the steps illustrated in 4.5.3.2. Figures 4.12 and 4.13 describe the procedures of sending and receiving a dictionary message considering all issues described.

4.6 Social Routing Agent

The *Social Routing Agent* is the most important part in the testbed architecture and that makes our testbed capable of evaluating social routing protocols. It consists of three main blocks. *Handler*, which is responsible for handling the different incoming messages concurrently. *Aggregation*, that provides different ways of aggregating past observed contacts to a social contact graph. *Metrics*, which decides the social metrics on which the routing algorithm is based.

4.6.1 Handler

Each node communicates with multiple peers concurrently. Therefore, a Handler is implemented to keep track of all incoming messages from the peers and to make sure that the communication flow is maintained correctly with each peer. Furthermore, it decodes every incoming message to

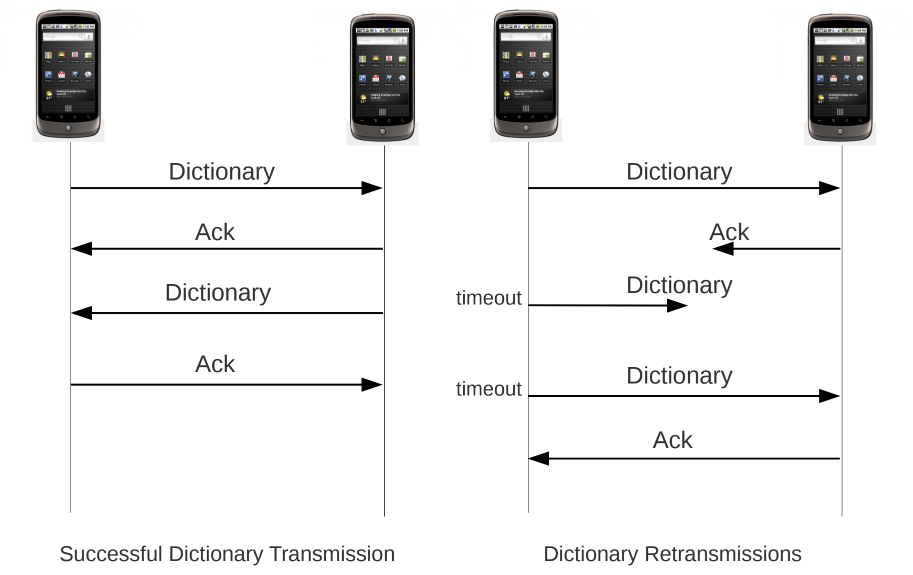


Figure 4.11: Dictionary Reliability Mechanism

extract data, source Id , port number and Ip address. To avoid opening a separate thread for each peer that could stay for a long duration and degrades the testbed performance, the Handler implemented starts a new thread for each message received, performs its functions then the thread is closed. The following flow chart 4.14 shows how the Handler works.

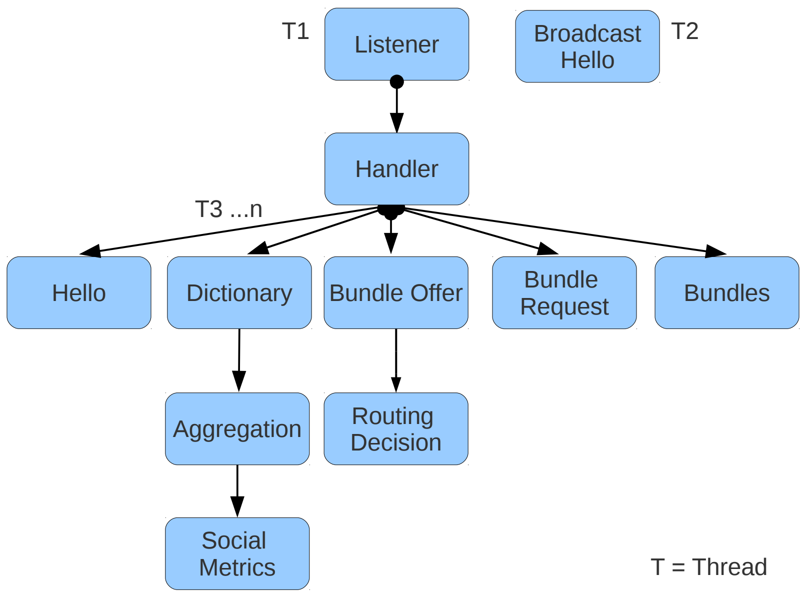


Figure 4.14: Handler starting different threads for incoming messages

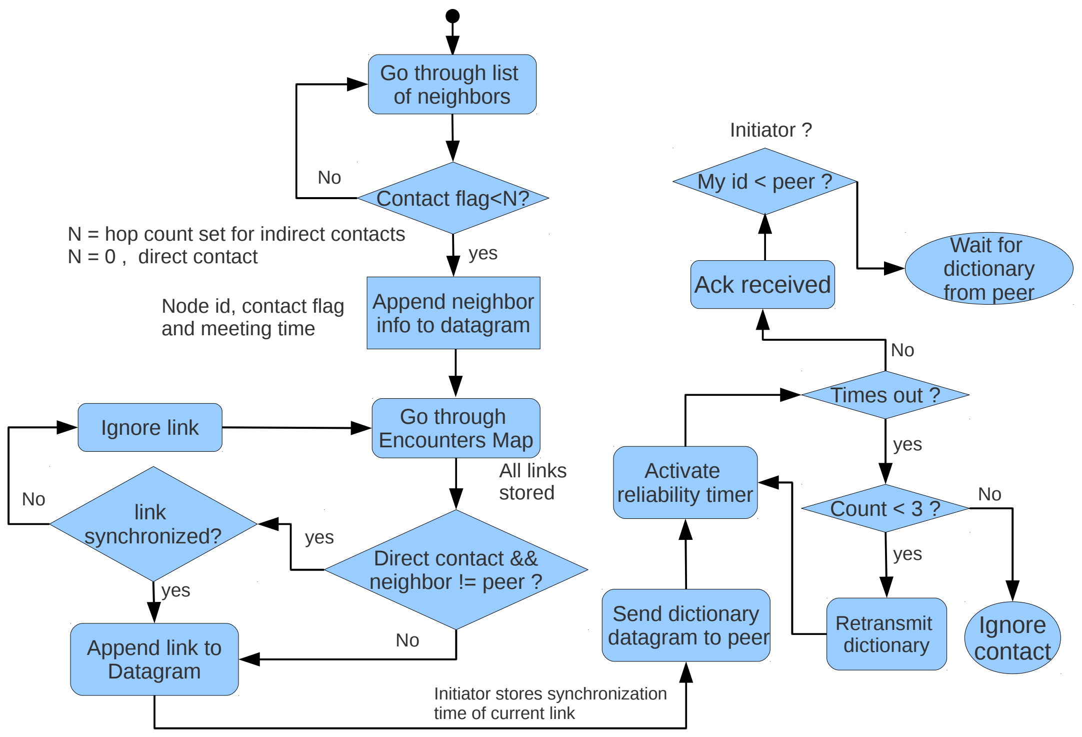

Go through list of neighbors
No
Contact flag<N?
N = hop count set for indirect contacts
N = 0 , direct contact
yes
Node id, contact flag and meeting time
Append neighbor info to datagram
Ignore link
Go through Encounters Map
All links stored
No
link synchronized?
yes
Direct contact && neighbor != peer ?
yes
Append link to Datagram
No
Initiator stores synchronization time of current link
Initiator ?
My id < peer ?
Ack received
Wait for dictionary from peer
No
Times out ?
Activate reliability timer
yes
Count < 3 ?
No
yes
Send dictionary datagram to peer
Retransmit dictionary
Ignore contact

Figure 4.12: Send dictionary flowchart

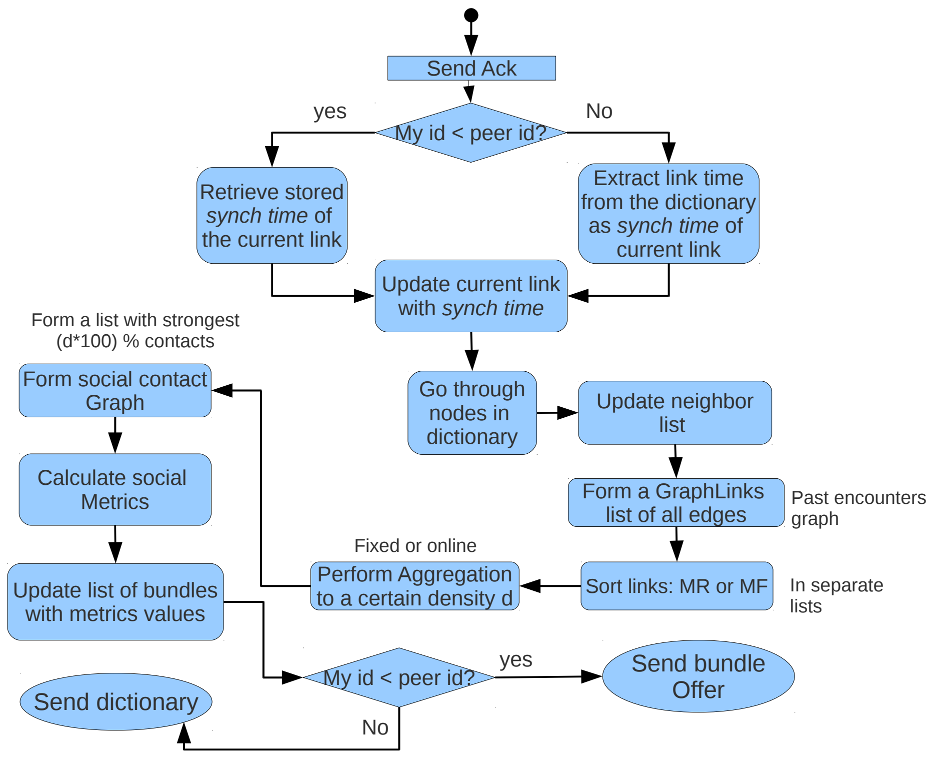

Send Ack
yes
My id < peer id?
No
Retrieve stored synch time of the current link
Extract link time from the dictionary as synch time of current link
Update current link with synch time
Form a list with strongest (d*100) % contacts
Form social contact Graph
Go through nodes in dictionary
Update neighbor list
Calculate social Metrics
Form a GraphLinks list of all edges
Past encounters graph
Fixed or online
Update list of bundles with metrics values
Perform Aggregation to a certain density d
Sort links: MR or MF
In separate lists
My id < peer id?
yes
Send bundle Offer
Send dictionary
No

Figure 4.13: Receive a dictionary message flowchart

4.6.2 Aggregation

Two ways are implemented in this work to aggregate past observed edges to a social contact graph, Fixed and Adaptive.

- Fixed : a node can operate the social graph at a certain density, say 0.3 . The node chooses the "best" edges according to a certain criterion, such that the aggregated edges have the desired cardinality. Next, two methods of picking the edges to fill the graph can be selected.

- Most Recent Contacts (MR): Each edge in the graph is assigned a timestamp according to the occurrence of the last contact. Here, we keep all the recent contacts which happened after a pre-determined timestamp $t_{threshold}$.
- Most Frequent Contacts (MF): Each occurrence of an edge is counted, i.e., whenever two nodes encounter each other a counter is incremented to keep track of the frequency of contact. Only those edges are included in the final graph G_n with a contact frequency at least equal to $F_{threshold}$.

- Adaptive : A node can operate the social graph at an optimum density where the underlying (social) structure guiding mobility best correlates with the social structure that can be observed on the aggregated contact graph. This is done by implementing an *Online Aggregation Algorithm.* Whenever a node encounters other nodes, it keeps a histogram of its similarity values observed. Then by means of *Spectral Analysis* and robust clustering, a node chooses the optimum density at which the separation between regular (high similarity) and random (low similarity) clusters are maximized. See section 3.1.3 for the detailed explanation.
Testers can choose the desired way of operation from the configuration files. See Appendix for more details.

4.6.3 CNA Metrics

Several social routing protocols are proposed with different social metrics that is mainly used to asses the utility of a node to carry messages. This block is responsible for the implementation of social metrics calculated. For example similarity and centrality used by *SimBet* algorithm that is implemented in this work, in addition to the flexibility to corporate different routing algorithms based on different metrics like *BubbleRap* [8], which uses *a Community Detection algorithm* rather than similarity to forward a message to a destination. Moreover, the *contact duration* observed between nodes could be a measure on which is the routing decision is based. Testers can define the desired metrics/social scheme from the configuration files.

The last part of the testbed architecture, *Performance Statistics* is described later in chapter 5.

Chapter 5

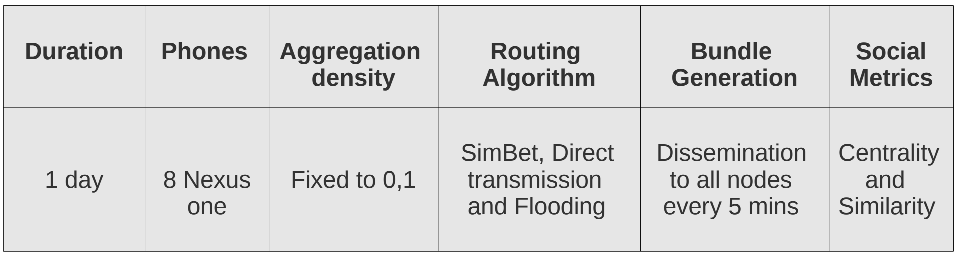
Validation and Experiment

The aim of this work was to design, implement and deploy a *DTN testbed* as a platform, for evaluating the performance of social-aware routing protocols. For this purpose, different steps have been taken to validate the implementation of the *message exchange flow* between nodes, *SimBet* routing protocol and *density-based aggregation* to the social contact graph. Initially we ran simulations to make sure that the implementation is efficient before porting the code on the devices. Then, different concurrency, reliability and synchronization scenarios have been considered to end up with an Android testbed, capable of evaluating social-aware routing protocols. To this end, we run many experiments as a validation and several evaluation results are stated.

5.1 Simulation

In this section we describe the simulations used to validate *SimBet* algorithm based on locally collected information at each node and without any global view of the network. For this, a real mobility trace (**ETH** trace) has been utilized. **ETH** trace [9] contacts were collected by 20 students and staff working on the same floor of a building of ETH Zurich. They were carrying 802.11 enabled devices for 5 days. At an interval of 0.5s, each device sent out a beacon message, the reception of which was logged by all devices in 802.11 radio proximity. This trace contains more than 23,000 reported contacts and is unique in terms of time granularity and reliability. On average there were more than 1000 reported contacts per device.

Different nodes are simulated on one computer. Each node initially creates a message for each other node, in this case 19 messages per node. Contacts are handled sequentially from the trace file, and nodes in contact exchange all the control messages (*Dictionary, BundleOffer* and other types) to decide upon the routing decision (*SimBet*) as discussed previously. By these simulations, design choices of the way each node maintains/sends information were confirmed. Moreover, aggregation of past observed contacts to a social contact graph is performed according to a fixed aggregation density at all nodes. We made sure that nodes in contact base their decision on the same network view correctly according to SimBet routing.



Duration	Phones	Aggregation density	Routing Algorithm	Bundle Generation	Social Metrics
1 day	8 Nexus one	Fixed to 0,1	SimBet, Direct transmission and Flooding	Dissemination to all nodes every 5 mins	Centrality and Similarity

Figure 5.2: Experiment Setup

5.2 Experiment

In order to validate the testbed, we run several experiments in a typical office environment. The objective of our experiments is to validate our design choices and to evaluate the ability of our testbed to disseminate data between nodes according to different social routing strategies.

8 *Nexus One* Android phones (see figure 5.1) are carried by research colleagues during a day of normal office activities. These include meetings, office work, and lunch outside the offices. A phone carries a bundle destined to all other phones and new bundles are generated every 5 mins. Each bundle is assigned a routing flag indicating which routing strategy used to forward such bundle. This provides the ability to evaluate different routing protocols under the same conditions. Every time a phone encounters another phone, they both exchange control messages, perform aggregation of past contacts, and finally apply the social routing strategy specified (*simBet*) to decide upon the best carrier a bundle. Table 5.2 summarizes the experiment setup.

Figure 5.1: 8 Nexus One Phones used in the experiment

5.3 Performance Evaluation

Our testbed enables each node to log detailed information with different measures. This function is mainly performed by the *Performance Statistics* block of the testbed architecture described in 4.2.

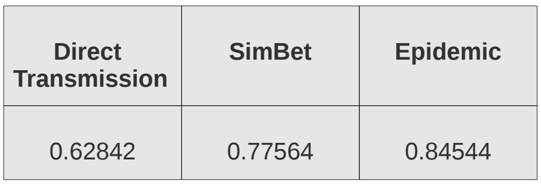
- Contact Information : The real time and the contact duration of encounters are reported. A contact duration is defined as the time period between connection and disconnection times. This is useful in case the contact duration is used as a metric for forwarding decisions. Moreover , it is an appropriate way to collect real contact data for mobility traces.

- Bundles Generated : The generation time of all bundles along with their destinations and the routing flags assigned.
- Bundles Carried : According to the social routing strategy, nodes carry bundles for other nodes as they are more likely to meet the destinations. Information about bundles carried (source, destination and via which node) are available.
- Bundles Received : When a node receives a bundle, information about its reception time, generation time, end-to-end delay, routing used, number of hops to arrive is reported.
- Forwarding Calculations : At each contact, a node calculates *SimBet* coefficients, as illustrated in equation 3.4. These details are stored at each node's database, for routing analysis.
- Contact Graph View after Aggregation: To keep track of the contact graph maintained by aggregation at each contact.

As a performance measure, we compare *SimBet* as social routing protocol, relative to two other protocols. *Direct Transmission*, where the source keeps the message until it meets the destination, and *Epidemic routing* where a copy of each bundle is given to every encountered node (*Flooding*). The measures are based on the following criteria.

5.3.1 The Delivery Ratio

The delivery ratio R_j to a user j is the fraction of messages that were generated by all users and are delivered to user j over the total number of messages generated by all users [34]. The following table 5.3 summarizes the results we obtained for the three routing protocols.



Direct Transmission	SimBet	Epidemic
0.62842	0.77564	0.84544

Figure 5.3: Experiment Results: Delivery ratio comparison

5.3.2 Total Number of Messages Delivered

The ultimate goal of any social-aware routing protocol is to achieve delivery performance as close to Epidemic routing as possible [7]. This is because *Epidemic* routing always finds the best possible

path to the destination by flooding a message across the whole network and therefore it represents the upper bound for the possible delivery performance.
The following chart 5.4 shows the total number of messages delivered by the three routing protocols in 8.3 hours. Three observations can be noticed. First, *Direct Transmission* serves as a lower bound on the total number of messages delivered, as nodes keep the messages until they meet the destinations with no effort to find better carriers. Second, the behavior of *SimBet* and *Direct Transmission* is quite similar at the beginning (first 1.6 hours), since no enough contacts have occurred, and hence the *SimBet* similarity and centrality metrics are not yet properly defined. Third, as one would expect the *Epidemic* routing represents the baseline for the best possible delivery performance if buffers are not limited.

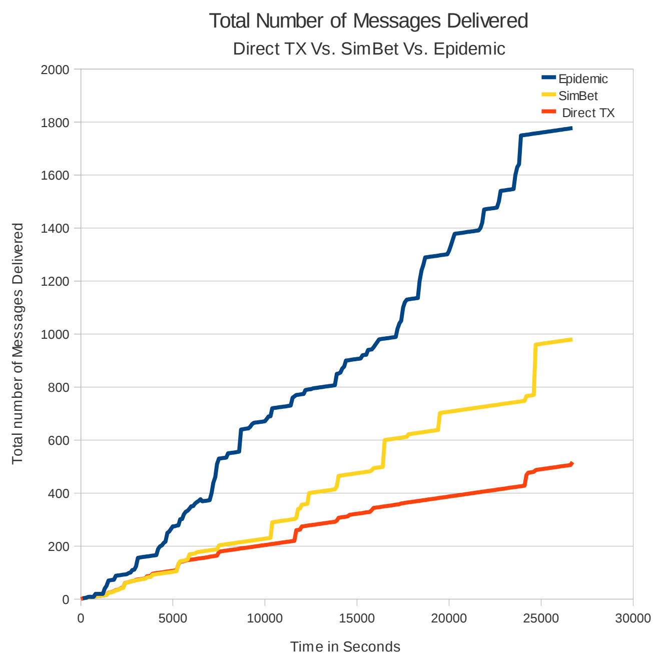


Figure 5.4: Experiment Results: Total number of messages delivered

5.3.3 Average End-to-End Delay

End-to-End delay is an important concern in performance evaluation. The following figures show a comparison of the End-to-End delay for the three different protocols. It is observed that Epidemic routing achieves a good baseline for the minimum end-to-end delay, in comparison to *SimBet* and *direct Transmission,* having in mind that *SimBet* assumes one single copy of each bundle.

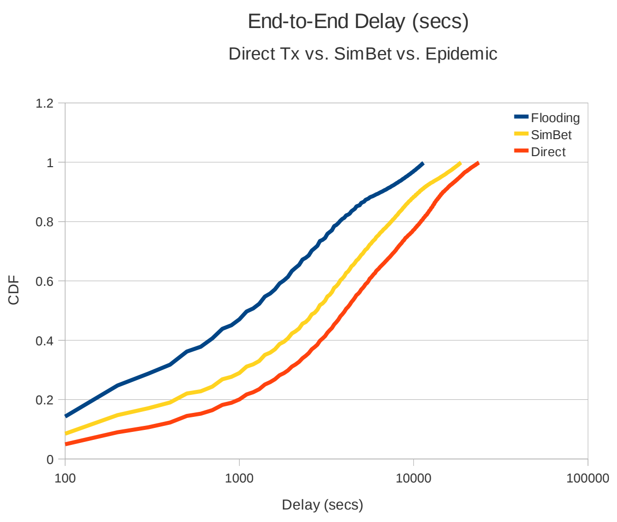


Figure 5.5: Experiment Results: End-To-End Delay (Log Scale)

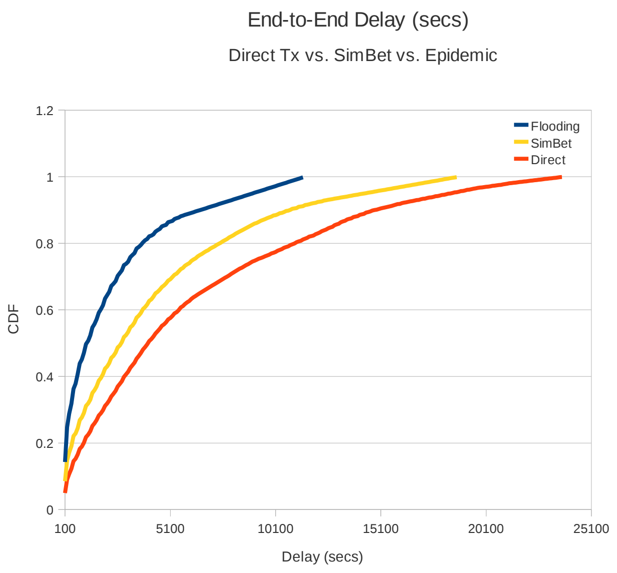


Figure 5.6: Experiment Results: End-To-End Delay (Linear Scale)

5.3.4 Average Number of Retransmissions

SimBet assumes the existence of one copy of the message on the network, and the message is deleted from the carrier list of bundles if only an acknowledgement is received from the destination confirming a successful reception. It has been observed during many experiment trials, that bundles are not successfully transmitted between peers in contact from the first transmission. Instead, frequent retransmissions are performed to guarantee a successful bundle delivery. We suspect that this is due to short contact durations and expected indoor network conditions in such office environment. The observed average number of retransmissions to guarantee a successful delivery is 1.87 retransmissions. From this, we propose to increase the maximum number of retransmissions allowed per node.

5.3.5 Average number of Hops per Message

It is desirable to minimize the number of hops a message takes in order to reach the destination. The average number of hops per message achieved by *SimBet* is 3.4, that is very close to *Epidemic* routing which is 2.8.

5.3.6 Total Number of Forwards

It is useful to have a measure of the overhead in the network, in terms of how many times a message forward occurs. It is observed that Epidemic routing is costly in this sense compared to SimBet. This is because *SimBet* assumes only a single copy of a message in the network. This measure is beneficial in case two routing protocols achieve the same delivery performance, then the one with less overhead is preferred for deployment.

5.4 Power Consumption

A limiting factor in our experiment is the battery life time of the mobile devices. The first thing we hence examine is the impact of our testbed application on a mobile phone‘s power consumption. As a baseline, we run a phone isolated with only neighbor discovery on Wifi ad hoc. This gives us an estimate of the impact of the neighbor discovery mechanism implemented. A listener that captures events of battery change is implemented. It returns integer values for the battery level and is stored in a file with the corresponding accumulated time from the beginning of the application. From different trials we observe that the beacon interval chosen for neighbor discovery, bears different weights on the battery life. As figure 5.7 shows, the battery drains faster if the beacon interval is set to 0.5 seconds, relative to 1 seconds. The battery of an active node during experiments that sends and receives data, lasted up to 6 hours before recharging.

For a longer battery life, a mechanism is proposed to operate the testbed at different neighbor discovery modes based on the network density around. A node will sleep or increase the beacon interval if it predicts no active neighbors around in a specific time window.

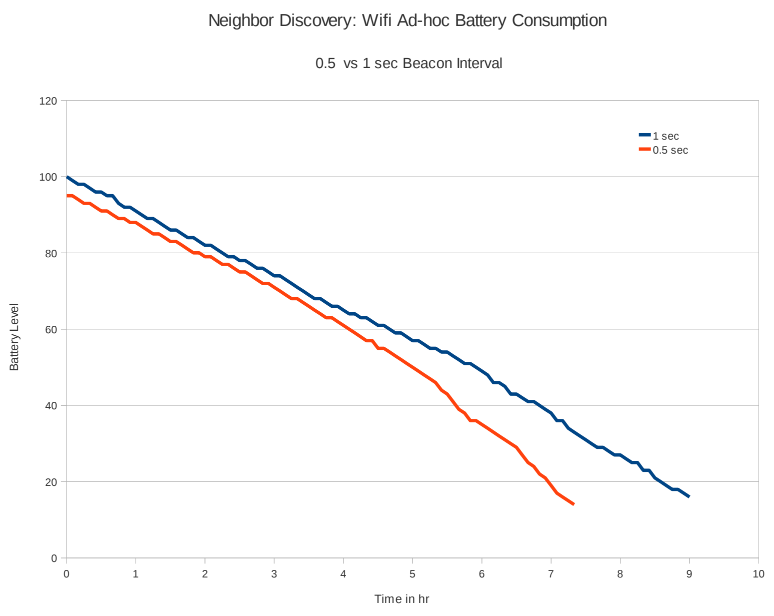


Figure 5.7: Experiment Results: Power consumption due to *neighbor discovery*

Chapter 6

Conclusion And Future Work

Delay Tolerant networks are the class of networks where no end-to-end connectivity exists. Due to disconnections and many challenges introduced by this environment, routing data from source to destination has been a challenging research area. Several social routing protocols are proposed to improve the message delivery performance.

In this work, we have designed, deployed and implemented a *testbed* on *Android* phones as a platform for evaluating social routing protocols in DTNs. Whenever two devices encounter each other, they communicate with WiFi ad hoc. A shell script that runs automatically on each device is implemented for this purpose. Devices in proximity exchange different control messages, with several functionalities, to eventually forward the messages to the node which is more likely to meet the destination. Forwarding decisions implemented mainly rely on two major techniques. First, aggregation of past observed contacts to a *social contact graph*, where edges indicate strong ties between nodes, thus a prediction for future contact opportunities. In this context, we have implemented two methods to perform aggregation to a certain density, fixed and adaptive. Second, *SimBet* is implemented, which is a social routing protocol that operates over the social contact graph to assess the utility of a node to carry a message, based on *CNA (Complex Network Analysis)* social metrics centrality and similarity. The testbed structure is flexible, so that researchers can test their proposed DTN social routing protocols in comparison to different social metrics.

Finally, the testbed has been validated by running several experiments to compare the performance of a social routing protocol (*SimBet*), relative to epidemic routing and direct transmission. 8 Nexus one phones have been carried by researchers during a day of normal office activities. Communication and routing decisions between devices in different scenarios have been observed and validated by performance measures, showing that the testbed is a suitable platform for DTN social routing protocols tests.

In future work, we propose the following :

- Longer experiments to validate the *online aggregation algorithm* impact on the performance of social routing protocols, as it requires enough number of contacts, in addition to computation issues that have not been well experimented.
- As the social routing implementation works properly and tested on *Android* phones, integration with existing *opportunistic twitter* applications is possible.
- With modifications, it can act as a standalone application to forward unicast messages to destinations over infrastructure-less and challenging environments (e.g. disasters).
- Enhancements to the testbed Graphical User Interface (GUI) to act as real time monitor during experiments, and to provide users with more flexible configuration options.

Appendix A

Code

A.1 Class Diagram

The application is written in Qt/C++ on Android *Nexus One* phones, consists of about 5500 lines of code. The following figure shows the class diagram of the implementation.

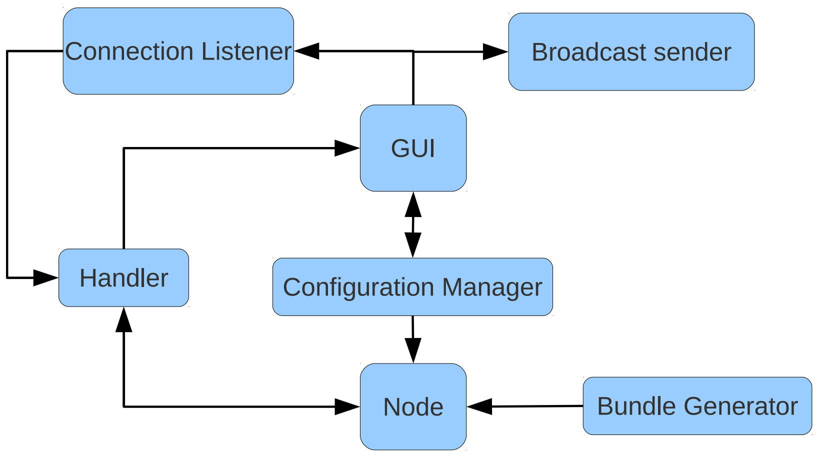


Figure A.1: Testbed Implementation Class Diagram

- Node Class : In this class, a node instance is defined with datastructures used for maintaining the neighbors list, bundles, similarity values, and timers.
- GUI Class : The graphical user interface that starts the application is defined here. The GUI has an interface with the configuration files and starts the both the listener, and the broadcast sender.
- Connection Listener : A listener is implemented here, to read any incoming datagram from the socket. The application uses a *UDP* socket that is binded on port 3001 on the devices. Ip

addresses, node Id, and data are extracted from these datagrams, then passed to the handler.

- Connection Broadcast : Datagrams are written on a UDP socket, then broad-casted to the address *X.Y.Z.*255 (broadcast address). The letters are the host address part, that is determined according to the Ip addresses assigned to the devices.
- Handler Class : This class handles all messages, executes all control functionalities, and perform the routing procedure.
- Configuration Manager : This deals with reading configuration files that initializes each node, and sets routing, reliability and timer parameters.
- Bundle Generator : During experiments, bundles are generated periodically. This class creates different variations of bundles, with respect to size, destinations and routing.

The following figure A.2 shows a snapshot of the configuration files read at initialization.

```
NODEID=4
BROADCASTAddress=192.168.1.255
Beacon_Interval=500
Reliability_Timer=3000
Retransmission_count = 3
Bundle_Timer= 3000
BETA=0.50                          % for SimBet
GAMMA=0.50                         % for SimBet
Aggregation_value = 0.1
Aggregation = 1                    % 1 = Fixed ; 2 = Online
DictionaryTimer=5000
Disconnection_Period=6000
STORAGEPATH=/data/data/SDTN/
TTLVALUE=_
CNA_Metric= S,b
Routing_Algorithm = SimBet
```

Figure A.2: Configuration File

A.2 Wifi Ad-hoc Shell Script

We implement a shell script to enable nodes to create/detect WiFi ad-hoc network. The script content is illustrated in the figure below A.3. This script runs automatically from the application main function by executing the following command:

system("su -c /system/bin/sh -c /data/data/wifi.sh"); where */data/data* is the path of the file on the phone.

```
#!/system/bin/sh
echo $(whoami)
cd /system/lib/modules/
insmod bcm4329.ko
#output=$(insmod /system/lib/modules/bcm4329.ko)
iwconfig eth0 mode ad-hoc essid dTn
ifconfig eth0 192.168.1.1 up
```

Figure A.3: WiFi ad-hoc shell script

Important steps are required to successfully run this script:

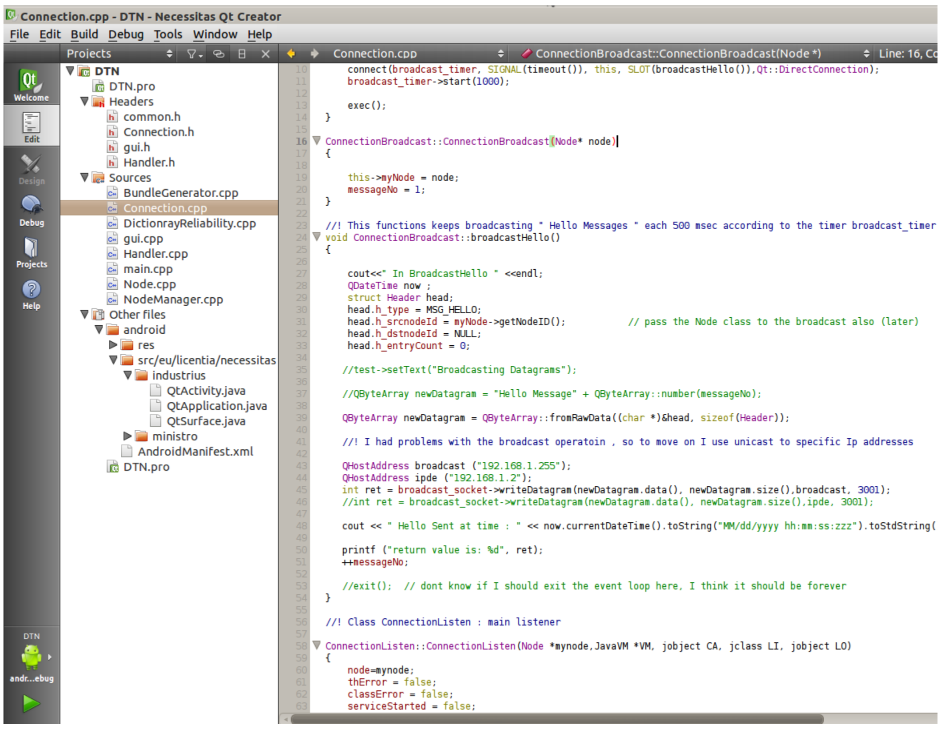
- Root phones (See Appendix B)
- Switch off the WiFi interface manually from the phone settings
- Give the file root, read, and write permissions:
  chmod u+rx wifi.sh
  chmod 777 wifi.sh

Appendix B

Android Issues

B.1 Necessitas Tool and Java Native Interface

Android main development environment is Java. Since our implementation is in Qt/C++. We use *Necessitas* tool [42] that is based on Java Native Interface (JNI) [39] to run C++ code on top of Java. *Necessitas* is an IDE letting developers manage, develop, deploy, run and debug Qt Applications on Android Devices. It is based on *Java Native Interface (JNI)* which enables aquiring a pointer of the *Dalvick virual machine* of Android, and use to it call Java APIs from the C++ code. Figure B.1 shows a snapshot of Necessitas.



```
    connect(broadcast_timer, SIGNAL(timeout()), this, SLOT(broadcastHello()),Qt::DirectConnection);
    broadcast_timer->start(1000);

    exec();
}

ConnectionBroadcast::ConnectionBroadcast(Node* node)
{

    this->myNode = node;
    messageNo = 1;
}

//! This functions keeps broadcasting " Hello Messages " each 500 msec according to the timer broadcast_timer
void ConnectionBroadcast::broadcastHello()
{

    cout<<" In BroadcastHello " <<endl;
    QDateTime now ;
    struct Header head;
    head.h_type = MSG_HELLO;
    head.h_srcnodeId = myNode->getNodeID();           // pass the Node class to the broadcast also (later)
    head.h_dstnodeId = NULL;
    head.h_entryCount = 0;

   //test->setText("Broadcasting Datagrams");

   //QByteArray newDatagram = "Hello Message" + QByteArray::number(messageNo);

   QByteArray newDatagram = QByteArray::fromRawData((char *)&head, sizeof(Header));

   //! I had problems with the broadcast operatoin , so to move on I use unicast to specific Ip addresses

   QHostAddress broadcast ("192.168.1.255");
   QHostAddress ipde ("192.168.1.2");
   int ret = broadcast_socket->writeDatagram(newDatagram.data(), newDatagram.size(),broadcast, 3001);
   //int ret = broadcast_socket->writeDatagram(newDatagram.data(), newDatagram.size(),ipde, 3001);

   cout << " Hello Sent at time : " << now.currentDateTime().toString("MM/dd/yyyy hh:mm:ss:zzz").toStdString(

   printf ("return value is: %d", ret);
   ++messageNo;

   //exit();  // dont know if I should exit the event loop here, I think it should be forever
}

//! Class ConnectionListen : main listener

ConnectionListen::ConnectionListen(Node *mynode,JavaVM *VM, jobject CA, jclass LI, jobject LO)
{
    node=mynode;
    thError = false;
    classError = false;
    serviceStarted = false;
```

Figure B.1: Necessitas IDE

The following steps are presented to get the Android device detected by the computer to be able to install Qt C++ applications on it :

- Install *SDK, NDK, Necissitas* (follow video on QT-Android lighthouse project [42]).
- Now we have an emulator, but to make the device detected by the computer, follow "Setting up a Device for Development" on android developers website [43] (do not forget to change the device ID).
- Add the path to *Platform_tools folder* of the SDK to the Linux path variable by typing the following command:
  export PATH=${PATH}:/home/alhussab/android/android-sdk-linux_x86/platform-tools
- From the shell on linux, we make sure that the device is detected, by typing the command: *adb devices* after CD to *platform_tools.*

B.2 Root Android Phones

Here we provide steps to root Android phones, which is important to enable Ad-hoc and obtain full root access.

- Download *nexus_one_softroot.tar* and extract contents to the same folder as adb
- Open up your terminal, cd to the same folder as adb and the extracted files, and enter the following commands:

```
sudo ./adb push psneuter /data/local/tmp/psneuter
sudo ./adb push busybox /data/local/tmp/busybox
sudo ./adb push su /data/local/tmp/su
sudo ./adb shell chmod 755 /data/local/tmp/psneuter
sudo ./adb shell chmod 755 /data/local/tmp/busybox
sudo ./adb shell chmod 755 /data/local/tmp/su
sudo ./adb shell
```

After this you should see only a $ which tells us that we at the android command line with user privileges only. Type:
$ cd /data/local/tmp
$ ls
$ ./psneuter
property service neutered, killing adb (should restart in a second or two). Then, this wil take you out of the shell
Enter again:
sudo ./adb shell
After this you should see only a # which tells us that we became root . If you want to double check issue this command:
# id uid=0(root) gid=0(root)

From this point we will install *busybox* and *su* which will make root permanent
Code:
# mount -o remount,rw -t yaffs2 /dev/block/mtdblock3 /system
# cd /data/local/tmp # ./busybox cp busybox /system/bin

```
# chmod 4755 /system/bin/busybox
# busybox cp su /system/bin
# chmod 4755 /system/bin/su
# exit
```

We enter again to finish the procedure of rooting:

```
sudo ./adb shell
# su
# mount -o remount,ro -t yaffs2 /dev/block/mtdblock3 /system
# exit
# exit
```

- Application "Superuser" is needed (get from market)
- You can access each time the device shell by sudo ./adb shell , then type " su " command to have the root access to your device
- We use application " ROM manager" to check new ROMS, then we used ***CyanogenMod 6.1.0*** ROM where WiFi ad-hoc is enabled.

Another method used to install new custom ROMs on Nexus One :

- Unlock bootloader
- Flash recovery image
- Backup
- Install your desired version of firmware (Cyanogen mode) from Sdcard
- Wipe data before you install any version (Format).
- A useful guide for the whole previous steps is [44]

B.3 Debugging

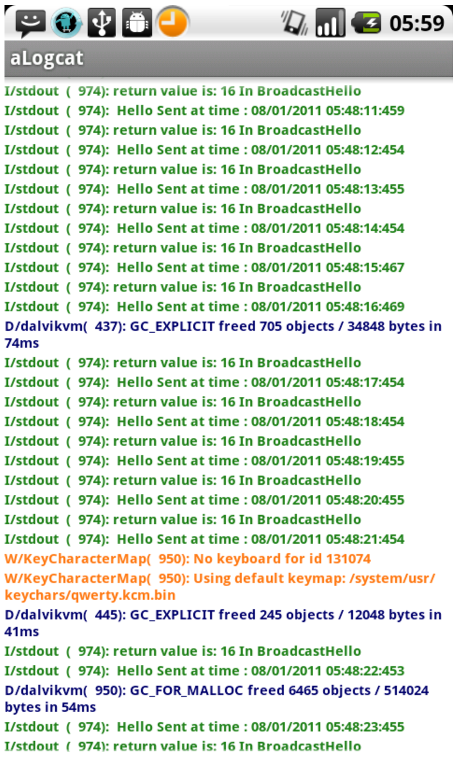
We use Shell *adb logcat* as a real-time debugging and monitoring tool for the code running on the phones. It runs on the compute‘s linux shell, and requires the phone to be connected to the computer by the USB cable. As we program in C++, to view the stdout logs, the following system script sould be copied to the phone:

Local.prop which contains this command:

```
log.redirect-stdio = true
```

B.4 Useful Tools During Experiments

- Alogcat App : Application allows using Logcat on the phone, to have real time monitor of the activities running in the background.
- Clocksynch App : Devices participating in the experiments are not perfectly time synchronized. Clock synchronization in DTNs has been a very challenging area in research. Due to the lack of time, we used this App, to synchronize the clocks before starting the experiments.
- ScreenOn : Sometimes we had problems in receiving all incoming datagrams while the screen is off. This can be due to wireless chipset problems, in addition to the android version used. This requires more research. Therefore, we used this App, that keeps the screen On if our testbed application is running in the background. It is recommended to set the screen brightness to the lowest, for avoiding a very fast drain of the battery. Figure B.2 shows a snapshot of the alogcat app.



```
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:11:459
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:12:454
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:13:455
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:14:454
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:15:467
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:16:469
D/dalvikvm(  437): GC_EXPLICIT freed 705 objects / 34848 bytes in
74ms
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:17:454
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:18:454
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:19:455
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:20:455
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:21:454
W/KeyCharacterMap(  950): No keyboard for id 131074
W/KeyCharacterMap(  950): Using default keymap: /system/usr/
keychars/qwerty.kcm.bin
D/dalvikvm(  445): GC_EXPLICIT freed 245 objects / 12048 bytes in
41ms
I/stdout  (  974): return value is: 16 In BroadcastHello
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:22:453
D/dalvikvm(  950): GC_FOR_MALLOC freed 6465 objects / 514024
bytes in 54ms
I/stdout  (  974):  Hello Sent at time : 08/01/2011 05:48:23:455
I/stdout  (  974): return value is: 16 In BroadcastHello
```

Figure B.2: Broadcast Hello shown in alogcat

Bibliography

[1] Chart: Egyptian text messages still largely blocked, http://irnglobal.com/?p=27146.

[2] Anders Lindgren, Avri Doria, and Olov Schelen. Probabilistic routing in intermittently connected networks. *ACM*, 2003.

[3] Theus Hossmann, Thrasyvoulos Spyropoulos, and Franck Legendre. Know thy neighbor: Towards optimal mapping of contacts to social graphs for dtn routing. *InfoCom*, 2009.

[4] Android compare, http://androidcompare.com/.

[5] Alex Pentland, Richard Fletcher, and Amir A. Hasson. A road to universal broadband connectivity. *Proceeding of 2nd International Conference on Open Collaborative Design for Sustainable Innovation*, 2002.

[6] Scott Burleigh, Adrian Hooke, and Leigh Torgerson. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine*, 2003.

[7] Elizabeth Daly and Mads Haahr. Social network analysis for routing in disconnected delay-tolerant manets. *MobiHoc*, 2007.

[8] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay tolerant networks. *MobiHoc*, 2008.

[9] Vincent Lenders, Jörg Wagner, and Martin May. Measurements from an 802.11b mobile ad hoc network, 2006.

[10] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, May 2006.

[11] T. Clausen and P. Jacquet. Optimised link state routing protocol (olsr). *RFC 3626 , IETF Network Working Group*, 2003.

[12] Charles. E. Perkins and Pravin Bhagwat. Highly dynamic destination- sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM Computer Communication Review, 24(4):234–244*, 1994.

[13] C. Perkins and E. Belding-Royer. Ad hoc on-demand distance vector (aodv) routing. *RFC 3561, IETF Network Working Group.*, 2003.

[14] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing, volume 353 of The International Series in Engineering and Computer Science, pages 153–181*, 1996.

[15] Graham Williamson. *Routing in Human Contact Networks.* PhD thesis, National University of Ireland , Dublin, 2010.

[16] Kevin Fall and Steven Farrell. Dtn: An architectural retrospective. *IEEE Journal on Selected Areas in Communications*, June 2008.

[17] Delay tolerant networking research group, http://www.dtnrg.org/.

[18] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. *In Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM 04, page 145, New York, New York, USA, ACM Press.*, 2004.

[19] Evan Jones, Lily Li, and Paul Ward. Practical routing in delay- tolerant networks. *ACM press*, 2005.

[20] Sze-Yao Ni, NewAuthor2, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *MobiCom*, 2004.

[21] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. *ACM/IEEE international conference on Mobile computing and networking*, 2005.

[22] V. Erramilli, A. Chaintreau, M. Crovella, and C. Diot. Diversity of forwarding paths in pocket switched networks. *IMC*, 2007.

[23] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli. Age matters: Efficient route discovery in mobile ad hoc networks using encounter ages. *MobiHoc*, 2003.

[24] M. E. J. Newman. The structure and function of complex networks. *SIAM review, vol 45, pp. 167-256*, 2003.

[25] Complex network analysis, http://sites.google.com/site/reuvenaviv/home/my-research/complex-network-analysis.

[26] Gautam S. Thakur, Ahmed Helmy, and Wei-Jen Hsu. Similarity analysis and modeling in mobile societies: The missing link. *ACM*, 2009.

[27] Kristina Lerman, Rumi Ghosh, and Jeon Hyung Kang. Centrality metric for dynamic networks. *USC Information Sciences Institute*, 2010.

[28] P Bonacich. Eigenvectorlike measures of centrality for assymetric relations. social networks. *23:191–201*, 2001.

[29] K Stephenson and M Zelen. Rethinking centrality: Methods and applications. *Social Networks*, 1989.

[30] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[31] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra. Implementing delay tolerant networking. *Tech. Rep. IRB-TR-04-020, Intel Research Berkeley*, 2004.

[32] H. Falaki and E. Oliver. Performance evaluation and analysis of delay tolerant networking. *(New York, NY, USA), pp. 1–6, ACM.*, 2007.

[33] Joshua Reich, Vishal Misra, and Dan Rubenstein. Roomba madnet: a mobile ad-hoc delay tolerant network testbed. *CS EE Department, Columbia University, New York, NY, USA*, 2007.

[34] Erik Nordström, Per Gunningberg, and Christian Rohner. A search-based network architecture for mobile devices. Technical Report 2009-003, Department of Information Technology, Uppsala University, January 2009.

[35] Anna Kaisa Pietiläinen, Earl Oliver, Jason Lebrun, George Varghese, and Christophe Diot. Mobiclique: middleware for mobile social networking. In *ACM SIGCOMM Conference*, pages 49–54, 2009.

[36] F. R. K. Chung. spectral graph theory. *American Society*, 1997.

[37] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2001.

[38] Cross platform application and ui framework, http://qt.nokia.com/.

[39] Oracle java native interface specifications http://download.oracle.com/javase/6/docs/technotes/guides/jni/spec/

[40] Bundle protocol specification - ietf rfc 5050 http://tools.ietf.org/html/rfc5050.

[41] Sacha Trifunovic, Bernhard Distl, Dominik Schatzmann, and Franck Legendre. Wifi-opp: Ad-hoc-less opportunistic networking. *ACM*, 2011.

[42] Necessitas qt on android, http://sourceforge.net/p/necessitas/home/necessitas/.

[43] Android developers. http://developer.android.com/index.html.

[44] Step by step guide to install cyanogen mod rom on nexus one, http://nexusonehacks.net/.

[45] Lily Li and Paul A. S. Ward Evan P. C. Jones. Practical routing in delay-tolerant networks. *Electrical & Computer Engineering University of Waterloo Waterloo, Ontario, Canada*, 2006.

[46] M. I. Jordan and Y. Weiss A. Y. Ng. On spectral clustering: Analysis and an algorithm. *In Advances in Neural Information Processing Systems, MIT press*, 2001.

[47] D. J. Watts. The dynamics of networks between order and randomness. *Princeton University Press*, November 2003.