There are three kinds of death in this world. There's heart death, there's brain death, and there's being off the network.

– Guy Almes

University of Alberta

Exploiting periodicity within mobile data for routing in Delay Tolerant Mobile Networks

by

Zhiyu Wang

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

To my beloved aunt Ming Wang, I miss you.
To my parents Xing Wang and Jufeng Yang, aunt Liang Wang and my wife Lin Li.
I love you all.

Abstract

Delay Tolerant Mobile Networks (DTMNs) provide communication despite the occasional presence of disconnected subnetworks. They rely on finding a set of sequential opportunistic encounters between pairs of mobile nodes. In this context, understanding mobile node behaviour is essential to design effective and efficient network protocols. Previous studies aimed to predict future encounters where predictions depend on exploring the probability/age of encounters and integrated interactions in the mobile data. However, those previous solutions suffer from unstable predicted encounters with lack of routing information such as encounter times. As an alternative to prediction, we propose to exploit periodicity within mobile data to find stable (periodic) encounters for routing in DTMNs. In this thesis, we first present a generic methodology to model and find periodic encounter patterns by using the auto-persistence function and detection techniques derived from it. Secondly, we propose a novel graph model to capture periodic encounter patterns where routing problems can be modelled and solved as optimization problems. Lastly, to connect disconnected subnetworks that are strongly connected inside, *e.g.*, by periodic encounters, in the networks we introduce stationary relay nodes whose deployment is modelled as various *k*-connectivity problems. Taking advantage of our studies, the experimental results demonstrate that in the environment of DTMNs with the presence of disconnected sub-networks, message delivery can benefit greatly from the underlying periodicity within mobile data. In addition, exploiting periodicity opens up new research frontiers in several aspects such as designing novel routing protocols, query dissemination and collection, and preserving privacy and security in environments with the presence of periodic behaviours.

Acknowledgements

I am grateful for the assistance and help from many people over the last several years.

First of all, I would like to give my sincere thanks to my supervisors Dr. Mario Nascimento and Dr. Mike MacGregor for their guidance, patience and extensive support. I not only benefited from their knowledge but also the opportunities they provided me with. In addition, they were always more than happy to help me with my personal issues. Without their support, this work would be impossible for me. No word can express my gratitude and appreciation to them.

I must thank some faculty members and graduate students for their help during my Ph.D program. I would like to thank Dr. Lorna Stewart for all genuine and valuable discussions regarding graph problems. The course I took from her inspired me to the graph related problems in this thesis. In addition, I would like to thank Dr. Janelle Harms, Dr. Ehab Elmallah and Dr. Martha Streenstrup for their valuable comments and feedback at our research group meetings. My thanks also go to Dr. Israat Tanzeena Haque, Ke Li, Sheehan Khan, Katherine Chen, Rick Valenzano, Levi Santana, Shahab Jabbari, Dr. Zachary Friggstad, Dr. Babak Behsaz, Siamak Ravanbakhsh for fruitful discussions.

I really appreciate the financial support from the Computing Science Department at the University of Alberta, from NSERC's DIVA Strategic Network and from NSERC's Discovery Grant program.

Finally, I would like to thank my parents and my aunts for their long-term understanding, encouragement and love. My utmost gratitude goes to my wife Lin Li for her unconditional love through my entire Ph.D journey.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AP	Access point
APF	Auto-persistence function
APG	Auto-persistence graph
APL	Average path length
DFT	Discrete Fourier Transform
DTMN	Delay Tolerant Mobile Network
DTN	Delay Tolerant Network
ETS	Edmonton Transit System
GCC	Global clustering coefficient
LCC	Local clustering coefficient
MANET	Mobile ad-hoc network
MAP	Mobile Access Points
NOT	NO reTransmissions during the same phase
ONE	Opportunistic Networking Environment
WSN	Wireless Sensor Network
VANET	Vehicular ad hoc network
VDTN	Vehicular Delay Tolerant Network

List of Symbols

Chapter 2

$S_{x,y}$	An encounter series for a unique pair of nodes x and y
n	The length of an encounter series
$R_{x,y}$	The encounter rate for a unique pair of nodes x and y
m	The period length of a periodic encounter pattern
$P^m_{x,y}$	An encounter pattern with a period length m
p_i	A data point in an encounter pattern
d_i	A data point in an encounter series
T	The user-defined threshold where $T \in [0,1]$
k	A lag in a graph
m_u	least common multiple of different pattern length
$P^{m_u}_{x,y}$	The fundamental encounter pattern of an encounter series
$\overline{m}$	The mean value of APG values from an encounter series
σ	The standard deviation of APG values from an encounter series
K	A constant
p_α	The probability of missing the periodic encounter at a data point
p_β	The probability of an unexpected/occasional encounter at a data point
T_i	The timestamp of an encounter within mobile data
T^{AVG}	The average inter-contact time
I_i	An interval corresponding to one period of a periodic encounter pattern
θ	The matching probability of periodic encounters within an I_i
Δ	The acceptable number of intervals with missing periodic encounters
N	The number of nodes in a network
d	The average node degree

Chapter 3

m	The period length of a periodic encounter pattern
$P^m_{x,y}$	An encounter pattern with a period length m
p_i	A data point in an encounter pattern
M^k	The encounter matrix for mobile node o_k
m^k_{jt}	An encounter between o_j and o_k at time t
V^R	A set of receiving vertices in a graph
$v^R_{t,i}$	The receiving vertex for node o_i at time t
V^T	A set of transmitting vertices in a graph
$v^T_{t,i}$	The transmitting vertex for node o_i at time t
V^D	A set of destination vertices in a graph

v_i^D	The destination vertex for node o_i
E^H	A set of horizontal edges in a graph
$e_{t,i}^H$	The horizontal edge connecting vertices $v_{t,i}^R$ and $v_{t,i+1}^R$
E^V	A set of vertical edges in a graph
$e_{t,i,j}^V$	The vertical edge connecting vertices $v_{t,i}^T$ and $v_{t,j}^R$
E^I	A set of internal edges in a graph
$e_{t,i}^I$	The internal edge connecting vertices $v_{t,i}^R$ and $v_{t,i}^T$
E^D	A set of destination edges in a graph
$e_{t,i}^D$	The destination edge connecting vertices $v_{t,i}^R$ and v_i^D
D	A set of desired destinations in multicast routing
d_i	A desired destinations where $d_i \in D$
d	The total number of desired destinations where $d = \|D\|$
s	The source vertex in the multicast routing
T^{OPT}	The optimal directed Steiner tree for multicast routing
V_i^{OPT}	A vertex on T^{OPT} where $V_i^{OPT} \in T^{OPT}$
n	The total number of nodes in a graph
r	The radio range

Chapter 4

k	A value for connectivity in a graph
T	A set of terminals
t_i	A terminal
$C_{i,j}$	The commodity in a network between terminals t_i and t_j
R	A set of relays
r_i	A relay
f_{uv}^{ij}	The flow in a commodity $C_{i,j}$ along an edge $e_{u,v}$
d_{r_i}	The degree of a relay r_i in a network
d	The degree constraint
n	The number of terminals in the network
m	The number of relays in the network
p	The percentage of terminals in average each relay connects
r	The radio range
s	The message generation rate
b	The buffer size of nodes in the network
b_r	The buffer size of relays

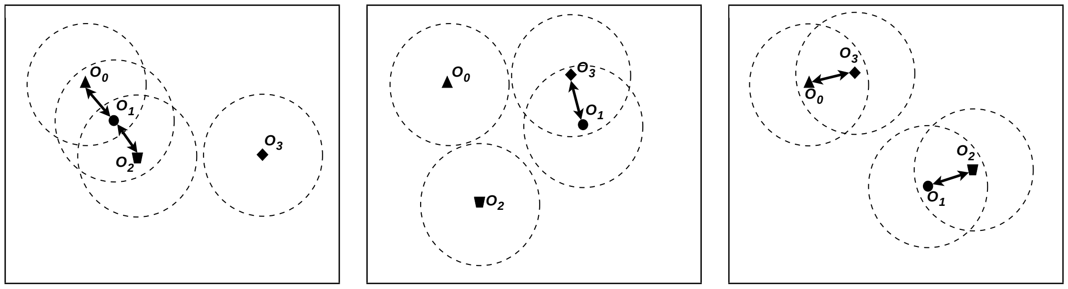
Chapter 1

Introduction

1.1 Delay tolerant mobile networks

The increasing popularity of wireless mobile devices such as sensors, laptops, tablets and smart-phones has created demand for novel techniques for effective and efficient communications in mobile networks. Mobile ad-hoc network (MANET) is one type of mobile network that has been studied extensively for the past decade. A MANET is a highly dynamic network that is established amongst mobile devices, without any need from other supporting infrastructure [90]. Each node participating in the network acts as both an end-host and a relay. Even though wireless devices are mobile within MANET, they are typically assumed to constitute a connected network where at least one path between any pair of nodes always exists. In addition, MANET has continuous bidirectional end-to-end connectivity with symmetric communications and a relatively short propagation delay. Thus, communications are possible if routing protocols adjust the routes in response to mobility. In general, logical structures such as paths and trees on top of the physical topology are used to guarantee optimal (or near-optimal) costs between sources and destinations. However, in some application domains the physical topology may change often and nodes (or entire sub-networks) may not always be connected. Disconnections in the network are possible due to various forms of node movement as well as energy management or interferences [34].

Previous studies have shown that node movement does impact the connectivity of mobile networks [20, 83, 125]. Mobile movements may lead to a situation called network partition where an end-to-end path may never exist because the network is divided into several isolated sub-networks from time to time. In addition, nodes travelling at high speeds in some environments, *e.g.*, vehicular ad-hoc networks (VANETs) [67], means that disconnection in the network is the norm rather than the exception. In this type of environment, traditional MANET route-discovery protocols are inefficient and ineffective regarding both



(a) Snapshot at interval I_0 (b) Snapshot at interval I_1 (c) Snapshot at interval I_2

Figure 1.1: Routing with opportunistic encounters

delivery ratio and resource consumption [6, 64, 70, 75, 106, 107]. Frequently discovering new routes imposes control overhead to the communication task and sometimes fails to return any result. In mobile wireless sensor networks, because each sensor has limited battery capacity [7], the communication overhead of frequent route discovery is a severe drawback. In summary, directly applying protocols from MANET does not meet the specific requirements of disconnected networks.

As an alternative solution, Delay Tolerant Networks (DTNs) [34], also called Disruption or Disconnection Tolerant Networks, provide communication despite the occasional presence of disconnected sub-networks. In this thesis, we focus on Delay Tolerant Mobile Networks (DTMNs) where a lack of continuous network connectivity is caused by node mobility. To address the routing problem in such networks, communications depend on finding a set of sequential opportunistic encounters between pairs of mobile nodes in a store-and-forward fashion where an encounter is defined as a time period long enough so that mobile nodes can directly communicate with each other. Unlike their roles in MANET, participants in DTMNs are not only end-hosts and relays, but also serve as temporary storage for messages during routing. Messages will be exchanged when opportunistic encounters arise.

Figure 1.1 illustrates an end-to-end communication using opportunistic encounters due to lack of continuous connection in a DTMNs. Let us assume that source o_0 has a message for destination o_3. An opportunistic encounter, indicated by a bidirectional arrow, arises during interval I_0. At this encounter, o_0 forwards the message to an intermediate node o_1. Node o_1 carries the message until it encounters destination o_3 during interval I_1. At this encounter, the message is indirectly delivered. As an alternative route, o_0 could hold the message until it meets destination o_3 directly during I_2. Because communications are asymmetric in DTMNs, the same path from o_0 to o_3 is not applicable if o_3 has a message for o_0.

Within DTMNs, opportunistic encounters are critical for routing. In this thesis, we propose to utilize stable encounters, more specifically periodic encounter patterns, within mobile data for routing in DTMNs.

1.2 Routing protocols in delay tolerant mobile networks

In this section, we concentrate on routing protocols and node mobility patterns that have already been used in DTMNs. There are mainly four categories of routing protocols in DTMNs: 1) mobility-unaware routing where protocols choose opportunistic encounters without considering overall node movement, 2) mobility-aware routing where protocols use the past history of node mobility to predict future encounters, 3) social-aware routing where protocols compute routes based on social networks that are derived from aggregating past encounters, and 4) periodicity-aware routing where protocols forward messages using regular and periodic encounters within node movement.

1.2.1 Mobility-unaware routing

Under the circumstance that there is no knowledge about when and which pair of nodes will encounter in the future, the *epidemic routing protocol* targets at providing promising delivery by utilizing all possible opportunistic encounters [129]. By flooding the message, every node holding a copy of the original message transmits a replica to the one without a copy at every opportunistic encounter. This protocol guarantees the delivery with the shortest delay. However, it is inefficient with regard to energy consumption because of the massive number of retransmissions and is also inefficient regarding the storage where a large number of replicas is distributed in the network. In contrast to epidemic routing, in *direct delivery* [119] the source delivers the message if and only if it encounters the destination. This protocol guarantees minimal storage and energy usage but with a potentially low delivery ratio because the probability of meeting the destination could be low. Epidemic and direct delivery routing protocols set upper and lower bounds on resource usage and delivery ratio in DTMNs.

To improve epidemic routing, researchers have proposed protocols that control the flooding in the network by limiting the number of replicas [128], diverting replicas individually [122], reducing the lifetime and the distance that replicas can travel in the network [69] and limiting the maximum number of relay hops to reach the destination [45]. However, it is difficult to accurately tune parameters in these protocols to adjust different mobile environments for better performances. As alternative solutions to direct delivery, some protocols aim to maintain one copy of the message in circulation in the networks [66, 123]. For example, the *first-contact routing protocol* always forwards the message to the first encountered node [66], and *randomized routing protocol* randomly transmits the message to a relay based on a forwarding probability [123]. In general, this single-copy forwarding protocol has very low delivery ratio because of long delays and low encounter probabilities. In contrast, flooding-based routing protocols achieve better performance while consuming a large amount of network resources. Now, the trend of designing protocols in DTMNs exploits node mobility where encounter patterns can be applied in routing.

With respect to inter-contact times and reachability of nodes, it has been shown that mobility and pair-wise encounters can benefit communications in disconnected networks [20, 125, 144]. In order to improve routing performance, solutions have been proposed to exploit node mobility patterns. For example, *FRESH* [31] selects the node whose last encounter with the destination is the most recent. *PROPHET* [85] uses the history of past encounters to estimate the nodes' delivery probabilities. In *MaxProp* [16] each node keeps track of the probabilities of encountering other nodes. *BubbleRap* [60] relies on social structures of the network for routing, which are aggregated on encounter histories. As we can see, different aspects of node mobility have been used to predict opportunistic encounters. Both mobility-aware and social-aware routing protocols are discussed next.

1.2.2 Mobility-aware routing

Mobility-aware routing protocols use the past history of node mobility to make forwarding decisions. In general there are two categories of statistics being used to design mobility-aware routing protocols. One category of protocol predicts the encounter probability based on past encounters. The other category of protocol learns from the age of encounters such as elapsed times since last encounters.

Probabilistic protocols

In previous studies, many protocols have been proposed to utilize encounter probabilities. For example, *MV* exploits the frequency of encounters in the past [17]. *OPF* replicates messages to achieve the maximum delivery probability [86]. 3R predicts the probabilities to meet other nodes at a certain hour during weekdays and weekends [130]. In this section, we focus on the most popular routing protocols in this category, which are *PROPHET*, *MaxProp*, *PREP* and *Prediction*.

PROPHET was introduced to direct communications by learning from past encounters and their transitivity properties [85]. In that paper, the authors introduced a metric called *delivery probability* at every node to indicate the probability that it will deliver the message to a given destination. Therefore, each node needs to maintain a list of delivery probabilities for different nodes in the network. At the beginning, each node can only record delivery probabilities for one-hop (directly encountered) neighbours. If a pair of nodes do not encounter each other in a while, delivery probabilities in both nodes decrease as time elapses. Later, the property of transitivity helps to calculate delivery probabilities for indirectly encountered nodes. The observation is that if node A frequently encounters node B, and node B frequently encounters node C, it implies that node B could be a good relay for messages destined to node A from node C, and vice versa. Nodes exchange summary vectors containing delivery probabilities and update their summary vectors accordingly. Last, if a node has a message, it only delivers it to the nodes that have a higher delivery probability to the destination than the current holder. Experiments showed that *PROPHET* achieved a good performance while consuming less network resources, *i.e.*, number of transmissions, comparing to epidemic routing. However, one problem is that *PROPHET* was evaluated using an unrealistic random mobility model so that, statistically, encounter probabilities do not vary with time. In addition, delivery probabilities that were cumulated as long as the experiments ran could not capture changes of encounter patterns within sub-periods.

MaxProp is another history-based protocol that calculates the delivery likelihood of a message, which is the probability of meeting other mobile nodes [16]. Similar to *PROPHET*, each node collects encounter probabilities for one-hop neighbours and then shares this local information with other nodes. Whenever there is a message, the source needs to compute the path with the best delivery likelihood based on collected probabilities. The difference between *MaxProp* and *PROPHET* is that *MaxProp* concentrates on local encounter probabilities where the probabilities of all possible paths for a source-destination pair can be computed in order to find the best one, while *PROPHET* finds end-to-end probabilities without further computation. In addition, *MaxProp* used *acks* for a better storage management where nodes can drop acked messages from their buffers. However, just like *PROPHET*, delivery likelihoods are cumulated for the whole duration of experiments, where changes of sub-periods could not be captured.

Both *MaxProp* and *PROPHET* exchange up to $O(n)$ number of messages at every encounter, where n is the number of nodes in the network. The total number of this exchange process in the network could be up to $O(n^2)$ times. To address the problem of massive exchanges, *Encounter-Based Routing protocol* (*EBR*) was proposed based on the observation that nodes experiencing a large number of encounters are more likely to successfully pass the message to the destination than the ones who have less encounters [98]. In *EBR*, a metric called *Encounter Value* was introduced to measure a node's past rate of encounters within a certain window length as an exponentially weighted moving average. The higher the Encounter Value is, the higher the probability to deliver a message is. A sender always transmits a copy of the message to one-hop neighbours with a higher Encounter Value. In this protocol, control overheads are saved by only comparing their Encounter Values. However, using *Encounter Values* regardless of the identities of encountered nodes is not enough to guarantee a good performance. For example, if high *Encounter Values* within a set of nodes are mainly contributed by encounters among themselves, messages destined to other nodes could get stuck within this set of nodes and never get delivered.

PREP is another protocol that takes advantage of node mobility. It plans routing by predicting availabilities of encounters between mobile nodes and proposes message prioritization to reduce the burden on storage and bandwidth [113]. In *PREP*, a metric called *average availability* is attributed to every pair of nodes to measure the average fraction of time that two nodes encountered in the past. This value is reset to zero if both nodes have not encountered for a while. With a neighbour discovery algorithm, each node obtains the duration of encounters to its one-hop neighbours in the past. Then, average availabilities are calculated to determine the availabilities of those encounters in the future. Whenever there is a change of average availability at a particular node to its one-hop neighbours, an update is disseminated into the network using epidemic routing. Therefore, the source node uses Dijkstra's shortest path algorithm to find the best route in terms of *average availability*. To conserve network resources, a priority schema is used to decide when to drop a message and which message to send first. This schema in *PREP* allows effective usage of network resources leading to a good performance on delivery ratio. However, this priority schema depends strongly on a set of thresholds. How to choose proper values for them in different applications has not been addressed by the authors in that paper.

Another protocol called *Prediction* was proposed to use historical contact information to estimate the encounter probability between every pair of nodes in the network [119]. *Prediction* protocol estimates the contact probability based on a metric called *timely-contact probability* that is the contact frequency between two nodes. To calculate this probability, the duration of each input trace is partitioned into sequential intervals with the same duration. For each pair of directly encountered nodes, their *timely-contact probabilities* measure the fraction of how many intervals actually contain at least one encounter between them. Over time, these direct contact probabilities are shared with all nodes in the network. Therefore, indirect contact probabilities can be derived from them. If a sender has a message, it transmits the message to a set of encountered nodes whose *timely-contact probability* to the destination is higher than a threshold. Note that this protocol replicates messages during routing. Comparing to *PROPHET* and *MaxProp* that are forwarding-based protocols, *Prediction* pushes replicas into the network, which is inefficient regarding network resources. In addition, different applications could have different mobile behaviours. Selecting a proper granularity for partitioning and choosing a proper threshold for replicating to achieve good performances are critical issues. How to do that is not addressed in that work.

Encounter age-based protocols

Besides predicting encounter probabilities from past histories, other protocols aim at using the age of last encounter to measure the reliability of encounters between nodes in the future.

FRESH is a protocol that directs routing by picking the next hop that encountered the destination earlier in the past [31]. Each node in the network maintains records of time elapsed since the most recent encounter with all other nodes. When a sender has a message for the destination, it searches for one-hop neighbours who encountered the destination more recent than the sender as the next relay. Then newly added relay keeps searching for the next relay who encountered the destination more recently. The same procedure repeats until the destination is reached. The assumption behind this protocol is that the distance travelled during an interval is positively correlated with the duration of this interval. For example, nodes encountered few minutes ago are probably closer than nodes that were encountered few hours ago. *FRESH* improves the usage of network resource without exchanging the global knowledge of networks. However, in heterogeneous networks, time may not be positively correlated to distance. Nodes who recently met the destination may not always be the ones who are closer to the destination. In addition, it is incorrect to assume that all nodes within the network encounter with each other. Added to that, in the scenarios that networks are partitioned into sub-regions with very little interaction among them, messages destined to other regions may never get delivered.

MEED is a method that keeps track of the disconnection time between node pairs and uses this information to make routing decisions [71]. In the network, each node records disconnection times with its one-hop neighbours. After sharing this information with other nodes, each node obtains the topology of entire network described as a graph, whose vertices are nodes, and edges connect pairs of encountered nodes with weights describing the estimated delays between two consecutive encounters. If a node has a message, it computes the path with the *minimum estimated expected delay* and sends the message to the first relay on this calculated path. A note here is that nodes may not have exactly the same graph at the same time. A relay may find a better path in its own graph that has more recent information than the sender. Therefore, nodes have to locally recompute the path for every received message to find a next hop. In addition, frequently distributing topology updates may also limit the performance of *MEED*.

Similarly to using contact history in *MEED*, another link-state protocol, proposed by Su *et al.*, uses the history of inter-contact times to measure the relationship between two nodes [126]. First, each node maintains perceived states of the network referred as a link-state graph. In the graph vertices represent nodes, and edges represent encounters between pairs of nodes. A weight on an edge is modelled as an exponentially weighted moving average that captures the changes of inter-contact times. Over time, nodes will update their link-state graphs and share updated entries with other nodes. The idea for routing is to find the path with the minimum weight (inter-contact time) on the graph. Just like *MEED*, frequently updating link-state graphs imposes control overheads in the network.

The *EASE* protocol exploits node mobility based on the time and the location of its last encounter with every other node in the network [46]. Because every node only needs to maintain a database of direct encounter information, no communication overhead is incurred to the network. To be a good relay, a node must either have a more recent encounter with the destination or physically closer to the destination than the current holder. Again, *EASE* assumes every node meets the rest of the network, which is not necessarily true. Another problem is that mobility patterns and speeds could degrade the performance in terms of route length because both the time and location of last encounter become less stable in irregular moving patterns or fast moving destinations.

Predict and Relay is another protocol that studies mobility history, specially contacts [141]. Because mobile nodes move between landmarks, where associated nodes can directly contact with others, according to their social schedules, *Predict and Relay* protocol addresses routing by calculating transition probabilities between landmarks based on contact history at a certain time. Therefore, inter-node contact probabilities can be derived. After sharing transition probabilities with the rest of the network, every node can calculate its own probability to meet the destination. Messages are forwarded using a greedy approach to select next hops with the highest encounter probability with the destination. The experiments showed that this protocol had good performance in terms of delivery delay and ratio. However, its performance will degrade as the number of landmarks increases in the network. Increasing the number of landmarks increases the communication overhead because each node needs to exchange up to $O(m^2)$ transition probabilities where m is the number of landmarks in the network. In some scenarios, this is even worse than other probabilistic protocols such as PROPHET and *MaxProp* with only $O(n)$ updates, where n is the number of nodes.

1.2.3 Social-aware routing

The performance of mobility-aware routing protocols depends highly on tuning parameters and/or global sharing of local collected probabilities, which may lead to significant consumption of energy, processing capability and bandwidth. As an alternative, social-aware routing was proposed to use social relationships between nodes to make forwarding decisions. This is because social relationships are thought to be less volatile than mobility behaviours [60]. In social-aware routing, connectivity is modelled using a graph called a social network. Each node is represented with a vertex, and an edge is assigned between two vertices if they have a social relationship. The idea of communities is also introduced in the network where people from the same community have stronger social relationships compared to people from different communities.

LABEL is one of the first methods that exploited social relationships, especially communities for routing in disconnected networks [59]. In that work, the authors distributed iMotes to participants at *INFOCOM* where iMotes used Bluetooth to collect encounter information. Before handing out the mobile devices, four groups/communities of people were carefully selected by labelling them with a group name. The inter-contact and intra-contact distributions from collected contacts showed that people from the same group encountered each other more frequently than people who are friends from different communities. In addition, friends from different communities encountered more frequently than people that are neither friends nor from the the same community. In their *LABEL* strategy, group labelling technique allows to forward messages directly to the destination or to a relay who also belongs to the same community as the destination. Experiments showed that forwarding based on communities improved the delivery ratio and delivery cost in terms of total number of transmissions. Additionally, forwarding between friends from different communities can slightly improve the delivery ratio. However, the *LABEL* strategy depends on relays from the same group as the destination. The delivery will fail if the sender never encounters any member from the same group as the destination.

As we have seen in *LABEL*, sources forward messages to any member from the same community as the destination if both source and destination do not meet directly. However, in some scenarios, the source may never meet any member from the same group as the destination. Even if the source does, relays that the source chooses may never meet the destination within their own group. In those cases, choosing popular nodes within communities and important nodes that connect different communities becomes critical. *SimBet* proposed methods to identify both types of nodes within the context of social networks by measuring nodes' betweenness centrality and social similarity [27]. Betweenness centrality captures how important a node is for connecting two other nodes from different communities who never encounter. Social similarity predicts the probability of future encounters between two nodes. Both metrics are used to calculate the *SimBet* utility function where the effects of the two metrics can be adjusted by tunable parameters in different applications. When two nodes meet each other, they exchange their contact lists along with their similarity and betweenness values. As a result, both nodes can update their metrics accordingly. Forwarding decisions are based on local calculations to find next hops with stronger social characteristics, *i.e.*, similarity and betweenness. The results showed that this protocol had good performance regarding delivery ratio and extremely low delivery cost in terms of total number of transmissions. However, there is one big issue about *SimBet*. When two nodes encounter, a lot of processes need to be performed such as sharing contact histories, computing utility values (both betweenness and similarity) and exchanging a set of messages. Its performance can easily degrade in scenarios where encounters have short durations.

Unlike *SimBet*, *BubbleRap* combines the knowledge of community structures and ranks of nodes to make forwarding decisions [60]. Instead of manually labelling each group as in their previous work (LABEL) the authors first classified nodes into communities based on their social characteristics such as number of contacts and identities of encountered nodes. Then, classified communities and local/global rankings of nodes derived from betweenness centrality are applied to forward messages. Here, global ranking describes the popularity of a node in the entire network whereas local ranking captures the popularity of a node within its own community. Because a node may belong to multiple communities, it can have multiple local rankings. The forwarding strategy of *BubbleRap* is relatively simple. If a node has a message, it first checks whether the destination is within the same community. If it is, the message is bubbled up to relays who have higher local rankings until one of relay encounters with the destination. If the destination is from another community, the message greedily bubbles up to relays with higher global rankings until a relay from the same community as the destination is found. Then, the same procedure in the first case is applied. By detecting communities and forwarding with social rankings, this protocol achieved a good performance regarding high delivery ratio and fewer transmissions. However, messages originated at nodes with low ranking could cause flooding because of exhaustively searching for better relays from the bottom to the top of social ranks. Such a scenario will affect the performance of this protocol.

There are also other protocols that take advantage of social relationships such as *SocialCast* [26] and *PeopleRank* [97], which all use centralities and utility functions to find relays with better social characteristics.

One criticism of using social networks for routing is that the delay is unknown because we do not know when an encounter is going to take place in the social network [130]. Another problem is that social networks grow in complexity over time [54]. A social network is built by aggregating encounter information. After a long time period, the social network will become very complex, sometimes even becoming a complete graph. This affects the utility function, and degrades the performance of routing. In addition, it has been shown that a small number of good nodes, *i.e.*, with high centrality, carry a significant amount of the traffic in the network [110]. The performance of these protocols will significantly degrade if any good node fails.

1.2.4 Periodicity-aware routing

Neither mobility-aware routing nor social-aware routing capture the regularity and periodicity of node mobility. For example, even if two nodes are not highly socially connected or have a low encounter probability in the past, they might still have a promising encounter during a certain time every week. In this section, we discuss the most relevant routing protocols that make forwarding decisions using periodic encounters.

In 2004, an algorithm was proposed for routing in disconnected networks based on an assumption that every node has full knowledge of future encounters in the network [49]. Even though this assumption has nothing to do with regular and periodic nodal encounters, it provided foresight to exploit deterministic encounter within node mobility. Because every node is assumed to have the global knowledge of all encounters during all intervals, a tree rooted at the source can be built for forwarding, where each level of the tree represents encounters at a certain interval, *k*, that is *k* intervals away from the root. If nodes do not have global knowledge of encounters, they can exchange local encounter knowledge with their one-hop neighbours. The same tree built on top of collected information is used to make forwarding decisions. In a distributed approach, because nodes within the network do not have exactly the same knowledge about the network, relays with a better path can make local decisions to reroute the message through a better path. Unfortunately, no experiment was conducted to demonstrate the performance of this work. The problem of this approach is that routes are computed with time-varying encounters. If nodes do not move as predicted, the path will be broken accordingly. This is the reason why we need to find those promising encounters within node mobility. Another problem of this work is the tree itself. If nodes have global information, the constructed tree could be very big because it has to record all encounters in the network.

In another work, the authors introduced the idea of contacting sequence that describes a sequential set of repetitive encounters within a time window [68]. It was assumed that repetitive encounters are given and repeat themselves with a given period in the future. Based on this assumption, a contact graph was built on top of repetitive patterns. Within this contact graph, each vertex represents a node whereas an edge links two nodes with a weight to describe the contact time within the period. Because in real life scenarios contacts may not repeat at exactly the same time, a probability of the occurrence of a contact is introduced as a part of the weight on an edge. Therefore, the weight on an edge is a tuple with two parameters, the contact time and probability of occurrence. Within the graph, the path to deliver a message is a sequence of contacts that occur in an increasing order of time based on the weight on edges. At the beginning, a centralized algorithm is required to calculate the best probability to access each edge from a specific source by examining them in the right order of time. To find the best path (highest delivery probability) for a given source-destination pair, another algorithm is used in a reverse fashion from the destination by greedily adding the edge with the highest probability step by step. Finally, a protocol makes forwarding decisions by only transmitting the message to the nodes that are on the pre-calculated path. Experimental results showed it can achieve high delivery ratio with no message replication. However, the problem is that a 2-tuple weight on an edge makes computation extremely complicated. This is the reason that two algorithms are used to find an end-to-end path. In that paper, a centralized algorithm was used to sort all edges according to their contact times globally to calculate access probabilities of using an edge for a specific source for routing. It is impractical to apply that approach in distributed environment because getting such global information in real-time is extremely inefficient. The worse problem is that algorithms in this protocol have to recompute critical information, *i.e.*, access probabilities, for every source because the algorithm is source-oriented.

One problem of existing solutions in periodicity-aware routing is that they all assumed that repetitive encounter patterns are given. However, as far as we know, no previous method is able to extract periodic patterns that can be used for routing. One objective of our studies is to fill this gap.

1.3 Periodic behaviours within mobile data

One interesting characteristic that has emerged from node mobility is that mobile nodes often have periodic behaviours [44, 68, 118]. In many domains, mobile nodes have relatively predictable and periodic patterns of movement. They tend to have a repeated, stable sequence of activities because of their social and/or controlled behaviours. For example, mobile robots in an automated factory have relatively invariant mobility traces because they have predefined schedules and areas to work, and people use relatively stable routes daily to commute to and from work. One outcome from these repeated activities is that each mobile node may encounter a set of other nodes periodically, for example at approximately the same time every day. We call these sequences of periodic encounters *encounter patterns*. Compared to previous works that predict next relays for routing, periodic encounters reveal useful information within node mobility where the time and node pairs of encounters are much more stable. Hence, if we can obtain such periodic encounter patterns, we can use this information to find routes to forward network packets in a more reliable way. Recalling the example in Figure 1.1, if all nodes have full knowledge of periodic encounters during all three intervals, then indirect delivery has less delivery delay, and direct delivery saves energy. Therefore, different performance metrics can be evaluated in the domain of periodicity.

Mobile devices are usually carried by animals, human beings or equipment that is programmed or operated by humans. These devices may exhibit regular and periodic movement patterns reflecting predefined schedules or social activities. Many applications such as vehicular networks (*i.e.*, public transportation network) [16], sensor networks for wild life monitoring [143] and PeopleNet [96] rely on opportunistic encounters among mobile nodes. If promising periodic encounters exist within node mobility, we should use them for routing instead of wasting network resources because of inaccurate predications. We believe the periodicity within node mobility is more promising to make forwarding decisions because both the time of periodic encounters and periods of periodic behaviours fill the gap within previous works. In addition, all periodic nodes are equally treated where a fair routing can be managed whereas traffic in mobility-aware and social-aware routings is heavily loaded on hot nodes.

1.3.1 Applications

Many applications of DTMNs can be extended to our problem because underlying mobile nodes may exhibit periodic behaviours. In this section, we discuss some of the applications that are closely related to our studies in this thesis.

Vehicular ad-hoc networks

One of the well-known DTMNs is a VANET where vehicles are mobile nodes in the network. With on-board sensors and computers, it aims to provide both vehicle-to-vehicle and vehicle-to-infrastructure communications. More specifically, for example, vehicle-to-vehicle and vehicle-to-infrastructure communications help enhance highway safety with cooperative collision avoidance [13], commercial advertising dissemination [81], entertainment services such as on-line gaming [103] and health data collection [99]. In those applications, vehicles as mobile nodes in the network are mediums reflecting drivers' movement patterns. They do not move randomly but follow the driver's intention to move around. One of their patterns is periodic movement patterns.

Another typical example with periodic patterns of movement is a public transportation system. A testbed called UMass DieselNet at the University of Massachusetts has been deployed [120]. In this testbed, buses follow the same route periodically. The testbed consists of 35 buses each with an on-board wireless device that allows information to be shared and exchanged between buses. In addition, there is a commercialized product called First Mile Solution [1], which implements the studies of DakNet [104]. Its objective is to provide internet services to remote villages with a relatively lower cost, and such systems have already been deployed in India and Cambodia. Instead of building underlying infrastructure, the system is composed of stationary kiosks and mobile access points (MAPs) that can be buses running between remote kiosks and internet gateways, and data is transmitted in a store-and-forward fashion. In DakNet, the system does not provide real-time communications where only asynchronous services such as email and voice mail are provided. Because MAPs are usually equipped on vehicles, they have relatively high capacities in terms of buffer space. As a result, it can provide a relatively high throughput compared to traditional telephone modem. In the setting of DakNet, vehicles running between cities and remote villages have predefined routes and schedules. Because data transmission solely depends on MAPs with periodic movement patterns, communications in such environments could significantly be benefited by techniques presented in this thesis.

Wireless ad-hoc networks

One example of wireless ad-hoc networks is wireless sensor networks. Wireless sensor networks (WSNs) consists of spatially distributed mobile or stationary sensor nodes to monitor physical or environmental conditions. Moreover, wireless sensors cooperate to pass data through network to base stations. In WSNs, a key constraint for sensors is energy. Because each sensor has limited battery capacity [7], different energy management frameworks have been introduced. One of the strategies is to schedule sensor's active time slots [8]. While designing wake/sleep scheduling algorithms in WSNs, we can incorporate periodicity in the design of scheduling by adopting the store-and-forward in DTNs. As a result, instead of having a connected sensor network at all the time, sensor networks could be disconnected. If data acquisition consumes significantly less energy than transmission, queries can be disseminated and collected by scheduling a set of sequential connected sensors between data sources and sinks.

Other than the famous ZebraNet [143], WSNs also have great impacts on measuring and monitoring dynamic changes in agriculture [133]. Wireless technologies have been introduced into agriculture to improve productivity and provide long term monitoring to prevent harmful events. For example, several cattle monitoring systems have been proposed in recent years with coverages from local farms to national wide control [111, 112, 136, 137]. Most studies just simply adopted routing protocols from MANET without considering the constraint of energy consumption in the network. Because the monitored animals often roam around separately in physical clusters that are disconnected from a central backbone infrastructure, naively adopting routing protocols from MANET could reduce the life-time of sensors, and it is very expensive and difficult to replace sensors if there are hundreds or thousands of animals in the system. It has shown that animals have relatively invariable mobility patterns because of their habits [15]. For example, cattle have predefined schedules for feeding, milking, exercising and so on, and wide-life such as zebras know where to feed and where to stay during the night. Understanding their movement patterns, specially periodic patterns, could help set up proper networks and design efficient routing protocols to disseminate and collect data from those mobile nodes.

In addition, other projects such as the European Union's Haggle Project [2], SARAH project [5] and ResiliNets [61] have also been proposed to utilize opportunistic communications between mobile nodes. We believe exploiting periodic behaviours within mobile data could help to design delay tolerant distributed services, resilient and survivable networks.

In summary, we discussed two major applications where periodic movement patterns can be applied. In those applications, mobile nodes exhibit strong periodic movement patterns. However, a total lack of knowledge about underlying periodicity limits the usage of stable periodic encounters where existing solutions simply adopt routing protocols from MANETs and DTNs that target predicting future encounters. In addition, some applications in MANETs and DTMNs, and VANETs particularly, need the assistance from stationary base-stations or roadside-units to enhance connectivity in the network. Sophisticated analyses aimed at different metrics are conducted for optimal deployment without considering the diversity in the network, so that if a critical base-station or roadside-unit failed, connectivity in the network could be significantly degraded.

Our studies are not limited by those applications. Utilizing periodic patterns is suitable to any application with the involvement of people, animal and schedules. Because mobile nodes must exhibit some types of movement patterns, as long as there is periodicity within mobile data, our studies in this thesis could benefit communication services.

1.3.2 Opportunities and challenges in periodicity detection and utilization

As we have discussed in Section 1.2, both mobility-aware and social-aware routing suffer from unstable predicted encounters with lack of routing information such as encounter times. Protocols using probabilities or social relationships alone are not enough to guarantee good performance. For example, even if two nodes are not highly socially connected or have a low encounter probability in the past, they might still have a promising encounter during a certain periodic time window. As an alternative solution, we propose to use regular and periodic mobile behaviours from node mobility to address communications in DTMNs. We believe the periodicity within node mobility is promising to make forwarding decisions because both the time and the period length of periodic behaviours fill the gap within previous works. In addition, all periodic nodes are treated equally where a fair routing can be managed.

Fundamental challenges

There are two fundamental challenges in order to take advantage of periodic patterns of node movements. First, we need to detect and extract periodic patterns from node mobility. Secondly, we need to model and exploit the obtained periodic behaviour.

- **Detecting and extracting periodic behaviours** Studies have been done to understand characteristics of different network environments and user groups, but it is difficult to generalize these findings to other applications. For instance, in [74], the periodic property of node mobility was introduced to reveal periods within node mobility and periods of association patterns from access points. There are also other studies detecting periodicity with different statistics. For example, the re-appearance probabilities of nodes to specific locations [56, 58, 118] and the probabilities of users returning to specific locations after a certain amount of time [44]. By examining repetitive patterns of users and encounters, previous work has observed periodic behaviour in real mobility traces. However, finding routes in mobile networks needs more than that. We need to find pairs of objects that encounter each other periodically. In addition, besides encounters between two objects, we need to know which encounters are periodic, and which ones are not (since they may not be reliably used). Finally, there are two important pieces of information from a periodic encounter pattern: the period length and the promising encounter phase(s). The length defines the duration of the pattern. The phase indicates the time(s) within the period where periodic encounters occur. Because a pair of nodes may have several events where their encounters for each event are periodic, we should be able to exploit them all. For example, we may observe a frequent weekly encounter on Mondays and a less frequent bi-weekly encounter on Thursdays. If we have to choose only one of these, we may degrade routing performance for some source-destination pairs.

- **Modelling periodicity** Once we obtain the periodic patterns, the next step is to define a formal model that is flexible enough to reflect different assumptions and goals. In conventional logical topologies graphs are usually applied to model network connectivities at a particular time. Within a graph, an edge between two vertices means that two mobile objects can directly communicate with each other, *i.e.*, an encounter occurs. If objects in the networks are mobile, the graph has to change whenever there is a change in the physical topology. As we have discussed before, periodic patterns contain both the period length and the periodic encounter phase(s). Within a periodic pattern there may be multiple periodic encounters at different times. Integrating encounters from multiple time instances is impractical because we will lose track of the phases of periodic encounters. In addition, representing repetitive encounters in the model becomes even more challenging.

- **Relay node deployment** In DTMNs where the network is partitioned into completely disconnected sub-networks, deploying stationary relay nodes is one of the approaches that enhance the connectivity by increasing encounter opportunities. In previous solutions, the problem of deployment is treated as different optimization problems such as the smallest connected dominating set. As a result, to satisfy different measurement metrics such as minimizing the number of relays in the network, there may be only one path between most of the node pairs. Because mobile networks can be very dynamic, if a mobile node or a relay that is used to join disconnected sub-networks fails, the whole network would fall apart and become disconnected again. Therefore, increasing route diversity, *i.e.*, the existence of multiple disjoint paths in the network, is also very important with respect to relay deployment. How to properly model route diversity in terms of mobile node failure, relay node failure and link failure is very challenging.

Open challenges

As extensions to the fundamental problems, exploiting regularity and periodicity within node mobility opens up new research frontiers in various aspects such as (1) designing novel routing techniques that adopt periodic encounters, (2) disseminating and collecting queries in delay tolerant environments, (3) integrating mobile nodes from different network types and (4) preserving security and privacy in DTMNs.

Routing protocols In our proposal of modelling periodic patterns a centralized graph model is built by integrating all periodic encounter patterns. Therefore, optimal algorithms such as Dijkstra's can be applied to find paths with the minimum weight for end-to-end communications. In addition, the minimum weighted Steiner Tree can be applied for mutlicast routing with the minimum cost. Theoretically, communication can be routed through those pre-calculated routes. However, in distributed environments each mobile node has to discover periodic encounters online, and can only obtain local information about its one-hop neighbours. Routing protocols with distributed solutions are required with cost-efficient methods to utilize local information. In the discussion of mobility-aware protocols, there are several problems about exchanging and updating local information. For example, updating local routing information implies control overhead, which wastes network bandwidth. In addition, the inconsistency of local information is also a problem. For example, because of diverse encounter times, the global sharing of local information is not consistently distributed through the network. As a result, the optimal route to a destination computed from one node may be different from the one from other nodes. In this situation, the global optimization is not always guaranteed. How to balance or minimize those constraints within the routing protocol becomes very challenging.

In addition to designing protocols with distributed solutions, protocols should also be robust with respect to random and infrequent interruptions of periodic encounters. The performance of routing under the circumstances of losing stable periodic partners highly depends on the sensitivity of detection techniques to discover abnormal behaviours. Slow reaction of routing protocols with respect to periodicity changes may lead to poor performance with long delays and high costs of network resources. Therefore, designing simple and robust periodicity detection techniques is also critical in the design of routing protocols.

Querying Querying is also an interesting problem in DTMNs. Of course, query dissemination and result collection rely on different routing protocols depending on query types. However, simply applying routing protocols for query processing in DTMNs may not perform very well. In mobile ad-hoc networks where networks are always assumed to be connected, queries are disseminated by forming a routing tree originated at the source node called root [42]. A query is distributed along the tree from the source to destinations that are leaf nodes in the tree. Once results are ready, they are sent back to the source in a bottom-up fashion where query aggregating or filtering such as finding the MAX or MIN value is performed by intermediate nodes on the tree. Selecting a proper routing tree has a non-trivial impact on effectiveness and efficiency of query dissemination and result collection. Unfortunately, such strategy does not work in DTMNs because of 1) lack of connectivity in the network and 2) time-varying encounters. Lack of connectivity in DTMNs means a routing tree may not exist in the network. Even though a routing tree may be found for query distribution, this tree may disappear or its topology may change over time. If the topology for routing tree has changed for result collection, pre-scheduled aggregat-

ing or filtering at intermediate nodes in the old tree will fail.
In the domain of periodic encounters in DTMNs, routing trees can be established on top of periodic encounters. However, unlike sharing one tree for both dissemination and collection, two trees may be required because of time-varying encounters. An encounter may not be there when a result needs to be sent back from a node to its parent in the tree. Therefore, one tree is used to distribute queries to all destinations where the second tree is used to collect results, where results are aggregated and filtered at the intermediate nodes in the second tree.

Mobile nodes and networks integration In our studies, we have found that typically only a small portion of nodes exhibit strong periodic mobile behaviours. This is because interruptions or unexpected events could affect periodic encounter series so that their periodicities are too weak to be detected. Our studies so far only focus on periodic nodes of entire node population. For aperiodic nodes, we lose the benefit of encounters that can be used to compute routes for routing. In DTMNs integrating both periodic and aperiodic mobile objects is vital. In addition, there are various network types co-existing in real life. For example, cellular networks, wireless local area networks and networks based on Bluetooth encounters such as Pocket Switched Networks. Mobile objects from these networks could exhibit periodic behaviours, where they should not be isolated from each other. Otherwise, mobile objects from different applications could suffer from system constraints where communications are limited within separated networks. New frameworks should coordinate requirements and cooperate mobile objects from diverse DTMNs environments where communications cross platforms (networks) can be benefited. However, they (each network type) should not be treated in isolation. Rather, the possibility of cooperation between different networks should be explored. How to integrate mobiles objects from different networks and routing among objects in large-scale networks as well as non-mobile networks requires more investigation.

Security and Privacy Delay Tolerant Networks (DTNs) have been studied extensively for almost a decade. However, not much research has been done to address the security of data transmission and protect the privacy of mobile objects who are willing to participate routing processes. Because DTMNs are just general scenarios of DTNs, they also face all security issues that target DTNs. In particular, authentication cannot guarantee environments free of malicious nodes because it is difficult to only deliver private keys to non-malicious objects in mobile environments. Once malicious objects are inserted, messages such as queries and answers for queries could be captured on the fly or even be replaced by malicious code. How to guarantee data integrity and protect node privacy and confidentiality become even bigger problems. In the context of routing with periodic patterns, inserting a malicious node into networks is relatively easy. Without authentication, if the movement of malicious objects is pre-planned to be periodic, or even if stationary malicious objects are placed in the network, non-malicious objects encountering them regularly could consider malicious objects as stable periodic one-hop neighbours. Therefore, how to prevent intruders, protect privacy and avoid attacks are important issues to preserve secure and reliable services while still benefiting from periodicity in DTMNs.

1.4 Thesis contributions

Utilizing encounters within mobile networks from different perspectives can improve the performance of routing protocols in DTMNs. Instead of predicting future encounters, we concentrate on exploiting stable periodic encounter behaviours within mobile data. In this thesis, we describe a series of studies: (1) finding and extracting periodic encounter patterns from mobile data, (2) modelling periodic encounter patterns as a graph where routing problems can be addressed as optimization problems and (3) deploying stationary relay nodes in DTMNs to connect disconnected subnetworks. In summary, this thesis makes the following contributions.

Finding periodic encounter patterns within mobile data

Our study is the first research to propose a method for extracting the specific encounter patterns (both period and phase) for pairs of nodes that meet each other periodically. Other work has stopped short of this, only proposing methods for detecting node pairs that have periodic encounters. We conduct experiments on different network types including Bluetooth, cellular and wireless local area networks whereas previous studies only concentrated on wireless local area networks. We also investigate the persistence of detected periodic behaviours, *i.e.*, how long they reliably project themselves in the future. Furthermore, the small-world structure of networks formed by periodic encounters is examined. Our results show that the majority of networks have a small-world structure where messages between any pair of nodes can be delivered through a very small number of hops in the network.

A graph model for periodic encounter patterns

To take advantage of periodic encounter behaviours, we introduce a graph model which integrates all encounter patterns. We show that both the minimum delay and minimum energy consumption problems can be modelled as optimization problems. The only change between the graphs used to solve either problem is in the assignment of weights for the edges. The minimum delay problem can be modelled as the shortest path tree problem whereas the minimum energy consumption problem can be modelled as the minimum weighted Steiner Tree problem. With a modification, we can use Dijkstra's shortest path algorithm [29] to find an optimal solution for the minimum delay problem. However, because the minimum weighted Steiner Tree problem is NP-hard, we propose a polynomial time k-approximation algorithm, where k is the number of destinations, to find a multicast tree for routing. Experimental results, using both real and synthetic datasets, show that our proposed approaches always find better routes with regard to delivery delay and energy cost compared to other state-of-art protocols in DTNs [34].

Deploying relay nodes to connect isolated subnetworks

Our studies showed that even mobile nodes with periodic behaviours can still be partitioned into disconnected components. To address this disconnection problem, we propose deploying relay nodes in the network to connect disconnected components. In this thesis, we first present an extension of our proposed graph model that can capture encounters in different mobility environments. We also show that relay node deployment problem can be modelled as various k-connectivity problems with the objective to build fault-tolerant networks: the k-edge connectivity problem models the encounter failure problem; the k-vertex connectivity problem captures the node failure problem; and the k-element connectivity problems examines the relay node failure problem. In our study, we concentrate on the k-element connectivity problem where we want to minimize the number of relay nodes in the network such that the network is k-element connected. Finally, we present an integer programming formulation to solve the problem. In addition, three polynomial time heuristic algorithms are introduced to find solutions to k-element connectivity. Our simulation results show significant improvement in network performance regarding the delivery ratio and delivery delay by forming k-element connected networks.

1.5 Thesis organization

The rest of this thesis is organized as follows. In Chapter 2, we introduce our methodology to detect and extract periodic behaviours from mobile data. To characterize the performance, we examine our method against both synthetic data and real-world traces. In addition, we use real-world traces to explore periodic encounter behaviours and their properties. Chapter 3 presents our graph models regarding both unicast and multicast routing. Mathematical optimization and optimal algorithms that capture the semantics of the discovered encounter patterns are also discussed. Experiments against both synthetic data and real-world traces are conducted to examine the performance of our algorithms in the graph models with respect to different problems. In Chapter 4, we discuss the relay node deployment problem in order to obtain k-element connected networks. To address the problem, we first present an integer programming formulation of the problem. In addition, three polynomial time heuristic algorithms are used to solve the problem. Experiments and simulations are conducted to demonstrate the performance of our heuristic algorithms and deployment solutions. Finally, we conclude this thesis in Chapter 5 with a discussion of future research directions.

Chapter 2

Finding periodic encounter patterns within mobile data

2.1 Introduction

Mobile nodes have been observed to exhibit periodic behaviours [44, 68, 95]. They tend to have a relatively stable sequence of activities because of their social or controlled behaviours. For example, people often use the same routes daily to commute to and from work. Farm animals and animals in the wild also exhibit predictable mobility behaviours in terms of paths followed, places they rest, feed, etc. One result of these repeated activities is that each mobile node may encounter one or more other nodes periodically. If we can detect and extract these periodic encounters, we can know when and which pairs of nodes are going to encounter each other. Therefore, periodic encounters can be used to find and schedule routes for communication. In order to utilize periodic encounter patterns, we first need to detect and extract them from node mobility. In this chapter we propose a methodology to find periodic behaviour within real mobility traces including MIT [32], USC [57], Dartmouth [76] and Nokia-MDC [80] traces and examine connectivities among mobile nodes with periodic behaviour. Our studies in this chapter have the following contributions:

1. We propose a method for extracting the specific encounter patterns (both period and phase) for pairs of nodes that meet each other periodically. Previous solutions have stopped short of this, only proposing methods for detecting node pairs that have periodic encounters.

2. We conduct experiments on different network types including Bluetooth, cellular and wireless local area networks whereas previous studies only concentrated on wireless local area networks.

3. We also investigate the persistence of detected periodic behaviours, *i.e.*, how long they reliably project themselves in the future, where similar work has never been presented before.

4. Furthermore, we examine the small-world structure of networks formed by mobile nodes with periodic encounter behaviours.

The remainder of this chapter is organized as follows. An overview of existing routing protocols in DTN is discussed in Section 2.2. In Section 2.3, we introduce our method to detect and extract periodic behaviours from traces. Section 2.4 presents two sets of experiments that use synthetic traces and real mobility traces, respectively. Finally, future work and a summary are given in Section 2.6 and Section 2.7.

2.2 Periodicity detection

In previous studies, analyses of human social network connectivity have been conducted to demonstrate delivery between different human communities using their social relationships [14, 27, 52, 59, 60]. However, routing with a social network does not offer control over the time delay of delivery [130]. We know that people will encounter each other if they are socially connected. However, we do not know when encounters will take place. Therefore, mobility traces have also been studied to understand user mobility. Many researchers have studied movement patterns. With respect to inter-contact times and reachability of nodes, it has been shown that mobility and pair-wise encounters can benefit communications [20, 83, 125, 144]. With respect to understanding periodicity within node mobility, several authors recently have reported experiments to detect periodic behaviours [56, 95].

One study looks at the association of wireless devices with wireless access points (APs) [56]. Device associations were captured every minute in a real network. These associations were used to calculate a *location similarity index*, which is the percentage of observations for which a given device is associated with the same AP. The *network similarity index* was then calculated as the average of the location similarity index values for all devices for a given time lag between successive observations. High values of the network similarity index were observed close to a lag of 1 day. This indicates that nodes have daily periodic behaviours. In addition, the second highest index was observed at a lag of approximately 7 days. This suggests a weekly periodic behaviour.

Another study reports a spectral analysis of mobility traces to quantify the regularity and periodicity of nodal encounters [95]. Real user traces were converted into time series of nodal encounters. Each trace was split up into consecutive intervals based on a given granularity (*e.g.*, 1 hour or 1 day). A binary time series was then created for each unique pair of nodes. The length of each series equals the number of intervals in the trace at the given granularity. During any interval, if there was an encounter between a pair of nodes, a value of 1 was recorded. Otherwise, a 0 was recorded. The auto-correlation function was then calculated to find periodic encounters at each lag [22]. Then, the discrete Fast Fourier Transform was applied to the auto-correlation function to convert it from time domain to frequency domain. Analysis in the frequency domain was quite elementary, and consisted of looking for the frequency component with the largest amplitude. However, as the authors stated, the results from using the Discrete Fourier Transform (DFT) include an artifact that affects the accuracy of period detection. For example, the most significant period returned from DFT could be 7 days and 10 hours whereas the true underlying periodicity is 7 days. In our method, we examine the correlations between encounters at different times. If there is an underlying periodic encounter behaviour within the time series, our method guarantees an integer period instead of a fractional result.

In addition to spectral analysis, mining periodic events is a well-known problem in data mining. Previous studies concentrated on finding "exact periodicity" and "partial periodicity" in transactions with the characteristic that events are perfectly periodically aligned [18, 48, 89, 102, 140]. However, such a characteristic does not hold for real-user mobility traces. In human routines, an event may occur on a regular basis; however, it is unlikely to always occur at exactly the same time. To discover periodicity in human routines, a new method was proposed to find the minimum routine period of events that have some periodicity but do not always occur at the exactly the same time [10]. In that paper, given sequences of events with the same type plus their occurrence times, the authors proposed a method with two steps: (1) first, an algorithm is used to find a set of candidate period lengths, and (2) second, given a set of candidate period lengths returned by the first step, another algorithm using a sliding window is proposed to find the minimum period length of the routine in the input sequence, if there is one. In summary, the method is based on the use of a sliding window. Instead of examining every period length that is less than the input sequence length, the authors used pruning to filter out the period lengths that are not valid based on user-specified parameters. For a candidate period l that has been found, a sliding window is used to examine each consecutive interval with length l in the original sequence for aligned routines. A sequence has an aligned routing with period l if and only if the number of adjacent intervals containing exactly one occurrence of the input event satisfies the user-specified constrains. Because the method proposed in [10] is based on a sliding window, it depends on finding adjacent intervals containing exactly one occurrence of an event. Noise such as extra events within those adjacent intervals affect the performance of proposed method. As a result, some event sequences with periodic behaviour will be overlooked. In contrast, our objective is to reliably find underlying periodicity that may be buried under noise.

From previous studies, we learned that mobile nodes with regular encounters can be identified with sliding windows, autocorrelation and spectral analysis. However, finding a route in a mobile network needs more than that. We can find pairs of nodes that encounter each other periodically. Just as in a social network, messages can be routed via such pairs of nodes. However, we may lose control over delivery delay [130] without knowledge of when the encounters will take place. In addition, among encounters between two nodes, we need to know which encounter is a periodic encounter, and which encounter is just an occasional or even random encounter. A route will likely fail if an occasional or random encounter is selected. Also, in previous work, a pair of nodes can only have encounters of one period. For example, they are allowed to have either a daily encounter, or a weekly encounter – but not both. If there are multiple encounter events between a pair of nodes, we should be able to exploit them all. For example, we may observe a frequent weekly encounter on Mondays and a less frequent bi-weekly encounter on Thursdays. If we have to choose only one of these, we may degrade routing performance for some source-destination pairs.

In traces with periodic encounter behaviours, encounters occur on a regular basis; however, they may not be perfectly periodic. To address this problem, we use *granularity* to wrap encounters within consecutive intervals whose length equals the granularity. This serves the same purpose as the "envelope" introduced in [10]. The challenge in our studies is to differentiate periodic encounters from random/occasional encounters. In addition, missing periodic encounters are another challenge. If a trace is perfectly periodic, methods using sliding windows or the Fast Fourier Transform are capable of detecting underlying periodicity. However, noise and missing encounters can interfere with the Fast Fourier Transform, leading to incorrect results. With sliding windows, noise encounters affect the detected intervals where some of them may contain multiple occurrences of the same event. As a result, those intervals are not valid, and some traces with periodic encounter behaviours will be overlooked.

In our work, in addition to detecting node pairs with periodic encounters, we propose a method to extract their pattern(s) of encounters. From a periodic pattern, two important pieces of information can be found: (1) period length and (2) promising encounter phases. For example, if a pair of nodes exhibits a weekly encounter pattern, our method can identify the weekdays that encounter takes place, *e.g.*, Monday and Thursday. It is possible that a trace contains multiple periodic encounter events between two nodes. Other work has stopped short of this, only detecting one period. Our method finds the least-common multiple at which encounters occur. For example, if encounters occur every two days and every five days, our method would highlight an encounter pattern every ten days. The resulting 10-day pattern will include both 2-day and 5-day patterns because both patterns synchronize every 10 days.

2.3 Proposed methodology

In this section, we introduce the methodology used to find periodic patterns. First, we define the terminology and model mobility traces. Second, we present our method to analyze time series and produce periodic encounter patterns.

Definition 1. *An encounter is defined as a time-period where wireless devices are physically close enough to communicate with each other.*

A mobility trace has a duration that is the monitoring time of mobile movement. Given a granularity, *e.g.*, 1 hour or 1 day, this duration can be broken into consecutive intervals whose duration is equal to the granularity. We create an encounter series for every unique pair of nodes where the length of the series equals the number of intervals after partitioning the trace at the chosen granularity. If a pair of nodes has an encounter during a specific interval, we record a 1 in that interval, otherwise, a 0 is recorded.

Definition 2. *Given a monitoring duration σ and a granularity τ, an encounter series for a unique pair of nodes (x,y) is defined as:*

$$S_{x,y} = \{d_1, d_2, d_3, \cdots, d_n\} \textit{ where}$$

$$n = \lfloor \frac{\sigma}{\tau} \rfloor \textit{ and}$$

$$d_j = \begin{cases} 1 & \textit{if } x \textit{ and } y \textit{ encounter at interval } j \in [1,n] \\ 0 & \textit{otherwise} \end{cases}$$

Obviously, an encounter series is just a binary time series. All series derived from the same mobility trace have the same length because all nodes are being monitored over the same period of time.

(a) A perfect encounter series

1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0

(b) An imperfect encounter series

1	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0

Figure 2.1: Encounter series

Fig. 2.1 presents two periodic encounter patterns. Assuming the granularity of observation is one day, each series models a total of 21 days. The first one is perfectly periodic, while the second one is imperfect. The perfect encounter series shows two nodes encountering each other every Monday and Friday, assuming the series starts on a Monday, whereas the imperfect encounter series also has periodic encounters on every Friday with noise and missing periodic encounters.

The general process of our method is shown in Figure 2.2. In this section, we will focus on the core component of the process. We use the auto-persistence function, defined below, to study the encounter series for each pair of nodes in the mobility trace. We normalize the auto-persistence values by applying a filter to remove noise from occasional encounters and missed periodic encounters. By analysing the result, we decide whether the encounter series for a particular pair has periodic encounter patterns. We repeat the same process for each node pair. If an encounter series has periodic encounter behaviours, its encounter pattern is extracted by pattern recognition. In this section, we introduce three sub-steps of the proposed method: (1) applying auto-persistence function and, (2) pre-processing the results and (3) recognizing periodic encounter patterns.

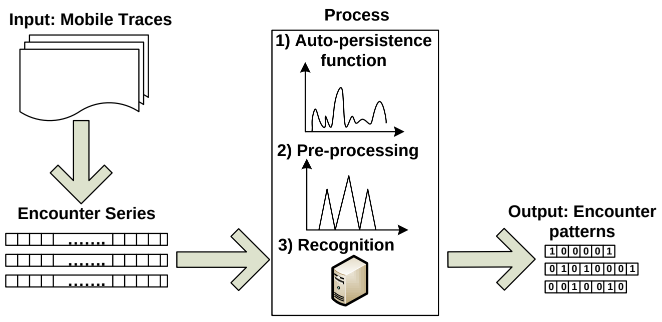


Figure 2.2: General procedure of our proposed method

2.3.1 Auto-persistence function

Previous work used the auto-correlation function, which examines encounters as a function of time between them [95]. As other works have suggested [124, 131], while the auto-correlation function is good for real-valued series, it provides less statistical information for binary time series.

The auto-persistence function is calculated from the conditional probabilities of the different combinations of two values in a binary time series that are separated by a given lag k, that is k positions away in encounter series. There are, of course, four combinations of two binary values. We concentrate on the combination where both intervals k-lags away from each other have encounters. We define the auto-persistence function (APF) as:

$$APF(k) = P(d_{t+k} = 1 | d_t = 1) \tag{2.1}$$

where k is the lag between intervals.

As an empirical counterpart to the auto-persistence function, the auto-persistence graph (APG) is defined as:

$$APG(k) = \frac{n}{n-k} \frac{\sum_{t=1}^{n-k-1} I\{d_{t+k} = 1, d_t = 1\}}{\sum_{t=1}^{n} I\{d_t = 1\}} \tag{2.2}$$

where n is the size of encounter series, k is the lag, and I is the binary indication function, whose value equals 1 if and only if the condition is satisfied.

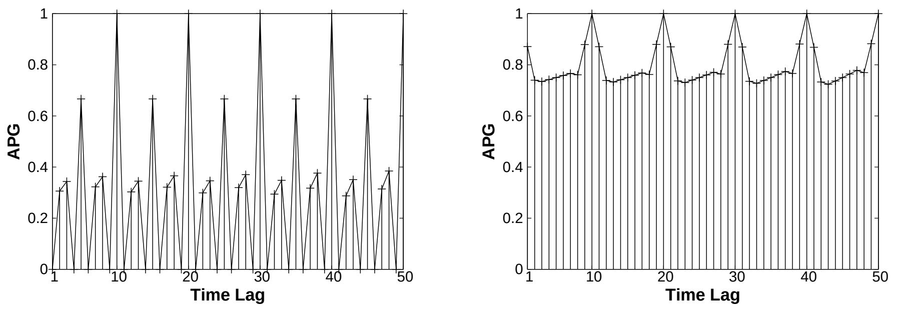
Similar to the auto-correlation function, as the lag increases, the correlation between the current interval and the interval that is k lags ahead usually decreases. However, in testing for an encounter of period k, an encounter in a given interval will be highly correlated with other encounters k intervals away. As the lag, k, increases, patterns with longer periods are being tested. Therefore, to compensate for the decreasing number of intervals at greater lags within a series of fixed length, the scalar $n/(n-k)$ is introduced in the APG.

In the APG, different encounter rates have different behaviours where the encounter rate of an encounter series is defined in Definition 3.

Definition 3. *The encounter rate is the fraction of intervals within the encounter series containing encounters:*

$$R_{x,y} = \frac{\sum_{j=1}^{n} d_j}{n}$$

As an example, the APGs for two perfect encounter series, each with a time series of length 100, are shown in Figure 2.3. The first encounter series has an encounter pattern with period 10, with 3 phases containing regular encounters. The second series has an encounter pattern with the same period length of 10, but with 8 encounter phases. As a result, the first encounter series has a lower encounter rate of 30% while the second has an encounter rate



(a) Encounters at 3 phases (b) Encounters at 8 phases

Figure 2.3: APG graphs for two series with encounters of period 10

of 80%. The figure shows that both series have periodic behaviours because of the wave representation. Waves appear at multiples of the minimum lag of 10 in both examples. The repetition in the APGs comes from the underlying periodicity in the encounter series. If an encounter series has an encounter pattern with length m, then this encounter series also has encounter patterns with length $2*m$, $3*m$ and so on. An encounter pattern is defined as:

Definition 4. *Given an encounter series $S_{x,y}$ and a user-defined threshold $T \in [0,1]$, if an encounter pattern exists in $S_{x,y}$, this encounter pattern is defined as:*

$$P_{x,y}^{m} = p_1, p_2, \cdots, p_m \textit{ where}$$

$$m < \lfloor \frac{n}{4} \rfloor \textit{ and}$$

$$p_j = \begin{cases} 1 & \textit{iff } \sum_{i}^{\lfloor \frac{n}{m} \rfloor} d_{i*m+j} \geq T * \lfloor \frac{n}{m} \rfloor \textit{ for } j \in [1,m],\ i \in [0, \lfloor \frac{n}{m} \rfloor - 1] \\ 0 & \textit{otherwise} \end{cases}$$

This encounter pattern repeats itself with a period of m within $S_{x,y}$. There are two important parts to an encounter pattern: the pattern period and the encounter phase(s). The period defines the duration of the pattern. The phase indicates the interval(s) within the period where periodic encounters occur. In the original encounter series, $d_j = 0$ means there is no encounter between two nodes during this interval, and $d_j = 1$ means there is an encounter. However, in the encounter pattern, a $p_j = 1$ means there is a *regular periodic* encounter, whose probability is above the user-defined threshold, and has been observed between the two nodes at this phase in the pattern, while $p_j = 0$ means the opposite. To successfully identify a periodic encounter pattern, we need to observe at least two repetitions of the encounter patterns at the corresponding time lags m and $2*m$. Therefore, the length of the input series must be greater than $4*m$, and this is the reason that m has the constraint $m < \lfloor \frac{n}{4} \rfloor$ in Definition 4. Furthermore, when an encounter appears at the same phase during all observed periods with a frequency greater than the threshold that is $T * \lfloor \frac{n}{m} \rfloor$, we consider this encounter a periodic encounter. This explains the constraint for p_j in the definition. Figure 2.4 presents encounter patterns corresponding to the encounter series in Figure 2.1.

(a)Encounter pattern for the perfect series

1	0	0	0	1	0	0

(b) Encounter pattern for the imperfect series

0	0	0	0	1	0	0

Figure 2.4: Encounter patterns

The encounter *pattern* for the series in Figure 2.1(a) is shown in Figure 2.4(a). Even through the imperfect encounter series has occasional aperiodic encounters, and is missing a regular encounter on Monday in the third week, this does not mean that this series has no periodic encounters. It still has the encounter pattern as shown in Figure 2.4(b) where a periodic encounter happens every Thursday. In both sample encounter series, the user-defined threshold is 100%, T=100%. However, if $T = 60\%$ for periodic encounters, meaning that encounters appearing 60% of time at a certain phase can be considered to be periodic, the imperfect encounter series has another valid pattern shown in Figure 2.4(a) with at least a 60% matching probability in its phases.

In addition, it is possible that an encounter series has events at multiple periods. This leads to multiple encounter patterns for the series. For example, the APG shown in Figure 2.5 arises from an encounter series with periodic encounters at two phases. One pattern has a period of 4, and the other has a period of 5. Spikes occur in the APG at lags that are multiples of four, and at lags that are multiples of 5. One interesting observation is that the patterns overlap at lags that are multiples of both period lengths, in this example at lags 20 and 40. The overlapping of encounter patterns produces a fundamental encounter pattern $P_{x,y}^{m_u}$. Instead of finding individual encounter patterns, selecting the maximum value in the APG identifies this fundamental encounter pattern for a series with multiple encounter events.

Lemma 1. *If an encounter series $S_{x,y}$ has multiple encounter patterns $P_{x,y}^{m_1}, P_{x,y}^{m_2}, \cdots, P_{x,y}^{m_j}$ where no encounter length is the multiple of the minimum encounter length, then series $S_{x,y}$ has a fundamental encounter pattern $P_{x,y}^{m_u}$ where m_u equals the least common multiple of all individual period lengths $m_1, m_2, \cdots, m_j$. The wave representation in the APG identifies this fundamental encounter pattern.*

Proof. Given such a series, let us assume this series is composed of several perfect periodic

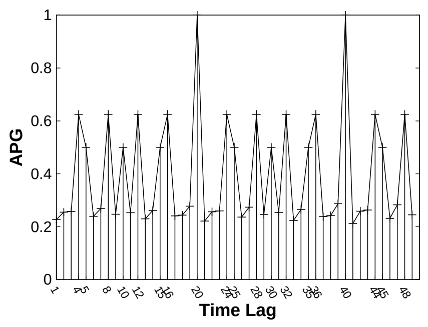


Figure 2.5: A series with two periodic encounter patterns

encounter series whose period lengths are m_1, m_2 and up to m_j. In addition, let us assume, all encounter lengths are not multiples of each other. An example of APGs for such a series is shown in Figure 2.5.

When the time lag equals the period of each individual encounter pattern such as $lag = m_1$, $lag = m_2$ or $lag = m_j$, their periodic encounters match. Therefore, the conditional probabilities for the individual encounter patterns reach their maximum values. When the time lag equals m_u, the least common multiple, every individual encounter pattern has its periodic encounter aligned on the encounter series. At this lag,

$$\begin{aligned} APG(m_u) &> APG(m_1) \\ APG(m_u) &> APG(m_2) \\ &\cdots \\ APG(m_u) &> APG(m_j) \end{aligned}$$

Each inequality holds because at $lag = m_u$ all periodic encounter patterns contribute to the conditional probability at that lag. As a result, the maximum values in the APG are represented by the fundamental lag and its multiples. Thus, according to the wave representation of periodicity, the encounter pattern with the period of the fundamental lag will have the maximum value. $\square$

As suggested by Lemma 1, if there are multiple encounter events within a series, our detection technique will return the fundamental encounter pattern, $P_{x,y}^{m_u}$.

2.3.2 Pre-processing

The APG is the tool we use to detect periodicity within each encounter series. With APGs such as those in Figure 2.3, we can easily find the top-k lags within an encounter series and then check whether these lags are multiples of each other. This approach works for a perfect encounter series. However, occasional or random encounters will add noise to our encounter series, and mask periodicity to a greater or lesser extent. The APG for a noisy series still exhibits periodic behaviour but requires extra processing, which we call pre-processing.

We start the discussion by presenting an example of a real encounter series. Figure 2.6 shows the APG for an encounter series derived from real mobility traces with a 1 hour granularity. We choose a 1 hour granularity because this scale of APG is easy to demonstrate. This figure exhibits a strong periodic behaviour represented by the dashed line. Three spikes appear at lags 168, 336 and 504. These lags represent exactly 7 days, 14 days and 21 days.

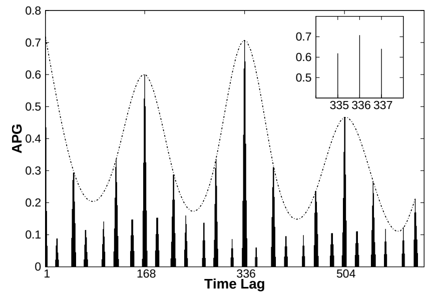


Figure 2.6: APG for a real trace [57]

This example shows that, unlike perfect encounter series, the periodicity in a noisy series is implicitly embedded within the APG spikes. The inset in Figure 2.6 shows the APG coefficients immediately adjacent to lag 336. In this example, the top three lags are at 335, 336 and 337 rather than at 168, 336 and 504. Therefore, just selecting the top-k spikes in the APG will not identify the top-k periods. We propose two steps to pre-process the APG: (1) filtering and (2) spike simplification.

Filtering One problem with the auto-persistence function is that there is still a low conditional probability for two encounters separated by some arbitrary length of time, even when those encounters are not periodic. Small conditional probabilities like these cause the small spikes in Figure 2.6. We propose to filter out these spikes as follows.

Definition 5. *A filter is a boundary chosen by considering the overall property of*

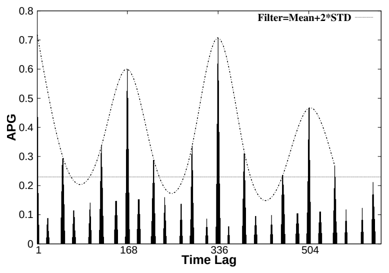


Figure 2.7: APG filtering

APG values to remove insignificant values. It is defined as:

$$F = \overline{m} + K * \sigma$$

where $\overline{m}$ is the estimated mean of the APG values, σ is the estimated standard deviation and K is a constant.

We assume the APG coefficients at different lags are normally distributed, and set $K = 2$ to filter out all but approximately the top 2% of the spikes. Figure 2.7 shows the filter value as a horizontal line. Compared to Figure 2.6, spikes above the filter value are more significant. This makes the APG much less sensitive to spikes from aperiodic behaviours.

Spike simplification The second step is to concentrate spikes. Figure 2.6 and 2.7 show that each spike is composed of a group of APG values from consecutive lags. To have a better representation of the spike, in this example we merge each spike into a single lag by using the highest value in the range to represent the whole spike. Figure 2.8 is the simplified APG of Figure 2.7.

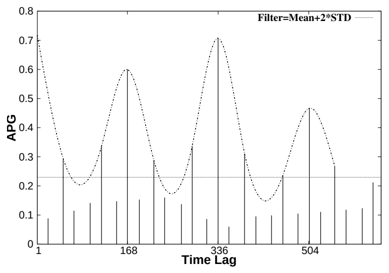


Figure 2.8: APG simplification

With these two steps, we have removed irrelevant spikes and simplified the APG. From this simplified APG, if an encounter series has periodic encounter behaviours, their representation can be found. We now look for spikes with maximum values, where a spike is a lag with smaller preceding and following APG values. For example, there are three spikes in Figure 2.8, at lags 168, 336 and 504. We conclude that this encounter series exhibits a weekly periodic encounter because the other two lags are multiples of the fundamental, 168.

2.3.3 Periodic pattern recognition

From the APG after pre-processing, we are able to declare whether an encounter series has a periodic encounter behaviour or not. In addition, we can identify the period of the behaviour. Once the period is known, we can extract the encounter pattern from the original encounter series using alignment, where each encounter series is split into several disjoint

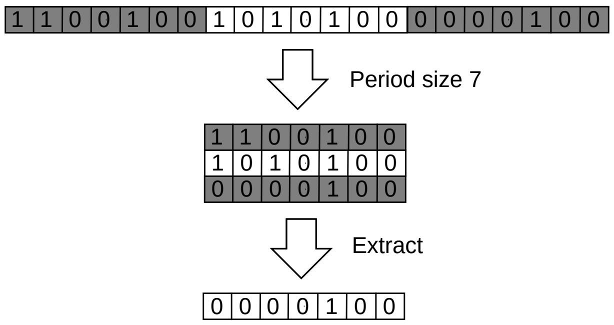


Figure 2.9: Pattern recognition

segments, each of whose length equals the period previously detected. We note that, in general, multiple encounter behaviours with different periods could appear in a lengthy segment; thus, we use the term *period* to refer specifically to the interval over which a behaviour repeats (irrespective of phase), and *segment* to refer to a fixed-length portion of a trace. After breaking the trace into segments, we stack the segments vertically as shown in Figure 2.9. In each vertical phase, we examine the probability that encounters appear. Given the length of the input series, n, and the period length, m, there would be $\lfloor \frac{n}{m} \rfloor$ complete segments. We consider an encounter a periodic encounter if it has a frequency greater than the threshold, $T * \lfloor \frac{n}{m} \rfloor$, at the same phase during all observed periods. As an example, given the input series $S_{x,y}$ and $T = 100\%$ in Figure 2.9, the periodic encounter pattern is $P^7_{x,y} = \{0,0,0,0,1,0,0\}$.

In the detection process, an encounter is considered to be noise if an encounter appears at different phases. For example, let us assume we have the encounter series in Figure 2.10 containing observations of an event over 28 days. For example, this may represent a person attending church, where his/her cellphone connects to the WiFi access point in the church. In the example series, this person goes to church every Sunday (assuming the week starts on Monday), and may occasionally drop by on another day during the week. The encounter series has an underlying weekly encounter pattern on Sunday. In other words, this encounter series has a period length of 7 days and the phase of the encounter is Sunday.

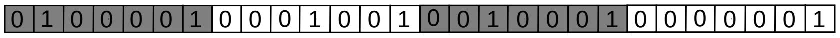


Figure 2.10: A noisy series

In this example, encounters happening other than on Sunday would be considered as noise. If any encounter is missing on Sunday, such a missing encounter would be considered a missing periodic encounter.

In summary, our method takes a set of encounter series as inputs. Each series represents the sequential encounters of a unique pair of nodes. The analysis described above is applied to each individual encounter series separately. If an encounter series has periodicity after pre-processing and recognition, an encounter pattern is derived from this encounter series by alignment. The process repeats for the next series for the next unique pair of nodes.

2.4 Experimental results

In this section, we present a set of experiments to examine our method. There is no previous work to which we can compare our method, so we first conducted a set of experiments in a controlled environment. We controlled the synthetic generation of encounter series. After this, we conducted experiments using real mobility traces.

2.4.1 Experiments using synthetic encounter series

We use synthetic traces to understand the performance of our method. By constructing synthetic traces, we can control the characteristics of the encounter patterns, including period lengths and the phase(s) of encounters within a pattern. This allows us to see how robust and noise-tolerant our method is.

We propose four controlled variables to create synthetic encounter series.

1. **Period length** m is used to control the length of an encounter pattern $P^m_{x,y}$.

2. **Probability** p_α is used to control the chance of missing an encounter within a periodic pattern. In other words, p_α controls the probability that a periodic encounter does not occur.

3. **Probability** p_β is used to control the chance of an unexpected/occasional encounter occurring at a non-encounter phase.

4. **Threshold** T defines the probability of a periodic encounter appearing at a phase.

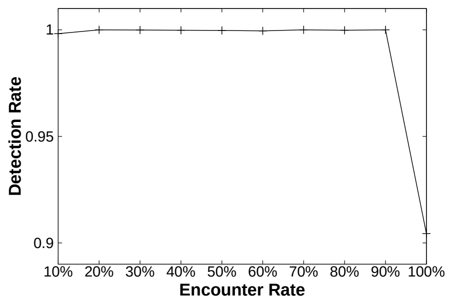
Period length is controlled for consistency. Given fixed values for p_α and p_β, the longer the period, the higher the chance that an event will be changed. In order to keep probabilities consistent, period length m is controlled as well.

To produce an encounter series, we need to select a series length, n. In our experiments, n is set to be 1440 to model a duration of one day with a one minute granularity, or 60 days with a 1 hour granularity and so on. We first choose a period length and create an encounter pattern with the chosen length. Note that there is at least one encounter within an encounter pattern. Then, encounter patterns are concatenated until the desired series length n is reached. If probabilities p_α and p_β equal zero, then we have a perfect encounter series. As both probabilities increase, they bring more and more noise into the encounter series.

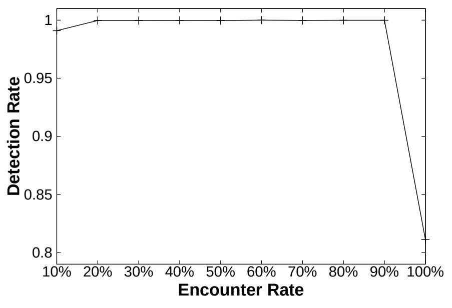
For the following, we set $T = 80\%$. We generated 100,000 encounter series for each combination of the remaining three variables and applied our method to produce the results in Figure 2.11 and 2.12. For all the results in Figure 2.11, $m = 100$. In the figures, each x-axis label includes the encounter series with a encounter rate that is less or equal to the current label but greater than the previous label. One note is that the series without any encounter is not covered by point 10% in all figures. For example, the point at 30% encounter rate on the x-axis covers series with $20\% < R_{x,y} \leq 30\%$

In Figure 2.11(a), we set both p_α and p_β to zero in order to see the performance of our method for several perfect encounter series. The results present the detection rates for encounter series with ten different encounter rates, $R_{x,y}$. The point at 10% includes series with $R_{x,y} \leq 10\%$ where rate 0 is not included, the point at 20% covers series with $10\% < R_{x,y} \leq 20\%$, etc. Figure 2.11(a) shows that our method is able to find over 99.9% of the encounter patterns except for patterns with high encounter rates ($R_{x,y} > 90\%$). Series with $R_{x,y} = 100\%$ are not detected because those series have no wave representation. However, a pre-scan of the series can easily solve this problem.

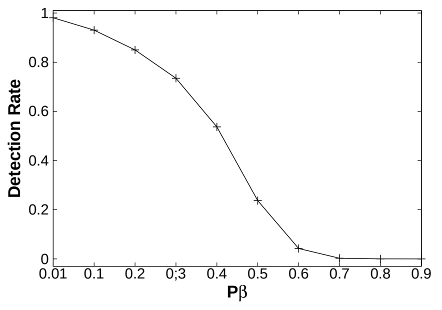
In Figure 2.11(b), we increase both p_α and p_β slightly to 0.01. The results are similar to Figure 2.11(a), however the detection rate for series with low $R_{x,y}$ ($\leq 10\%$) and high $R_{x,y}$ ($> 90\%$) has decreased. Similar results were observed in previous work as well [56]. The conditional probabilities for series with encounter rates at both ends of the scale are sensitive to noise. The encounter probabilities are easily changed with even a small amount



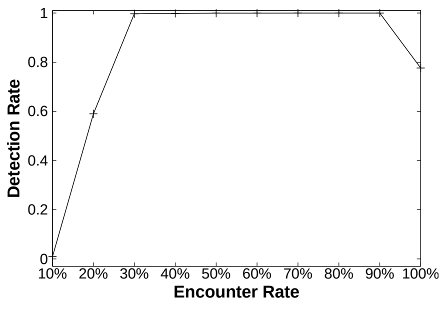
(a) $p_\alpha = 0$, $p_\beta = 0$



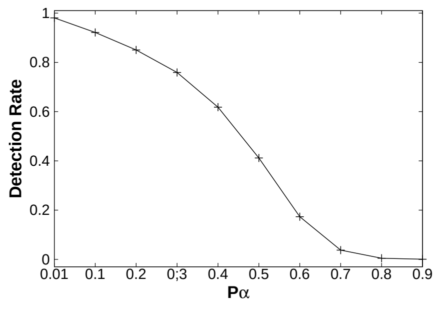
(b) $p_\alpha = 0.01$, $p_\beta = 0.01$



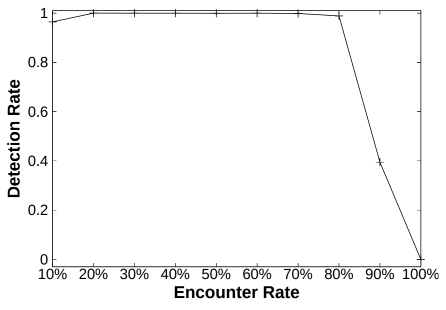
(c) $p_\alpha = 0.01$, varying p_β



(d) $p_\alpha = 0.01$, $p_\beta = 0.1$

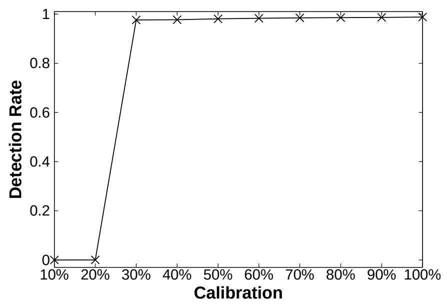


(e) Varying p_α while $p_\beta = 0.01$

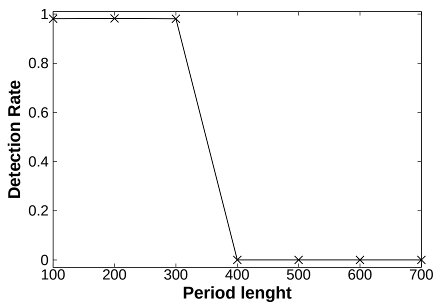


(f) $p_\alpha = 0.1$, $p_\beta = 0.01$

Figure 2.11: Effects of p_α and p_β



(a) $m = 100$, $p_\alpha = 0.01$ and $p_\beta = 0.01$



(b) $m = 100$, $p_\alpha = 0.01$, $p_\beta = 0.01$ and $T = 80\%$

Figure 2.12: Varying m and calibration

of noise.

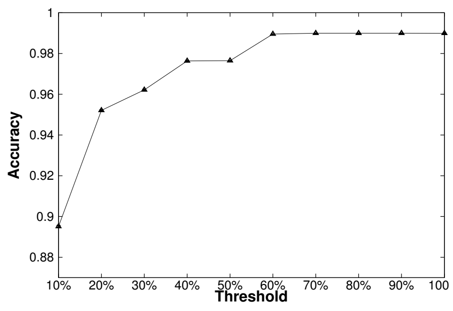
In Figure 2.11(c), we fixed the period length $m = 100$ and $p_\alpha = 0.01$ to examine the effect of p_β. Increasing p_β decreases the detection rate dramatically, as expected. The increasing amount of noise in the encounter series degrades the APG. Therefore, we are not able to detect periodic encounters.

We also tested the effect of encounter rate on detection rate (see the results in Figure 2.11(d)). Here, the detection rates are very low for series with $R_{x,y} \leq 20\%$. The lack of periodic encounters lets unexpected encounters dominate the results, so the encounter series lose their periodicity.

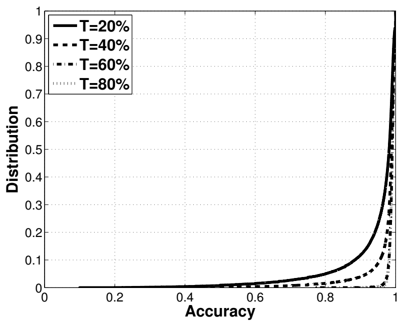
We tested the effect of p_α the same way we tested p_β. We fixed the period length to be 100 and set $p_\beta = 0.01$. Figure 2.11(e) is very similar to Figure 2.11(c). The increasing amount of noise makes the encounter series less and less representative.

Lastly, we examined detection rates at different encounter rates (see Figure 2.11(f)). The results are symmetric to those in Figure 2.11(d). p_α affects series with high $R_{x,y}$ while p_β affects series with low $R_{x,y}$. Missing periodic encounters clearly affect the APG. Encounters from other lags and phases are able to dominate the conditional probability and mask the true periodicity when the encounters are very dense.

The results shown in Figure 2.12 are produced by fixing p_α and p_β to be 0.01 while varying m and the *calibration*. Calibration is the percentage of the total length of the series used to detect periodicity. For example, if the calibration is 40%, then we use the first 40% of the series in our analysis. The purpose of varying m is to examine how sensitive the detection is to the period of the pattern. The purpose of varying calibration is to see how sensitive detection is to the length of the acquired series. In Figure 2.12(a), the period length is 100 and the length of each encounter series is 1440. The results show that no encounter



(a) Varying T



(b) CDF of accuracy

Figure 2.13: Varying T with 50% calibration

patterns are found until the calibration is over 30%. This is because we need an encounter series that is at least 4 times the period length for detection. The first portion of an encounter series whose length is twice the period is used to detect the first repetition of the pattern. Then the second repetition is used to confirm the pattern. Figure 2.12(b) demonstrates the same result from a different point of view. Patterns with length greater than 360 could not be found in these experiments because the time series are only of length 1440.

With the APG, we are able to detect periodicity and extract encounter patterns. However, we still need to validate the extracted patterns to ensure they persist throughout the series. We do this by examining the series, and the extracted patterns, at a given calibration. For example, with 50% calibration, we use the first 50% of the series for periodicity and pattern detection. If we detect a periodic encounter in the first half of the series, we extract the corresponding pattern and label it *promising*. Then we use the second 50% of the series for validation. For each promising pattern, we calculate the fraction of times an encounter appears where it should. We call this the accuracy of the pattern.

We now illustrate the results of accuracy testing. In the following set of experiments, we set $m = 100$, $p_\alpha = p_\beta = 0.01$ and varied T. We used 50% calibration to find the encounter patterns in the series, and then we used the remaining 50% to check the accuracy of detection. Figure 2.13(a) shows that increasing the detection threshold increases the accuracy. This is because we include fewer non-periodic encounters that affect the accuracy of detection. Figure 2.13(b) supports this conclusion by presenting the CDF of accuracy. With $T = 20\%$, 10% of the encounter patterns are detected with an accuracy less than 90%. However, by increasing the threshold to 40%, less than 5% of the patterns are detected with an accuracy lower than 90%. Therefore, our experiments suggest that a high detection threshold should be used.

These experiments show that our method is robust and tolerates a certain amount of noise. Figure 2.11(c) and (e) show that approximately 20% to 30% noise is acceptable (above this range the slope of the curve changes dramatically). The stability of periodic encounters, which is affected by p_α, and the stability of non-encounter phases, which is affected by p_β, are equally important to our method. We also learned that encounter series whose encounter rate is in the middle of the range, *i.e.*, $R_{x,y} = 50\%$, exhibit stronger periodicity than those whose encounter rate is nearer either extreme.

2.4.2 Experiments using real mobility traces

Now we change our focus to real traces. A database called CRAWDAD provides numerous datasets that have been collected by researchers over the years [28]. In general, there are two types of publicly available real-world traces [56, 95]: (1) encounter-based traces and (2) AP-based (access point based) traces. In encounter-based traces, each wireless mobile device such as an iMote [32, 82] or NOKIA 6600 [116] is equipped with a Bluetooth module. The devices frequently broadcast a beacon signal to announce their presence in a neighbourhood. They also acknowledge the beacon signal from other devices. Once a device receives an acknowledgement, an encounter is recorded with a timestamp. AP-based traces are usually GSM traces [32] or WLAN traces [9, 57, 76]. When a mobile devices enters the coverage area of an AP, the AP records the arrival of the device with a timestamp or the duration of the association. If other mobile devices are associated with this AP at the same time, we first assume there is an encounter between each pair of the then-associated devices. Durations of encounters are derived from the association times with the AP. We then treat each access point as a stationary relay in the network. Encounters are derived when two mobile nodes or a mobile and a stationary node are physically close enough. Experiments are conducted for both scenarios.

Encounter-based traces are ideal for our analysis. However, as shown in Table 2.1, the available encounter-based traces are composed of small numbers of nodes observed for very short durations. In contrast to encounter-based traces, the available AP-based traces are rich in nodes and cover a much longer duration. In previous studies, encounters are derived from AP-based traces by assuming that two mobile devices encounter each other if they are associated with the same AP at the same time. In our study, we do not make this assumption. However, we consider each cellular tower or AP as a stationary relay in the network. The reason is because the association between a mobile node and a cellular tower or an AP is a type of encounter. If two mobile nodes never encounter, no message can be

Table 2.1: Real-world traces

Source	nodes	Duration	Type
Milano [94]	44	19 days	PMTR
UMPC [82]	36	12 days	iMote
Cambridge [116]	12	6 days	iMote
MIT Bluetooth [32]	95	282 days	Bluetooth
MIT Cellular network [32]	95	282 days	Cellular
USC Summer 05 [57]	5580	111 days	WLAN
USC Spring 06 [57]	25481	95 days	WLAN
Dartmouth [76]	13888	1178 days	WLAN
Nokia-MDC Bluetooth [80]	38	548 days	Bluetooth
Nokia-MDC GSM [80]	38	548 days	Cellular
Nokia-MDC WLAN [80]	38	548 days	WLAN

Milano = University of Milano
PMTR = Pocket Mobility Trace Recorder
UMPC = University Pierre et Marie Curie
USC = University of South California

exchanged directly between them. However, if both of them associate with the same AP or cellular tower at different times, the message could be exchanged between the mobile nodes indirectly by using the AP or cellular tower as a relay. Although relaying is not currently a feature of cellular base stations or APs, we believe there is merit in exploring the potential benefit of adding this functionality, as it would enable the diverse applications of DTNs.

Initially, we looked at the MIT traces [32], Dartmouth College traces [76], USC traces [57] and the Nokia-MDC traces [80] shown in Table 2.1. In the MIT, Dartmouth College and USC traces, the duration of each encounter is given. However, in the Nokia-MDC traces, each encounter has a corresponding timestamp. From that, we need to calculate the duration of each encounter. In Figure 2.14, mobile node A encounters nodes B and C at timestamp T_i. At the next timestamp, A only meets node C. We assume that A stops seeing B right before T_{i+1}. Therefore, the encounter for node pair {A,B} starting at T_i has a duration of $T_{i+1} - T_i$ time units. Similarly, the encounter for node pair {A,C} starting at T_i has a duration of $T_{i+2} - T_i$ time units. Using this approach, there are two complete encounters, {A, B} and {A, C}, starting at T_i.

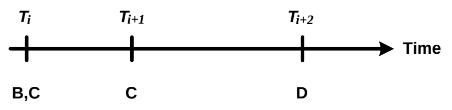


Figure 2.14: Encounter history of mobile node A

At the time of examining the Nokia-MDC traces, we also considered another approach to define encounter duration that is based the characteristics of inter-encounter times. We use Figure 2.14 to explain the second approach. In Figure 2.14, mobile node A encounters nodes B and C at timestamp T_i. At timestamp T_{i+1}, A only meets node C. Similarly, A only encounters node D at timestamp T_{i+2}. We first extract the individual inter-encounter times. In this example, there would be two inter-encounter times: (1) $T_{i+1} - T_i$ and (2) $T_{i+2} - T_{i+1}$. We then calculate the average of all inter-encounter times as T^{AVG}. In our example, the encounter for node pair $\{$A,B$\}$ starting at T_i has an inter-encounter time of $T_{i+1} - T_i$ time units. If the inter-encounter time of $T_{i+1} - T_i$ time units is less then T^{AVG}, the duration of this encounter will be $T_{i+1} - T_i$ time units. Otherwise, the duration of this encounter is set to be T^{AVG}. We conducted experiments on the Nokia-MDC traces using both methods for calculating encounter durations. The results showed that both methods returned very similar results. This is because all encounter series are derived with a 1-day granularity. The assumptions about encounter duration have a very small impact on generating encounter series, and thus they have a very small impact on our final results. In the rest of this chapter, durations derived from real-world traces are based on the first approach, and encounter series are derived with a 1-day granularity. With respect to periodicity, the USC traces were the only ones to exhibit periodicities extending over their full lengths because the USC traces are relatively short, *i.e.*, 95 days, as noted in Table 2.1. We will discuss the reason for this shortly while in this section we present our experimental results based on the USC Spring traces from 2006.

From our previous results, we know we need a series of at least 4 times the period length to detect a periodic encounter pattern. This means we will only be able to detect patterns with periods less than 23 days in the USC spring traces. By using the complete length of each series (100% calibration), we were able to detect 2536 encounter series, where the encounters involve 2394 unique nodes. There are 25481 nodes in the USC spring traces, so only a relatively small fraction of them – less than 10% – are involved in periodic encounters.

As before, we find that the number of patterns detected changes with the calibration. Figure 2.15 shows a zigzag curve for the number of patterns detected while increasing the calibration. This is because different lengths of encounter series can only accommodate encounter patterns with a certain period. For example, 30% calibration can detect periods up to $\lfloor (95 * 0.3)/7 \rfloor = 4$.

We note a drop in the detection curve at 60% calibration. At 60% calibration, each encounter series contains encounter information from $\lfloor (95 * 0.6)/7 \rfloor = 8$ weeks. Because the user-defined threshold T was set at 80%, an encounter series is noted as having underlying periodic encounter behaviour if and only if it has the encounter at the same phase in $\lceil 8 * 0.8 \rceil = 7$ out of 8 weeks. Many of the encounter series fail to pass this high threshold value because of missing encounters. As a result, we observe this drop in the detection curve. The increase in the number of patterns detected after 60% calibration suggests: (1) a new encounter series with a 14-day periodicity has been detected, and (2) an encounter series that failed at 60% calibration has been detected again due to including more encounter information.

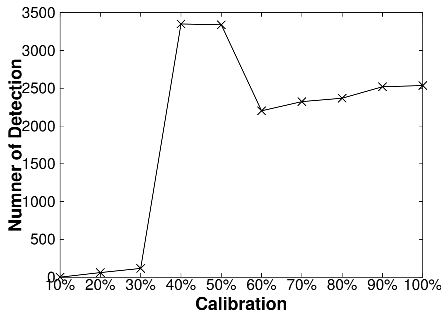


Figure 2.15: Number of patterns detected

Figure 2.16 presents the CDF for periods found at different calibrations. When the calibration is 20%, the majority of patterns have lengths of 2, 3 or 4 days. However, when the calibration reaches 40%, the majority of patterns have length 7 days. As we increase the calibration, over 99% of the patterns are of weekly or multi-week periodicity.

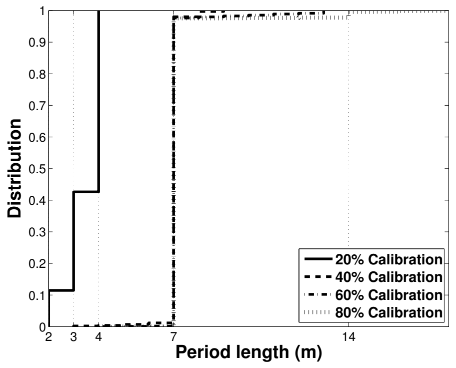


Figure 2.16: Detected period length using different calibrations

In Figure 2.17, we present the average accuracy of encounter patterns for each calibration. This figure shows that using a longer series in detection leads to better accuracy. However, there is a drop in accuracy from calibration 80% to 90%. This is an artifact because the last 10% of a series may not be as long as the period of the pattern being validated.

This leads to a degradation in performance.

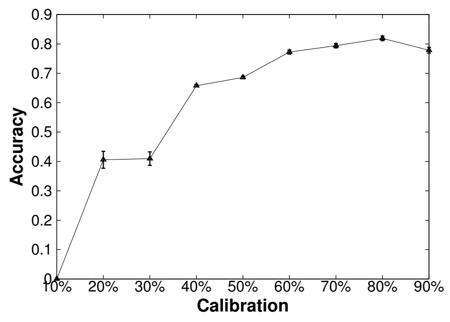


Figure 2.17: Accuracy using different calibrations

Figure 2.18 shows the CDF for accuracies at different calibrations. Here again, we see that longer encounter series provide better results. When the calibration reaches 80%, approximately 50% of the detected patterns have 100% accuracy.

We have examined the effects of varying calibrations in terms of the quality of detected patterns. Furthermore, we conducted experiments to examine the effect of the threshold, which controls the promising phases. In the following experiments, we use 50% calibration to detect encounter patterns, and the remaining 50% of the series to examine the accuracy of the detected patterns.

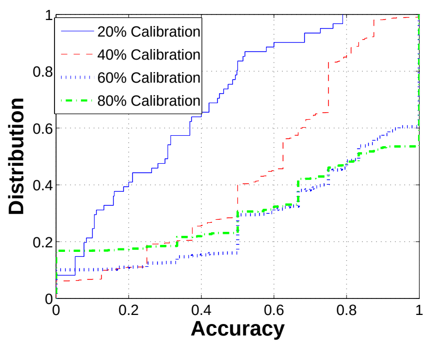


Figure 2.18: Accuracy distributions using different calibrations

As we noted earlier, one of the inputs to our method is the user-defined threshold T. This defines whether the encounter occurs over a significant enough fraction of time at a particular phase to be considered periodic. By increasing the threshold, we place heavier constraints on the phases that qualify. On one hand, fewer patterns satisfy the threshold, leading to a decrease in detections as shown in Figure 2.19. On the other hand, the phases yielded by a higher threshold value provide encounter patterns with better accuracy, as shown in Figure 2.20.

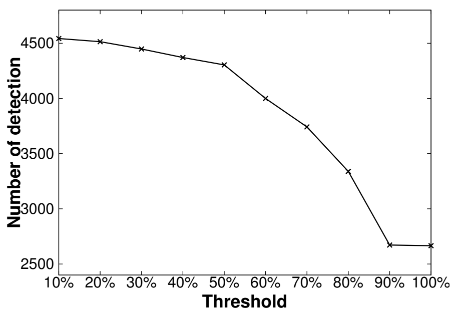


Figure 2.19: Number of patterns detected

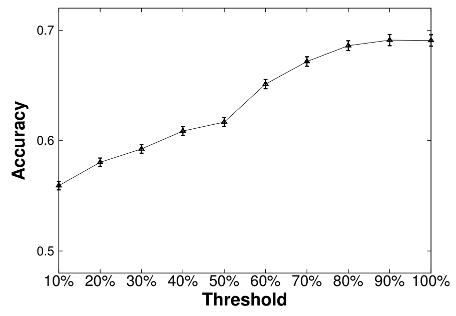


Figure 2.20: Accuracy with different thresholds

2.5 Analysis of results

Our experimental results indicated that there are detectable weekly periodic encounter patterns within real-world traces. In order to apply those periodic encounters for routing in DTMNs, we need to understand the characteristics of periodic encounter patterns from three main perspective: (1) the persistence of periodic encounters, (2) the connectivity among nodes with periodic encounter behaviours and (3) the structure, *i.e.*, small-world structure, of the networks formed by periodic nodes. In this section, we present the results regarding these characteristics.

2.5.1 The persistence of periodicity within real mobility traces

In our experiments, we applied our detection techniques to four different traces. Only the USC traces exhibit periodic behaviour for the full duration of the series. However, this does not mean that there is no periodic behaviour that lasts for shorter periods in the other traces.

To discover short-term periodic behaviour, we break the traces into disjoint segments with the same length. For example, each daily encounter series in the Nokia-MDC traces could be broken down into 19 segments where each segment is 30 days in length. For daily encounter series, we apply our detection techniques to segments with length varying from

1 day to the maximum duration of each trace, with a 1 day increment.

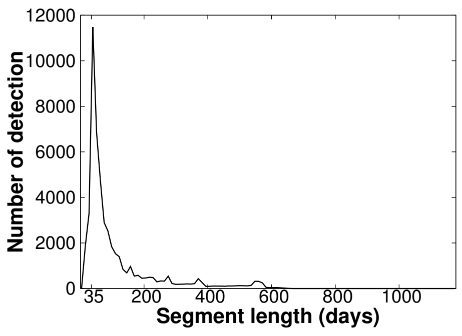
As shown in Figure 2.21 (a), (c), (e) and (g), 35-day segments in the Dartmouth, MIT and Nokia-MDC traces have the largest number of detected periodic encounters. However, in Figure 2.21(g), 33-day segments have the largest number of detected periodic encounters in the USC traces. The USC Spring traces in Figure 2.21(g) have a sudden drop after the 33-day segments. This is caused by an artifact. The USC Spring traces have a duration of 95 days. At 34-day segments, each encounter series could be broken down into 3 segments. The first two segments each cover 34 days whereas the last segment only contains 27 days. As a result, even though there may be weekly periodic patterns within the last segment, we can only detect periodic patterns up to 6 days, $\lfloor \frac{27}{4} \rfloor = 6$. Therefore, we lose a large number of detections in the last segment of all encounter series.

To understand these periodic behaviours at the segment size with the largest number of detected periodic encounters, Figure 2.21(b), (d), (f) and (h) provide more details about detected periodic behaviour by presenting the CDF of their periods. The results indicate that there is a strong weekly encounter behaviour in all the mobility traces.

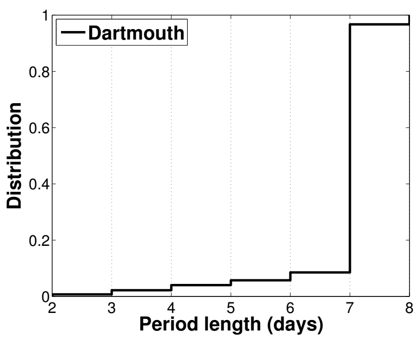
Once the period length is known, we can extract periodic encounters from the original segments using alignment. For alignment, each segment is split into disjoint sub-segments where the length of each sub-segment equals the period length previously detected. Then, we stack all sub-segments vertically to examine the probability that encounters appear in a particular column. When the probability of encounters in a column is greater than a user-defined threshold, we consider that encounter as a periodic encounter. This process is the same as the pattern recognition described before.

From our discussion above, we discovered mobility traces do not have long-term periodicity. If we still examine the accuracy of detected periodic encounters within segments against the whole traces, the overall accuracy for each periodic pattern will be very low. Therefore, instead of evaluating the accuracy, we examine how long those periodic patterns reliably appear in the future. We call this time range the *projected persistence*.

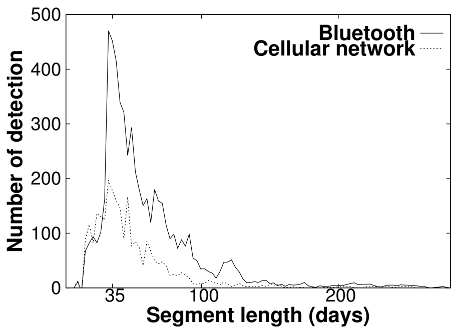
After extracting periodic encounters, we evaluate their projected persistence by using the remaining encounter series following a given segment. For example, if the very first 35-day segment in the Nokia-MDC traces has periodic encounters, we use the remaining 548-35=513 days to evaluate the persistence of detected periodic encounters. Similarly, if the second 35-day segment has periodic encounters, we use the remaining 513-35=478 days to evaluate its persistence. We first partition the remaining encounter series into disjoint intervals, I, whose size equals the previously detected period. To evaluate projected



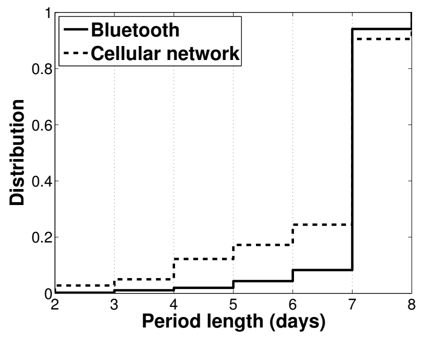
(a) Dartmouth



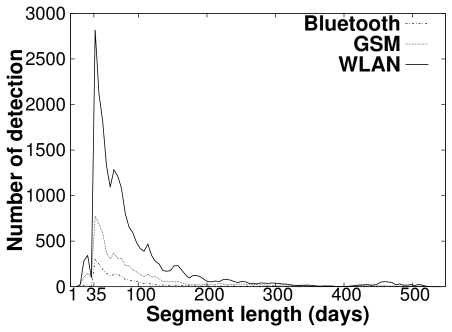
(b) Dartmouth at 35-day segments



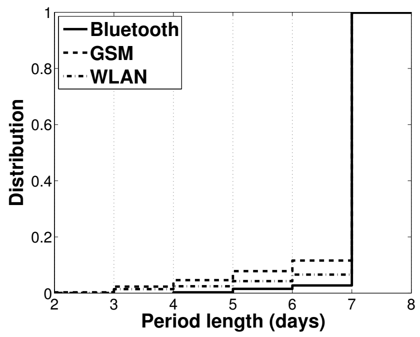
(c) MIT



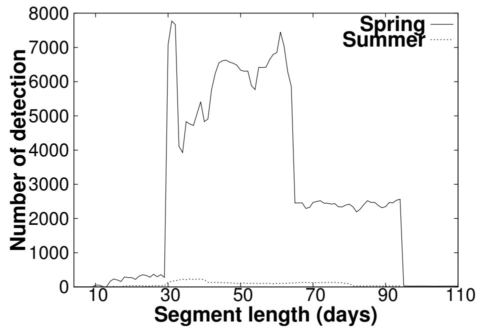
(d) MIT at 35-day segments



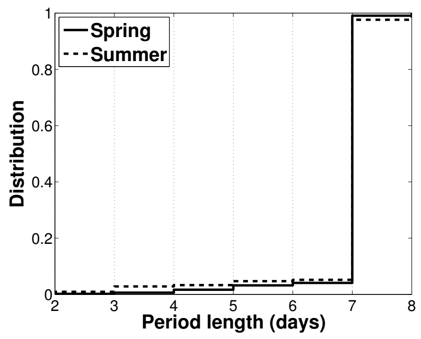
(e) Nokia-MDC



(f) MDC at 35-day segments



(g) USC



(h) USC at 33-day segments

Figure 2.21: Effects of segments

persistence, we introduce two thresholds:

1. the matching probability of periodic encounters within an interval, θ. Given an interval, θ measures the probability that this interval has periodic encounters. For example, if detected periodic encounters take place every Monday and Friday, and a given interval only has an encounter on Monday, then $\theta = 1/2 = 0.5$ for the given interval.

2. the acceptable number of intervals without periodic encounters, Δ. Because of unexpected events, periodic encounters may be interrupted. Therefore, Δ is used to tolerate the number of intervals that suffer from missing periodic encounters because of interruptions.

Using Figure 2.22 as an example, let us assume that from the very first 35-day segment, we detected periodic encounters repeating every 7 days. Since the period length is 7 days, we partition the remaining encounter series into intervals whose size equals 7 days and calculate θ for each interval. Given thresholds at $\theta = 0.9$ and $\Delta = 0$, only I_1 satisfies the θ threshold; therefore, the projected persistence is 7 days. If thresholds are $\theta = 0.9$ and $\Delta = 1$, by treating I_2 as the tolerated non-periodic interval, I_4 is the interval furthest in the future for which the thresholds are satisfied, therefore, the projected persistence is 28 days. Similarly, if thresholds are $\theta = 0.9$ and $\Delta = 2$, the projected persistence is 42 days.

Tables 2.2 and 2.3 show the average projected persistence for each mobility trace at the segment length with the largest number of detected periodic encounters. On one hand, as θ increases, the average persistence decreases because we are looking for intervals with highly persistent and stable periodic encounters. On the other hand, as Δ increases, we have longer projected persistence because we are more tolerant of interruptions in the patterns.

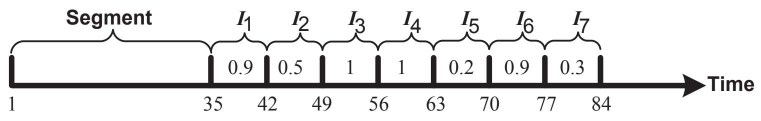


Figure 2.22: Accuracy with different thresholds

This explains why we did not detect any long-term periodic behaviour in the full-length traces: their persistence is interrupted from time to time. Due to various limitations such as students changing schedules each semester, or seasonal road construction, it is possible that periodic behaviours have been changed, or those behaviours just change. Therefore, long-term periodic behaviour is not detected.

Table 2.2: $\Delta = 0$, projected persistence by varying θ

Trace types	Threshold θ				
	0.1	0.3	0.5	0.7	0.9
Dartmouth 35-day segments	87.01 days	63.49 days	46.14 days	30.32 days	19.31 days
MIT Bluetooth 35-day segments	40.5 days	32.56 days	26.65 days	22.04 days	15.56 days
MIT Cellular 35-day segments	20.37 days	19.84 days	19.01 days	13.84 days	12.67 days
USC Spring 33-day segments	23.77 days	22.49 days	20.12 days	15.21 days	14.9 days
USC Summer 33-day segments	27.07 days	26.08 days	24.12 days	21.14 days	18.38 days
Nokia Bluetooth 35-day segments	54.68 days	39.9 days	29.01 days	22.28 days	14.59 days
Nokia GSM 35-day segments	70.72 days	54.45 days	33.72 days	23.64 days	14.18 days
Nokia WLAN 35-day segments	61.63 days	47.87 days	34.94 days	23.55 days	14.58 days

Table 2.3: $\theta = 0.9$, projected persistence by varying Δ

Trace types	Threshold θ			
	0	1	2	3
Dartmouth 35-day segments	19.31 days	20.74 days	27.83 days	35.15 days
MIT Bluetooth 35-day segments	15.56 days	18.89 days	20.17 days	23.88 days
MIT Cellular 35-day segments	12.67 days	14.03 days	15.92 days	17.54 days
USC Spring 33-day segments	14.9 days	20.11 days	26.15 days	29.98 days
USC Summer 33-day segments	18.38 days	20.15 days	20.92 days	21.04 days
Nokia Bluetooth 35-day segments	14.59 days	17.29 days	23.41 days	29.17 days
Nokia GSM 35-day segments	14.18 days	14.18 days	25.83 days	31.41 days
Nokia WLAN 35-day segments	14.58 days	14.58 days	25.28 days	28.54 days

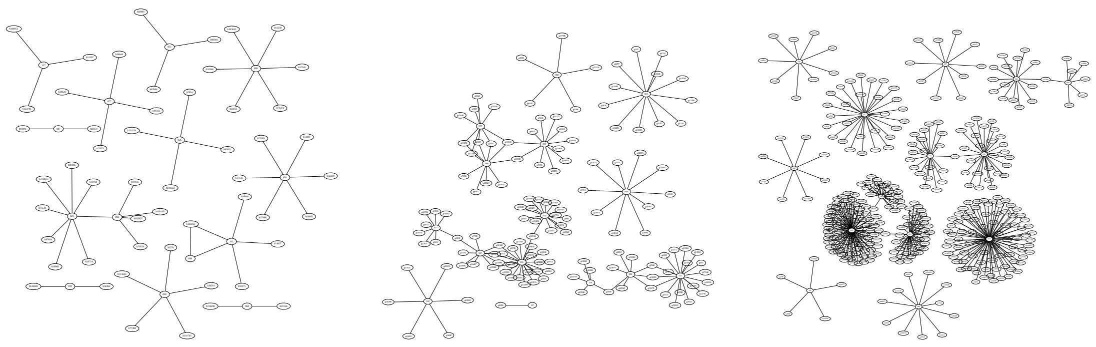

(a) Bluetooth (b) GSM (c) WLAN

Figure 2.23: Connectivity in different network types from Nokia MDC traces

In addition, both 2.2 and 2.3 reveal information about the persistence range. As shown in this table, for a certain value of Δ, periodic behaviour can last up to a few months on average. For example, given $\Delta = 3$ in the Dartmouth traces, the projected persistence can be up to 35+35.15=70.15 days (segment length plus projected persistence).

2.5.2 Network connectivity

We also built graphs based solely on periodic encounter series. The connectivity graph $G = \langle V, E \rangle$ is an undirected graph, which is defined as follows. Given a set of encounter series with periodic behaviour, P, V is the set of nodes participating in P. For a pair of nodes u and v, the edge $e_{u,v} \in E$ if and only if $s_{u,v} \in P$.

The Nokia-MDC datasets include mobility datasets from different network types. The three types of datasets are Bluetooth, GSM and WLAN. Bluetooth traces track direct encounters between mobile nodes. Both GSM and WLAN traces record associations between mobile nodes and either access points or cellular towers. Using the Nokia-MDC traces as an example, the connectivity graphs for 35-day segments shown in Figure 2.23 represent connectivity between nodes with periodic behaviour from different network types. In this figure, connection graphs derived from Bluetooth, GSM and WLAN traces consist of several star shapes. In the Nokia-MDC datasets, we have a total of 38 participants whose encounters are recorded. As a result, the centre of each cluster is one of the participants whereas leaf nodes are external MAC addresses whose encounter series is unknown, except for one encounter with a non-leaf node. All three graphs are composed of multiple isolated clusters where nodes from different network types cannot communicate with each other. In real life there are various co-existing network types. For example, in the Nokia-MDC datasets we have cellular networks, wireless local area networks and Bluetooth encounters.

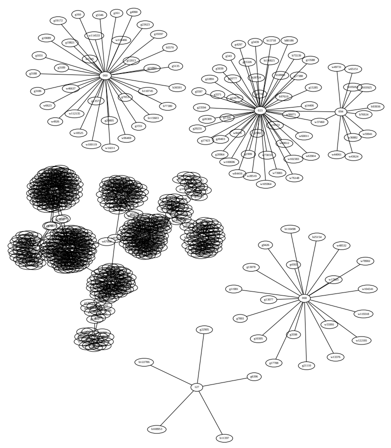


Figure 2.24: The integrated network from Figure 2.23

Mobile nodes could exhibit different periodic behaviours within each network type. As shown in Figure 2.24, integrating periodic encounter series from different network types presents a better connected graph, which could substantially improve the effectiveness, and likely efficiency, of communication overall. Nonetheless, there still exist isolated clusters. As shown in Figure 2.23, the networks containing periodic nodes are composed of isolated star-shape components where the centre of each component is one of the 38 participants in the Nokia-MDC datasets.

Unlike social networks, which are generally connected, the periodic nodes in our studies do not form a single connected component. As a result, conventional DTN routing protocols cannot be immediately applied to take advantage of periodic encounter behaviour. Providing services between the clusters in such networks still requires further research. One solution would be using some underlying infrastructure such as the Internet to connect access points in isolated clusters. An alternative solution could be deploying relays (throw-boxes [149]) in the network.

2.5.3 Small world structure

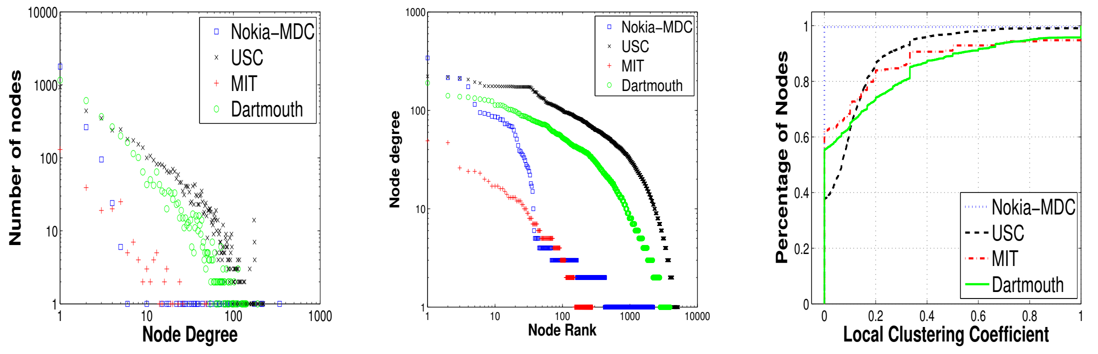
In this section, we examine the properties of the connectivity graphs that are composed by periodic patterns in different mobility traces. To create connectivity graphs, we selected the periodic patterns detected using the segments with the highest number of detections, *i.e.*, 35-day segments from the Nokia-MDC, MIT and Dartmouth traces but 33-day segments from the USC traces. The connectivity graph shown in Figure 2.23 is an example derived from the Nokia-MDC traces. In these graphs, we may have several clusters. Since some evaluations can only be used on connected graphs, we present our analyses on the largest cluster in each graph from each mobility trace.

Degree distribution

The node degree distribution for the largest clusters in each mobility trace follows the power-law distribution with a heavy tail (see Figure 2.25(a)). However, these large clusters contain a relatively small fraction of the users. To investigate further, we rank nodes in descending order based on their node degree. Plotting the sorted nodes in Figure 2.25(b), we find that the USC, MIT and Dartmouth traces follow the power-law distribution. Even though the overall distribution of the Nokia-MDC traces is non-trivial, the first 38 nodes in the distribution follow the power-law distribution. This number matches the total number of participants in the Nokia-MDC datasets whereas all other nodes are external nodes whose complete mobility is unknown. This strongly suggests that by building a network using nodes with periodic behaviour, we can obtain a scale-free network rather than a random network.

The small-world network

Based on the Watts-Strogatz mechanism [134], a graph with a small-world structure must satisfy two properties: (1) short average path lengths, and (2) clustering coefficient is orders of magnitude higher than its random equivalent. Table 2.4 presents calculations regarding small-world structure. In the Nokia-MDC traces, the average node degree is $d = 2.51$. Investigating the small-world structure in this network, we find that the average path length, L, between any pair of nodes in the network is 3.12. We observe that the average path length is approximately equal to the logarithm of the number of nodes in the network, $3.12 \approx log(2025) = 3.30$. This is a good indication that the network has a small-world structure [134]. In addition, we calculate the clustering coefficients for all nodes in the network. There are two types of cluster coefficient: local and global, which measure the number of triangles with respect to the number of open triplets in the graph [53, 134]. In other words, cluster coefficients measure whether a graph forms a clique. If it is not, it tells how close the graph is to a clique. Our calculations show that 99.9% of the nodes in the Nokia-MDC datasets are external nodes whose local cluster coefficient equals zero as shown in Figure 2.25(c). Therefore, the global cluster coefficient of the network equals the relatively small value of 0.0015. If we consider a node in a random graph, the probability that two of its neighbours are connected is equal to the probability that two randomly selected nodes are connected. Consequently the global cluster coefficient of a random graph equals d/N.



(a) Node degree distribution (b) Node degree rank (c) Local clustering coefficient

Figure 2.25: Comparison between datasets for small-world structure in log-log scale

Since the global cluster coefficient for the network in the Nokia-MDC datasets is very close to that from a random graph, $2.51/2025 = 0.0012$, we conclude that the network does not have a small-world structure even though the average path length still satisfies the small-world bound. One possible reason is that the traces of external nodes are unknown, and we may be losing periodic encounter series among them. As a result, the graph still consists of multiple star-shaped components.

To show that networks formed by nodes with periodic behaviour have a small-world structure, we present our studies about the other three traces. In the USC connectivity graph, the largest cluster contains 4961 mobile nodes. As shown in Figure 2.25(a) and 2.25(b), the node degree distribution in the USC traces follows the power-law distribution. It is clear that these traces come from a scale-free network. To further explore the network, we calculate the average path length for its connectivity graph. The value at 3.34 is a good indication of the small-world structure, where $3.34 \approx log(4961) = 3.69$. In addition, the global cluster coefficient shown in Table 2.4 is much greater than the one from a random graph. Therefore, we conclude the network formed by the periodic nodes in the USC traces has a small-world structure. The same analyses can be applied to the MIT and Dartmouth traces where we also conclude that graphs composed by the periodic nodes have a small-world structure.

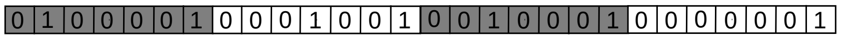
2.6 Future work

Similar to previous work [56], we have an encounter series for every pair of nodes in the mobility traces. Our objective is to detect and extract periodic encounter patterns in the encounter series with the presence of random/noise encounters that could affect the detection. However, experimental results showed that our method had a low detection rate for

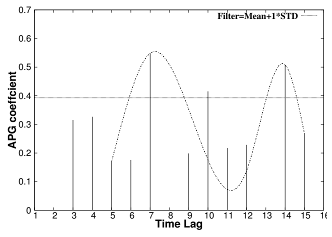
Table 2.4: Small-world statistics from different mobility traces

Network	N	APL	d	GCC
Nokia-MDC	2025	3.12	2.51	0.0015
Random Graph	2025	3.12	2.51	0.0012
USC	4961	3.34	19.44	0.139
Random Graph	4961	3.34	19.44	0.0039
MIT	268	2.72	4.07	0.13
Random Graph	268	2.72	4.07	0.011
Dartmouth	3642	3.75	9.47	0.197
Random Graph	3642	3.75	9.47	0.0026

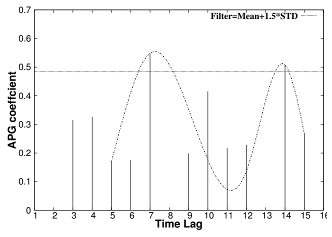
N=number of nodes in the network
APL=average path length
d=average node degree
GCC=global clustering coefficient



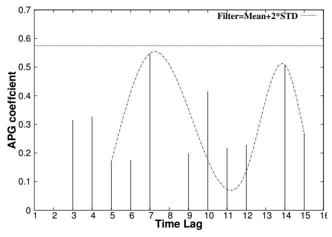
(a) Encounter series shown in Figure 2.25



(b) $K = 1$



(c) $K = 1.5$



(d) $K = 2$

Figure 2.26: The effects of using different thresholds by varying constant K

node pairs with frequent encounters where such pairs are clearly important relays. If we can clean the encounter series by distinguishing regular encounters from the random ones, we may detect more patterns and/or have detection with better persistence. Therefore, there could be two ways in general to improve our techniques: 1) pruning the search space by excluding series with only random encounters, and 2) cleaning random encounters within each encounter series. One possible solution inspired by other studies is to generate random mobile traces and prune encounter series or encounters within series that have similar characteristics comparing to the ones from random traces.

In our method, the filtering described in Section 2.3.2 plays an important role to identify periodic lags in auto-persistence functions. To prune spikes representing random/aperiodic behaviours, we proposed a threshold defined as $F = \overline{m} + K * \sigma$ where $\overline{m}$ is the estimated mean of the APG values, σ is the estimated standard deviation and K is a constant. Given an encounter series, in the definition of the threshold its value is controlled by the constant K because both $\overline{m}$ and σ are fixed. Therefore, how to choose a proper constant becomes critical. In Figure 2.26, examples are presented to demonstrate the effects of using different K values. Given the encounter series in Figure 2.26 (a), we have shown that this encounter series has an underlying weekly encounter pattern. When K equals 1, because three spikes found at lags 7, 10 and 14 are not correlated, we can not detect the underlying periodicity. When K is equal to 1.5, spikes located at lags 7 and 14 indicate the weekly patterns. However, if we increase the value of K to 2, no spike can be discovered leading to a failed detection. From those examples, the value of the threshold can affect detection results because a tight threshold could overlook periodic encounters whereas a loose threshold could include aperiodic behaviours in the detection process. To address the problem, we could examine some properties such as rankings of all returned APG coefficients to find a proper threshold instead. An alternative solution could define a range for K instead of having a constant value, and examine every possible value of K with a granularity within the range for periodic encounter patterns.

2.7 Summary

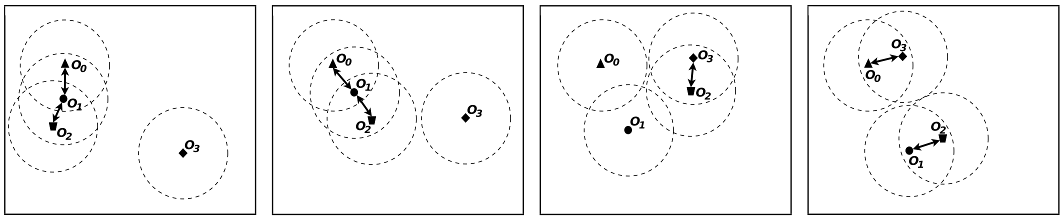
Understanding user mobility, and the patterns of encounters between nodes, is critical for communications within a DTN. Instead of blindly using opportunistic encounters, periodic encounters can be used in routing messages between nodes. In this chapter, we explore the periodicity and regularity within encounter series by applying the auto-persistence graph. Our experimental results show that our method is robust and can detect and extract the periodic behaviours in synthetic traces. It can also detect and extract encounter patterns in real traces with up to 100% accuracy. Our experiments show that noise in the form of unexpected encounters, and in the form of missing periodic encounters, affect the fidelity of our method equally. Nonetheless our experiments showed that there is no long-term periodicity within mobility traces whereas periodic behaviours could be interrupted by unexpected events from time to time. Experimental results suggest that periodic behaviours are highly likely to change after a few months. This helps us to determine how stable and persistent a pattern is, which explains why one is not able to find a pattern that lasts the whole duration of the mobility trace. To understand the properties of networks formed by periodic encounter series, our studies show that networks are scale-free networks, and some of them satisfy a small-world structure where messages between any pair of nodes can be delivered through a very small number of hops in the network.

Chapter 3

A graph model for periodic encounter patterns

3.1 Introduction

In Chapter 2, we proposed a methodology to detect and extract periodic encounter patterns within mobile data. Given periodic encounter patterns, the next task is how to properly model them in order to find routes to deliver packets through periodic encounters. To illustrate the benefits of encounter patterns, an example is presented in Figure 3.1 where we have four nodes. In this example, node o_0 is a stationary base station, while the other three are mobile nodes. Encounters, represented by directed lines, occur during four consecutive time intervals. In the example, at the beginning of interval I_0, node o_0 wants to send a packet to node o_3. If routing protocols from mobile ad-hoc networks start route exploration immediately at interval I_0, no route will be found until interval I_3. Until then the routing discovery process is simply a waste of energy. In contrast to those protocols, and assuming one knows the encounter pattern for this scenario, two routes can be found to disseminate the packet from o_0 to o_3. The first route is that node o_0 holds the packet and sends it to destination o_3 during interval I_3. The second route is that node o_0 transmits the packet to



(a) Snapshot at interval I_0 (b) Snapshot at interval I_1 (c) Snapshot at interval I_2 (d) Snapshot at interval I_3

Figure 3.1: Encounters during four consecutive intervals

node o_1, and node o_1 carries the packet and disseminates it to o_2. The packet is eventually delivered to node o_3 by node o_2 during interval I_2. The first route saves energy, but is not the most efficient in terms of delay, whereas the second route has less delay but consumes more energy.

In this chapter, we introduce a graph model which integrates all periodic encounter patterns. Using the graph model, both the minimum delay and minimum energy consumption problems can be modelled as optimization problems. The only change between the graphs used to solve either problem is in the assignment of weights for the edges. The minimum delay problem can be modelled as the shortest path tree problem whereas the minimum energy consumption problem can be modelled as the minimum weighted Steiner Tree problem. With a modification, we can use Dijkstra's shortest path algorithm [29] to find an optimal solution for the minimum delay problem. However, because the minimum weighted Steiner Tree problem is NP-hard, we propose a polynomial time approximation algorithm to find a multicast tree for routing. The experiments show that our algorithm can quickly find sub-optimal routes whose energy cost is typically 2-3 orders of magnitude smaller than the cost of state-of-art solutions in delay tolerant networks [34].

The remainder of this chapter is organized as follows. An overview of existing models to capture encounters and periodic patterns is discussed in Section 3.2. In Section 3.3, we introduce a graph model that captures the semantics of the discovered encounter patterns. Mathematical optimization and algorithms are presented in Section 3.4 to solve routing problems. Experimental results shown in Section 3.5 present the performance of our algorithms regarding both unicast and multicast routing. Finally, conclusions and suggestions for future work are given in Section 3.7.

3.2 Existing graph models

In communication networks, graphs play an important role in route design and optimizations. In MANETs, a graph is used to capture the network connectivity during a certain period of time with the assumption that the topology does not change. Many routing protocols can take advantage of this assumption to provide end-to-end, *i.e.*, unicast, communication in the network [64, 70, 107]. However, in DTNs, such a static graph is not sufficient to capture topology changes. To address the problem, previous studies proposed a complex graph system, called time-varying graphs [19], dynamic networks [78] or temporal graphs [135]. Although different names are used, they are very similar to each other, and all of them can model topology changes in DTNs. The proposed graph systems contain a set of consecutive static graphs where each static graph captures the network topology at a particular timestamp or during a certain interval. The problem of the proposed graph systems is that they generally are very large and complicated. To model large mobile traces, they will create hundreds or thousands of static graphs for different snapshots or intervals. In addition, the same node in different static graphs at different time instances is not connected. It is difficult to track the same node over multiple time instances. Finding routes among static graphs over multiple time instances becomes even more difficult.

With respect to periodic encounter patterns, several studies have been proposed to use different logical structures to model them. In [49], the authors assumed every node has the global knowledge of all encounters during all intervals. A tree rooted at the source can be built for forwarding, where each level of the tree represents encounters at a certain interval, k, that is k intervals away from the root. The problem of this approach is that different trees have to be built at different intervals in order to accurately capture future encounters. In another work, a contact graph was built on top of repetitive patterns [68]. Within this contact graph, each vertex represents a node whereas an edge links two nodes with a weight to describe the contact time within the period. Because in real life scenarios contacts may not repeat at exactly the same time, a probability of the occurrence of a contact is introduced as a part of the weight on an edge. The problem of the contact graph is that a 2-tuple weight on an edge makes computation extremely complicated. This is the reason that the authors used two algorithms to find an end-to-end path.

The previous work shows different approaches to model encounters and encounter patterns. However, they fall short in two ways: (1) they do not present a formal model that is simple and flexible enough to reflect different assumptions and goals, and more importantly, (2) they do not provide an algorithmic approach to solve the problem optimally. The research presented in this chapter fill both gaps.

3.3 Proposed graph model

Given a set of mobile nodes $O = \{o_0, o_1, o_2, ...o_{n-1}\}$, we assume that a set of *periodic encounter patterns* with period length m is given. The encounter patterns are considered given for the purposes of this chapter. In a periodic encounter pattern, the duration of each phase is equal to a given granularity; this term was defined in Chapter 2. As a result, each phase has the same duration. Let us define the duration of a phase as τ.

The period length m of the encounter pattern indicates the total time over which the pattern repeats itself. For example, if the duration of each phase is one day and $m = 21$, the encounter pattern represents the regular interactions between nodes x and y over a three-week period. The phase t indicates the interval within the period where regular encounters occur. For example, continuing the previous example, if $p_6 = 1$ then nodes x and y encounter each other regularly on the first Sunday in the three-week period, assuming p_0 occurs on a Monday.

During an encounter between two nodes, and at no other time, messages can be exchanged between the two encountering nodes. For instance, in Figure 3.1(a), o_0 encounters o_1 during phase I_0. Furthermore, mobile nodes can relay packets among themselves. For instance, as illustrated in Figure 3.1, a message can be routed from o_0 to o_2 through o_1 at the cost of two transmissions.

Table 3.1: Encounter matrix for node O_0.

	p_0	p_1	p_2	p_3
O_1	1	1	0	0
O_2	0	0	0	0
O_3	0	0	0	1

Table 3.2: Encounter matrix for node O_1.

	p_0	p_1	p_2	p_3
O_0	1	1	0	0
O_2	1	1	0	1
O_3	0	0	0	0

Table 3.3: Encounter matrix for node O_2.

	p_0	p_1	p_2	p_3
O_0	0	0	0	0
O_1	1	1	0	1
O_3	0	0	1	0

Table 3.4: Encounter matrix for node O_3.

	p_0	p_1	p_2	p_3
O_0	0	0	0	1
O_1	0	0	0	0
O_2	0	0	1	0

Using encounter patterns, we can solve a variety of problems in routing a message from a source to one destination (unicast), or from a source to a set of destinations (multicast), such that either the energy cost or the time delay for message delivery is minimized.

Definition 6. *The* encounter matrix *summarizes all periodic encounters for a given node. We define the encounter matrix $M^k = [m^k_{jt}]$ for mobile node o_k as:*

$$m^k_{jt} = \begin{cases} 1 & \textit{iff } p_t = 1 \in P^m_{k,j} \\ 0 & \textit{otherwise} \end{cases}$$

where $0 \le k \le (n-1)$, $k \ne j$ *and* $0 \le t \le (m-1)$.

We note that the number of nodes encountered by any given node, and thus the number of rows in M^k, is expected to be much smaller than n. The encounter matrices for the four nodes in Figure 3.1 are given in Tables 3.1-3.4.

3.3.1 The encounter graph for unicast routing

If we combine the encounter matrices for all the mobile nodes in the network, we can create an *encounter graph* that represents all the observed periodic patterns for a given set of nodes. For example, Figure 3.2 gives the encounter graph corresponding to the behaviours shown in Figure 3.1.

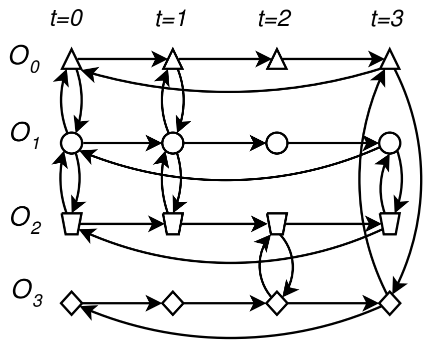


Figure 3.2: An unicast graph model reflecting the encounter patterns in Figure 3.1

In this graph, there are four nodes in the network, and all node pairs have periodic encounter patterns with a period of four intervals. In order to represent a mobile node during different intervals, each node is modelled using a vertex with the same shape at four time intervals in the graph. To denote the same node at different interval and encounters between node pairs, two types of edges, horizontal edges and vertical edges, are used in the graph respectively:

- *Horizontal edges* connect vertices with the same shape to represent the same node at different intervals. This type of edge only connects vertices with the same shape during two adjacent intervals. Because this graph models periodic encounters, when the last interval of a period is reached, a horizontal edge connects the node back to the first interval of the next period to reflect the encounter pattern repeating in the next period.

- *Vertical edges* link two nodes at the same interval to illustrate encounters between these nodes. Encounters happen only within one time interval, so links from one node at one interval to another node at another interval are not permitted.

With this model, unicast routing with respect to both the minimum end-to-end energy consumption and minimum end-to-end delay problems can be treated as shortest path problems. The two problems can be instantiated by assigning different edge weights in the graph.

- **Communication Delay** In this problem, if a node decides to store the message and carry it on to the next time interval to disseminate it in the future, this yields a communication delay equal to the duration of the current time interval. Because all intervals in the graph model have the same duration, each horizontal edge has the same weight. In contrast, if a node decides to transmit the packet immediately, it needs to prepare the message for transmission, and then the message needs to propagate over the wireless channel. However, these delays have already been embedded in the duration of the corresponding interval. Thus, all vertical edges have a zero weight.
- **Energy consumption** In the energy consumption problem, if a node decides to carry the message on to the next time interval and disseminate it in the future, this requires a very small amount of energy for data storage during the current time interval. Therefore, all horizontal edges have a relatively small non-zero weight. In contrast, if a node transmits the message during an interval, each vertical edge has a constant energy cost that is much larger compared to the energy required for message storage.

Another important function from networks is to deliver messages to multiple destinations, *i.e.*, multicast routing. However, our model presented in Figure 3.2 targeting at single source and destination is not able to address problems of multicast routing. In wireless communications, specially in time division multiple access, all neighbours within the transmission range can hear from the source. The energy cost only needs to be counted once no matter how many receivers are within the neighbourhood. However, in our previous graph model, costs of transmission to neighbours will be added separately. For example, in Figure 3.1, node o_1 can transmit to nodes o_0 and o_2 at the same time during interval I_0. In the graph, because both outgoing edges from o_1 to o_0 and o_2 have non-zero energy weights, the total energy cost for this single transmission will be counted twice. Thus, the graph model in Figure 3.2 is not suitable for multicast routing.

3.3.2 The encounter graph for multicast routing

To mimic scenarios of multiple receivers of a single transmission, in the graph model for multicast routing shown in Figure 3.3, two vertices now are used to represent a node during an interval: one describes the receiving/processing module carried by this node, and the other one stands for the transmitting module. The receiving/processing module is described using a regular vertex, and the transmitting module is represented by a dotted vertex with the same shape. As an example shown in Figure 3.3, this graph model has the same connectivity as the graph model for unicast routing in Figure 3.2.

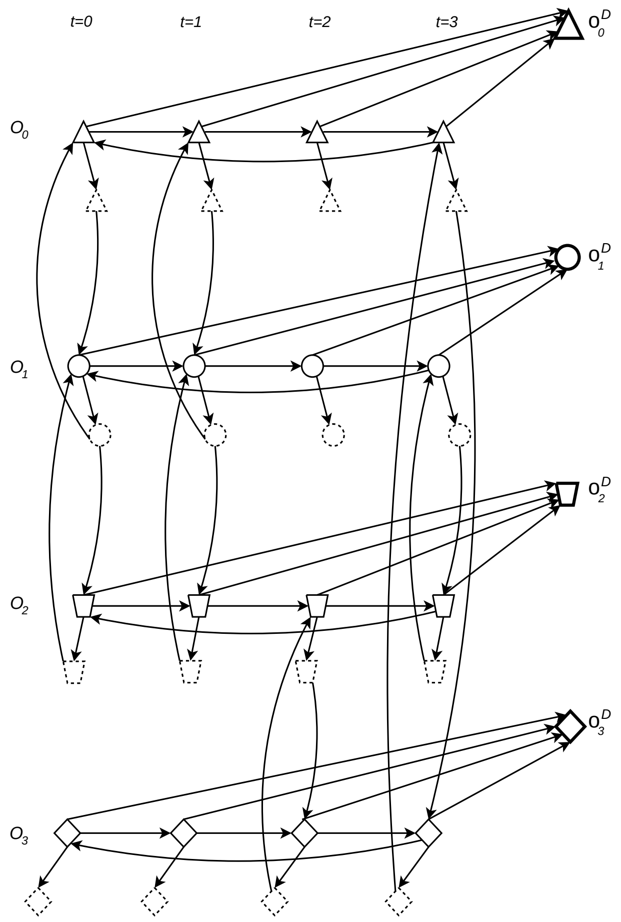


Figure 3.3: A multicast graph model reflecting the encounter patterns in Figure 3.1

There are four nodes whose encounters are represented in the graph. The nodes are each represented by a different shape (triangle, circle, trapezoid and diamond). There is an implicit time axis in this graph, from left to right. The four different solid-line instances of each shape represent the same node during four successive time intervals. That is, the left-hand solid triangle represents node o_0 during period $t = 0$, and the right-hand solid triangle represents o_0 during period $t = 3$.

The horizontal left-to-right directed edge from one solid-line shape to the next instance of the same solid-line shape to the right represents the possibility for a node to buffer a message and carry it forward to successive instants in time. The final edge, from the furthest right-hand instance of a solid-line shape back to the furthest left version of the same solid-line shape represents the possibility for a message to be carried forward in time to the next repetition of the observed behaviours.

In this example, all encounter patterns have a period length of $m = 4$, so the graph has four instances of each solid-line shape. If we had patterns with different period lengths, the graph would have a width equal to the least-common multiple of the periods.

There are four solid-line instances of each shape, and four dotted-line instances. They enable us to differentiate between the node in receiving mode (solid), and the same node in transmitting mode (dotted). For example, the possibility of a transmission from o_0 to o_1 during $t = 1$ is represented by the directed vertical arc from the dotted triangle in interval $t = 1$ to the solid circle in the same interval. Messages can only be transmitted from one node to another during the same interval. Thus, transmission arcs cannot cross intervals. They may, however, be directed vertically either up or down, depending simply on the relative positions where two nodes have been positioned in the graph. The vertical arcs represent the encounters in the binary time series; it is these encounters that make message transmission possible.

Lastly, there is a single "darker solid" lined version of each shape. This is a convenient representation of each node as a potential destination vertex in the graph model. For example, if our goal is to find the minimum delay path for a message from o_0 to o_2, one possibility is to send it from o_0 to o_1 during $t = 1$, and then from o_1 to o_2 during $t = 2$. This route could be found algorithmically by asking for the shortest path (under a few constraints) from o_0 at $t = 0$ to o_2^D, *i.e.*, to the destination version of o_2.

In summary, we use three types of vertices in our graph model: receiving vertices V^R, transmitting vertices V^T and destination vertices V^D. The edges used to connect these vertices can be grouped into four categories:

- Horizontal edges connect vertices with the same shape to represent the possibility for a node to buffer and carry a message from one phase to the next. We denote these edges as $e^H_{t,o}$ where subscript t indicates the phase number, and subscript o defines the node number.

- Internal edges connect a node's receiver to its transmitter. The weight on an internal edge can be used, for example, to represent the energy cost of transmission. These edges are denoted $e^I_{t,o}$.

- Vertical edges link two nodes in the same phase, and represent an encounter between these two nodes. They are denoted $e^V_{t,o1,o2}$, where $o1$ and $o2$ are the indices of the two nodes.

- Destination edges link receivers to their destination vertex to capture the fact the message has been delivered to its destination. They are denoted $e^D_{t,o}$.

We now use this terminology to describe the situation where a node can deliver a message to a set of destinations. Here, we assume that each node at a particular phase either broadcasts the message to all encountered neighbours or does not transmit at all. Using the encounter matrices $M^k, 0 \leq k \leq n-1$, the encounter graph is the weighted directed graph $G = (V, E, W)$ where:

- $V^R = \{v^R_{t,i} \mid 0 \leq i \leq n-1,\ 0 \leq t \leq m-1\} \in V$ denotes the receiving modules for a set of n nodes during a period of length m phases,
- $V^T = \{v^T_{t,i} \mid 0 \leq i \leq n-1,\ 0 \leq t \leq m-1\} \in V$ denotes the transmitting modules for a set of n nodes during a period of length m phases,
- $V^D = \{v^D_i \mid 0 \leq i \leq n-1\} \in V$ denotes destination vertices for a set of n nodes,
- $E^H = \{e^H_{t,i} = (v^R_{t,i}, v^R_{t+1,i})\ \forall o_i \in O,\ 0 \leq t \leq m-1\} \in E$ denotes the horizontal edges,
- $E^V = \{e^V_{t,i,j} = (v^T_{t,i}, v^R_{t,j})\ \forall o_i, o_j \in O,\ 0 \leq t \leq m-1\} \in E$ if and only if $m^i_{jt} = 1\}$ denotes the vertical edges,
- $E^I = \{e^I_{t,i} = (v^R_{t,i}, v^T_{t,i})\ \forall o_i \in O,\ 0 \leq t \leq m-1\} \in E$ denotes the internal edges,
- $E^D = \{e^D_{t,i} = (v^R_{t,i}, v^D_i)\ \forall o_i \in O,\ 0 \leq t \leq m-1\}\} \in E$ denotes the destination edges and
- W represents a function on an edge returning the weight assigned to it.

This graph is generic enough to be used to represent both energy and delay costs simply by adjusting the weights W assigned to each edge in G. We discuss this next.

Communication delay

The encounter graph can be used to solve the minimum delay problem as follows. If a node decides to store the message and carry it on to the next phase to disseminate it in the future, this yields a communication delay equal to τ, *i.e.*, the duration of the current phase p_t. Transmission preparation (the weight for internal edges) yields a small amount of delay between the receiving module and the transmitting module, and signal propagation (the weight for vertical edges) also introduces another delay. However, we can treat both delays as zero because both are embedded within the weight of the horizontal edges τ, which is the duration of each phase. In addition, the weight of destination edges is always zero because they are artificial edges which do not add any delay in the physical environment. Therefore, the weights of the edges for minimum delay routing are (see Figure 3.4):

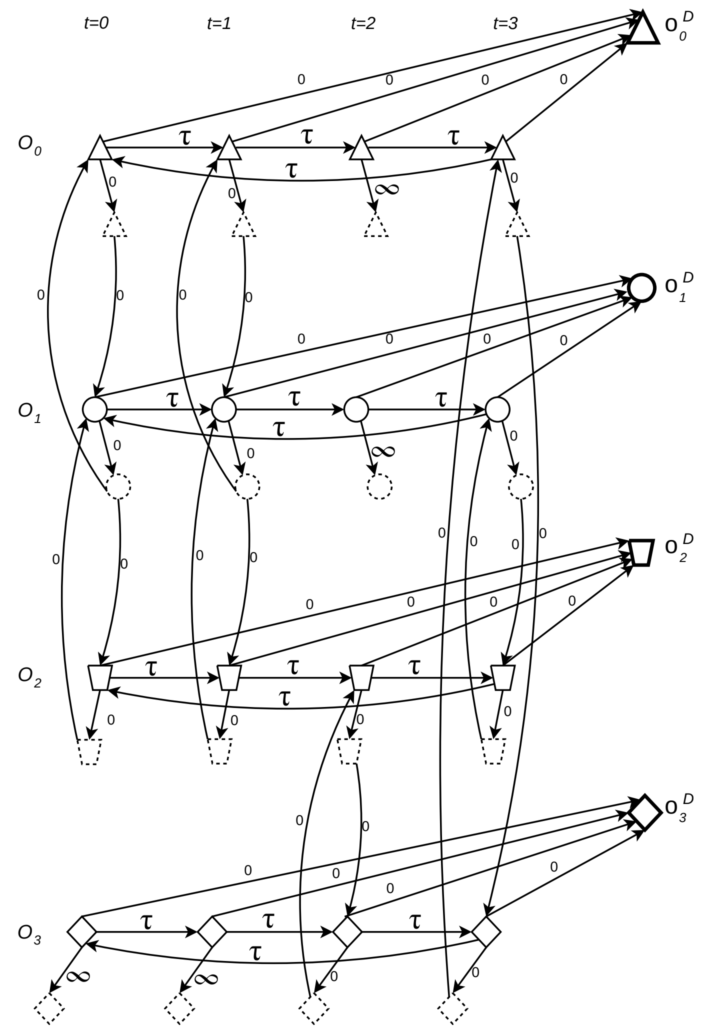


Figure 3.4: Graph model for the minimum delay problem

- $\forall e^H \in E^H$, $W(e^H) = \tau$.
- $\forall e^V \in E^V$, $W(e^V) = 0$.
- $\forall e^I \in E^I$, $W(e^I) = 0$.
- $\forall e^D \in E^D$, $W(e^D) = 0$.

Energy consumption

The encounter graph can also be used to find the minimum energy consumption for routing a message. If a node decides to carry the message on to the next phase and disseminate it in the future, this requires a very small amount of energy for data storage during the current phase p_t. In this chapter, we let ε be the energy cost for a node to store data during a phase with duration τ. If a node carries the message on to the next phase, this results in an energy cost, ε, to store the message. In contrast, if the node decides to transmit the message

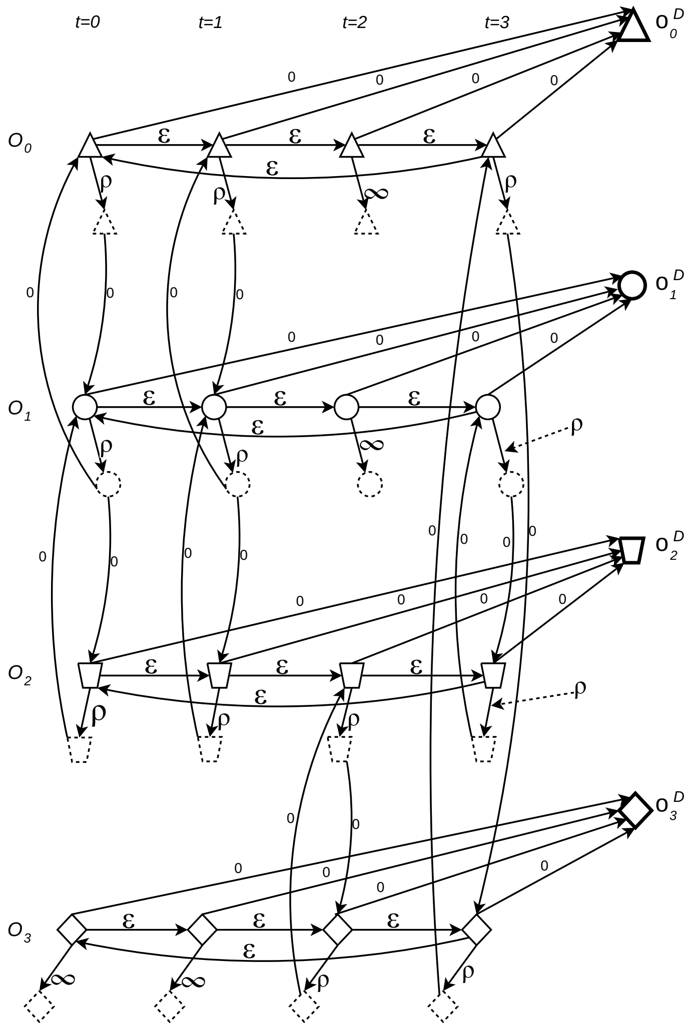


Figure 3.5: Graph model for the minimum energy consumption problem

immediately to a set of encountered nodes during the current phase, a certain amount of energy, depending on the application, is required. In this chapter, we assume homogeneous networks where each transmission costs the same amount of energy, ρ. However, our model can be used to model the case where the energy required to transmit a message between two nodes depend on when and where they meet. For instance, in some encounters the nodes can be closer or further apart. Formally, to model energy costs for routing, the weights of horizontal and vertical edges can be defined as follows (see Figure 3.5):

- $\forall e^H \in E^H$, $W(e^H) = \varepsilon$.
- $\forall e^V \in E^V$, $W(e^v) = 0$.
- $\forall e^I \in E^I$, $W(e^I) = \rho$.
- $\forall e^D \in E^D$, $W(e^D) = 0$.

3.3.3 Domain constraint

In some scenarios, an intermediate node within two consecutive encounters during the same phase may participate in the optimal route. For example, if node o_0 has a packet for node o_2 during phase p_0, the route with the minimum delay in Figure 3.3(a) is $v^R_{0,0} \rightarrow v^T_{0,0} \rightarrow v^R_{1,0} \rightarrow v^T_{1,0} \rightarrow v^R_{2,0} \rightarrow v^D_2$ with zero delay if the propagation delay and processing delay are negligible. In this route, the intermediate node o_1 acts as a bridge for two consecutive transmissions during the same phase. However, because the encounters between node o_1 and nodes o_0 and o_2 both take place during the same phase and the order of encounters is unknown in the graph model, we forbid such type of retransmissions during the same phase. Therefore, we propose a domain-oriented constraint called NOT (NO reTransmissions during the same phase in the graph). However, if a retransmission is required for a node during a particular phase, *e.g.*, node o_1 at phase p_0 with a transmission from node o_0 during the same phase, it has to wait for the same phase during the next repetition of the periodic encounters. As a result, each retransmission costs an additional delay that equals the total period of time of such a periodic pattern. Because of this domain-oriented constraint, conventional shortest path algorithms cannot be used to find optimal routes satisfying the constraint. In the next section, we discuss mathematical optimization and modifications of Dijkstra's algorithm [29] that enforce the NOT constraint in their results for both the minimum delay and minimum energy consumption problems.

3.4 Mathematical optimization and algorithms

One of basic functions of a network is to distribute information to various groups of destinations with the objective to minimize the usage of resources including overall energy consumption or/and communications delay. Given a set of destinations, our objective is to distribute the packet to them using the minimum delay or the minimum energy consumption. We need to find a tree structure rooted at the source that connects desired destinations in the graph. The tree in our graph model is a *Steiner Tree*. Our objective is to find the minimum weighted Steiner Tree that connects the source to all destinations [91, 92].

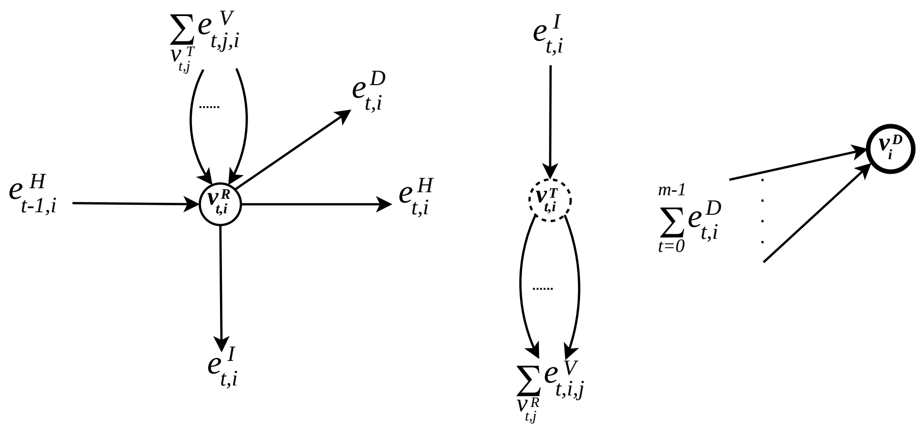
In this section, we introduce two approaches, binary integer programming and optimal/approximate algorithms, to find routes for communications in our graph model. We discuss the multicast version of the problem as the unicast is a special case of multicast.

3.4.1 Binary integer programming

Mathematical programming is a powerful tool that can be used to solve optimization problems. Targeting both the minimum energy consumption and minimum delay problems by assigning different weights on our graph model, the optimal route is a set of sequential edges where the summation of weights on those edges is the minimum. If we treat all edges as variables, an edge either does participate in an optimal route (its value is 1) or does not (its value is 0). This general approach is called the binary integer programming. In this section, we define the formulation to find a minimum weighted Steiner Tree that satisfies our constraints in the graph model.

The minimum delay problem

In the minimum energy consumption problem, our goal is to find the Steiner Tree whose total weight is the minimum. This is because the energy cost on the path from the source to a destination is isolated from other paths. The total weight of a tree is the cumulative cost of all these paths. However, this strategy is incorrect for the minimum delay problem where there are multiple destinations in a packet. The reason is that the time to deliver a packet to one destination may partially overlap the time to deliver the packet to other destinations. For example, as shown in Figure 3.3 a), if node o_2 has a packet for nodes o_0 and o_3 at the beginning of phase p_0, the minimum delay will be 2τ instead of 3τ because the delay (τ) to deliver the packet from o_2 to o_0 is embedded within the delay (2τ) of path from o_2 to o_3. Therefore, when we count the minimum delay, it is incorrect to simply add delays from all paths. This special phenomenon cannot be modelled by linear programming, but we can solve this problem algorithmically; this is discussed in detail in the next subsection.



$\sum_{V^T_{t,j}} e^V_{t,j,i}$
$e^D_{t,i}$
$e^H_{t-1,i}$
$V^R_{t,i}$
$e^H_{t,i}$
$e^I_{t,i}$
$e^I_{t,i}$
$V^T_{t,i}$
$\sum_{V^R_{t,j}} e^V_{t,i,j}$
$\sum_{t=0}^{m-1} e^D_{t,i}$
V^D_i

Figure 3.6: Illustrations for edge notations

The minimum energy consumption problem

In this binary integer program, each edge in the graph has a weight. Each edge either participates in the minimum weighted Steiner Tree or it does not. Therefore, if we set each edge as a variable, this variable has value either one or zero. Because each edge also has a weight, our goal is to minimize the total weight of the Steiner tree being used in order to connect all desired destinations. To define the formulation of all edges, we first have a look at different types of vertices. As we described before, we have three types of vertices in our graph model. Receiving and processing vertices have five categories of edges in either incoming and outgoing directions. Transmitting vertices have only incoming internal edges and outgoing vertical edges. Lastly, destination vertices have only incoming destination edges.

A vertex $v^R_{t,i} \in V^R$ has five types of incoming and outgoing edges as shown in Figure 3.6: a set of vertical incoming edges, a set of horizontal incoming edges, a set of horizontal outgoing edge, a set of vertical outgoing edge to the transmitting module and an edge to v^D_i. The combinations of their corresponding values are also presented in Table 3.5. The total weight of all incoming edges must be zero or one. Otherwise, it violates the definition of a tree structure where there is only one path from the root to every node on the tree. In addition, a vertex $v^R_{t,i} \in V^R$ has four different roles in routing. First, it can be the source vertex. As shown in the seventh column of Table 3.5 where the cross sign represents the invalid combination of values, it has no incoming edge, and the edge to its destination vertex must not be enabled. Second, this type of vertex can be the vertex on the tree which directly connects to its destination vertex. The third role for this type of vertex is to be an intermediate vertex, which does not directly connect to the destination vertex. The last one is a vertex that does not participate to form the tree. Similar to the last two roles, intermediate vertices and vertices which are not on the tree also have some invalid combinations. In Table 3.5, the last column summarizes all invalid scenarios for the last three roles of receiving and transmitting vertices.

A vertex $v^T_{t,i}$ has the following properties shown in Figure 3.6:

1. if $v^T_{t,i} \in V^T$ is an intermediate vertex on the tree, $e^I_{t,i} = 1$ while $\sum_{v^R_{t,k}} e^V_{t,i,k} \geq 1$ for all $o_k \in O$ if $m^i_{kt} = 1$.

2. if $v^T_{t,i} \in V^T$ is not on the tree, $e^I_{t,i} = 0$ while $\sum_{v^R_{t,k}} e^V_{t,i,k} = 0$ for all $o_k \in O$ if $m^i_{kt} = 1$.

A vertex c^D_i has the following properties shown in Figure 3.6:

$$y = \sum_{e \in E} w(e) \cdot e$$

subject to

$$e = \begin{cases} 1 & \text{if used in the minimum Steiner tree} \\ 0 & \text{otherwise} \end{cases} \quad (C_1)$$

$$\sum_{t=0}^{m-1} e^D_{t,i} = \begin{cases} 1 & \text{if } v^D_i \in D \\ 0 & \text{otherwise} \end{cases} \quad (C_2)$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} + e^H_{t-1,i} + e^D_{t,i} = 0 \text{ where } v^R_{t,i} = v^R_{st,s} \quad (C_3)$$

$$e^I_{t,i} + e^H_{t,i} \geq 1 \text{ where } v^R_{t,i} = v^R_{st,s} \quad (C_4)$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} \leq 1 \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_5)$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} + e^H_{t-1,i} \leq e^I_{t,i} + e^H_{t,i} + e^D_{t,i} \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_6)$$

$$e^I_{t,i} \leq e^H_{t-1,i} \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_7)$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} + e^H_{t-1,i} \geq e^H_{t,i} \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_8)$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} \geq e^D_{t,i} \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_9)$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} + e^H_{t-1,i} \leq e^I_{t,i} + 1 \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_{10})$$

$$\sum_{v^T_{t,j}} e^V_{t,j,i} + e^H_{t-1,i} \leq e^H_{t,i} + 1 \text{ where } v^R_{t,i} \neq v^R_{st,s} \text{ and } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_{11})$$

$$e^I_{t,i} \leq \sum_{v^R_{t,j}} e^V_{t,i,j} \text{ where } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \quad (C_{12})$$

$$e^I_{t,i} \geq e^V_{t,i,j} \text{ where } \forall o_i, o_j \in O \text{ and } 0 \leq t \leq m-1 \text{ if } m^i_{jt} = 1 \quad (C_{13})$$

Figure 3.7: Binary Integer program for routing in our graph model

1. $\sum_{t=0}^{m-1} e_{t,i}^{D} = 1$ if v_i^D is one of the destinations

2. $\sum_{t=0}^{m-1} e_{t,i}^{D} = 0$ if v_i^D is not one of the destinations

To enforce all properties regarding different types of vertices, we have the formulation shown in Figure 3.7. Given a directed graph G, let $w(e)$ be the cost of an edge $e \in E$. Given a source vertex $v_{st,s}^{R}$ and a set of destination vertices D, we present our BIP formulation for the minimum edge-weighted directed Steiner tree problem.

Table 3.5: Combinations of edge values to identify $i \in V^R$

Constraint	$\sum_{v_{t,j}^T} e_{t,j,i}^{T}$	$e_{t-1,i}^{H}$	$e_{t,i}^{I}$	$e_{t,i}^{H}$	$e_{t,i}^{D}$	$v_{st,s}^{R}$	$v_{t,i}^{R} \in V^R$
	0	0	0	0	0	$\times$	$\surd$
C_9	0	0	0	0	1	$\times$	$\times$
C_8	0	0	0	1	0	$\surd$	$\times$
C_8, C_9	0	0	0	1	1	$\times$	$\times$
C_7	0	0	1	0	0	$\surd$	$\times$
C_7, C_9	0	0	1	0	1	$\times$	$\times$
C_7, C_8	0	0	1	1	0	$\surd$	$\times$
C_7, C_8, C_9	0	0	1	1	1	$\times$	$\times$
C_6	0	1	0	0	0	$\times$	$\times$
C_9	0	1	0	0	1	$\times$	$\times$
	0	1	0	1	0	$\times$	$\surd$
C_9	0	1	0	1	1	$\times$	$\times$
	0	1	1	0	0	$\times$	$\surd$
C_9	0	1	1	0	1	$\times$	$\times$
	0	1	1	1	0	$\times$	$\surd$
C_9	0	1	1	1	1	$\times$	$\times$
C_6	1	0	0	0	0	$\times$	$\times$
	1	0	0	0	1	$\times$	$\surd$
	1	0	0	1	0	$\times$	$\surd$
	1	0	0	1	1	$\times$	$\surd$
C_7	1	0	1	0	0	$\times$	$\times$
C_7	1	0	1	0	1	$\times$	$\times$
C_7	1	0	1	1	0	$\times$	$\times$
C_7	1	0	1	1	1	$\times$	$\times$
C_6	1	1	0	0	0	$\times$	$\times$
C_6	1	1	0	0	1	$\times$	$\times$
C_6	1	1	0	1	0	$\times$	$\times$
C_{10}	1	1	0	1	1	$\times$	$\times$
C_6	1	1	1	0	0	$\times$	$\times$
C_{11}	1	1	1	0	1	$\times$	$\times$
	1	1	1	1	0	$\times$	$\surd$
	1	1	1	1	1	$\times$	$\surd$

There is a total of 13 constraints in our formulation.[1]

- Constraint C_1 treats each edge as a variable to determine whether that edge is on the optimal Steiner tree.
- Constraint C_2 controls destination vertices. All desired destination vertices on the tree must have one incoming edge. Other destination vertices that do not belong to the destination set D must not have any incoming edge at all.
- Constraints C_3 and C_4 are used to constrain the source. A source has no incoming edge (see C_3). In addition, it must have at least one outgoing edge except the destination edge (see C_4).
- Constraints C_5 to C_{11} are used to regulate receiving and processing vertices. For example, constraint C_5 requires that every receiving and processing vertex can only have one incoming vertical edge.
- Constraints C_{12} and C_{13} are used to control each transmitting vertex. If its internal edge is on the tree, at least one of the vertical outgoing edges must also be on the tree. Otherwise, both incoming and outgoing edges must not be included on the tree.

Even though we have discussed the domain-oriented constraint in the previous section, no constraint in our formulation is dedicated to it. However, if we have a close look at Table 3.5, our domain-oriented constraint is enforced by combining different constraints in the formulation. We know that in a logical tree, each vertex has only one incoming edge and at least one outgoing edge. However, if we look at two valid scenarios presented by the last two rows in Table 3.5, both incoming vertical and horizontal edges can be included on the tree. It seems that they are contradictions to the definition of a logical tree, but this is valid because of our domain constraint and the repetition of periodic encounters. For example, if a node receives a packet from another node (one of the incoming vertical edges is enabled), this node can store the packet and transmit it in the future. If the best time to transmit this packet is at the same phase during the next period, as a result, both incoming horizontal and vertical edges will be included on the tree.

3.4.2 Optimal and approximate algorithms

We have to treat the minimum energy consumption and minimum delay problems differently. This is because the nature of time in the minimum delay problem is quite different

[1]Table 3.5 provides details about the functionalities of each constraint in this group. In that table, cross signs represent violations; check signs represent valid scenarios.

from that of energy usage in the minimum energy problem. In this section, we present approaches that address these two problems separately.

1) The minimum delay problem:

The minimum delay problem in routing is relatively simple. The main result can be found in the following theorem.

Theorem 1. *The minimum delay in routing for a set of destinations equals the maximum of all pair-wise minimum delays between the source and destinations.*

Proof. We prove this theorem by induction. Let us use d to represent the number of destination nodes in the graph.

If $d = 1$, then a path with minimum delay from the source to that node must also yield the the minimum delay for unicast routing.

If $d = 2$, we can pick one node first and find a path with the minimum delay from the source to it. Then, we find a path with the minimum delay from the source to the second node. If the delay to reach the second node is less than the delay to reach the first node, the delay to reach the second node must be embedded in the delay to reach the first node. Otherwise, the minimum delay to reach the second node must include the delay to reach the first node. Therefore, in either case, the maximum of the minimum pair-wise path delays among two nodes is the minimum delay in multicast routing.

Let us assume our theorem is true when $d = n - 1$.

If d equals n, for every combination of $n - 1$ destination nodes, the minimum delay to reach those $n - 1$ destinations is the maximum of all minimum path delays between the source and $n - 1$ destinations. By introducing one extra destination node, we need to show that the theorem statement holds as well. We can now use the same argument for the case where $d = 2$ to complete the induction argument. □

From our theorem, many classical shortest path algorithms can be used to solve the minimum delay problem with small modifications. The algorithm used in this chapter for the minimum delay problem is a modification of Dijkstra's algorithm [25]. Its pseudo-code is presented as Algorithm 1. There is one difference with respect to the original algorithm in [25]: the if-statement at lines 18 to 21 is inserted to enforce the NOT constraint. This if-statement does not re-direct the optimal route; it only adjusts the weight of related vertices to avoid transmissions that violate the NOT constraint. For example, in line 19, if the predecessor and successor of the current vertex on the route are from the same phase, it is a violation of the NOT constraint. If the retransmission must be part of the optimal route, that means the edge in the graph for the retransmission represents the only encounter between both intermediate nodes. In other words, there is only one path between the source and the destination. Because of our domain-oriented constraint, the retransmission to its successor has to wait, or the packet has to be held until the same phase in the next period. As shown in line 20, data storage requires an additional energy cost, $\varepsilon \times m$ units of energy, to store the message for a whole period with m phases. The principle of the modification in Algorithm 1 is: whenever there is a violating retransmission, we eliminate the retransmission by having the node to store the packet until the same encounter at the next period in time. Because the

Algorithm 1 ModifiedDijkstra(G, D, w, s)

```
1: for each vertex $v \in V[G]$ do
2:     $d[v] \leftarrow \infty$          //initialize the distance to be infinite
3:     $p[v] \leftarrow null$       //set the predecessor to be empty
4: end for
5: $d[s] \leftarrow 0$          //initialize the source distance
6: $S \leftarrow \emptyset$
7: $Q \leftarrow V[G]$              //initialize the priority queue
8: while $Q \neq \emptyset$ do
9:     $u =$ EXTRACt-MIN(Q)
10:    if $u \in D$ then
11:        $D \leftarrow D \backslash \{u\}$
12:    end if
13:    if $D == \emptyset$ then
14:        break
15:    end if
16:    for each vertex $v \in Adj[u]$ do
17:        if $d[v] > d[u] + w(u,v)$ then
18:            //check whether the predecessor/successor and the current vertex are from the
               same phase
19:            if phaseOf($v$) == phaseOf($p[u]$)
                  && phaseOf($v$) == phaseOf($u$) then
20:                $d[v] \leftarrow d[u] + \varepsilon * m$
21:            else
22:                $d[v] \leftarrow d[u] + w(u,v)$
23:            end if
24:            $p[v] \leftarrow u$
25:        end if
26:    end for
27: end while
28: // retrieveTree function retrieves the tree by traversing the predecessor recorded at each
    destination back to the source
29: T = retrieveTree(G,D,s)
30: return T
```

cost of storing the message has already been included in the final cost in Algorithm 1, the route returned is an optimal route that does not violate the NOT constraint. Note that if we remove lines 18 to 21 from the algorithm, this part of the algorithm is exactly the same as Dijkstra's algorithm. In other words, the path to each destination generated by the algorithm which satisfies our constraint is guaranteed to be the optimal route, and furthermore, it can be found in polynomial time based on the complexity of Dijkstras algorithm. In summary, our modified Dijkstra's algorithm finds a shortest-path tree for a given set of destinations. Based on Theorem 1, the minimum delay to reach all destinations equals the weight of the longest shortest path on the tree.

In order to prove the correctness of Algorithm 1, we prove that the tree returned from that modified algorithm is the shortest path tree.

Theorem 2. *Given a source and a set of destinations, Algorithm 1 returns a shortest path tree satisfying the NOT constraint.*

Proof. As we discussed before, the only difference between Algorithm 1 and Dijkstra's algorithm is that we enforce the NOT constraint from line 18 to 21. As a result, Algorithm 1 only examines paths that satisfy the NOT constraint, and does not change the logic of Dijkstra's algorithm. Therefore, the path to each one of the destinations returned by Algorithm 1 satisfies the NOT constraint, and must be the shortest based on the correctness of Dijkstra's algorithm. Because all optimal paths originate from the same source, if we combine those optimal paths, the tree returned must also be a shortest path tree satisfying the NOT constraint. □

Using Theorems 1 and 2, the weight of the longest path between the source and any destination returned by Algorithm 1 is the minimum delay in order to deliver a packet to multiple destinations.

It is obvious that the if-statement from line 18 to 21 only updates the weight of the corresponding vertices. It does not bring additional complexity into the algorithm. Therefore, based on the complexity of Dijkstra's algorithm, our algorithm has polynomial complexity.

2) The minimum energy consumption problem:

As we mentioned before, the minimum energy consumption problem can be solved by finding the minimum weighted Steiner Tree in the graph where the destination vertices are leaf nodes [101]. However, the minimum weighted Steiner Tree problem is a classical NP-hard problem [41] and has been well studied [62, 63, 138]. The minimum weighted Steiner tree problem can be further divided into two categories: node-weighted Steiner tree and edge-weighted Steiner problem. The minimum energy consumption problem in our graph model falls into the second category. In graph theory, the minimum weighted Steiner tree problem is NP-hard in general; however, studies have shown that the Steiner tree problem can be solved in polynomial time in many special graph types such as Series-Parallel graphs, Halin graphs, K-planer networks and strongly chordal graphs [121, 138]. Unfortunately, our graph model does not belong to any one of these graph types.

Previous works have presented both exact algorithms and approximation algorithms that are based on some heuristics. The exact algorithms that have been proposed are all exponential algorithms [30, 47, 84]. They are not practical for graph models with a large number of vertices. Therefore, we focus on approximation algorithms which can find the result faster, although without the guarantee of optimality. Many works have been conducted on heuristics [77, 108, 114, 127]. The most well-known heuristics are from Kou et al. [77] and Takahashi and Matsuyama [127]. In their paper, Kou et al., used the minimum spanning tree heuristic on source and destinations; Takahashi and Matsuyama used

Algorithm 2 Shortest Path Heuristic to find minimum weighted directed Steiner tree(G, D, w, s)

```
1:  T ← {s}          //initialize the result tree
2:  shortestPathTree = null
3:  while D ≠ ∅ do
4:      minCost ← ∞
5:      P ← ∅          //the set used to hold the selected path
6:      $\hat{v}$ ← $null$
7:      $\hat{d}$ ← $null$
8:      for each vertex v ∈ T do
9:          if shortestPathTree[v] == null then
10:             shortestPathTree[v]←ModifiedDijkstra(G,D,w,v)
11:             for each destination d ∈ D do
12:                 if cost(path(v,d)∈shortestPathTree[v])<minCost then
13:                     minCost = cost(path(v,d))
14:                     $\hat{v}$ ←v
15:                     $\hat{d}$ ←d
16:                     P←path(v,d)
17:                 end if
18:             end for
19:         end if
20:     end for
21:     T←T∪P\{$\hat{v}$}
22:     D←D\{$\hat{d}$}
23: end while
24: return T
```

the minimum cost paths heuristic. Both algorithms can achieve a $O(kn \log n)$ complexity where k is the number of destinations; however, these algorithms do not work on directed graphs.

In order to find a heuristic for directed graphs, we decided to modify previous heuristic algorithms for undirected graphs. The minimum spanning tree heuristic cannot be applied to our graph model because the destination vertices has no outgoing edges. The algorithm cannot construct a complete graph among the source and all destination vertices. Therefore, we focus on the minimum cost path heuristic.

In the minimum path cost heuristic [127], the algorithm first finds a path from the source to one of the destinations that has the minimum cost among all destinations. Using the path as the base of a tree, the algorithm selects the next destination that is the closest to the base tree and merges the selected path to that tree. The algorithm keeps doing this until the last destination vertex has been included in the tree. This is why the algorithm has $O(kn \log n)$ complexity. There are k destinations in the tree, and each one of them is required to run Dijkstra's algorithm once. As we just mentioned, the destination vertices in our graph model have no outgoing edge. Thus, there is no path to any vertex in the graph by using the destination vertex as the source. To change that, we make a small modification from the original algorithm. Instead of running Dijkstra's algorithm from destination vertices, the next closest destination vertex can be used by running Dijkstra's algorithm from all intermediate vertices on the result tree. A detailed algorithm is described in Algorithm 2. In the algorithm, we introduce a storage space, *shortestPathTree*, to store the shortest-path tree rooted at each intermediate vertex that is returned by our modified Dijkstra algorithm in Algorithm 1. In addition, *path(v,d)* represents the shortest path from vertex v to destination d in the shortest-path tree rooted at v. In summary, Algorithm 2 incrementally builds the heuristic tree by inserting a path to one of the destinations at a time.

In the following analysis, we show that our algorithm is a k-approximation algorithm in the worst case.

Theorem 3. *Given any directed graph $G = (V, E)$, a source vertex s and a set of destinations D where $D \subseteq V$ and $|D| = k$, Algorithm 2 always provides a solution to the minimum weighted directed Steiner tree problem that is at most k times the optimal solution.*

Proof. According to Algorithm 2, there is an order among all destinations for the sequence that each one has been added into the tree. For simplicity, we name the destination vertices as $d_1, d_2, \cdots d_k$. A destination vertex with a smaller index is inserted into the result tree earlier. Correspondingly, the same sequence can be used to name the destination vertices in the optimal directed Steiner tree.

Given the optimal tree for the minimum weighted directed Steiner tree problem for k destinations, T^{OPT}, let $c^{OPT}(u,v)$ be the weight of the path between vertices u and v on the optimal tree. For the tree that is incrementally built by Algorithm 2, let $c(x,y)$ be the weight of the path between vertices x and y. Vertices without OPT notation are in the tree built by Algorithm 2, and the ones with OPT notation are from the optimal tree.

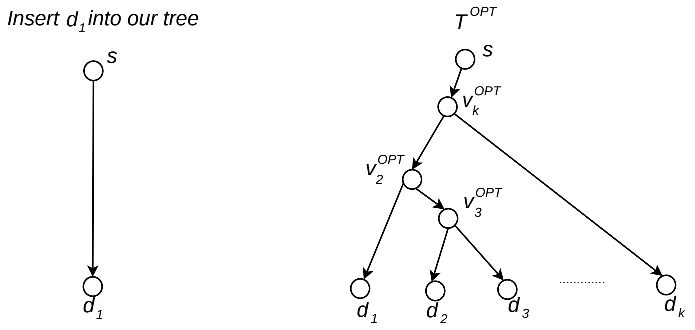


Figure 3.8: Partial tree T_1 and T^{OPT}

At the beginning, our algorithm creates a path with the smallest cost connecting the source and one of the destinations. As a result, our partial tree T_1 is a single path as shown in the left part of Figure 3.8 [2].

Because the path selected by Algorithm 2 is the shortest path between the source and any one of the desired destinations, the following inequality holds.

$$c(s,d_1) \leq c^{OPT}(s,d_1) \tag{3.2}$$

Next, the algorithm adds another destination d_2 into the tree. For illustration, please refer to Figure 3.9. When our algorithm adds the second destination d_2 into the tree, there is an intermediate vertex on the path (s,d_1) in the left part of Figure 3.8 that has the smallest cost to connect d_2. Let us call this vertex v_2. Both v_2 and d_2 in this example can be mapped to $\hat{v}$ and $\hat{d}$ in Algorithm 2 at each iteration. Similarly, there is a corresponding vertex on the optimal tree, T^{OPT}, that connects d_2 to the path between s and d_1. Let us call it v_2^{OPT}. Because the path between v_2 and d_2 has the smallest cost over all paths between s and d_2, we have the following inequality.

[2]All the figures in this proof are for demonstration only, vertices such as v_2, v_2^{OPT} and so on can be any vertex on the corresponding trees.

$$\begin{aligned} c(v_2, d_2) &\leq c(shortestPath(s, d_2)) \\ &\leq c^{OPT}(s, v_2^{OPT}) + c^{OPT}(v_2^{OPT}, d_2) \end{aligned} \tag{3.3}$$

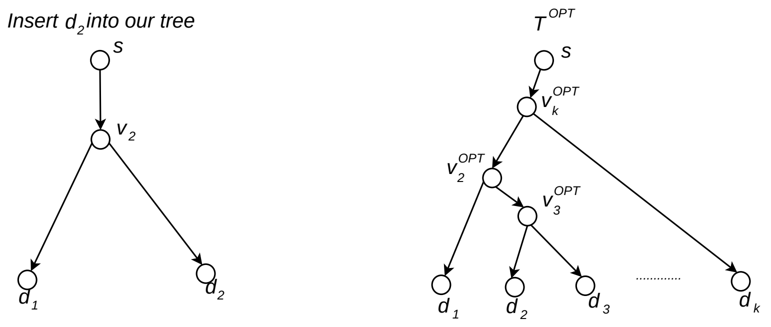


Figure 3.9: Partial tree T_2 and T^{OPT}

When our algorithm adds the third destination d_3, with the same argument before, there are vertices v_3 and v_3^{OPT} in our tree and the optimal tree connecting d_3. Because of the shortest path heuristic in Algorithm 2, the cost between path v_3 and d_3 in our tree must be less than or equal to the any path between s and d_3 in the graph. Therefore,

$$\begin{aligned} c(v_3, d_3) &\leq c(shortestPath(s, d_3)) \\ &\leq c^{OPT}(s, v_3^{OPT}) + c^{OPT}(v_3^{OPT}, d_3) \end{aligned} \tag{3.4}$$

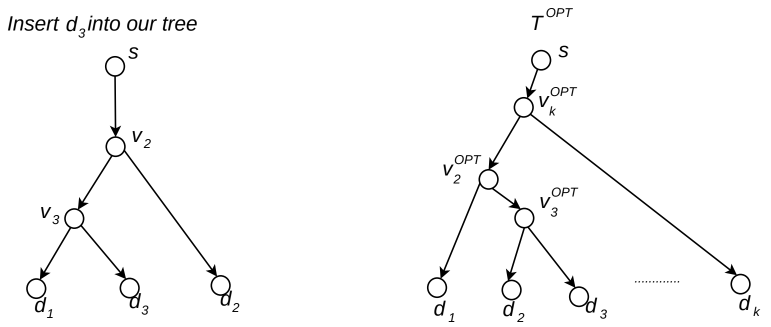


Figure 3.10: Partial tree T_3 and T^{OPT}

The same analysis applies to each insertion of one destination. When the last destination d_k is inserted, we have the inequality shown in Equation 3.5.

$$\begin{aligned} c(v_k, d_k) &\leq c(shortestPath(s, d_k)) \\ &\leq c^{OPT}(s, v_k^{OPT}) + c^{OPT}(v_k^{OPT}, d_k) \end{aligned} \tag{3.5}$$

After including all destinations, the cost of our tree, T, is:

$$c(T) = c(s, d_1) + c(v_2, d_2) + \cdots + c(v_k, d_k) \tag{3.6}$$

Similarly, the cost of the optimal tree, T^{OPT}, is:

$$\begin{aligned} c^{OPT}(T) &= c^{OPT}(s, d_1) + c^{OPT}(v_2^{OPT}, d_2) \\ &\quad + \cdots + c^{OPT}(v_k^{OPT}, d_k) \end{aligned} \tag{3.7}$$

By applying the inequalities in Equations 3.2, 3.3, 3.7 and so on, we have:

$$\begin{aligned} c(T) &\leq c^{OPT}(s, d_1) + c^{OPT}(s, v_2^{OPT}) + c^{OPT}(v_2^{OPT}, d_2) \\ &\quad + \cdots + c^{OPT}(s, v_k^{OPT}) + c^{OPT}(v_k^{OPT}, d_k) \\ &= c(T^{OPT}) + \sum_{i=2}^{k} c^{OPT}(s, v_i^{OPT}) \end{aligned}$$

and

$$\frac{c(T)}{c(T^{OPT})} \leq 1 + \frac{\sum_{i=2}^{k} c^{OPT}(s, v_i^{OPT})}{c(T^{OPT})} \tag{3.8}$$

Therefore, our algorithm is a $1 + \frac{\sum_{i=2}^{k} c^{OPT}(s, v_i^{OPT})}{c(T^{OPT})}$ approximation where $c(s, v_i^{OPT})$ is the shared weight with the existing subtree in the optimal solution for a destination when it is added into the tree. There are two extreme cases for this shared weight. First, the newly added destination does not use any edge from existing subtree; second, the newly added destination fully utilize edges on existing subtree to be reachable by the source s. Therefore, it is easy to see that $0 \leq c^{OPT}(s, v_i^{OPT}) \leq c(T^{OPT})$. In the worst case scenarios where all $c^{OPT}(s, v_i^{OPT}) = c(T^{OPT})$, we have

$$\frac{c(T)}{c(T^{OPT})} \leq 1 + \frac{(k-1) * c(T^{OPT})}{c(T^{OPT})} = k$$

As a result, $c(T) \leq k * c(T^{OPT})$ and Algorithm 2 is a k-approximation algorithm. □

In Algorithm 2, almost every vertex on the result tree has to run the modified Dijkstra's algorithm in Algorithm 1 using itself as the source. Only vertices from the path to connect the last destination do not need to do that. Therefore, given the resulting tree from our algorithm with $|T_k|$ vertices, the running complexity of our algorithm is at most $O(|T_k|n\log n)$.

In 1997, Zelikovsky showed that there exists a l-restricted (level-restricted) Steiner tree that can achieves an approximation guarantee of factor $k^{\frac{1}{l}}$ to the optimal tree for directed *acyclic* graphs [142]. To generalize and extend that study to arbitrary directed graphs, Charikar *et al.* presented a $l(l-1)k^{\frac{1}{l}}$-approximation algorithm with the running time of $O(n^l k^{2l})$ for any tree level $l \in [1,n]$, where n is the number of vertices, and k is the number of terminals [21]. An improvement on this approach, resulting in a time complexity of $O(n^l k^l + n^2 k + nm)$, where m is the number of edges was presented by Hsieh et al. in [55].

Compared to previous results, our algorithm has several advantages. First of all, our bound of k is the worst case, and the results from our algorithm on most cases are much closer to the optimum and sometimes are the optimum. Secondly, our algorithm is simple. Our k-approximation runs in $O(n\log n)$ time complexity, which is dominated by the complexity of Dijkstra's algorithm, and can be implemented more efficiently using data structures such as binary heap and Fibonacci heap. In other studies, the algorithm has a running time of $O(n^2k^2)$ even for a 2 level restricted Steiner tree. Lastly, our approximation only depends on one parameter, k, whereas the $l(l-1)k^{\frac{1}{l}}$ approximation mainly depends on the factor of $l(l-1)$. In some graphs (like the ones in our graph mode), there is no guarantee about the depth of the returned l-restricted Steiner tree. In other words, there may not exist l-restricted Steiner tree with a small number l. Therefore, as the returned level l increases, the factor $l(l-1)$ will become worse. For example, if k is equal to 5, a 3-restricted Steiner tree with a factor $3(3-1)5^{\frac{1}{3}}$ is worse than ours with a factor of 5.

3.4.3 Unicast and broadcast versus multicast

In the previous discussion, distributing packets aiming at a set of destinations is called *multicast* whereas *unicast* and *broadcast* are just special cases of multicast. When there is only one destination, finding the minimum Steiner tree requires finding the shortest path between the source and the destination. Both the minimum delay and minimum energy consumption problems in unicast routing are relatively easy to solve by using Algorithm 1. To achieve broadcast in our model, our objective is to connect the source to all destination vertices with the minimum Steiner tree rather than finding the minimum spanning tree to connect all vertices in the graph. With some variations, our graph models can also solve unicast and broadcast routing by treating them as special cases of multicast routing.

3.5 Performance evaluation

In order to evaluate our method, we propose two sets of experiments where one set uses synthetic traces, and the other one uses real traces.

3.5.1 Evaluation metrics

The complexity of the network is controlled by two different parameters: the number of nodes in the network and each node's radio range. Increasing the number of nodes will increase the number of vertices and horizontal edges. In addition, increasing the radio range increases the coverage area of each sensor, which leads to more encounters. In this section, we first present experiments using synthetic traces in which we evaluate the effectiveness and efficiency of our algorithms with respect to two parameters: (1) the number of nodes, n, and (2) the radio range, r. In addition, a third parameter that is the number of destinations in a packet, d, is introduced for multicast routing. Next, experiments using real mobile traces are presented. We compare our algorithm to those results obtained via two straightforward delay tolerant network routing protocols, *epidemic* [129] and *direct delivery* [119] protocols, because both protocols exhibit optimal bounds on either delivery ratio or network resources. In order to make a fair comparison, we make the assumption that there are unlimited buffers and energy supply in each node to guarantee the maximum delivery ratio.

Two metrics are examined in our experiments: (1) delivery delay and (2) energy cost. Regarding both metrics, the lower their values are, the better the performance is. In addition, in order to have accurate results, each measurement is calculated by taking the average of each metric from running 100 packets individually with different sources and destinations. In our simulation, we arbitrarily choose an integer number as the base unit to calculate the energy costs at different radio range. However, the energy costs in real-life applications can be obtained based on different transmission standards [38].

In our experiments, we evaluate both unicast and multicast communications. This is because in some application one type of communications may be preferred over the other.

3.5.2 Experiments using synthetic traces

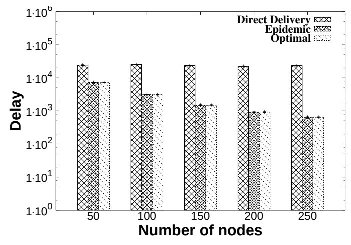
First, we examine the running time of our algorithm. Our experiments are implemented in Java with JDK 1.6. All mobility traces are generated using NS2 [3] random way-point mobility model. Because this mobility model is complex and takes far too long to generate traces for a large number of nodes, the maximum number of nodes used in our experiments is 250. In addition, experiments are conducted within a 400×400 square meter area and all nodes have a radio range up to 50 meters. Finally, we make the assumption that transmitting a message is much more expensive than simply holding on to a message during a phase. This is related to how one assigns cost to the vertical and horizontal edges in our graph model, respectively. We set the ratio of the energy cost to transmit a message over the energy cost to store-and-carry a message to be 1000:1. In our experiments, the energy cost grows quadratically with the radio range [39]. In addition, the duration of each phase is set to be 60 seconds.

Unicast

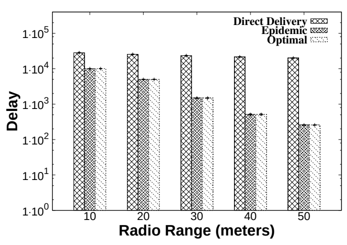
In unicast routing, we focus on a special case of multicast where there is one destination. As we mentioned earlier, both the number of nodes in the network, n, and node's radio range, r, are controlled in our experiments. By default, $n = 150$ and $r = 30$. The performance of our algorithm is presented in Figure 3.11 that contains two sets of experiments. The first column presents experimental results with varying n while maintaining a fixed 30 meters radio range. The second column shows results for varying radio range while keeping 150 nodes in the network. We compare our algorithm with direct delivery and epidemic routings. In order to be as fair as possible to epidemic routing we just accounted for the energy and delay spent until a route is found. This applies to the rest of our experiments.

Figures 3.11(a) and (b) show the end-to-end delay obtained by running our algorithm. Recall that both our approach and the epidemic routing provably yield the optimal delay, thus a comparison is not relevant in this case. As expected, with more nodes (or with greater radio range) the larger the number of encounters and the higher the probability that a shorter route is found. This causes decreasing trends in all three approaches in both experimental settings. However, the direct delivery method has the highest delay because the source has to wait until it meets the destination directly. In addition, this method does not have obvious decreasing trends because of the random mobility model where the probabilities to meet other nodes do not vary significantly.

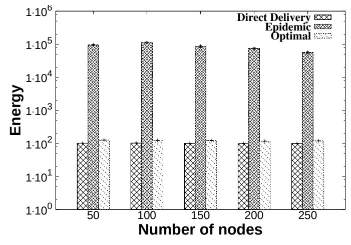
Figure 3.11(c) shows that the obtained energy cost is fairly insensitive to the the number of nodes in the network. It turns out that for a given default radio range, all packets can be optimally delivered in a fairly small number of hops. We believe this is an artifact, rather than a characteristic, due to the size of the area of interest. In this experiment, direct delivery uses the least amount of energy because only one transmission is required to deliver the packet whereas epidemic protocol costs the most because of flooding the packet to



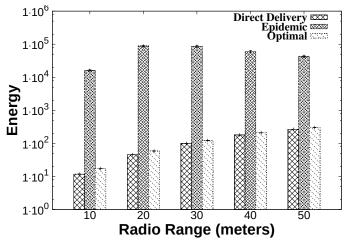
(a) Varying n while $r = 30$



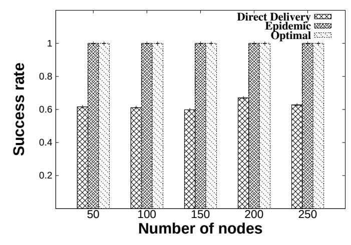
(b) Varying r while $n = 150$



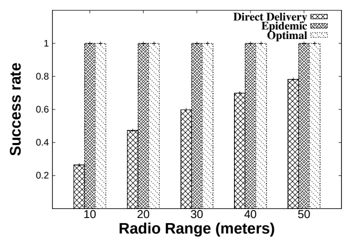
(c) Varying n while $r = 30$



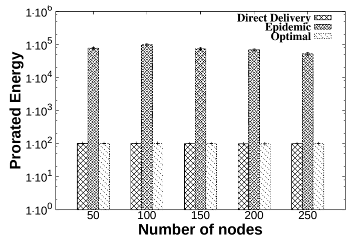
(d) Varying r while $n = 150$



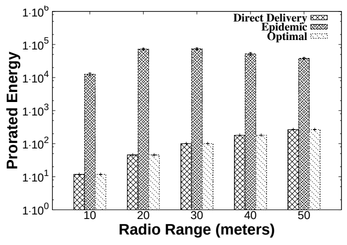
(e) Varying n while $r = 30$



(f) Varying r while $n = 150$



(g) Varying n while $r = 30$

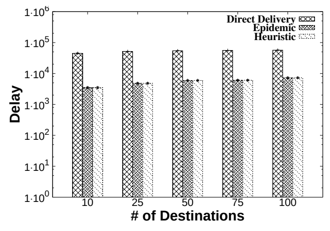


(h) Varying r while $n = 150$

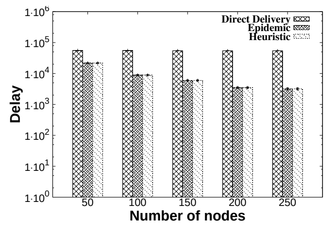
Figure 3.11: Unicast performance regarding delay and energy cost

every encountered node that does not have a copy of the packet. Our algorithm has a performance very similar to that of direct delivery. When the radio range increases as shown in Figure 3.11(d), the energy cost increases for both direct delivery and our algorithm.

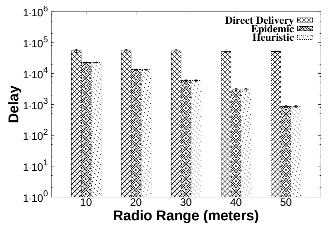
This may sound contradictory as larger radio ranges lead to more encounters and thus more chances of creating better routes. However, as in the case of Figure 3.11(c), the actual number of hops in the optimal is fairly constant. The curves goes up due to the well known fact that the energy cost increases quadratically (at least) with the radio range. This increase



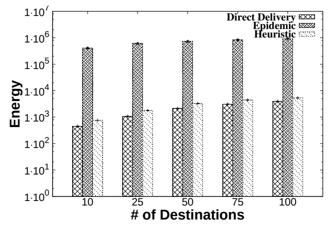
(a) Varying d while $n = 150$ and $r = 30$



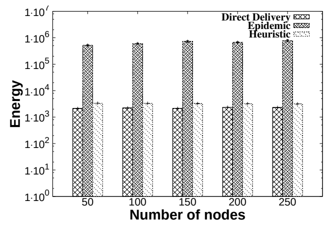
(b) Varying n while $r = 30$ and $d = 50$



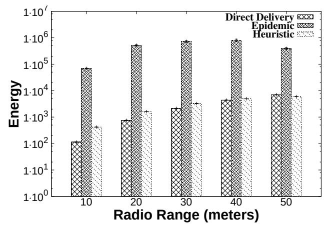
(c) Varying r while $d = 50$ and $n = 150$



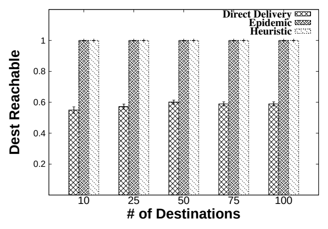
(d) Varying d while $n = 150$ and $r = 30$



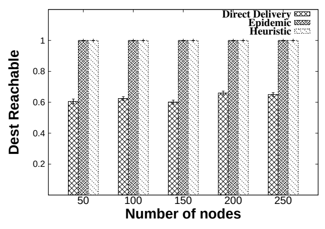
(e) Varying n while $r = 30$ and $d = 50$



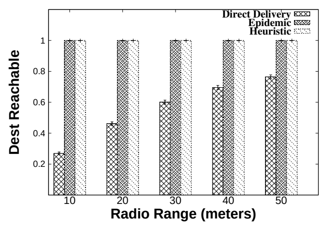
(f) Varying r while $d = 50$ and $n = 150$



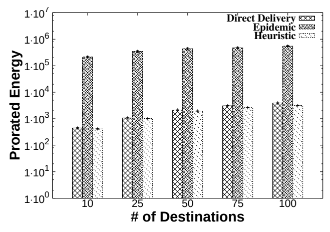
(g) Varying d while $n = 150$ and $r = 30$



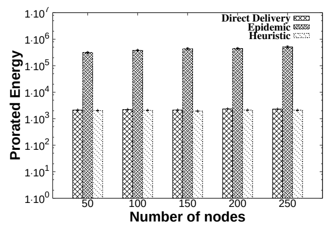
(h) Varying n while $r = 30$ and $d = 50$



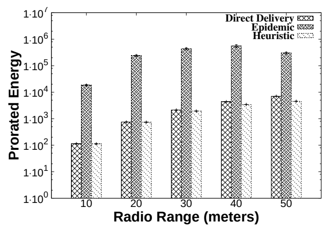
(i) Varying r while $d = 50$ and $n = 150$



(j) Varying d while $n = 150$ and $r = 30$



(k) Varying n while $r = 30$ and $d = 50$



(l) Varying r while $d = 50$ and $n = 150$

Figure 3.12: Multicast performance regarding delay and energy cost

in energy usage itself dominates any energy savings by having better routes. In contrast, the energy cost for epidemic algorithm increases at the first, then it drops after a certain point of time. This is because the increasing usage of energy first dominates the performance, then as the number of encounters increases, the saving from having better routes dominates the energy cost. If we keep increasing the radio range to a certain point of time where all nodes encounter each other during the same phase, the epidemic protocol will always find a route with only one transmission just like direct delivery.

In Figure 3.11(e), direct delivery has only approximately 60% of successful delivery ratio because of the random mobility model. This ratio increases to approximately 75% as the radio range increases in Figure 3.11(f) because more encounters in the network increases the probability of meeting a certain node. The performance of direct delivery in the previous studies is calculated based on successful deliveries whose total number is much less than that of our algorithm and the epidemic protocol. Therefore, its performance could be overestimated. To cancel this effect, we compare the three approaches considering only the packets that are deliverable using direct delivery. The result in Figures 3.11(g) and (h) shows that our algorithm performs as well as direct delivery.

Multicast

In addition to the number of nodes in the network, n, and the node's radio range, r, we introduced the number of destinations, d, in our experiments for multicast routing. By default, $n = 150$, $r = 30$ and $d = 50$. Three sets of experimental results are presented in Figure 3.12. The first column presents experimental results with varying the number of destinations while maintaining 150 nodes in the network and the radio range to be 30 meters. The second column shows results for varying the number of nodes in the network while keeping 30 meters radio range and 50 destinations. The third column gives experimental results for varying radio range where we use 150 nodes in the network and packets with 50 destinations. In general, the experimental results are similar to the results for unicast routing except multicast routing costs more time and energy.

Figures 3.12(a), (b) and (c) show the packet delay obtained by the different methods when varying three parameters. Those results show that direct delivery has the highest delay in all circumstances because of the random mobility model where the probabilities to meet other nodes do not vary significantly. As the number of destinations increases, both our algorithm and the epidemic protocol spend more time searching in the network to deliver the packet to all destinations. For the other two parameters, as expected with more nodes and greater radio range both cause more encounters in the network leading to less time to reach all destinations.

For energy cost, epidemic routing has the highest energy cost in all three settings because of its nature of flooding, and direct delivery always has the lowest cost. Figure 3.12(d) shows that more energy is required in order to reach more destinations whereas the results in Figures 3.12(e) and (f) are very similar to the results for unicast routing. As shown in Figure 3.12(e), energy cost is fairly insensitive to the the number of nodes in the network because of the artifact from the random mobility model due to the size of the area of interest. The energy cost increases as the radio range increases because the energy cost is quadratically related to the radio range while the actual number of hops in the result is fairly constant. In general, our algorithm has performance similar to direct delivery.

Similar to what we have discussed in unicast routing, direct delivery is not able to reach all destinations. Figures 3.12(g), (h) and (i) present the percentage of reachable destinations for each method. Direct delivery can only reach approximately 60% of the destinations on average if we vary the number of destinations and the number of nodes in the network. This probability is determined by the random mobility model. However, the percentage of reachable destinations increases as the radio range increases because there are more encounters in the network. Because our results are calculated by taking an average of delivering packets to all reachable destinations, only delivering to a portion of destinations makes direct delivery appear to be the best approach. For a fair comparison, we compare the results of three approaches prorated to the percentage of reachable destinations in direct delivery. In Figures 3.12(j), (k) and (l) our algorithm has either equal or better performance compared to direct delivery after prorating the result.

3.5.3 Experiments using real mobile traces

In this section, we use traces collected from real mobile nodes. There are three traces used in our experiments. Table 3.6 presents the details of these traces. We derived encounters from each trace and assume derived encounters are periodic encounters, which repeat themselves at every discrete period of time.

Table 3.6: Real user traces

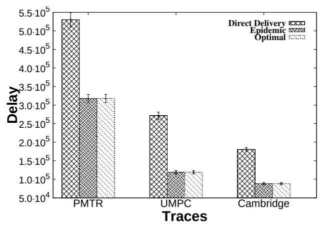
Source	Number of Users	Duration	Type	Analysed Users	Analysed Duration
Milano [94]	44	19 days	PMTR encounters	44	19 days
UMPC [82]	36	12 days	iMote encounters	36	12 days
Cambridge [116]	12	6 days	iMote encounters	12	6 days

Milano = University of Milano
PMTR = Pocket Mobility Trace Recorder
UMPC = University Pierre et Marie Curie

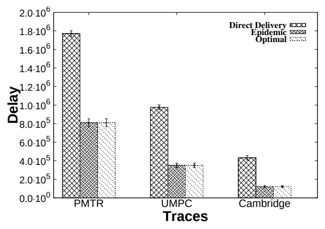
In our experiments, we again compare our algorithm to epidemic and direct delivery. Each measurement is calculated by taking the average of 100 unicast/multicast packets where packets are generated randomly. For each trace, we use the same set of packets across three routing approaches to guarantee a fair comparison.

Unicast

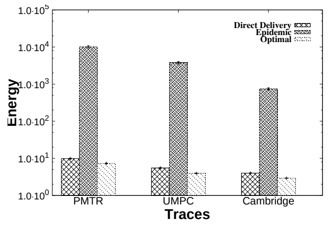
Regarding delay, the optimal results from our algorithm are equal to those returned by the epidemic routing protocol. This is shown by the experimental results in Figure 3.13(a). When we compare our method to direct delivery routing, the routes found by direct delivery always have longer delays than our method.



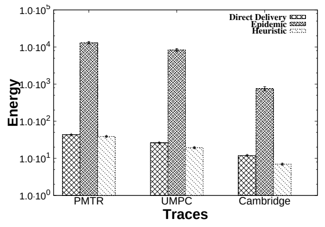
(a) Experimental results for delay in unicast



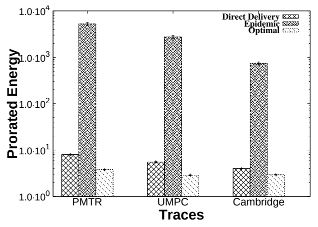
(b) Experimental results for delay in multicast



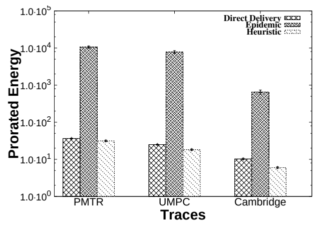
(c) Experimental results for energy in unicast



(d) Experimental results for energy in multicast



(e) Prorated energy cost in unicast



(f) Prorated energy cost in multicast

Figure 3.13: Unicast and multicast performance regarding delay and energy cost in real mobile traces

For the energy cost, a node has to transmit whenever there is an encounter in the epidemic protocol. This approach causes many more redundant transmissions compared to direct delivery and our optimal routing. As shown in Figure 3.13(c), the average energy cost from epidemic routing is an order of magnitude higher than for direct delivery and our optimal algorithm. Our algorithm can always find an optimal route that has an energy cost less than or equal to that from direct delivery. The prorated energy cost is presented in Figure 3.13(e).

Multicast

For the delay problem, our algorithm finds the shortest path tree whose delay equals the delay of the longest branch. This delay is the same as the delay returned by the epidemic protocol. As shown in Figure 3.13(b), the result of our algorithm is the same as the epidemic algorithm. In the direct delivery protocol, it always has to wait for the direct encounter. In general, the source has to wait until all reachable destinations are encountered. Therefore, the delay for each packet in direct delivery protocol is longer than the delay to deliver the message to destinations in our algorithm and the epidemic algorithm.

For the energy cost in the multicast, our algorithm and direct delivery consume much less energy compared to the epidemic protocol as shown in Figure 3.13(d). Experimental results show that our heuristic multicast tree works better than direct delivery in all three traces. Again, the prorated energy analysis is presented in Figure 3.13(f).

3.6 Future work

In order to simplify our model, we assume that periodic encounter patterns have the same period length. However, our assumption may not be true in some scenarios. For example, a person may go to a gym everyday, but only go to a church once a week. As a result, our graph model is not able to capture periodic encounter patterns with distinct period lengths. In this section, we present two approaches that modify encounter patterns and our graph model so that periodic encounter behaviours with different period lengths can be accommodated. We use three periodic encounter patterns in Table 3.7 for the following discussion. One note here is that the proposed modifications currently only focus on unicast routing. We leave multicast routing for future work.

Table 3.7: Periodic patterns with different period length

Node pair	Periodic pattern
$o_0 \leftrightarrow o_1$	$P^3_{o_0,o_1} = 1\ 1\ 0$
$o_0 \leftrightarrow o_1$	$P^4_{o_0,o_1} = 1\ 0\ 1\ 1$
$o_1 \leftrightarrow o_2$	$P^6_{o_1,o_2} = 1\ 0\ 1\ 1\ 1\ 0$

To combine encounter patterns with different lengths, we can expand them because of their property of repetition. If an encounter series has an encounter pattern with length m, then this encounter series also has encounter patterns with length $2m$, $3m$ and so on. After calculating the least common multiple of all period lengths, encounter patterns shown in Table 3.7 can be extended to the ones in Table 3.8. Because the least common multiple equals 12 for integers 3, 4 and 6, in the extended encounter patterns, each pattern just repeats itself until it reaches the length of 12.

Table 3.8: Extended periodic patterns

Node pair	Periodic pattern
$o_0 \leftrightarrow o_1$	$P^{12}_{o_0,o_1} = 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0$
$o_0 \leftrightarrow o_1$	$P^{12}_{o_0,o_1} = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1$
$o_1 \leftrightarrow o_2$	$P^{12}_{o_1,o_2} = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1$

There are two ways we can model extended patterns: (1) building the encounter graph based on extended encounter patterns and (2) modifying the encounter graph to adopt extended patterns.

Applying the proposed graph model

Similar to our discussion in Section 3.3, if we combine all the extended patterns, we can create an encounter graph where all periodic patterns are synchronized by their least common multiple. For example, Figure 3.14 presents the encounter graph corresponding to the extended patterns in Table 3.8. In the encounter graph, we have 12 intervals matching 12 phases in the extended patterns.

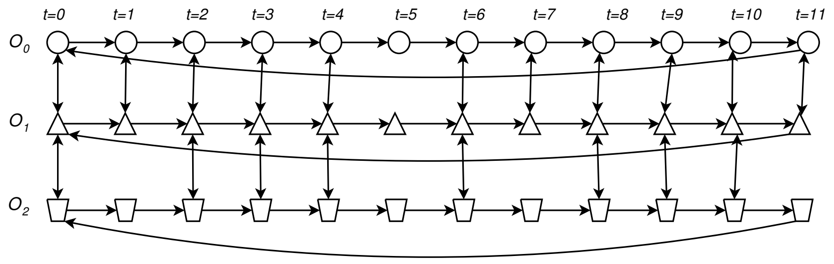


Figure 3.14: The encounter graph adopting extended patterns

Applying the proposed graph model to model a pattern with different lengths, our analyses presented in this chapter can be applied to solve routing problems in the proposed graph model. With respect to both unicast and multicast routing, the minimum energy consumption and minimum delay problems can be solved by finding the shortest path and minimum weighted Steiner tree. However, this approach uses the extended patterns whose lengths are usually multiples of the original ones. The resulting graph can be very large and be composed of a substantial number of vertices.

Modifying the proposed graph model As an alternative, we can modify the original

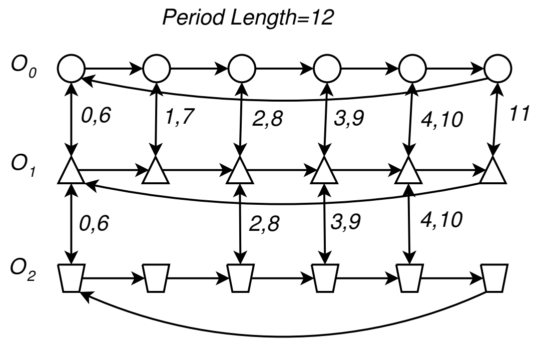


Figure 3.15: The compact encounter graph

encounter graph to have a compact graph with many fewer vertices. Because periodic encounters are synchronized by extending the patterns to the length of their least common multiple, the input to the modified encounter graph is still the extended patterns. However, instead of containing 12 intervals as in our last approach, we compress the total number of intervals into 6, which is the length of the longest encounter pattern in its original form in Table 3.7. The modified encounter graph is shown in Figure 3.15 where the constraints on vertical edges indicate the phases of periodic encounters. For example, at interval $t = 13$, there is a periodic encounter between o_0 and o_1 because constraint $13 \equiv 1 \pmod{12}$ is indicated on the second vertical edge between o_0 and o_1. Similarly, there is no periodic encounter among three nodes at interval $t = 17$ because no vertical edge in the encounter graph has a constraint $17 \equiv 5 \pmod{12}$.

The modified encounter graph is relatively simple and small. However, the phase constraints on vertical edges need to be considered in the design of algorithms and routing protocols. In addition, in our graph model we already have another constraint called NOT (NO reTransmissions during the same phase). Adopting both constraints could complicate our graph model.

We briefly presented two approaches to model encounter patterns with different lengths. Each one of them has different advantages and disadvantages. However, more studies are required with respect to routing problems in the modified encounter graphs.

In addition to assuming patterns with the same length, we assume that each mobile node has exactly the same mobility behaviours during every consecutive period in time. However, mobility behaviours are often disturbed by unexpected events. Furthermore, unexpected events may cause delays on a sequences of activities. How to handle these incidents requires further study. Our future work is to extend the proposed models to account for uncertainty in the encounter patterns. Also, all calculations and computations are currently performed in a centralized manner. In applications where there are thousands of mobile nodes, this may be not practical. Like conventional routing protocols, mobile nodes should be able to make local decisions based on their mobility patterns. How to distribute this process is another interesting direction for further research.

3.7 Summary

Assuming a scenario of a mobile wireless network where disconnections are more the norm than the exception, we explored the ways mobility patterns of nodes can be used to improve routing in such networks. We proposed a novel graph model built on top of mobility patterns that is flexible enough to accommodate several different assumptions. Analyses on two hard problems in wireless communications were conducted to show the benefits of having the knowledge about mobility patterns. Algorithms were developed to solve the complementary problems of forwarding a packet with minimum delay, and with minimum energy, both in polynomial time. In addition, our experiments show the effectiveness and efficiency of our solutions.

Chapter 4

Relay node deployment

4.1 Introduction

In DTMNs, communications depend on finding a set of sequential opportunistic encounters between pairs of mobile nodes in a store-and-forward fashion. However, it is still possible that the network is partitioned into isolated subnetworks where there is no path between mobile nodes from unconnected subnetworks. Messages that must travel between disconnected subnetworks cannot be delivered simply because no path exists.

To connect isolated components, additional encounter opportunities are required to deliver messages across different subnetworks. In general, there are two approaches to address the problem: (1) message ferrying and (2) deploying stationary relay nodes. Message ferrying aims to provide extra encounter opportunities by either having dedicated nodes or changing the trajectories of existing nodes in the network. The key problem of message ferrying is to design effective routes for ferries. Deploying relays adds additional nodes into the network that serve as buffers for nodes that visit the same relays at different times. Therefore, finding the optimal locations for deployment becomes critical. Having dedicated ferries or using existing mobile nodes as ferries by changing their original movements can be expensive and infeasible in some applications. Therefore, we focus on the relay deployment problem, which aims to optimally place relays to enhance connectivity in the network.

When the network is disconnected, there is no direct encounter between mobile nodes from different isolated subnetworks. However, it is possible that mobile nodes from different subnetworks visit the same locations but at different times. For example, in the university people from different departments rarely physically encounter each other, but they frequently meet people from the same department. Most of the time different groups of people are disconnected from each other. It would be very difficult to deliver a message through people from one department to the other. However, there are locations that both groups of people may visit, *e.g.*, cafeteria during lunch time and LRT/bus stations during commute time. By setting up message boards at these locations, people from one department can write message on the boards, and people from another department can read the message when they visit the locations of the message boards. In this example people can be treated as mobile nodes whereas message boards serve as relays in the network. In disconnected environments, effectively deploying relays can help increase encounter opportunities leading to the improvement of network connectivity and performance.

In relay deployment, relay failure may pose significant degradation to the performance of the whole network. In addition, encounters in mobile networks may be interrupted by unexpected events leading to link failures. If a link is critical to join disconnected subnetworks, its failure will disconnect the whole network. Similarly, mobile node failure is also detrimental to network connectivity. Therefore, instead of deploying the minimum number of relays in the network simply to reduce delivery delay for example, the route diversity in the network is also very important to improve routing robustness so that the network is more resilient to relay node failures, encounter (link) failures and mobile node failures. In this chapter, *route diversity* is defined as the existence of multiple disjoint paths for routing between mobile nodes in the network.

The relay deployment problem can be modelled as various *k*-connectivity problems: (1) node failure can be modelled as the *k*-vertex connectivity problem, (2) encounter failure can be modelled as the *k*-edge connectivity problem, and (3) relay node failure can be modelled as the *k*-element connectivity problem. In this chapter we concentrate on the *k*-element connectivity problem in which we find the minimum number of relays where the network is tolerant regarding relay node failure. Our study in this chapter has the following contributions:

- We extend the graph model presented in Chapter 3 to capture encounters in different mobility environments. With the extended graph model, the relay deployment problem can be addressed in both AP-based and trajectory-based mobility traces.

- We model the relay deployment problem as a *k*-connectivity problem. In disconnected environments, relays can be deployed to guarantee route diversity in the network where *k* disjoint paths exist between mobile nodes.

- We propose two different approaches to solve the relay deployment problem: (1) exact solutions using linear programming and (2) heuristic algorithms. Experiments and simulations are used to evaluate the performance of our heuristic algorithms and understand how various deployment approaches impact the network connectivity.

The remainder of this chapter is organized as follows. An overview of message ferrying and relay deployment is discussed in Section 4.2. In Section 4.3, we introduce the extended graph model that captures encounters in different mobility environments. Section 4.4 presents the formal problem statement and approaches to solve the problem. To examine the performance of proposed algorithms and deployment solutions, experimental and simulation results that use synthetic data and real mobility traces are discussed in Section 4.5 and Section 4.6, respectively. Finally, future work and a summary are given in Section 4.7 and Section 4.8.

4.2 Related work

There are mainly two approaches to join disconnected subnetworks: message ferrying and deploying relays. Message ferries act as mobile relays that travel between disconnected nodes whereas stationary relays buffer messages for mobile nodes who visit the same places at different times.

4.2.1 Message ferrying

In message ferrying, the network is composed of two components: partitioned regular nodes that can be either mobile or stationary and message ferries. Dedicated ferries or regular nodes acting as ferries change their trajectories to carry and forward messages between regular nodes in permanently partitioned environments [51, 145, 146, 148]. This technique can be applied in battlefield communication, communications after disasters, wide area sensing and surveillance where ferries, for example, can be unmanned aerial vehicles [132, 147]. Message ferrying introduces new node movements to the network and exploits such movements to improve message delivery ratio. Therefore, the key problem in message ferrying is the message ferry route problem targeting at finding the optimal routes. In previous studies, researchers addressed the problem using optimization techniques such as solutions from Travelling Salesman problem [147] and connected dominating set problem [109]. However, changing or disrupting the original trajectories of regular nodes is not feasible in many applications.

Most studies of message ferrying assumed that ferries have the full knowledge of the network. In other words, every ferry knows the exact locations of all other nodes at all times. However, because of obstacles and limited communication range, ferries may only cover a sub-region of the network. This is called partial ferry observation. To solve the problem, studies were conducted using partially observable Markov decision process with the assumption of the Markovianess of node mobility [50, 87]. However, because mobile nodes in real-life such as animals and vehicles do not move randomly, the assumption of randomness is unrealistic in some applications.

4.2.2 Relay node deployment

Another solution to enhance network connectivity is to deploy stationary relays, also called throw-boxes [149] in DTNs. Its objective is to build an unconnected infrastructure that helps message delivery by increasing transmission opportunities, *i.e.*, connectivity.

In DTNs, previous studies have shown the impact of adding relays to the network with respect to increasing connectivity (delivery ratio) and decreasing average delivery delay [12, 35, 37, 117, 149]. In [149], the authors deployed throw-boxes (relays) to impose additional encounter opportunities to enhance message delivery ratio. In their analyses, different deployment approaches are considered based on the availability of contact histories and/or traffic demands of mobile nodes in the network. To extend that work, the authors in [11] proposed an energy management strategy for throw-boxes to increase their performance in DTNs. Another study also considered the use of base stations [12]. In that study, the authors investigated the trade-offs of three different infrastructure deployment: relays, base stations and meshes. The authors showed that with the ease and cost of deployment, a mesh network or disconnected relays is a better choice compared to deploying base stations in many scenarios. However, to achieve similar performance in a network with base stations, it requires twice the number of mesh nodes or five times the number of relay nodes.

Aiming at Vehicular Delay Tolerant Networks (VDTNs), a special network of DTMNs, Pereira *et al.* presented different challenges in VDTNs [105]. To address the routing problem in VDTNs, the same research group focused on the optimal deployment of relay nodes. The authors argued that finding the smallest Connected Dominating Set is a feasible solution to their problem. Because the complexity of finding an optimal deployment in general is a NP-hard problem, two greedy-based heuristic algorithms were proposed to minimize (1) the average hop count and (2) the average message delivery time with the deployment of relays [35, 37].

To connect a large area of isolated regions, Farahamand *et al.* proposed a wireless burst switching network architecture targeting at providing low-cost communications for isolated regions with no infrastructure [36]. In the scenarios where the network is composed of disconnected communities that are internally connected, relays are placed where mobile vehicles can pick up and drop off messages which are destined to different e-koisks known as community gateways. To build such a network, the authors introduced three greedy-based heuristic algorithms to solve the relay deployment problem with the objectives to minimize the number of relays in the network and/or the delivery delay.

For the relay deployment problem, previous studies assumed that trajectories of mobile nodes are given. However, in many applications, no trajectory is provided. For example, real mobility traces such as Dartmouth, USC and MIT traces only contain associations between mobile nodes and APs, *e.g.*, wireless routers or cellular towers. Therefore, solutions from previous studies cannot be generalized to those applications. Furthermore, previous solutions mainly aimed to increase delivery ratio and reduce delivery delay without considering route diversity in the network. Because mobile networks can be very dynamic, if relays that are used to join disconnected subnetworks fail, the whole network would fall apart and become disconnected again. Therefore, increasing route diversity is also very important with respect to relay deployment.

4.3 Extended graph model

In this section, we extend the graph model presented in Chapter 3 to capture encounters from both AP-based and trajectory-based traces with the presence of relays.

In AP-based traces, associations are recorded when mobile nodes associate with APs. If two nodes associate with the same AP at the same time, we assume that there is a direct encounter between both mobile nodes. Although APs are not explicitly expressed, there exists an underlying AP for every derived encounter from AP-based traces. This is because APs act as intermediates to forward messages between associated nodes. Since there is no geographical information in AP-based traces, the potential locations for relay deployment cannot be determined. Our motivation is to use APs as candidates for relays. The locations of APs are referred as *logical locations*. Although relaying is not a feature of APs, we propose to explore the potential of adding this functionality in DTMNs. Regarding the relay deployment problem, in AP-based traces we target at finding potential APs as relays to connect isolated subnetworks. In trajectory-based traces, we are able to track the movement of mobile nodes. With the help from trajectories, we aim to determine the optimal locations in a geographical space for relay deployment. We refer such locations in trajectory-based traces as *physical locations*.

In the following discussions, deploying a relay at a *logical location* means treating an AP as a relay in AP-based traces; and deploying a relay at a *physical location* means placing a new relay in the network at a physical location in trajectory-based traces.

4.3.1 Mobile data with logical locations

An example of AP-based traces is presented in Table 4.1. In this example, we have three mobile nodes and two access points with the associations of three intervals. For example, mobile node B associated with access point M_1 during interval I_0. It then moved to a remote location with no record, and finally moved to the coverage area of access point M_1 at time I_2. By assuming that two mobile nodes encounter each other if both associate with the same AP at the same time, the derived encounters are presented in Table 4.2.

Table 4.1: AP associations

Time	Node	AP
I_0	A	M_1
I_0	B	M_1
I_0	C	M_2
I_1	A	M_2
I_1	C	M_1
I_2	A	M_1
I_2	B	M_1

Table 4.2: Derived encounters

Time	Node	Node	AP
I_0	A	B	M_1
I_2	A	B	M_1

Recalling our graph model for periodic encounters in Chapter 3, it can be used to model derived encounters between mobile nodes. Figure 4.1 presents a graph that models the derived encounters in Table 4.2. In the graph, node C is disconnected from other nodes. The only difference between the graph in Figure 4.1 and the ones capturing periodic encounters is the absence of horizontal edges to connect the same node from the last phase to the first one. This is because we do not model periodic encounter patterns in the relay deployment problem, and the extended graph model is adopted to capture opportunistic encounters within mobile data. One note here is that the graph model in Chapter 3 can also be extended using the methods discussed in this section to capture the relay deployment problem in the network formed by mobile nodes with periodic behaviours.

When a relay is deployed at a logical location, *i.e.*, adding the relaying functionality to an AP, the AP becomes a real node in the network rather than an underlying infrastructure supporting encounters between mobile nodes under our assumption. Once an AP is active

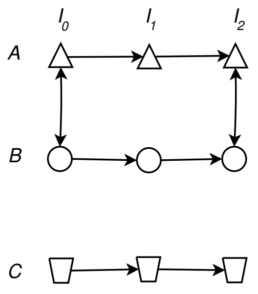
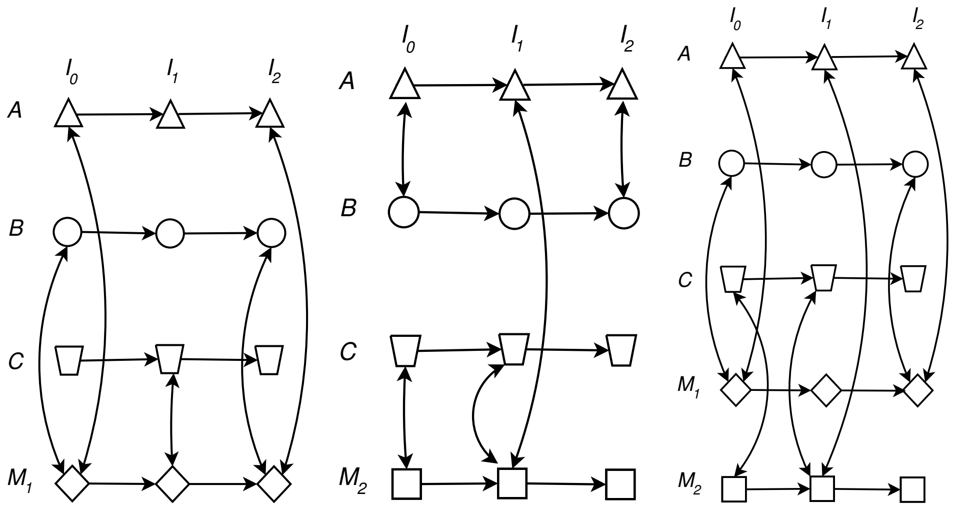


Figure 4.1: The graph model



(a) Using M_1 as a relay (b) Using M_2 as a relay (c) Using both M_1 and M_2 as relays

Figure 4.2: Relay nodes deployment

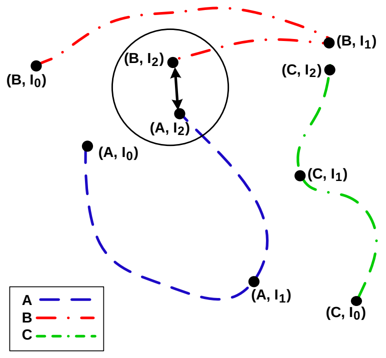
as a relay, the associations between the AP and mobile nodes introduce new transmission opportunities to the network.

To connect two isolated components in Figure 4.1, Figure 4.2 presents examples of deploying relays at different logical locations. In those examples, a set of vertices with the same shape is inserted into the graph to mimic each newly deployed relay during different intervals. In addition, each encounter derived from an AP, which is now a relay, is split into two encounters where each encounter corresponds to an association. To demonstrate the changes in the graph, we select access point M_1 as a relay. In Figure 4.2 (a), a set of three connected vertices is inserted to represent M_1 at different intervals. In addition, because the underlying AP is M_1 for derived encounters between nodes A and B during intervals I_0 and I_2, we remove each vertical edge in the original graph and replace it by two vertical edges where each edge represents the association between the relay and one of the mobile nodes. Furthermore, because node C is associated with M_1 during interval I_1, another vertical edge is inserted to represent the encounter between C and M_1 during interval I_1. By treating M_1 as a relay, the network is now connected.

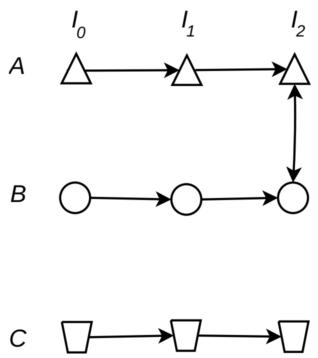
4.3.2 Mobile data with physical locations

In trajectory-based traces, trajectories are restricted in a geographical area. Given a relay at a specific location with a radius, it is trivial to find mobile nodes whose trajectory overlaps the coverage area of the relay. As a result, the search space in trajectory-based traces is much larger than that of AP-based traces. Unlike derived encounters in AP-based traces, encounters in trajectory-based traces take place when mobile nodes are physically close enough. Therefore, adding relays into the network does not affect existing encounters.

We use the example presented in Figure 4.3 to demonstrate how to map trajectory-based traces into our extended graph model. In this example, we have trajectories of three mobile nodes where the tuples in the figure represent the node movement. Each tuple is used to indicate the location of a mobile node during a particular interval: the first item in the tuple represents the identity of a mobile node whereas the second item defines the interval. For example, the dashed line in Figure 4.3 (a) represents the trajectory of node A. Tuple (A, I_0) defines the location of node A during interval I_0, tuple (A, I_1) indicates the location of node A during interval I_1, and the location of node A during interval I_2 is identified by tuple (A, I_2). There is no relay in this example, and only one encounter, which is emphasised by the circle in Figure 4.3 (a), can be observed when nodes A and B are physically close to each other during interval I_2. As a result, there is only one vertical edge in the corresponding graph in Figure 4.3 (b) to indicate the encounter.



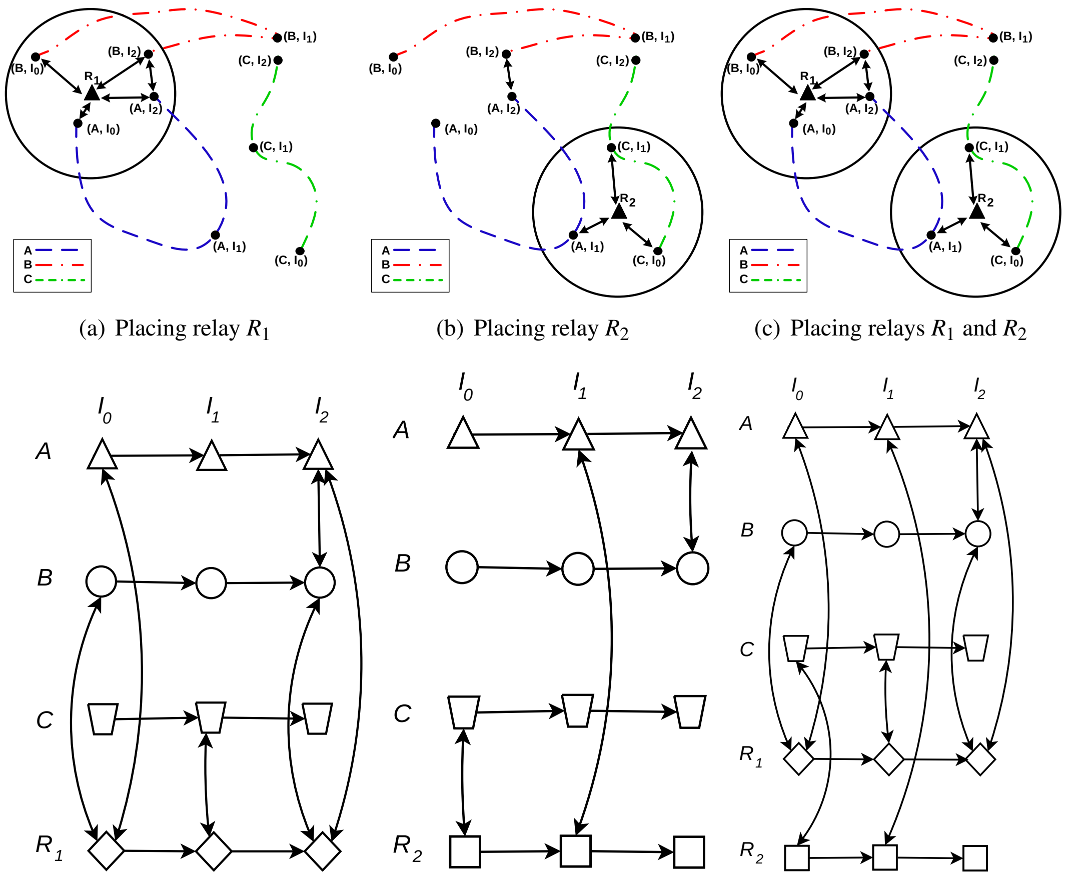
(a) Trajectories



(b) The graph model

Figure 4.3: Trajectories and the corresponding graph

When a relay is placed in the network, we add a set of connected vertices with the same shape to represent that relay. New vertical edges are also inserted into the graph between relays and mobile nodes if and only if they encounter each other. Examples of deploying relays at two different locations are presented in Figure 4.4. As we can see, once relays have been placed at different physical locations, the network connectivity can be increased significantly. When there is no relay, node C is disconnected from the rest of the network. By placing relay R_2, for example, as shown in Figure 4.4 (b) and (e), node C can transmit messages to R_2 where node A can receive them later and further distribute them to node B. As a result, placing relay R_2 connects two disconnected components.



(a) Placing relay R_1

(b) Placing relay R_2

(c) Placing relays R_1 and R_2

(d) Graph for the deployment in (a)

(e) Graph for the deployment in (b)

(f) Graph for the deployment in (c)

Figure 4.4: Graph models to capture different deployment approaches

Compared to the graph models for AP-based traces and trajectory-based traces, turning an AP into a relay (deploying relays at logical locations) breaks related derived encounters into two separated encounters whereas placing relays at physical locations does not affect any existing encounters. Similar to our graph model in Chapter 3, the extended graph is generic enough to represent both energy and delay costs simply by adjusting the weights assigned to each edge.

4.4 Relay deployment problem

To connect partitioned subnetworks, our objective is to deploy the minimum number of relays in the network while maintaining route diversity, *i.e.*, disjoint paths in the network. We propose to use the k-connectivity problem paradigm to model the relay deployment problem.

4.4.1 k-connectivity problems

In the network, metrics such as throughput, reachability, reliable transportation, fault-tolerance and load balancing can be modelled as or related to k-connectivity problems. There are three variations of the k-connectivity problem:

1. **k-vertex connectivity**: A graph is said to be k-vertex connected if and only if there does not exist a set of k-1 *vertices* whose removal disconnects the graph.

2. **k-edge connectivity**: A graph is said to be k-edge connected if and only if there does not exist a set of k-1 *edges* whose removal disconnects the graph.

3. **k-element connectivity**: In this problem, vertices in a graph are partitioned into two categories: terminals and non-terminals. The non-terminals are called elements. A graph is said to be k-element connected if and only if there does not exist a set of k-1 *elements* whose removal disconnects the graph. In other words, there are k element-disjoint paths connecting every pair of terminals; and these paths are allowed to share terminals.

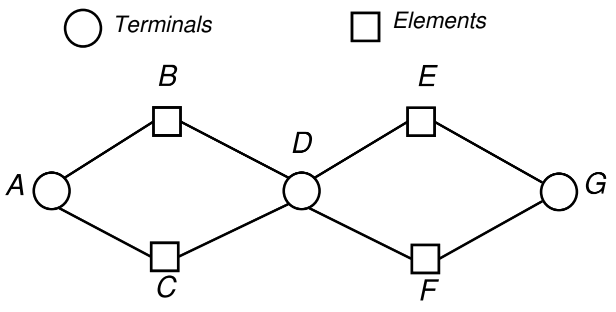


Figure 4.5: A sample graph

As an example, let us consider the sample graph in Figure 4.5. First of all, the sample graph is 1-vertex connected but not 2-vertex connected because removing vertex D disconnects the graph. Secondly, the graph is 2-edge connected because removing one edge does not disconnect the graph. In the figure, vertices with a circle-shape are terminals whereas the ones with a square-shape are elements. The sample graph is 2-element connected. Because vertex D is a terminal that can be shared by two element-disjoint paths, removing one of the elements B, C, E or F does not disconnect the graph.

For a given k, finding an optimal subgraph for all three variants of k-connectivity is NP-hard [40].

To the best of our knowledge, the best approximation ratios regarding general graphs are presented in Table 4.3. All those approximation algorithms are based on a linear programming relaxation of the problem, which can be solved in polynomial time by the ellipsoid algorithm [88]. The studies listed in the table focused on general graphs. In some special graph classes, there are better approximations. For example, the k-vertex and k-element connectivity problem in planar graphs can be approximated by $O(k)$ [23].

	Undirected graph		Directed graph	
	Edge weighted	Node weighted	Edge weighted[1]	Node weighted
Edge-connectivity	2-approx [65]	$O(k \log n)$ [100]	2-approx	N/A
Element-connectivity	2-approx [40]	$O(k \log n)$ [100]	2-approx	N/A
Vertex-Connectivity	$O(k^3 \log n)$ [24]	$O(k^3 \log n)$ [24]	$O(k^3 \log n)$	N/A

Table 4.3: Approximation results

Although finding an optimal subgraph with k-connectivity is NP-hard, finding the maximum k-connectivity in a graph or checking whether a graph is k-connected can be solved in polynomial time using max-flow algorithms based on Mengers theorem [93]. Max-flow algorithms such as Ford-Fulkerson algorithm [43] and Edmonds-Karp algorithm [33] can be used to find edge-disjoint paths in directed graph. By assigning each edge capacity 1, a max-flow algorithm actually computes the maximum number of edge-disjoint paths for a given source and destination. As a result, the maximum k of the input graph equals the minimum max-flow over all node pairs. To address node weighted graphs and undirected graphs, some reductions are required. The same technique can also be used to find the maximum k-vertex connectivity and maximum k-element connectivity of the input graphs. Once again, reductions are required.

The main objective of using k-connected graphs is to build fault-tolerant networks, also called survivable networks [73]. In addition to fault-tolerant networks, k-connected graphs

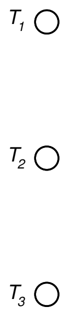
[1]Lando *et al.* showed that an edge-weighted directed graph can be reduced to an undirected graph [79]. Therefore, any edge-weighted connectivity problem in directed graph can be solved by the ones from undirected graph in the complexity of reduced format.

can also be used to address other network issues. For example, k vertex/edge-disjoint paths provide multiple routes to deliver messages from a source to a destination. As a result, the throughput can be increased by utilizing these independent routes. Another example is load balancing. With multiple vertex-disjoint, *i.e.*, node disjoint, paths, a naive approach would be using round-robin scheduling to inject traffic into those independent paths.

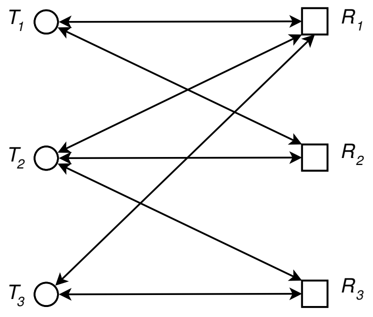
The problem statement

In Section 4.3, we introduced an extended graph model modelling encounters in different mobility environments. Using the graph model, routes can be selected for communications. In our simulations, we used the extended graph model to find routes to deliver messages. However, in the relay deployment problem, we concentrate on whether a relay can be used to join isolated subnetworks or not. The horizontal edges in the extended graph model mimicking time-flow are not important any more in our problem domain. Therefore, we propose to simplify the graph to model the relay deployment problem.

Disconnected networks are composed of isolated subnetworks. In this study, we assume mobile nodes within subnetworks do not fail. Thus, we can use a single vertex to represent each isolated subnetwork. Because subnetworks are disconnected, their representative vertices are also disconnected. In other words, there is no path between any pair of representative vertices. The problem now is to deploy the minimum number of relays to connect those vertices such that the failure of relays does not disconnect the network. As we discussed before, this problem can be modelled as a k-element connectivity problem where vertices representing subnetworks are terminals, and vertices representing relays are elements in the graph.



(a) A disconnected network with three isolated subnetworks



(b) A connected network with the presence of relays

Figure 4.6: A sample scenario to the k-element connectivity problem

An example of building an input graph to our problem is presented in Figure 4.6. As shown in Figure 4.6 (a), the network is composed of three isolated subnetworks represented by three terminals. In order to connect those subnetworks, three candidate relays can be deployed as shown in Figure 4.6 (b). By treating each subnetwork as a single vertex, *i.e.*, a terminal in the graph, we finally obtain an input graph to the relay deployment problem as shown in Figure 4.6 (b). It is not hard to see that the graph is a bipartite graph where terminals and relays are two disjoint sets, and no two vertices within the same set are connected. In this example, the minimum deployment to build a 1-element connected subgraph connecting all terminals is using relay R_1. Similarly, the minimum deployment to build a 2-element connected subgraph connecting all terminals is using all three relays because the failure of one relay will not disconnect the network.

In disconnected networks, by assuming mobile nodes within isolated subnetwork do not fail, each isolated subnetwork is modelled as a single vertex called a terminal in the graph. Given an input bipartite graph whose vertices are divided into two disjoint sets terminals T and relays R such that every edge connects a vertex in T to one in R, we define the relay deployment problem as follows.

Definition 7. *Given an input k and an input bipartite graph $G = (T, R, E)$, the relay deployment problem in our domain is to find the minimum k-element connected subgraph of G. In other words, our objective is to find the minimum number of relays such that every pair of terminals in the subgraph formed by these relays has k relay-disjoint paths.*

It is clear that the relay deployment problem in our domain is just the minimum-cost (minimum number of relays) k-element connectivity problem in a special graph class.

4.4.2 Mathematical optimization

With the input graph, we can set each relay as a variable. If a relay participates in the optimal deployment, its value is 1; otherwise, its value is 0. Therefore, the relay deployment problem can be modelled by linear programming with the objective to minimize the summation of the values of all relay variables.

The relay deployment problem is similar to the multi-commodity flow problem. In the multi-commodity flow problem, we want to guarantee multiple commodities (flow demands) between different sources and destinations in the graph. In the relay deployment problem, if we assume each directed edge has a unit weight, each pair of terminals demands k units of flow. Therefore, we can model the relay deployment problem based on the multi-commodity flow problem by transforming undirected input graphs to directed graphs.

Given a flow network $G = (T,R,E)$ and a flow demand $k \geq 1$, let the set of vertices $V = T \cup R$, $T \cap R = \emptyset$, and edge $e_{u,v} \in E$ if and only if vertices u and v are not from the same set. In addition, let $|T| = n$, $t_i \in T$ and $t_j \in T$, there are $\frac{n*(n-1)}{2}$ commodities $C_{i,j}$ in the flow network, which is defined by $C_{i,j} = (t_i, t_j, k)$ where $i < j$, t_i and t_j are the source and sink of the commodity $C_{i,j}$. The flow of a commodity $C_{i,j}$ along an edge $e_{u,v}$ is defined as f_{uv}^{ij}. Let $r_x \in R$, our objective is to minimize the number of r_x used in the network such that the flow demands of all commodities are satisfied. To enforce the flow demand of each commodity $C_{i,j}$, we have the following formulation.

$$\text{minimize} \sum_{\forall r_x \in R} r_x \text{ where } r_x = \begin{cases} 1 & \text{if participates in the optimal deployment} \\ 0 & \text{otherwise} \end{cases}$$

subject to

$$\sum_{v \in R} f_{uv}^{ij} - \sum_{v \in R} f_{vu}^{ij} = \begin{cases} k & \text{if } i < j \text{ and } u = t_i \\ -k & \text{if } i < j \text{ and } u = t_j \end{cases} \tag{C_1}$$

$$\sum_{u,v \in V} f_{uv}^{ij} - \sum_{u,v \in V} f_{vu}^{ij} = 0 \text{ if } i < j \text{ and } u,v \notin \{t_i, t_j\} \tag{C_2}$$

$$0 \leq d_{r_x} \times r_x - \sum_{u \in T} f_{uv}^{ij} \leq d_{r_x} - 1 \text{ if } i < j \text{ and } v = r_x \text{ where } d_{r_x} = |e_{u,r_x}| \tag{C_3}$$

$$f_{uv}^{ij} \in \{0,1\} \tag{C_4}$$

In the formulation, T represents terminals, and R represents relays.

- Constraint C_1 defines the amount of flow from a source to a destination. It enforces every source terminal must have k units of outgoing flow whereas every destination terminal must have k units of incoming flow.

- Constraint C_2 controls the amount of flow through vertices that is neither the source nor the destination of a given commodity $C_{i,j}$. If a vertex participates the flow, the amount of its incoming flow must equal the amount of outgoing flow. Moreover, if a vertex does not play a part in the flow, both the amount of incoming and outgoing flow must be zero. In either case, the difference between both amounts must be zero.

- Constraint C_3 defines the behaviour of a relay. In the constraint, the number of incoming edges d_{r_x} is a constant based on the input graph. If a relay node r_x participates in a flow, its value must equal 1. Meanwhile, at least one of its incoming edges must carry a flow. The constraint on the outgoing flow of a relay node is enforced by Constraint C_2.

- Constraint C_4 controls the amount of a flow on each edge.

4.4.3 Heuristic algorithms

To find an optimal k-element connected subgraph, approximation algorithms based on a linear programming relaxation of the problem have been proposed. They can be solved in polynomial time using the ellipsoid algorithm. However, the ellipsoid algorithm is relatively computationally expensive. In this section, we introduce three heuristic algorithms, an iterative algorithm, a greedy algorithm and a shortest-path based algorithm, to solve the relay deployment problem.

Iterative Algorithm

As we described before, given an input k, a max-flow algorithm can be used to check whether a graph is k-element connected in polynomial time. Taking advantage of this approach, we present Algorithm 3 that checks the k-element connectivity every time a relay node is inserted into the graph. Initially, the subgraph only contains terminals. The algorithm iteratively adds relays into the subgraph and checks its connectivity. There are two phases in this algorithm.

1. The algorithm first sorts the relays according to the number of terminals they connect. It then gradually expands the graph by iteratively inserting relays that connect to the most terminals first into the subgraph in the decreasing order.

2. Once the subgraph is k-element connected, in the second phase, the algorithm prunes the relays that have been selected in the reverse order. The objective is to remove redundant relays in the subgraph.

Because checking whether a graph is k-element connected can be solved using a max-flow algorithm *i.e.*, Edmonds-Karp algorithm, the complexity of this algorithm is dominated by the underlying max-flow algorithm and the number of relays being inserted. The complexity of this algorithm can be improved in some scenarios. In lines 9 and 15 in Algorithm 3, when a relay is inserted to or removed from the subgraph, a max-flow algorithm is run to check whether the insertion or deletion of the relay changes the connectivity of the subgraph. In these scenarios, we do not have to always apply the max-flow algorithm. Because the k-element connectivity is less than the minimum degree of the graph, checking the minimum degree of the subgraph first can sometimes eliminate the unnecessary running of the underlying max-flow algorithm. In our experiments, we implemented this strategy to

Algorithm 3 Iteratively add relays to build a k-element connected graph($G = (T,R,E),k$)

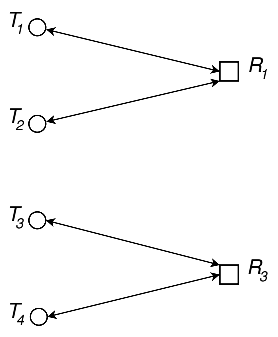
```
if G is not $k$-element connected then
    return null
end if
//phase one
$G' \leftarrow T$        //initialize a graph only with terminals
sort $R$ in the descending order based on the number of terminals each relay connects
for each relay $r \in R$ in the descending order do
    $G' \leftarrow G' \cup r$
    if $G'$ is $k$-element connected then
        break
    end if
end for
//phase two
for each relay $r \in G'$ in the reverse order of insertion do
    if $G'$ is $k$-element connected if $r$ is removed then
        $G' \leftarrow G' \setminus r$
    else
        break
    end if
end for
return $G'$
```

reduce the complexity of the iterative algorithm. Another observation is that this algorithm can be generalized to solve the k-vertex and k-edge connectivity problem by iteratively adding relays based on the number of vertices they connect in the network.

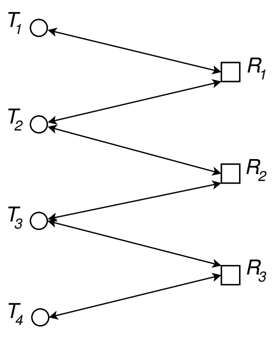
Greedy algorithm

The input graphs to our problem are bipartite graphs. It is easy to see that an input bipartite graph can be viewed as an instance of set covering. The objective is then to find a minimum subset of relays which cover all terminals. However, the relay deployment problem is not equivalent to the set cover problem. Let us look at an example. The universe includes four terminals where $U = \{T_1, T_2, T_3, T_4\}$. We let relay R_1 connect to terminals $\{T_1, T_2\}$, relay R_2 connect to terminals $\{T_2, T_3\}$, and relay R_3 connect to terminals $\{T_3, T_4\}$. It is clear that relays R_1 and R_3 is the minimum set cover to cover all terminals. However, deploying relays R_1 and R_3 does not connect the whole graph because subgraphs $\{T_1, R_1, T_2\}$ and $\{T_3, R_2, T_4\}$ are still disconnected. To connect the whole graph, relay R_2 is also required. To demonstrate the difference between the set covering problem and the relay deployment problem, Figure 4.7 presents two deployment approaches for the discussed example.

To solve the relay deployment problem, we develop a greedy algorithm inspired by the



(a) A solution for the set covering problem



(b) A solution to build a 1-element connected subgraph

Figure 4.7: The difference between the set covering problem and the relay deployment problem

greedy algorithm for the set cover problem. The pseudo-code of the algorithm is presented in Algorithm 4. The idea is simple, our greedy algorithm gradually builds a k-element connected subgraph by always building a (k-1)-element connected graph first. To be specific, the algorithm constructs the k-element connected subgraph by greedily connecting terminals with a degree of k-1 in the (k-1)-element connected subgraph.

In Algorithm 4, when i equals 1, set U contains all terminals because their degrees are less than i. To cover all terminals in U, the *greedy covering* function is called at line 14. The greedy covering function first chooses a relay containing the most uncovered terminals. After selecting the first relay, the greedy covering function chooses the second relay that covers the most uncovered terminals left in U. Meanwhile, the second relay must cover at least one terminal that has already been covered. The process of selecting the second relay repeats until all terminals are covered. By doing this, the greedy covering function guarantees that terminals in U are connected. It is not hard to see that the greedy covering builds a 1-element connected subgraph to connect all terminals in set U. As i increases, the subgraph returned by our greedy algorithm always connects terminals whose degree is less than i. The greedy covering function is applied to connect those terminals until i equals k. When i is equal to k, the returned subgraph is k-element connected.

Let us look at the if-statement at lines 18 to 24 in Algorithm 4 where the greedy covering function fails to return a feasible covering. Because greedy covering is used to find relays covering terminals with a degree less than the current connectivity i where $i \leq k$, if no relay can be used to connect those specific terminals, greedy covering fails to find a feasible solution, and thus the iterative algorithm is called to finish searching for relays.

An example to illustrate the failure is presented in Figure 4.8. In this example, it is

Algorithm 4 Greedy algorithm to find relays to build k-element connected graph($G = (T,R,E),k$)

```
1: if G is not $k$-element connected then
2:     return null
3: end if
4: $G' \leftarrow T$        //initialize a graph only with terminals
5: $U \leftarrow \emptyset$        //$U$ is used to contain terminals with a degree less than $i$
6: $C \leftarrow \emptyset$        //$C$ is used to contain a set of selected relays
7: $i \leftarrow 1$        //counter
8: while $i \leq k$ do
9:     for each terminal $t \in T$ do
10:        if degree($t$)$< i$ then
11:            $U \leftarrow U \cup t$
12:        end if
13:    end for
14:    $C \leftarrow$ calling greedyCovering($U$,$G'$,$G$) function
15:    if $C \neq \emptyset$ then
16:        $G' \leftarrow G' \cup C$
17:        $U \leftarrow \emptyset$
18:    else if $C == \emptyset$ then
19:        //no feasible solution is returned by the greedy covering function
20:        $C \leftarrow$ use the iterative algorithm to find the rest of relays to build $k$-element
21:                connected subgraph
22:        $G' \leftarrow G' \cup C$
23:        return $G'$
24:    end if
25:    $i++$
26: end while
27: return $G'$
28:
29: function greedyCovering ($U$,$G'$,$G$)
30:     $C \leftarrow U$        //initialize a graph only with terminals
31:     $S \leftarrow \emptyset$        //terminals that have been covered
32:     select a relay $r \in G$ and $r \notin G'$ that covers the most uncovered terminals in $U$
33:         $S \leftarrow S \cup S_r$        //$S_r$ is the set of terminals covered by $r$
34:         $C \leftarrow C \cup r$
35:         $U \leftarrow U \setminus S_r$
36:     while $U \neq \emptyset$ do
37:         select a relay $r \in G$, $r \notin G'$ and $r \notin C$ such that it covers the most uncovered
            terminals in $U$ and $S_r \cap S \neq \emptyset$
38:             $S \leftarrow S \cup S_r$
39:             $C \leftarrow C \cup r$
40:             $U \leftarrow U \setminus S_r$
41:     end while
42:     return $C$
43: end function
```

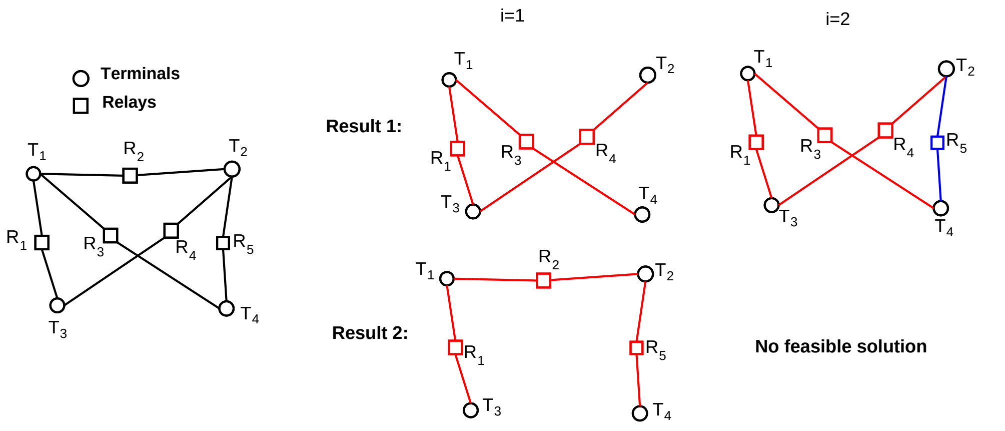


Figure 4.8: An example of failure of the set covering

trivial that the graph on the left side is 2-element connected. To find a 2-element connected subgraph, the greedy algorithm first builds a 1-element connected subgraph. When i equals 1, both results 1 and 2 are valid 1-element connected subgraphs that can be returned by greedy covering. When i is increased to 2, in result 1 the greedy algorithm will select R_5 to connect terminals T_2 and T_4 because their degree is less than 2 when $i = 1$. As a result, the subgraph is 2-element connected. However, in result 2 no such a relay can be found to connect terminals T_3 and T_4. Therefore, greedy covering fails to return a feasible solution. Thus, the greedy algorithm calls the iterative algorithm to find relays on top of the ones that have already been selected to build a 2-element connected subgraph. In this example, all five relays will be selected. Because both the subgraphs returned in results 1 and 2 are valid when i equals 1, if the greedy algorithm returns result 2, it would be more computationally expensive to find a feasible solution. In addition, the subgraph returned by calling the iterative algorithm will contain more relays compared to the number of returned relays in result 1.

We now prove the correctness of our greedy algorithm. In the cases where greedy covering fails to deliver a solution, the iterative algorithm is called to find a result. It is obvious that the iterative algorithm is correct. Therefore, the theorem focuses on the cases where the greedy algorithm does return a k-element connected subgraph.

Theorem 4. *Given an input k and an input bipartite graph $G = (T,R,E)$, the greedy algorithm returns a k-element connected subgraph if it returns a nonempty set.*

Proof. The greedy algorithm gradually builds a subgraph towards k-element connectivity. We let the set of relays being added at state i be R^i in this proof where $1 \leq i \leq k$.

When i equals 1, it is clear that the greedy algorithm builds a 1-element connected subgraph using a set of relays R^1.

When i equals 2, the greedy algorithm adds a set of relays, R^2, into the subgraph to connect the terminals with a degree 1 in the 1-element connected subgraph. As a result, the resulting subgraph is still connected if a relay $r \in R^1$ is removed. This is because the subgraph is connected by relays in R^2. Similarly, if a relay $r \in R^2$ is removed, the subgraph is also connected because the subgraph is 1-element connected when i equals 1. In the resulting subgraph, because there are two independent sets of relays connecting the subgraph, removing one relay from two independent sets does not disconnect the subgraph. Therefore, the subgraph generated is 2-element connected.

The same analyses apply as i increases. Therefore, we conclude that our greedy algorithm returns a k-element connected subgraph. $\square$

Shortest path Heuristic Algorithm

To deploy the minimum number of relays in a disconnected network, we can find the relay-disjoint paths with the minimum total weight in the corresponding graph if a unit weight is assigned to vertices whereas edges have a zero weight. To find relay-disjoint paths with the minimum total weight, we develop a shortest-path based heuristic algorithm to find a k-element connected subgraph.

Algorithm 5 A shortest-path based algorithm to build k-element connected graph($G = (T,R,E),k$)

```
1:  if G is not $k$-element connected then
2:      return null
3:  end if
4:  $w(t) = 1\ \forall t \in T$
5:  $w(r) = 1\ \forall r \in R$
6:  for every pair of terminals $u, v$ where $u \in T$ and $v \in T$ do
7:      $C \leftarrow$ use a shortest path algorithm to find $k$ relay-disjoint paths between $u \rightarrow v$
8:      for each relay $r$ in $C$ where $r \in R$ do
9:          if $w(r) == 1$ then
10:             $w(r) = 0$
11:         end if
12:     end for
13: end for
14: return the subgraph with all terminals and relays with a zero weight.
```

Initially, every vertex in the input graph has a unit weight, and edges have a zero weight. For each pair of terminals, the idea is to run a shortest path algorithm to find k relay-disjoint paths. The weight of relays that are selected by previous terminal pairs is set to zero in the graph. Changing their weights to zero is to mimic that the relays have been deployed for previous terminal pairs; therefore, it is free to reuse those deployed relays for the rest of terminal pairs to find their relay-disjoint paths since we want to minimize the number of relays being deployed towards a k-element connected subgraph containing all terminal pairs.

Our heuristic is presented in Algorithm 5. The complexity of this algorithm is bounded by the running time of a shortest path algorithm and the total number of terminals in the graph.

4.5 Experiments

In this section, we present experiments to evaluate the performance of the proposed heuristic algorithms. First, we conduct a set of experiments using real mobility traces. Because the instances derived from real mobility traces are relatively small, we also evaluate the three heuristic algorithms against synthetic data containing large instances.

4.5.1 Experiments using networks derived from real mobility traces

In trajectory-based traces, there exist a large number of candidate positions where relays can be set up. In order to reduce the search space, previous studies made different assumptions such as dividing the entire area into a grid of cells where relays are placed at the center of those cells [149]. However, making such assumptions may impact the results of the relay deployment problem. In this section, we concentrate on AP-based traces. In this type of mobility trace, we consider each AP as a candidate relay.

Table 4.4: Real-world traces

Traces	# of nodes	# of APs	Total duration	Examined duration	Examined nodes
Dartmouth [76]	13888	603	1178 days	14 days	2169
MIT Reality [32]	95	26956 [2]	282 days	14 days	68
MIT corporate WLAN [9]	1237	172	30 days	14 days	870
USC Summer 05 [57]	5580	61	111 days	14 days	4177
USC Spring 06 [57]	25481	137	95 days	14 days	5206

In the experiments, we focus on five real mobility traces listed in Table 4.4. By examining mobility traces over their entire durations, we found that the derived networks are connected as a single component. In order to have disconnected networks, we reduce the examined duration of each type of trace. In the real mobility traces, if the examined dura-

[2] This number includes both cellular towers and Bluetooth relays.

tion is too short, the networks formed by mobile nodes within the period can be completely disconnected. Deploying all candidate relays to the network still cannot connect them. As a result, in a totally disconnected network, there is no feasible solution for the relay deployment problem in our domain. Based on preliminary results, we found that a period of consecutive 14 days is a reasonable duration where disconnected subnetworks can be connected by relays. By randomly selecting a period of 14 days, the number of the examined mobile nodes in each trace is listed in Table 4.4. In addition to reducing the duration of the examined traces, we propose a constraint, d, on the degree of the vertices in a graph. In a graph, some vertices may be loosely connected to the rest of the graph, *i.e.*, with a degree equal to one. If we set the constraint $d = 2$, the vertices with a unit degree will become disconnected from other vertices. As a result, the graph will become disconnected. Using the degree constraint, we are able to derive disconnected networks from real mobility traces. Because the results of all five traces are qualitatively similar to each other, we concentrate on the Dartmouth College traces in this section.

(a) Dartmouth traces

	$d=2$	$d=3$	$d=4$
n	14	20	30
m	482	485	497
p	24.7%	18.2%	15.8%
k	3	2	2

(b) Relays for Dartmouth traces

Algorithm	# of returned relays		
	$d=2$	$d=3$	$d=4$
greedy	8	12	14
iterative	8	12	15
shortestPath	28	41	66

Table 4.5: Relay deployment in Dartmouth college traces

Table 4.5 presents the results regarding the Dartmouth College traces. In Table 4.5 (a), by varying the degree constraint d from 2 to 4, four characteristics have been measured in the generated networks:

- n is the number of disconnected subnetworks, *i.e.*, the number of terminals in the graph.
- m is the number of candidate relays that connect to at least two terminals.
- p is the percentage of coverage. It measures the percentage of terminals in average that each candidate relay connects.
- k measures the maximum element connectivity in the graph.

For example, when d equals 2, the generated network contains 14 terminals, 482 candidate relays where each relay in the network connects to 24.7% of the terminals in average, and the maximum element connectivity in the network is equal to 3.

As the degree constraint increases, both n and m increase. It is trivial that the number of disconnected components increases because more nodes are partitioned as d increases. In addition, the number of candidate relays increases because of more terminals in the network. However, p decreases because the increasing number of terminals impacts the average coverage. Lastly, the element connectivity decreases simply because candidate relays may not cover all terminals as the number of terminals increases in order to be k-element connected.

Let us look at the results returned by our heuristic algorithms in Table 4.5 (b). Because the minimum element connectivity equals 2 among three different scenarios of the degree constraint in the last row of Table 4.5 (a), Table 4.5 (b) presents the number of returned relays in order to form 2-element connected networks. In the results, on one hand, the greedy and iterative algorithms return the same number of relays to connect disconnected subnetworks except when the degree constraint equals 4. One note here is that even though the number of returned relays from both algorithms are identical, the actual candidates may not be the same. On the other hand, the shortest-path based algorithm returns more relays compared to the ones returned by the other two algorithms.

There are three observations from the results:

1. The number of returned relays increases for all three algorithms as the number of terminals n increases. This is because as n increases, more relays need to be deployed in order to connect all terminals.

2. The greedy algorithm returns the smallest number of relays whereas the iterative algorithm returns identical or similar results. However, the shortest-path based algorithm returns the highest number of relays to form 2-element connected subgraphs. We believe this is caused by the differences in our heuristic algorithms. In the greedy algorithm, the relays connecting the most uncovered terminals are always selected first. The greedy covering guarantees to reduce the number of returned relays. In the iterative algorithm, relays are selected in the descending order based on the number of terminals that they connect. Furthermore, the pruning process in the iterative algorithm eliminates the redundant relays to reduce the number of returned relays. The greedy covering and greedy selection in the greedy and iterative algorithms help to reduce the number of returned relays. However, in the shortest-path based algorithm, terminal pairs may have completely different sets of shortest relay-disjoint paths. As

a result, relays being selected may not be reused by other terminal pairs leading to a large number of returned relays.

3. The difference between the numbers of relays returned by the shortest-path based algorithm and the number returned by the greedy/iterative algorithm increases as the degree constraint increases. For example, the difference between the number of relays returned by the shortest-path based algorithm and the iterative algorithm equals to 20 (28-8) when $d = 2$, 29 (41-12) when $d = 3$ and 51 (66-15) when $d = 4$. We believe the cause is the increase of n and/or the decrease of p. Because in the shortest-path based algorithm terminal pairs may have completely different sets of shortest relay-disjoint paths, the increase in n introduces more relays to form relay-disjoint paths between terminal pairs. In addition, as the coverage p decreases, relays in average connect to fewer terminals. It also reduces the probability that the shortest-path based algorithm finds the relays that can be reused by multiple terminal pairs.

Because the network instances being examined in the real mobility traces are relatively small, we conduct experiments against synthetic data containing large network instances in the next section.

4.5.2 Experiments using synthetic networks

Because the disconnected networks derived from real mobility traces are small, in this section we use synthetic networks to understand the performance of different heuristic algorithms. By constructing synthetic networks, we can control the characteristics of synthetic networks such as the number of candidate relays and the percentage of coverage.

We use three control variables to create synthetic networks:

1. n controls the number of terminals in a network.

2. m defines the number of relays in a network.

3. p is used to control the percentage of terminals covered by relays in average. For example, if p equals 5%, it means that each relay in the network randomly connects to 5% of the terminals in average. Given a fixed p, we assume that the percentage of coverage from candidate relays in the network follows a normal distribution. This variable mainly controls the overall connectivity in a network.

Two metrics are examined in our experiments: (1) the number of returned relays and (2) the running time. The number of returned relays indicates how good each heuristic algorithm is whereas the running time measures the efficiency of each algorithm.

In the following experiments, 100 random networks are generated for each combination of these three variables. In those random networks, the maximum k-element connectivity can be equal to 7. We first set $k = 3$ by default in each combination of the three variables, and then vary k to examine heuristic algorithms in networks with different k connectivity.

Varying n

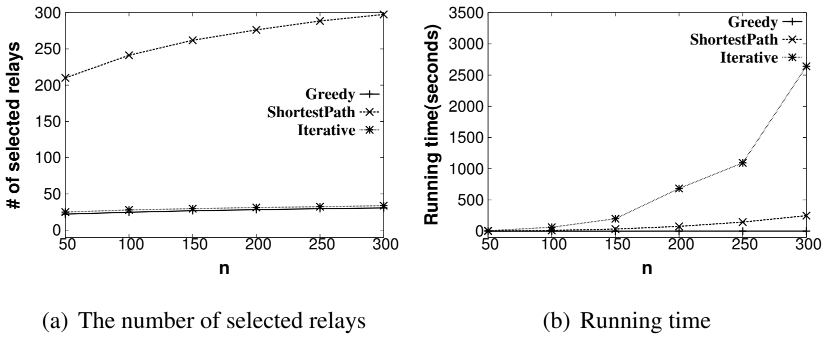
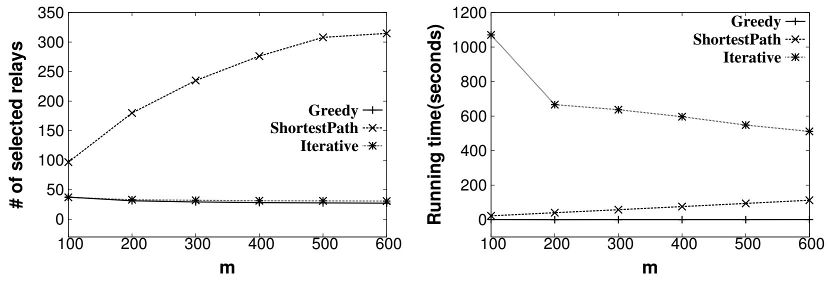


Figure 4.9: Varying n while $m = 400$, p =20% and $k = 3$

In Figure 4.9, we vary n while keeping the other three variables fixed. As the number of terminals increases, the number of relays required to connect those terminals increases as well. This explains the increases of the number of selected relays for all three heuristic algorithms. Among the three heuristic algorithms, the greedy and iterative algorithms return the smallest number of relays. This is because the greedy algorithm always selects the relays covering the most uncovered terminals first whereas the iterative algorithm always chooses the relays connecting to the most number of terminals first. The number of relays returned by the greedy algorithm at different n is slightly less than the one returned by the iterative algorithm; however, it may be difficult to recognize the difference in the figure. The shortest-path based algorithm returns the highest numbers of relays. In the shortest-path based algorithm, even though we set the weight of selected relays to be zero to reuse them, there could be so many relays that can be used to form the shortest relay-disjoint paths between terminal pairs that the ones can be reused by other terminal pairs may not be selected at all. This explains why the shortest-path based algorithm returns the largest number of relays.

Regarding the running time, all three algorithms use more time to find the results as the number of terminals increases. The greedy algorithm takes the smallest amount of time because of adopting the greedy algorithm from the set cover problem. The iterative algorithm uses a significant amount of time to find a solution because it needs to check k-element connectivity at every insertion of a relay. The running time of the iterative algorithm increases as the number of terminals increases. This is because the iterative algorithm needs to include more relays as the network size increases. The running time of the shortest-path based algorithm increases because the number of times that the underlying shortest path algorithm needs to be run increases as the number of terminals n increases in the network.

Varying m



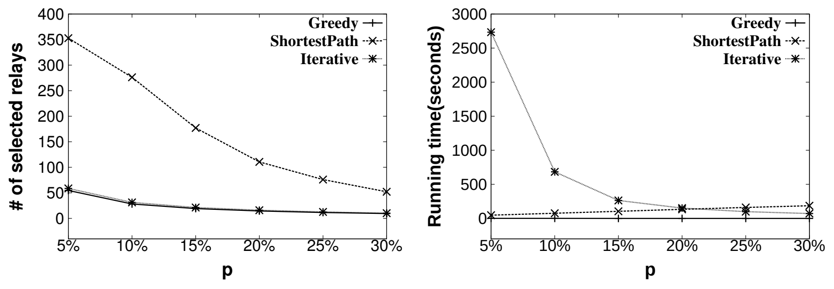
(a) The number of selected relays (b) Running time

Figure 4.10: Varying m while $n = 200$, p =20% and $k = 3$

In Figure 4.10, we examine the three heuristic algorithms by varying m. Given a fixed p, we assumed the percentage of coverage from candidate relays follows a normal distribution. As the number of candidate relays increases in the network, even though p is fixed, the absolute number of relays with a high percentage of coverage increases. Because the absolute number of candidate relays with a high percentage of coverage increases, both the greedy and iterative algorithms demand fewer relays to connect a fixed number of terminals. As a result, there is a slight decrease in the results of both algorithms with respect to the number of selected relays. Similar to our discussion for Figure 4.9, as the number of candidate relays increases in the network, more relays can be used to form the shortest relay-disjoint paths between different pairs of terminals so that the ones can be reused by other terminal pairs may not be selected by the shortest-path based algorithm. This causes the increase in the number of selected relays, and it also explains the reason that the shortest-path based algorithm still returns the largest number of relays.

By selecting fewer relays, the running time of both the greedy and iterative algorithms decreases because their greedy approaches only need to examine fewer relays. However, the increase of the network size due to the increase of m leads to the increase in the running time of the shortest-path based algorithm. This is because the underlying shortest path algorithm takes longer time to find solutions in large networks.

Varying p



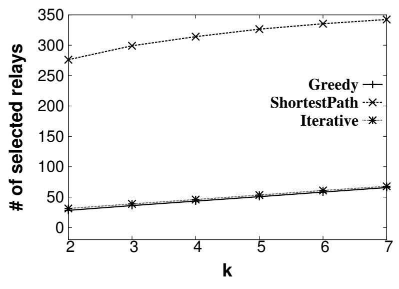
(a) The number of selected relays (b) Running time

Figure 4.11: Varying p while $n = 200$, $m = 400$ and $k = 3$

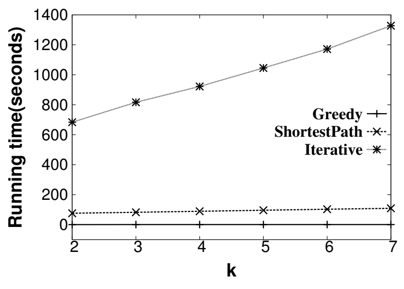
In Figure 4.11, we test the effect of varying the percentage of coverage p. As the average coverage increases, candidate relays now connect to more terminals. As a result, fewer relays are required to form a 3-element connected subgraph. This explains that all three algorithm return fewer relays as p increases in Figure 4.11 (a). In the figure, the number of relays returned by the shortest-path algorithm decreases very quickly. This is because the number of feasible solutions to build an optimal/sub-optimal 3-element connected subgraph increases leading to a high probability of selecting reusable relays by the shortest-path based algorithm.

As p increases, the resulting networks become dense. As a result, the shortest-path based algorithm takes longer time to find relay-disjoint paths. With respect to the greedy and iterative algorithm, both their running time decrease as the percentage of coverage increases because fewer relays are required to form 3-element connected subgraphs. Surprisingly, the running time of the iterative algorithm is lower than that of the shortest-path based algorithm once p is greater than 20%. As p increases, in Figure 4.11 (a), the iterative algorithm returns fewer relays. For example, only approximately 10 relays in average are returned by the iterative algorithm to form 3-element connected subgraphs when p equals 30%. As p increases, the number of times to checking the k-connectivity of subgraph decreases because fewer relays are required; however, the shortest-path based algorithm needs to find k relay-disjoint paths for every pair of terminals no matter what the value of p is. When p is large enough and the network is dense enough, the iterative algorithm takes less time than the shortest-path based algorithm.

Varying *k*



(a) The number of selected relays



(b) Running time

Figure 4.12: Varying k while $n = 200$, $m = 400$ and p =20%

When we set $n = 200$, $m = 400$ and p =20%, the maximum connectivity of the generated synthetic networks can be equal to 7. Figure 4.12 presents the results regarding the performance of the three heuristic algorithms in networks with different connectivity. As k increases, more relays are required in order to form a k-element connected network. This explains the increase in the number of selected relays in Figure 4.12 (a) for all three heuristic algorithms. However, because the shortest-path based algorithm may not select the reusable relays, the number of returned relays is significantly higher than the numbers returned by the other two algorithms in the networks with different connectivity.

In Figure 4.12 (b), the running time of all three algorithms increase as the value of k increases because they need to use more relays to build k-connected networks. The running time of the iterative algorithm is significantly higher than that of the other two algorithms because checking whether a subgraph is k-element connected after each insertion of a relay is very expensive. Because more relays are required when k increases, the number of times that the iterative algorithm checks k-connectivity in the subgraph increases too.

In summary, our experimental results against synthetic networks have confirmed our observations from using the real mobility trace. In addition, we found that the greedy algorithm has the best performance in both synthetic and real data regarding both the running time and the number of selected relays. The greedy algorithm using greedy covering gradually connects the terminals towards building a k-element connected subgraph where greedy covering guarantees to return a small number of relays. Because of greedy covering that does not need to check the k-connectivity of the subgraph, the greedy algorithm is very efficient and effective in terms of finding solutions for the relay deployment problem.

The iterative algorithm has competitive results compared to the greedy algorithm. Greedily selecting the next relays which connect the most terminals and pruning redundant relays help to reduce the total number of returned relays. However, the iterative algorithm takes the longest time to find solutions among the three algorithms because iteratively checking k-connectivity at every insertion of a relay is time-consuming. With respect to the shortest-path based algorithm, even though running a shortest path algorithm to find relay-disjoint paths for terminal pairs is relatively fast, the shortest-path based algorithm fails to utilize the reusable relays because the returned shortest-paths may not always choose those reusable relays.

4.6 Simulations

In the previous section, we presented experiments to evaluate the performance of the three heuristic algorithms. Because the greedy algorithm has offered the best performance, we use the greedy algorithm to find relays in the following simulations. After finding a set of relays to form a k-element connected network, we evaluate the performance of the deployment using simulations. We aim to evaluate the deployment in enhancing network performance regarding (1) the delivery ratio, (2) the average delivery delay and (3) the overhead ratio. In addition, we also examine how various deployment and routing approaches impact the performance. In this section, we first introduce our simulation settings and evaluation metrics, and then present our simulation results.

4.6.1 Simulation settings

There are two steps in our simulations: (1) computing feasible deployment and (2) evaluating deployment solutions using ONE simulator.

In our simulations, we adopt Edmonton Transit System (ETS) bus network scenarios [4]. In general, both AP-based and trajectory-based real-world traces are publicly available to use. Due to the limitation of ONE simulator, the AP-based traces cannot be simulated. This is mainly affected by the missing trajectories of moving among access points. In AP-based traces, only the associations between mobile nodes and access points are recorded without any trajectory capturing its movement from one access point to another. Even though we can generate synthetic movement, *i.e.*, shortest path, to fill the graph, this may affect the accuracy of AP-based traces and alter the underlying properties of mobile traces.

In trajectory-based traces, GPS positions are recorded to track node movement. It has been shown that the trajectory data is significantly affected by the low accuracy of GPS positioning because of missing trajectory segments and imprecise recording of positions due to the facts such as atmospheric and ionospheric disturbances and limited satellite visibility [139]. We decide to use ETS bus network scenarios for our simulations because the trajectories of buses are complete.

Each simulation is run for a period of 86400 seconds (1-day) in simulation time to capture the complete movement during a weekday. In order to create disconnected environments, in our simulations we randomly selected 20 buses as mobile nodes that are simulated to move in a 15×15 km^2 area within the City of Edmonton. Within this space, we have approximately 3000 bus stops. Instead of making any assumption about the potential locations for deployment, we use bus stops as candidate locations for relay deployment. The greedy algorithm is used to find potential locations to set up relays to build k-element connected networks in various input settings.

In our experiments, the packet-level simulations are conducted using the Opportunistic Networking Environment (ONE) [72] simulator. It provides simulations in delay tolerant environments supporting different mobility models and DTN routing protocols. However, buses in our simulations follow a map-based scheduled movement in which buses travel on the road between bus stops rather than the Euclidean shortest paths on the map. This movement model is currently not available in ONE simulator. To solve the problem, we implement the movement model.

To evaluate the performance of deployment, two routing protocols are used in our simulations: (1) epidemic routing and (2) scheduled routing. The epidemic protocol targets at providing delivery by utilizing all possible opportunistic encounters. By flooding the message, every node holding a copy of the original message transmits a replica to the ones without a copy at every encounter. In addition to the epidemic protocol, we propose a protocol called scheduled routing that takes advantage of our extended graph model in Section 4.3 to schedule routes between mobile nodes. Our scheduled routing implements the modified Dijkstra's algorithm from Chapter 3. Based on the bus schedules, we are able to obtain encounters between buses and relays whose locations are determined by the greedy algorithm. One note here is that encounters are directly derived from bus schedules; however, the actual delays of arrival time of buses at bus stops in the simulation follow a power-law distribution.

Due to the constraint on ONE simulator, each message being generated from a mobile node during the simulation only has one destination. In our simulations we focus on unicast routing and leave multicast routing for future work.

4.6.2 Evaluation metrics

Given a fixed number of mobile nodes, the complexity of the network is mainly controlled by each node's radio range. Increasing the radio range increases the coverage area of each node, which leads to more encounter opportunities. Because we are deploying relays in the network, the buffer size of relays and mobile nodes may also impact the performance of routing protocols. In addition, the message generation rate and the number of relays in the network also affect network performance. In general, higher message generation rates increase the traffic load in the network; more relays in the network create more encounter opportunities to forward messages. In this section, we present simulations using traces derived from ETS bus network scenarios in which we evaluate the performance of different deployment approaches and routing protocols with respect to five parameters:

1. the radio range, r.
2. the message generation rate, s.
3. the buffer size of all nodes including relays, b.
4. the buffer size of relays, b_r.
5. the number of relays in the network, m.

Three metrics are examined in our simulations: (1) the delivery ratio, (2) the average delay and (3) the overhead ratio. The delivery ratio is defined as the ratio of messages received by the destinations to those generated by the sources. The average delay is the average time from the generation of a message until it is delivered to the destination in the simulation. The overhead ratio defines the ratio between the number of transmitted messages (without the final deliveries) and the number of unique messages being delivered. It is defined as follows.

$$\text{Overhead ratio} = \frac{\textit{Total number of transmissions} - \textit{The number of messages being delivered}}{\textit{The number of messages being delivered}}$$

The overhead ratio is used to measure the amount of overhead, *e.g.*, duplicates in epidemic routing, in the network. For instance, the overhead ratio of the direct delivery protocol is zero because messages are forwarded only to the destination.

Regarding the delivery ratio, the higher the value is, the better the performance is. In contrast with the delivery ratio, the lower the average delay and overhead ratio are, the better the performance is. In addition, in order to have accurate results, we run the simulations three times with different sets of mobile nodes. Each measurement is calculated by taking the average of the results from three simulations.

4.6.3 Results

Each simulation is run for a period of 86400 seconds in simulation time. By default, the size of messages is 1 kB, $r = 300$ meters, $s = \frac{1}{60}$ (1 message every 60 seconds), $b = 5MB$, and $b_r = 5MB$. In addition, messages are transmitted from the buffer in a FIFO order (First In, First Out). When the buffer overflows, a node will drop the first message in the buffers. The transmission rate is set to be 1 Mbps by using the lowest transmission rate of IEEE 802.11b in outdoor space [115]. By default, the greedy algorithm is used to find relays to build 2-element connected networks in the simulations. Two other deployment strategies, grid deployment in which relays are placed to form a grid and random deployment in which relays are deployed at random locations, are considered in the simulations as well.

The effect of varying the radio range

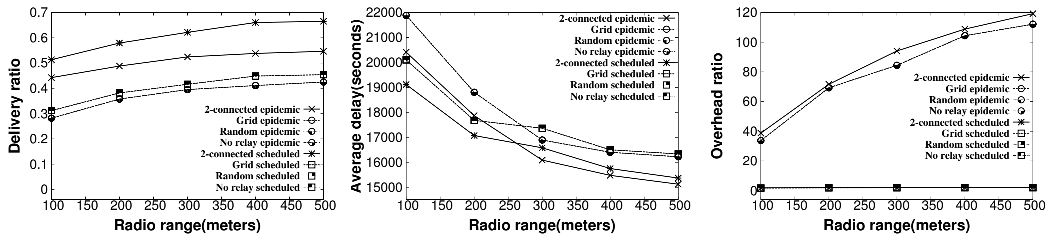
We first study how the radio range affects network performance. By varying the radio range from 100 meters to 500 meters, Figure 4.13 presents the results of two routing protocols under different deployment approaches. In our simulations, disconnected networks only have 2 or 3 isolated subnetworks. As shown in Table 4.6, it only requires approximately 2 relays in average to build 2-element connected networks. Under ETS bus network scenarios, due to a relatively large span of the area and a small number of relays in the network, both grid and random deployment tends to place relays at locations where mobile nodes do not visit. Both grid and random deployment do not introduce any transmission opportunity, and thus do not affect network performance. As a result, the curves of grid and random deployment overlap with the one without any relay. This is the reason that we can only observe four curves in Figure 4.13. The same argument applies to the following results unless stated otherwise.

As the radio range increases, the total number of encounters increases in the network. As a result, each node has more opportunities to forward messages leading to the increase of delivery ratios for both epidemic and scheduled routing in Figure 4.13 (a). With the presence of relays to form 2-element connected networks, there are much more transmission

	2-element connected
Simulation 1	2
Simulation 2	2
Simulation 3	3
Average	2.33

Table 4.6: The number of relays returned to form 2-element connected networks in each round of simulation

opportunities compared to grid and random deployment. As a result, the delivery ratios of both epidemic and scheduled routing are increased by at least 20% in the networks that are 2-element connected. Because epidemic routing generates a large number of duplicates in the networks, given a fixed buffer space in each node, the performance of epidemic routing is limited by the availability of buffer space. If the buffer overflows, messages that have not been delivered yet may be dropped leading to lower delivery ratios. This explains that scheduled routing in Figure 4.13 (a) always has better delivery ratios compared to epidemic routing.



(a) Delivery ratio (b) Average delivery delay (c) Overhead ratio

Figure 4.13: Varying radio range

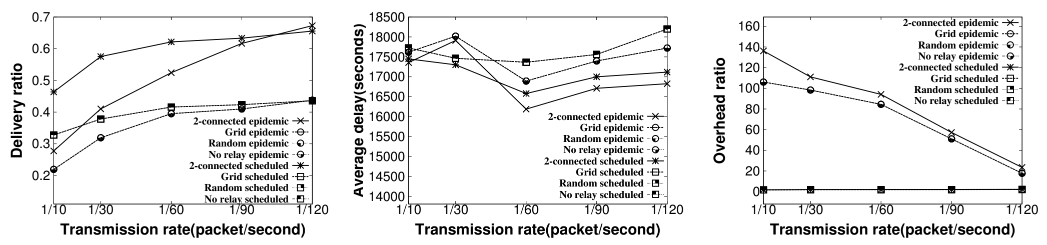
As the radio range increases, there are more forwarding opportunities in the network. As a result, each message has a better chance to be delivered leading to the decrease of the queueing delay for the messages in the network. As expected, in Figure 4.13 (b) the average delivery delay decreases as the radio range increases. With the presence of 2-element connected networks, there are more forwarding opportunities that can be used by epidemic and scheduled routing compared to other deployment approaches. As a result, epidemic and scheduled routing have the lowest delay at different radio ranges in the networks that are 2-element connected.

In epidemic routing every node holding a copy of the message transmits a replica to the ones without a copy at every encounter. As the radio range increases where each node has more transmission opportunities, the number of duplicates of the original message (redundant messages) increases in the network. As a result, as shown in Figure 4.13 (c), the overhead ratio of epidemic routing increases no matter whether there are relays in the networks or not. Epidemic routing in the networks without any relay or with grid/random deployment has a slightly lower overhead ratio because it has less transmission opportunities compared to the one in the 2-element connected networks. In the networks with different deployment approaches, scheduled routing has a very low overhead ratio, which approximately equals three. This is because in scheduled routing each message is only forwarded to a single next-hop based on a pre-calculated route. As a result, the overhead is generated only when an intermediate node forwards the message to the next-hop.

The effect of varying the message generation rate

We now focus on the effect of varying the message generation rate. In Figure 4.14, the message generation rate is reduced from 1 message every 10 seconds to 1 message every 120 seconds. As the traffic load decreases, fewer messages are generated in the network. This reduces the chances of buffer overflow leading to a result that more messages can be transmitted during encounters. As shown in Figure 4.14 (a), the delivery ratio increases for both protocols in different deployment approaches when the radio range increases. There are significant increases in delivery ratios of both epidemic and scheduled routing with the presence of 2-element connected networks compared to grid and random deployment. This is because the network is connected and there are more transmission opportunities in 2-element connected networks. As we described before, the performance of the epidemic protocol is limited by the buffer space. When the message generation rate decreases, fewer messages are buffered in the network where the chance that a message is dropped due to the buffer overflow reduces. Since fewer messages are dropped in the network, the chance that a message is delivered to the destination increases. As the traffic load in the network decreases, the impact of the buffer size on epidemic routing is reduced as well. In scheduled routing, a message delivery follows the pre-calculated route derived from bus schedules, and it may fail because buses may run behind schedule. This explains that in Figure 4.14 (a) epidemic routing outperformed scheduled routing when the traffic load in the network is low enough, *i.e.*, 1 message every 120 seconds.

Whenever the message generation rate changes, we have completely different sets of messages being generated. It is difficult to have a trend/pattern regarding the average delivery delay. As shown in Figure 4.14 (b), we have zigzag curves for both protocols in the networks with different deployment approaches. In general, the protocols in 2-element



(a) Delivery ratio (b) Average delivery delay (c) Overhead ratio

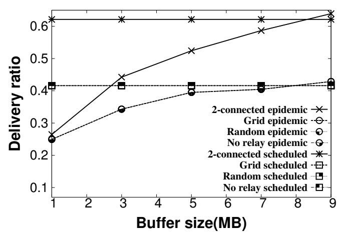
Figure 4.14: Varying the message generation rate

connected networks always have the lowest delay. This is because the queuing time of the messages decreases in the 2-element connected networks when there are more transmission opportunities in the network. When the traffic load is high, *i.e.*, 1 message every 30 seconds, in the network, scheduled routing has a lower delivery delay compared to epidemic routing because in epidemic routing messages are often dropped due to the buffer overflow. Once the message generation rate reduces, the impact of the availability of buffer space on epidemic routing reduces leading to lower delivery delays, which is shown in Figure 4.14 (b). This is because epidemic routing generates the shortest delay if it is not restricted by the network resources such as the buffer size.

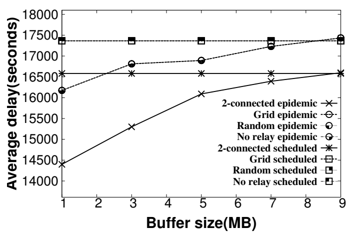
In Figure 4.14 (c), the overhead ratio of epidemic routing decreases as the message generation rate decreases. This is because less traffic in the network generates fewer replicated messages in the circulation. The overhead ratio of scheduled routing still maintains at a low value because of using a single-copy forwarding approach.

The effect of varying the buffer size

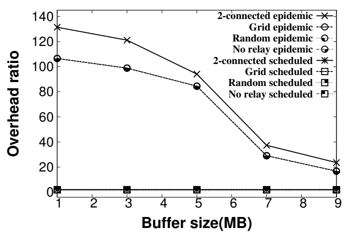
In our simulations, the size of a message is 1 KB. By varying the buffer size from 1 MB to 9 MB, each node including relays can hold up to approximately 9200 messages. Because the messages in a node may be dropped due to the buffer overflow, increasing the buffer size increases the lifetime of the messages in the network where each message has a better chance to be delivered. As expected, in Figure 4.15 (a) the delivery ratio of epidemic routing increases as the buffer size increases. In addition, as the storage space increases, epidemic routing has great improvement in the delivery ratios in 2-element connected networks compared to ones in the networks without any relay. For example, the delivery ratio is increased by 20% when the buff size equals 9 MB whereas the improvement is only about 10% when the buffer size is 3 MB. This is because as the buffer size increases, more messages destined to different subnetworks can be delivered through the deployed relays whereas the increase in the delivery ratios in grid and random deployment is caused by the increase of deliveries within the same subnetwork. Because single-copy forwarding only has one copy of the message being circulated in the network, scheduled routing only has a very small number of messages in the network. Because the size of the messages being stored does not exceed the buffer size, the availability of buffer size does not have any impact in scheduled routing in our simulations. This explains that the delivery ratios of scheduled routing in the networks with different deployment approaches do not change.



(a) Delivery ratio



(b) Average delivery delay



(c) Overhead ratio

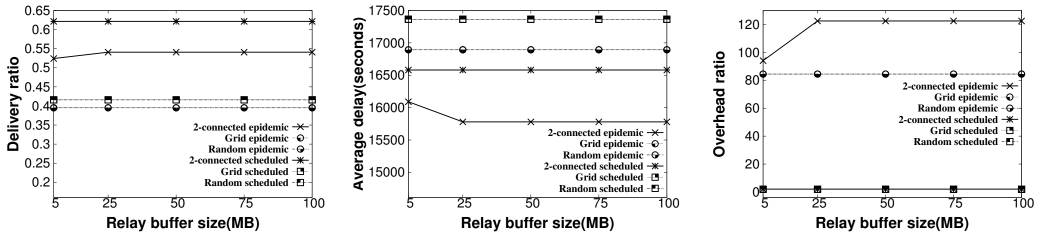
Figure 4.15: Varying buffer size

In contrast to the improvement in the delivery ratio, in Figure 4.15 (b) the average delay of epidemic routing increases as the buffer size increases. With a large buffer size, in epidemic routing messages with long delays do not get dropped and tend to stay in the network for a longer time. As a result, the delay for delivered messages increases. Because the number of messages in the network is relatively low, the availability of more buffer space does not affect scheduled routing. As a result, the average delay does not change.

As shown in Figure 4.15 (c), the overhead ratio of epidemic routing decreases. This is because fewer messages are dropped because of larger buffer sizes. The overhead ratios of scheduled routing in the networks with different deployment approaches do not change because the availability of buffer space has no impact when the number of messages in the network is low.

The effect of varying the relay buffer size

We have shown the impact of varying the buffer space on epidemic and scheduled routing in Figure 4.15. In those simulations, we increased the buffer size of both mobile nodes and relays. In this section, we only vary the buffer size of relays. Because relays are critical to deliver messages destined to different subnetworks that are used to be disconnected, we aim to evaluate the performance of the network by only varying the relay buffer size. By setting the buffer size of mobile nodes to be 5 MB, the buffer size of relays varies from 1 time and up to 20 times of that of mobile nodes. Because of examining relays, the performance of epidemic and scheduled routing in disconnected networks without any relay is not presented in the following figures.



(a) Delivery ratio (b) Average delivery delay (c) Overhead ratio

Figure 4.16: Varying relay buffer size

By increasing the buffer size of relays from 5 MB to 25 MB, the delivery ratio of epidemic routing in 2-element connected networks is increased by approximately 5% in Figure 4.16 (a). After that, the availability of more buffer space does not affect the delivery ratio any more, *i.e.*, the delivery ratio of epidemic routing in 2-element connected networks does not change even with larger buffer sizes. As the relay buffer size increases, relays are able to hold more messages without the buffer overflow. As a result, more messages can be delivered in the network. When the buffer size of relays is significantly large enough, given the same encounter scenarios, the messages received by the relays do not change any more. Therefore, the delivery ratio of epidemic routing in 2-element connected networks stays the same after the relay buffer size reaches 25 MB. Since grid and random deployment do not introduce any transmission opportunity, the delivery ratios of epidemic routing in both deployment approaches do not change. In scheduled routing, because of a relatively small number of messages in the circulation in the network, relays can always hold messages destined to other subnetworks without the buffer overflow. Increasing the buffer size of relays does not affect the delivery ratio of scheduled routing in different deployment approaches.

Increasing the buffer size of relays makes relays hold more messages. Therefore, those messages have better chances to be delivered. As a result, the delivery delay of epidemic routing decreases when the relay buffer size is increased to 25 MB. After that, the delivery delay does not change in Figure 4.16 (b) because there is no change in the total number of deliveries. Similarly, in other tested scenarios, there is no change in the delivery delay because there is no change in the deliveries.

As the buffer size increases, relays hold more messages and transmit more replicas to the network. As a result, the overhead ratio of epidemic routing with the presence of 2-

element connected networks increases when the relay buffer size is increased to 25 MB. After that, the delivery ratio does not change any more even in larger buffer sizes. This result is presented in Figure 4.16 (c). Similarly, the overhead ratio of other tested scenarios stays the same.

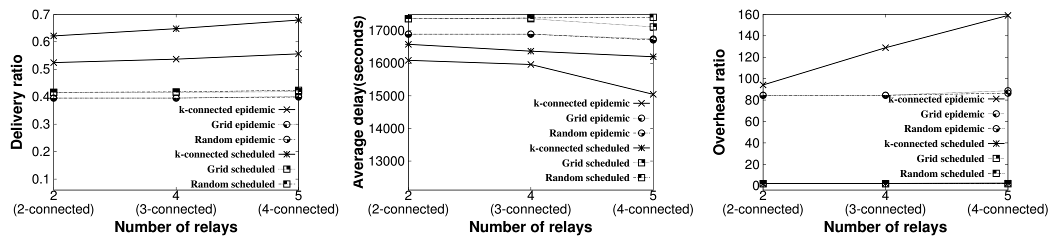
The effect of varying the number of relays in the network

In the previous simulations, we evaluated network performance in the 2-element connected networks. In this section, we increase the *k*-element connectivity to understand the performance implication of various routing and deployment approaches.

	2-element connected	3-element connected	4-element connected
Simulation 1	2	4	5
Simulation 2	2	3	4
Simulation 3	3	4	6
Average	2.33	3.66	5

Table 4.7: Number of relays selected to form *k*-element connected networks in each round of simulation

When we increase the *k*-element connectivity in the network, the number of relays required to form a *k*-element connected network increases as well. In our simulations, the average number of relays to form 2, 3 and 4-element connected graphs are presented in Table 4.7. As we described before, in our simulations disconnected networks only have 2 or 3 isolated subnetworks. Only a small number of relays are required to form a corresponding *k*-element connected network. In Figure 4.17, we rounded each average number of relays to 1 significant digit, *i.e.*, its nearest integer, and use them as tic marks along x-axis to indicate different connectivity. The actual number of relays in grid and random deployment equals the number of relays in corresponding *k*-element connected networks.



(a) Delivery ratio (b) Average delivery delay (c) Overhead ratio

Figure 4.17: Varying connectivity

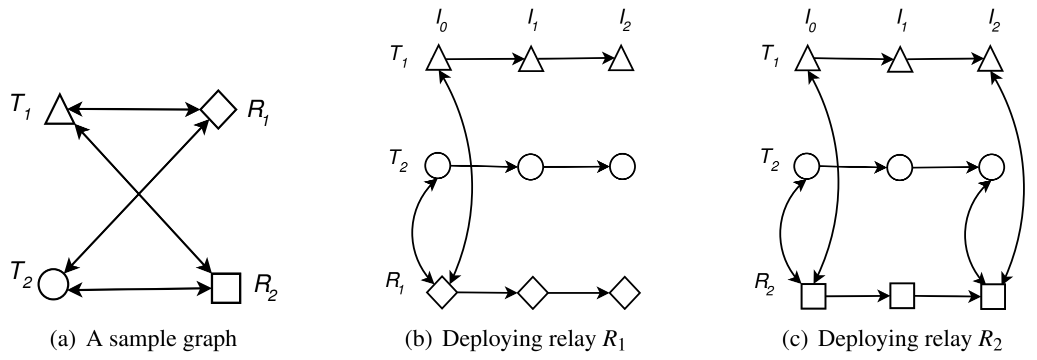
As the number of relays increases in the network, each mobile node has more transmission opportunities. In the networks that are *k*-element connected, the delivery ratios of both epidemic and scheduled routing increase as *k* increases. As shown in Figure 4.17 (a), the increases in the delivery ratio of both protocols may not be significant, *e.g.*, the delivery ratio is increased by 5% for scheduled routing when *k* is increased from 2 to 4. This may be caused by the small number of relays being deployed in the network, *e.g.*, 5 relays in average being deployed to the network when *k* equals 4. We believe that the connectivity (new transmission opportunities) introduced by deploying a few relays is insufficient for significantly increasing the delivery ratio.

As expected, in Figure 4.17 (b) the delivery delay decreases in both epidemic and scheduled routing because there are more forwarding opportunities in the networks. In Figure 4.17 (c) the overhead ratio of epidemic routing in the *k*-element connected networks increases as *k* increases. This is because more relays introduce more transmission opportunities to the network, and the number of replicated messages increases in the network leading to the increase in the overhead ratio.

In summary, to connect isolated subnetworks, deploying relays to build *k*-element connected networks has great impact on network performance with respect to the delivery ratio, the delivery delay and the overhead ratio in our simulations. Even with the presence of a very small number of relays in the network, we had approximately 50% improvement in the delivery ratio in general, *i.e.*, from 0.4 to 0.6 in 2-element connected networks. In addition, the experimental results showed that scheduled routing consumed much lower network resources such as the buffer space and the total number of transmissions compared to epidemic routing. Although we did not concentrate on periodic encounter patterns in this chapter, it is not difficult to see that our studies in this chapter can be easily generalized to the relay deployment problem under the network environments with periodic movement patterns.

4.7 Future work

In our problem domain, the input bipartite graph represents the aggregated connectivity between terminals and relays. However, in aggregated graphs we lose underlying encounter information such as how frequent the encounter is between a relay and a terminal and when those encounters take place. Moreover, even though two candidate relays connect to the same set of terminals in the graph, they may introduce different transmission opportunities to the network. As a result, one may be preferred over the other. To demonstrate the difference, we present an example in Figure 4.18. In the example, there are two subnetworks and two relays in the network where either of relays can be used to join both subnetworks. In the aggregated graph in Figure 4.18 (a), relays R_1 and R_2 are considered equivalent because they both connect to the same set of terminals. However, in the extended graphs representing the deployment of each one of the relays, it is trivial that relay R_2 imposes more encounters opportunities compared to relay R_1. If R_1 is deployed, messages generated after interval I_0 will never be delivered, but they can be delivered if R_2 is deployed.



(a) A sample graph (b) Deploying relay R_1 (c) Deploying relay R_2

Figure 4.18: The difference in network connectivity between two deployment approaches

If we can use the underlying encounter information for selecting the relays to form k-element connected networks, we may further improve network performance regarding the delivery ratio and the delivery delay. One possible solution to utilize the underlying encounter information is to assign a weight to each relay. In our example, relay R_2 should have a higher weight because it introduces more encounter opportunities. It would be a very interesting future direction to design a weight function to capture the differences between candidate relays in the network. By assigning weights to relays, the relay deployment problem becomes a multi-objective optimization problem where we want to minimize the number of relays to form k-element connected networks while maximizing the total weight of the selected relays.

In our discussion, we have introduced three variations of the k-connectivity problem: (1) the k-element connectivity problem, (2) the k-vertex connectivity problem and (3) the k-edge connectivity problem. In this chapter, we only focused on the relay deployment problem that is modelled as the k-element connectivity problem in a special graph class. In DTMNs, mobile nodes failures and encounter failures are the norm rather than the exception. How to properly model those failures in the scope of the k-connectivity problem in general graphs requires further investigations.

In our simulations, the examined protocols, especially scheduled routing, did not take advantage of relay-disjoint paths in the network. In scheduled routing, routes are pre-calculated based on the derived encounters. If a relay on the scheduled routes fails, messages travelling through that relay will never be delivered even though there exist other paths. Therefore, dedicated routing protocols are required in order to utilize *k*-element connectivity. Because of the existence of disjoint paths in the network, dedicated protocols should aim at utilizing different metrics such as throughput, fault-tolerance and load balancing. We will design routing protocols in *k*-connectivity environments in future study.

4.8 Summary

In DTMNs, mobile movements may lead to a situation called network partition where an end-to-end path may never exist because the network is divided into several isolated subnetworks. To connect isolated subnetworks, we proposed to deploy stationary relays to form *k*-element connected networks where there are *k* relay-disjoint paths between isolated subnetworks. The whole network is still connected if *k*-1 relays fail. In this chapter, we extended our graph model to capture the relay deployment problem in both AP-based and trajectory-based traces. To solve the relay deployment problem, we proposed three heuristic algorithms: a greedy algorithm, an iterative algorithm and a shortest-path based algorithm. The experimental results showed that the greedy algorithm returns the smallest number of relays to form *k*-element connected networks. Using the relays returned by the greedy algorithm, we conducted a set of simulations to evaluate the performance of the *k*-element connected network. Our simulation results showed significant improvement in network performance regarding the delivery ratio and delivery delay by forming *k*-element connected networks.

Chapter 5

Conclusion and future directions

5.1 Conclusion

New applications have created increasing demands for services in mobile networks. Currently, the majority of routing techniques have an underlying assumption that mobile nodes are within a single connected network. However, in some application domains the physical topology may change often and mobile nodes (or entire sub-networks) may not always be connected due to various forms of node movement as well as energy management or interferences. In our studies, we focused on mobile networks where a lack of continuous network connectivity is caused by node mobility, which are also called Delay Tolerant Mobile Networks (DTMNs). To provide effective and efficient communications, we proposed exploiting periodic encounter behaviours within node mobility for routing in DTMNs. In this thesis, we concentrated on the three problems: (1) detecting and extracting periodic encounter patterns within mobile data, (2) modelling and utilizing periodic encounter patterns and (3) connecting isolated sub-networks that are strongly internally connected, *e.g.*, by periodic encounters. In summary, we have made the following contributions.

1. We proposed a methodology for extracting the specific encounter patterns (both period and phase) for pairs of nodes that meet each other periodically. We also investigated the persistence of detected periodic behaviours and the small-world structure of networks formed by periodic encounters to understand their characteristics. The experimental results showed that the proposed method can detect and extract encounter patterns in real mobility traces with up to 100% accuracy. In addition, our studies showed that the majority of networks formed by the mobile nodes with periodic encounter behaviours have a small-world structure where messages between any pair of nodes can be delivered through a very small number of hops in the network.

2. To take advantage of periodic encounter behaviours, we introduced a graph model which integrates all encounter patterns. We showed how routing problems can be modelled as optimization problems. The only change in the graphs used to solve different problems is in the assignment of weights for the edges. To solve the routing problems, we proposed exact and approximation algorithms targeting at finding the routes for unicast and multicast routing. Using both real and synthetic datasets, the experimental results showed that our proposed approaches always find better routes with regard to delivery delay and energy cost compared to other state-of-art protocols.

3. To connect disconnected sub-networks, we proposed deploying stationary relays to the network to create new transmission opportunities. We showed that the relay node deployment problem can be modelled as the *k*-element connectivity problem with the objective to minimize the number of relays in the network such that the network is *k*-element connected, *i.e.*, *k* relay-disjoint paths. By deploying a very small number of relays to the disconnected network, our experimental results showed that the resulting *k*-element connected network had approximately 50% improvement in the delivery ratio in general.

Our three contributions are steps towards efficient and effective routing in DTMNs. Detecting and extracting periodic encounter patterns can identify mobile nodes that can be reliably used. The proposed graph model adopts the semantics of periodic patterns where routers can be scheduled for message delivery with respect to different metrics in the network. Relay deployment is used to join isolated sub-networks which are internally strongly connected, *e.g.*, by periodic encounters. With the combination of the solutions from the three problems, in the environment of DTMNs with the presence of disconnected sub-networks, message delivery can benefit greatly from the underlying periodicity within mobile data.

5.2 Future directions

There are still many interesting questions left open and worthy pursuing in the future. The following directions are some of the many we consider interesting in further the contributions presented in this thesis.

5.2.1 A real-time routing protocol

In this thesis the method in Chapter 2 for detecting and extracting periodic encounter patterns is currently performed in a centralized manner. In applications where there are thousands of mobile nodes, this may be not practical. To distribute the process of detection and extraction, one possible solution would be that a copy of the method can be implemented in each mobile node to discover its own patterns. In our studies, we showed that periodic encounter patterns do not last forever but change over a period of time. Pre-calculated routes will be seriously affected if the underlying periodicity changes. Our study regrading the persistence of detected periodic behaviours helps understand how long patterns reliably project themselves. As a result, after a certain period of time, the detection and extraction method can be re-executed to find new patterns in response to changes. Our future plan would be to design a routing protocol that utilizes periodicity for routing and dynamically adopts the changes in periodic behaviours.

5.2.2 A graph model capturing the probabilities of periodic encounters

In order to simplify our model, in Chapter 3 we assumed that every periodic encounter appears at its phase during every consecutive period in time. However, node movements are often disturbed by unexpected events. As a result, a periodic encounter may not be available to use at some phases of periods in time. To capture the stability of periodic encounters, probabilities can be used to describe how stable each periodic encounter is. We would like to design a graph model that is able to incorporate the probability of each periodic encounter. Using this type of graph model, the objective is to find the routes with the highest delivery probabilities as well as to minimize other metrics in the network. If there are disjoint paths, *e.g.*, relay-disjoint paths in this thesis, multiple copies of the message can independently travel through each route to increase the reliability of delivery.

5.2.3 Coordination among relays

In Chapter 4 the relay deployment problem discussed focused on building an unconnected infrastructure to help message delivery where there is no interaction between relays. By connecting relays in the network with underlying infrastructure, it is worthwhile to study coordination among relays to further improve network performance. For instance, redirecting messages between relays may lead to improvement. In addition, how to optimally deploy connected relays is another interesting direction for future research.

Bibliography

[1] First Mile Solution. http://www.firstmilesolutions.com/.

[2] Haggle project. http://www.haggleproject.org/.

[3] NS2. http://www.isi.edu/nsnam/ns/.

[4] ETS data for developers. http://www.edmonton.ca/transportation/ets/about_ets/ets-data-for-developers.aspx.

[5] SARAH: Delay-tolerant distributed services for mobile ad hoc networks. http://www-valoria.univ-ubs.fr/SARAH/presentation.shtml.

[6] M. Abolhasan, T. Wysocki, and E. Dutkiewic. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1–22, 2004.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[8] G. Anastasi, M. Conti, M. Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

[9] M. Balazinska and P. Castro. Characterizing mobility and network usage in a corporate wireless local-area network. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, pages 303–316, 2003.

[10] A. Bamis, J. Fang, and A. Savvides. A method for discovering components of human rituals from streams of sensor data. In *Proceedings of the International Conference on Information and knowledge management*, pages 779–788, 2010.

[11] N. Banerjee, M. Corner, and B. Levine. An energy-efficient architecture for DTN throwboxes. In *Proceedings of the International Conference on Computer Communications*, pages 776–784, 2007.

[12] N. Banerjee, M. Corner, D. Towsley, and B. N. Levine. Relays, base stations, and meshes: enhancing mobile networks with infrastructure. In *Proceedings of the 14th ACM International Conference on Mobile computing and Networking*, pages 81–91, 2008.

[13] S. Biswas, R. Tatchikou, and F. Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *IEEE Communications Magazine*, 44 (1):74–82, 2006.

[14] V. Borrel, F. Legendre, M. Dias de Amorim, and S. Fdida. SIMPS: Using sociology for personal mobility. *IEEE/ACM Transactions on Networking*, 17(3):831–842, 2009.

[15] M. Bouissou, A. Boissy, P. Neindre, and I. Veissier. Social behaviour in farm animals. In L. Keeling and H. Gonyou, editors, *The social behavior of cattle*, pages 113–145. CABI Publishing, 2001.

[16] J. Burgess, B. Gallagher, D. Jensen, and B. Levine. MaxProp: Routing for vehicle-based disruption-tolerant networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications.*, pages 1–11, 2006.

[17] B. Burns, O. Brock, and B. Levine. MV routing and capacity building in disruption tolerant networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 398–408, 2005.

[18] H. Cao, N. Mamoulis, and D. Cheung. Discovery of periodic patterns in spatiotemporal sequences. *IEEE Transactions on Knowledge and Data Engineering*, 19(4): 453–467, 2007.

[19] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. In *Proceedings of the 10th international conference on Ad-hoc, mobile, and wireless networks*, pages 346–359, 2011.

[20] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 6:606–620, 2007.

[21] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 192–200. Society for Industrial and Applied Mathematics, 1998.

[22] C. Chatfield. *Analysis of Time Series*. Chapman and Hall/CRC, 6th edition, 2003. ISBN 1584883170.

[23] C. Chekuri, A. Ene, and A. Vakilian. Node-weighted network design in planar and minor-closed families of graphs. In *Proceedings of the 39th international colloquium conference on Automata, Languages, and Programming*, pages 206–217, 2012.

[24] J. Chuzhoy and S. Khanna. An $o(k^3 log n)$-approximation algorithm for vertex-connectivity survivable network design. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 437–441, 2009.

[25] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill Science/Engineering/Math, 2nd edition, 2003. ISBN 0-26-2032937.

[26] P. Costa, C. Mascolo, M. Musolesi, and G. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 26(5):748–760, 2008.

[27] E. M. Daly and M. Haahr. Social network analysis for routing in disconnected delay-tolerant MANETs. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 32–40, 2007.

[28] Dartmouth College. CRAWDAD dataset. http://crawdad.cs.dartmouth.edu/data.php.

[29] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[30] S. Dreyfus and R. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[31] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli. Age matters: efficient route discovery in mobile ad hoc networks using encounter ages. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing*, pages 257–266, 2003.

[32] N. Eagle, A. S. Pentland, and D. Lazer. Inferring social network structure using mobile phone data. In *Proceedings of the National Academy of Sciences*, volume 36, pages 15274–15278, 2009.

[33] J. Edmonds and R. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1974.

[34] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.

[35] F. Farahmand, I. Cerutti, A. Patel, Q. Zhang, and J. Jue. Relay node placement in vehicular delay-tolerant networks. In *IEEE Global Telecommunications Conference*, pages 1–5, 2008.

[36] F. Farahmand, A. Patel, J. Jue, V. Soares, and J. Rodrigues. Vehicular wireless burst switching network: Enhancing rural connectivity. In *IEEE Global Telecommunications Conference*, pages 1–7, 2008.

[37] F. Farahmand, I. Cerutti, A. Patel, J. Jue, and J. Rodrigues. Performance of vehicular delay-tolerant networks with relay nodes. *Wireless Communications and Mobile Computing*, 11(7):929–938, 2011.

[38] L. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 3, pages 1548–1557, 2001.

[39] J. Feng, C.-W. Chang, S. Sayilir, Y.-H. Lu, B. Jung, D. Peroulis, and Y. Hu. Energy-efficient transmission for beamforming in wireless sensor networks. In *Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, pages 1–9, 2010.

[40] L. Fleischer, K. Jain, and D. P. Williamson. An iterative rounding 2-approximation algorithm for the element connectivity problem. In *42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 339–347, 2001.

[41] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1990. ISBN 0716710455.

[42] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3:46–55, 2004.

[43] A. Goldberg and R. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[44] M. Gonzalez, C. Hidalgo, and A. Barabasi. Understanding individual human mobility patterns. *NATURE*, 453:799–782, 2008.

[45] M. Grossglauser and D. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, 2002.

[46] M. Grossglauser and M. Vetterli. Locating mobile nodes with ease: learning efficient routes from encounter histories alone. *IEEE/ACM Transactions on Networking*, 14 (3):457–469, 2006.

[47] S. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[48] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the International Conference on Data Engineering*, pages 106–115, 1999.

[49] R. Handorean, C. Gill, and G. Roman. Accommodating transient connectivity in ad hoc and mobile settings. In *Pervasive Computing*, volume 3001, pages 305–322. 2004.

[50] T. He, K. Lee, and A. Swami. Flying in the dark: controlling autonomous data ferries with partial observations. In *Proceedings of the eleventh ACM international symposium on Mobile ad hoc networking and computing*, pages 141–150, 2010.

[51] T. He, A. Swami, and K. Lee. Dispatch-and-search: dynamic multi-ferry control in partitioned mobile networks. In *Proceedings of the 12th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 1–10, 2011.

[52] K. Herrmann. Modeling the sociological aspects of mobility in ad hoc networks. In *Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 128–129, 2003.

[53] S. Holland, P. W.; Leinhardt. Transitivity in structural models of small groups. *Comparative Group Studies*, (2):107–124, 1971.

[54] T. Hossmann, F. Legendre, and T. Spyropoulos. From contacts to graphs: Pitfalls in using complex network analysis for DTN routing. In *Proceedings of the International Conference on Computer Communications*, pages 1–6, 2009.

[55] M. Hsieh, E. Wu, and M. Tsai. FasterDSP: A Faster Approximation Algorithm for Directed Steiner Tree Problem. *Journal of Information Science and Engineering*, 22: 1409–1425, 2006.

[56] W. Hsu and A. Helmy. On nodal encounter patterns in Wireless LAN traces. *IEEE Transactions on Mobile Computing*, 9(11):1563–1577, 2010.

[57] W. Hsu and A. Helmy. CRAWDAD data set usc/mobilib (v. 2008-07-24). Downloaded from http://crawdad.cs.dartmouth.edu/usc/mobilib, 2008.

[58] W. Hsu, T. Spyropoulos, K. Psounis, and A. Helmy. Modeling spatial and temporal dependencies of user mobility in wireless mobile networks. *IEEE/ACM Transactions on Networking*, 17(5):1564–1577, 2009.

[59] P. Hui and J. Crowcroft. How small labels create big improvements. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 65–70, 2007.

[60] P. Hui, J. Crowcroft, and E. Yoneki. BUBBLE Rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589, 2011.

[61] D. Hutchison and J. P. Sterbenz. Resilinets: Resilient and survivable networks. *European Research Consortium for Informatics and Mathematics*, pages 30–31, 2009.

[62] F. Hwang and D. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[63] F. Hwang, D. Richards, and P. Winter. The Steiner tree problem. *Annals of Discrete Mathematics*, 53, 1992.

[64] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Proceedings of the International Multitopic Conference*, pages 62–68, 2001.

[65] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 448–457, 1998.

[66] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, 2004.

[67] J. Jakubiak and Y. Koucheryavy. State of the art and research challenges for VANETs. In *Proceedings of the Conference on Consumer Communications and Networking*, pages 912–916, 2008.

[68] R. Jathar and A. Gupta. Probabilistic routing using contact sequencing in delay tolerant networks. In *Proceedings of the 2nd International Conference on Communication Systems and Networks*, pages 1–10, 2010.

[69] Z. Jin, J. Wang, S. Zhang, and Y. Shu. Epidemic-based controlled flooding and adaptive multicast for delay tolerant networks. In *Proceedings of the 7th International Conference on Ubiquitous Intelligence Computing*, pages 191–194, 2010.

[70] D. Johnson, D. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In C. Perkins, editor, *In Ad Hoc Networking*, pages 139–172. Addison-Wesley, 2001.

[71] E. P. C. Jones, L. Li, J. K. Schmidtke, and P. A. S. Ward. Practical routing in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 6:943–959, 2007.

[72] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–10, 2009.

[73] H. Kerivin and R. Mahjoub. Design of survivable networks: A survey. *Networks*, 46 (1):1–21, 2005.

[74] M. Kim and D. Kotz. Periodic properties of user mobility and access-point popularity. *Personal Ubiquitous Computation*, 11:465–479, 2007.

[75] Y. Ko and N. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networking*, 6:307–321, 2000.

[76] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo. CRAWDAD data set dartmouth/campus (v. 2009-09-09). Downloaded from http://crawdad.cs.dartmouth.edu/dartmouth/campus, 2009.

[77] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[78] M. Lahiri and T. Berger-Wolf. Periodic subgraph mining in dynamic networks. *Knowledge and Information Systems*, 24(3):467–497, 2010.

[79] Y. Lando and Z. Nutov. Inapproximability of survivable networks. *Theoretical Computer Science*, 410(21):2122–2125, 2009.

[80] J. Laurila, D. GaticaPerez, I. Aad, J. Blom, O. Borneta, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Proc Mobile Data Challenge by Nokia Workshop, in conjunction with Int. Conf. on Pervasive Computing*, Newcastle, June 2012.

[81] S. Lee, J. Park, M. Gerla, and S. Lu. Secure incentives for commercial ad dissemination in vehicular networks. *IEEE Transactions on Vehicular Technology*, 61(6): 2715–2728, 2012.

[82] J. Leguay, A. Lindgren, J. Scott, T. Friedman, J. Crowcroft, and P. Hui. CRAWDAD data set upmc/content (v. 2006-11-17). Downloaded from http://crawdad.cs.dartmouth.edu/upmc/content, 2006.

[83] V. Lenders, J. Wagner, and M. May. Analyzing the impact of mobility in ad hoc networks. In *Proceedings of the ACM/Sigmobile Workshop on Multi-hop Ad Hoc Networks: from theory to reality*, pages 39–46, 2006.

[84] A. Levin. Algorithm for the shortest connection of a group of graph vertices. *Soviet Math*, 12:1477–1481, 1971.

[85] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In P. Dini, P. Lorenz, and J. Souza, editors, *Service Assurance with Partial and Intermittent Resources*, volume 3126, pages 239–254. Springer Berlin Heidelberg, 2004.

[86] C. Liu and J. Wu. An optimal probabilistic forwarding protocol in delay tolerant networks. In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, pages 105–114, 2009.

[87] C. H. Liu, T. He, K. Lee, K. K. Leung, and A. Swami. Dynamic control of data ferries under partial observations. In *Proceedings of the Conference on Wireless Communications and Networking*, pages 1–6, 2010.

[88] L. L. M. Grotschel and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 2nd edition, 1993. ISBN 3540567402.

[89] S. Ma and J. Hellerstein. Mining partially periodic event patterns with unknown periods. In *Proceedings of the International Conference on Data Engineering*, pages 205–214, 2001.

[90] J. Macker, V. Park, and M. Corson. Mobile and wireless internet services: putting the pieces together. *IEEE Communications Magazine*, 39(6):148–155, 2001.

[91] N. Maculan. The Steiner problem in graphs. *Annals of Discrete Mathematica*, 31: 185–222, 1987.

[92] Z. Melzak. On the problem of Steiner. *Canadian Mathematical Bulletin*, 4:143–148, 1961.

[93] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

[94] P. Meroni, S. Gaito, E. Pagani, and G. P. Rossi. CRAWDAD data set unimi/pmtr (v. 2008-12-01). Downloaded from http://crawdad.cs.dartmouth.edu/unimi/pmtr, 2008.

[95] S. Moon and A. Helmy. Understanding periodicity and regularity of nodal encounters in mobile networks: A spectral analysis. In *Proceedings of IEEE Conference on Global Telecommunications*, pages 1–5, 2010.

[96] M. Motani and V. Srinivasan. Peoplenet: engineering a wireless virtual social network. In *Proceedings of the International Conference on Mobile Computing and Networking*, pages 243–257, 2005.

[97] A. Mtibaa, M. May, C. Diot, and M. Ammar. Peoplerank: social opportunistic forwarding. In *Proceedings of the 29th conference on Information communications*, pages 111–115, 2010.

[98] S. Nelson, M. Bakht, R. Kravets, and A. Harris. Encounter based routing in DTNs. *Proceedings of SIGMOBILE on Mobile Computing and Communication Review*, 13: 56–59, 2009.

[99] H. Noshadi, E. Giordano, H. Hagopian, G. Pau, M. Gerla, and M. Sarrafzadeh. Remote medical monitoring through vehicular ad hoc network. In *Proceedings of the Conference on Vehicular Technology*, pages 1–5, 2008.

[100] Z. Nutov. Approximating Steiner networks with node weights. In *Proceedings of the 8th Latin American conference on Theoretical informatics*, pages 411–422, 2008.

[101] C. Oliveira and P. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32:1953–1981, 2005.

[102] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proceedings of the International Conference on Data Engineering*, pages 412–421, 1998.

[103] C. Palazzi, M. Roccetti, S. Ferretti, and S. Frizzoli. How to let gamers play in infrastructure-based vehicular networks. In *Proceedings of the International Conference on Advances in Computer Entertainment Technology*, pages 95–98, 2008.

[104] A. Pentland, R. Fletcher, and A. Hasson. DakNet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, 2004.

[105] P. Pereira, A. Casaca, J. Rodrigues, V. Soares, J. Triay, and C. Cervello-Pastor. From delay-tolerant networks to vehicular delay-tolerant networks. *IEEE Communications Surveys Tutorials*, 14(4):1166–1182, 2012.

[106] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing for mobile computers. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, 1994.

[107] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[108] J. Plesnik. A bound for the Steiner tree problem in graphs. *Mathematica Slovaca*, 31:155–163, 1981.

[109] B. Polat, P. Sachdeva, M. Ammar, and E. Zegura. Message ferries as generalized dominating sets in intermittently connected mobile networks. *Pervasive and Mobile Computing*, 7(2):189–205, 2011.

[110] J. Pujol, A. Toledo, and P. Rodriguez. Fair routing in delay tolerant networks. In *Proceedings of the International Conference on Computer Communications*, pages 837–845, 2009.

[111] M. Radenkovic and B. Wietrzyk. Mobile ad hoc networking approach to detecting and querying events related to farm animals. *International conference on Networking and Services*, 0:109–120, 2006.

[112] M. Radenkovic and B. Wietrzyk. Wireless mobile ad-hoc sensor networks for very large scale cattle monitoring. In *Proceeding of the International Workshop on Applications and Services in Wireless Networks*, pages 47–058, 2006.

[113] R. Ramanathan, R. Hansen, P. Basu, R. Rosales-Hain, and R. Krishnan. Prioritized epidemic routing for opportunistic networks. In *Proceedings of the 1st international MobiSys workshop on Mobile opportunistic networking*, pages 62–66, 2007.

[114] V. Rayward-Smith. The computation of nearly minimal Steiner trees in graphs. *International Journal of Mathematical Education in Science and Technology*, 14:15–23, 1983.

[115] P. Romano. The range vs. rate dilemma of WLANs. http://www.eetimes.com/design/communications-design/4009276/The-Range-vs-Rate-Dilemma-of-WLANs.

[116] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau. CRAWDAD data set cambridge/haggle (v. 2009-05-29). Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle, 2009.

[117] S. Shahbazi, A. Harwood, and S. Karunasekera. On placement of passive stationary relay points in delay tolerant networking. In *Proceedings of the International Conference on Advanced Information Networking and Applications*, pages 764–771, 2011.

[118] C. Song, Z. Qu, N. Blumm, and A. Barabsi. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

[119] L. Song and D. Kotz. Evaluating opportunistic routing protocols with large realistic contact traces. In *Proc. of the ACM CHANTS*, pages 35–42, 2007.

[120] H. Soroush, N. Banerjee, A. Balasubramanian, M. Corner, B. Levine, and B. Lynn. DOME: a diverse outdoor mobile testbed. In *Proceedings of the 1st ACM International Workshop on Hot Topics of Planet-Scale Mobility Measurements*, pages 21–26, 2009.

[121] J. Spinrad. *Efficient graph representations*. American Mathematical Society, 2003. ISBN 0821828150.

[122] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 79–85, 2007.

[123] T. Spyropoulos, K. Psounis, and C. Raghavendra. Efficient routing in intermittently connected mobile networks: The single-copy case. *IEEE/ACM Transactions on Networking*, 16(1):63–76, 2008.

[124] R. Startz. Binomial autoregressive moving average models with an application to u.s. recessions. *Journal of Business on Economic Statistics*, 26(1):1–8, 2008.

[125] J. Su, A. Chin, A. Popivanova, A. Goel, and E. Lara. User mobility for opportunistic ad-hoc networking. In *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 41–50, 2004.

[126] J. Su, A. Goel, and E. Lara. An empirical evaluation of the student-net delay tolerant network. In *Proceedings of the 3rd Annual International Conference on Mobile and Ubiquitous Systems*, pages 1–10, 2006.

[127] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24:573–588, 1980.

[128] K. P. Thrasyvoulos Spyropoulos and C. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proc. of the ACM SIGCOMM workshop on DTN*, pages 252–259, 2005.

[129] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, Duke University, CS-2000-06.

[130] L. Vu, Q. Do, and K. Nahrstedt. 3R: Fine-grained encounter-based routing in delay tolerant networks. In *Proceedings of IEEE International Symposium on WoWMoM*, pages 1–6, 2011.

[131] C. Wang and W. Li. On the autopersistence functions and the autopersistence graphs of binary autoregressive time series. *Journal of Time Series Analysis*, 32(6):639–646, 2011.

[132] T. Wang and C. Low. Reducing message delay with the general message ferry route (MFR*) problem. In *Proceedings of the International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 380–387, 2011.

[133] T. Wark, P. Corke, P. Sikka, L. Klingbeil, Y. Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley. Transforming agriculture through pervasive wireless sensor networks. *IEEE Pervasive Computing*, 6:50–57, 2007.

[134] D. Watts and S. Strogatz. Collective dynamics of small-world networks. *Nature*, 393 (6684):440–442, 1998.

[135] J. Whitbeck, M. Dias de Amorim, V. Conan, and J. Guillaume. Temporal reachability graphs. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 377–388, 2012.

[136] B. Wietrzyk and M. Radenkovic. Enabling large scale ad hoc animal welfare monitoring. *International Conference on Wireless and Mobile Communications*, 0:401–409, 2009.

[137] B. Wietrzyk, M. Radenkovic, and I. Kostadinov. Practical MANETs for pervasive cattle monitoring. *International Conference on Networking*, 0:14–23, 2008.

[138] P. Winter. Steiner problem in networks: a survey. *Networks*, 17:129–167, 1987.

[139] J. Wolf, S. Hallmark, M. Oliveira, R. Guensler, and W. Sarasua. Accuracy issues with route choice data collection by using Global Positioning System. *Transportation Research Record Journal*, 1660(2):66–74, 1999.

[140] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):613–628, 2003.

[141] Q. Yuan, I. Cardei, and J. Wu. Predict and relay: an efficient routing in disruption-tolerant networks. In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, pages 95–104, 2009.

[142] A. Zelikovsky. A series of approximation algorithms for the acyclic directed Steiner tree problem. *Algorithmica*, 18:99–110, 1997.

[143] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 227–238, 2004.

[144] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang. Study of a bus-based disruption-tolerant network: Mobility modeling and impact on routing. In *Proceedings of ACM MobiCom*, pages 195–206, 2007.

[145] Z. Zhang and Z. Fei. Route design for multiple ferries in delay tolerant networks. In *Proceedings of the International Conference on Wireless Communications and Networking*, pages 3460–3465, 2007.

[146] W. Zhao and M. Ammar. Message ferrying: proactive routing in highly-partitioned wireless ad hoc networks. In *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 308–314, 2003.

[147] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 187–198, 2004.

[148] W. Zhao, M. Ammar, and E. Zegura. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *Proceedings of the International Conference on Computer Communications Joint Conference of the IEEE Computer and Communications Societies*, pages 1407–1418, 2005.

[149] W. Zhao, Y. Chen, M. Ammar, M. Corner, B. Levine, and E. Zegura. Capacity enhancement using throwboxes in DTNs. In *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 31–40, 2006.