Providing Location-privacy in Opportunistic Mobile Social Networks

by

Rui Huang

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the MASTER OF APPLIED SCIENCE degree in
Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
School of Electrical Engineering and Computer Science
University of Ottawa

Abstract

Users face location-privacy risks when accessing Location-Based Services (LBSs) in an Opportunistic Mobile Social Networks (OMSNs). In order to protect the original requester's identity and location, we propose two location privacy obfuscation protocols utilizing social ties between users.

The first one is called Multi-Hop Location-Privacy Protection (MHLPP) protocol. To increase chances of completing obfuscation operations, users detect and make contacts with one-hop or multi-hop neighbor friends in social networks. Encrypted obfuscation queries avoid users learning important information especially the original requester's identity and location except for trusted users. Simulation results show that our protocol can give a higher query success ratio compared to its existing counterpart.

The second protocol is called Appointment Card Protocol (ACP). To facilitate the obfuscation operations of queries, we introduce the concept called Appointment Card (AC). The original requesters can send their queries to the LBS directly using the information in the AC, ensuring that the original requester is not detected by the LBS. Also, a path for reply message is kept when the query is sent, to help reduce time for replying queries. Simulation results show that our protocol has a higher query success ratio than its counterparts.

We have also developed a new OMSN simulator, called OMSN Routing Simulator (ORS), for simulating OMSN protocols more efficiently and effectively for reliable performance.

Acknowledgements

First, I would like to express my sincere gratitude to Prof. Amiya Nayak for his continuous support, motivation, advice and immense knowledge throughout my thesis work. I could not have imagined having a better mentor for my Master's study.

Also, I would like to thank my family and friends, especially Yichao Lin, for providing me with unfailing support and encouragement throughout the process of researching and writing my thesis. This accomplishment would not have been possible without them. Thank you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Location-privacy is becoming a major concern in the Opportunistic Mobile Social Networks (OMSN) which is a kind of Delay Tolerant Networks (DTNs) [3] featuring lack of continuous connectivity. More specifically, in OMSNs, it is not necessary for senders to have an end-to-end routing path to their destinations. Users make contact when they encounter each other. Location-Based Services (LBSs) are common applications in OMSNs and they are widely used in "military and government industries, emergency services and the commercial sector" [19], especially after the proliferation of localization technologies, like GPS. Many people access LBSs with their portable devices and send their location to LBS providers. In this case, LBS users face a continuous risk that their location may be leaked from LBS applications. That makes people unwilling to use LBSs. Thus, protecting location privacy has been a critical issue in LBSs.

1.0.1 Motivation and Objective

LBSs which use the location information of users can be considered as "two types of application design: push and pull services" [19]. For example, you may receive an advertisement when you enter an area, which is a push service; if you look for the nearest restaurant, you must pull information from the network. It is obvious that you must reveal your location if you use LBS. When you use a LBS application to find the nearest restaurant, you actually tell the LBS provider your location and your next destination which might have a restaurant nearby. If attackers can access the LBS provider's database, they can learn that information. Then you may receive a lot of advertisements from surrounding restaurants.

Such inference attack, in addition to bothering you with advertisements, can become more dangerous when you send more queries to the LBS provider. If you use the former LBS application 10 times in a day, attackers can learn your trace from the information so that they can infer more information including your identity and home address. According to [8], "Hoh used a database of week-long GPS traces from 239 drivers in the Detroit, MI area. Examining a subset of 65 drivers, their home-finding algorithm was able to find plausible home locations of about 85%, although the authors did not know the actual locations of the drivers' homes". Therefore, the LBS provider is considered as a semi-trusted party so that users must protect their location-privacy when they use LBS applications.

The OMSN is defined "as decentralized opportunistic communication networks formed among human carried mobile devices that take advantage of mobility and social networks to create new opportunities for exchanging information and mobile ad hoc social networking" [17]. In other words, OMSN is a kind of mobile ad hoc network, and the information and technology of social network are also used in it. People carrying smartphones which contain WiFi or Bluetooth can form a typical OMSN. Since the WiFi, Bluetooth on smartphones can be used for discovering other devices and direct communication between devices, users can communicate with others who are within their communication range without using any infrastructure, which is the basic requirement of an OMSN. Since users in OMSN could also use LBS applications, their location-privacy must be protected. Besides, the information in the social network allows users to identify friends. Therefore, it is necessary to have location-privacy protection protocol using the social networks.

1.0.2 Objectives

Disconnection, decentralization and highly delays are obvious features for OMSN so that a decentralized strategy may benefit for the location-privacy protection protocol in OMSN. Besides, decreasing interactions between users and servers can significantly cut down the time cost. To prevent attackers from learning users' private information, the protocol should obfuscate users' location and hide their identities.

1.0.3 Contribution

We propose two new decentralized location-privacy protecting protocols for OMSN so that they do not need a three-party server which is used to obfuscate queries for users. We also create a new simulator for OMSNs called OMSN Routing protocols Simulator (ORS) to evaluate our protocols.

The first protocol which we propose is a distributed location-privacy algorithm, called Multi-Hop Location-Privacy Protection (MHLPP). It guarantees location-privacy and achieves a higher query success ratio. The introduction of social networks enables us to hide the original requester's information behind his friends. When a user wants to send a query, he starts to look for friends based on information in his social network. He sends his query to the first encountered friend who is then responsible for forwarding the query to the intended location. This friend can also pass the query to one of his friends when they encounter. When the distance between the user carrying this query and the original requester exceeds a specified threshold, the user sends the query to the LBS server directly without having to find a friend to pass on. At that time, he also replaces the original requester's information with its own identity and location, which enables the LBS server to receive the query without any information about the original requester. After receiving the query, the LBS server replies to the last friend (the user sending the query to the LBS) who then transmits it to the original requester.

Our second protocol is also a distributed location-privacy algorithm called Appointment Card Protocol (ACP), which aims to guarantee location-privacy and reach a higher query success ratio. The introduction of social networks enables us to hide the original requester's information behind his friends. We introduce the Appointment Card (AC) as a kind of intermediary which records a serial of agents. The original requester sends his query using the identity of the first agent in the AC to the LBS, which prevent the LBS learns the identity of the requester. The reply of the query can be delivered back to the original requester along with the same serial of agents as in the AC one by one. The last agent called the trusted agent is a user in the social network of the original requester, in other words, he is a friend (or a friend of friends) of the original requester. The trusted agent separates the stranger-agents and the original requester so that no stranger knows the identity of the original requester. The query-delivery success ratio of the ACP is in a similar level as the no-privacy protocol comparison in our experiment.

Our new simulator ORS is inspired by a well-known simulator, ONE [7], which is used by many researchers to test their models and protocols in DTN. The major reason why we created a new simulator is that the ONE is not designed for OMSN and lacks social networking concepts for it to be effective in OMSN. Adding social network information into the ONE simulator must modify its basic structure, which is problematic and might not ensure correctness of our experiments. We used more efficient algorithms to make our simulator run two times faster than ONE.

1.0.4 Thesis Organization

This thesis is organized as follows:

Chapter 2 describes the concept of Mobile Ad hoc networks, Delay Tolerant Networks, Location-Based Services and social networks. We also give an overview of the existing location-privacy protocols.

Chapter 3 describes our first protocol, MHLPP, in detail. The performance of MHLPP has been compared against its counterpart, Hybrid and Social-aware Location-Privacy in Opportunistic mobile social networks (HSLPO) [26].

Chapter 4 elaborates on the second protocol, ACP. We describe how it works and how it enhances the location-privacy. Its performance is compared against Binary Spray and Wait (BSW) [20], distributed social based location privacy protocol (SLPD) [25], and our Multi-Hop Location-Privacy Protection (MHLPP).

Chapter 5 describes our new simulator which has been instrumental in simulating MHLPP and ACP in realistic settings. We introduce the key enhancements and the algorithms which are used in the simulator for making it more efficient.

Chapter 2

Background and Related Work

2.1 Mobile Ad hoc Networks

"Ad-hoc networks are infrastructure-less and cooperation-based networks which means that the network topologies must be decided by the network sensors themselves" [28]. In other words, the major feature of ad-hoc networks is that they have no infrastructures.

The Mobile Ad hoc NETwork (MANET), also called the Wireless Ad-hoc NETwork (WANET), is a decentralized wireless network with no pre-existing infrastructure. In MANET, because users move independently, the topology changes frequently. Without the help of infrastructure, users can only communicate when they encounter each other. In other words, users communicate only if they are covered by each other's communication range. Because of the above characteristics, the MANET can be viewed as a kind of Delay-Tolerant Network (DTN) that lacks continuous network connectivity. The MANET can hardly establish instantaneous end-to-end paths, which compels the routing protocols in MANET to use a "store-and-forward" strategy in which users forward messages when they encounter others, or they store messages for later delivery. As shown in Figure 2.1, the user A has a message for C but does not know where C is, so he sends the message to B when they encounter each other. The user B then stores the message and moves along.

The message is then delivered to C when both B and C come in contact with each other at a later time. It is obvious that whether the message can be delivered depends on the movement of the users. If B will never meet C or the message is held by a user for a long period of time, it must be dropped after some timeout. The delivery process can lead to low success ratio and is often time-consuming.

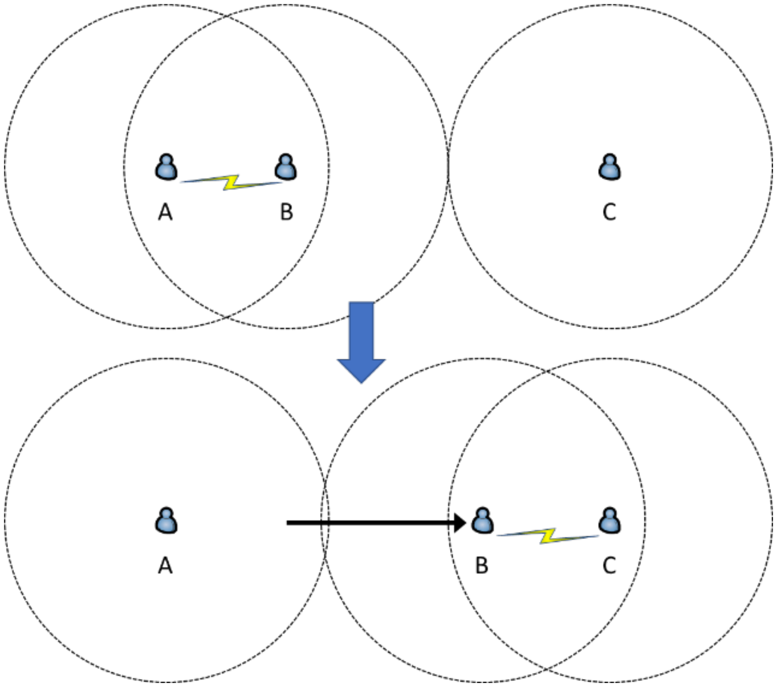
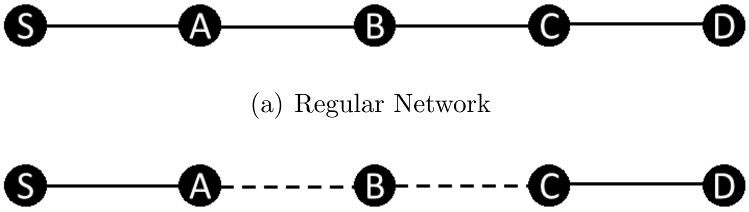


Figure 2.1: MANET

2.2 Delay Tolerant Network

Regular networks, e.g. Internet, always have end-to-end paths. As shown in Figure 2.2(a), two nodes S and D are connected through nodes A, B, and C. The maximum round-trip time is not excessive, and their drop probability is also small. Compared to the regular network, there is a class of challenged networks [3] which lack end-to-end path and suffer from high latency and long queuing delays. As shown in Figure 2.2(b), when node B is down or non-existent, there is no connection between nodes A and C, for which extensive latency is inevitable if the source S sends a message to the destination D.



(a) Regular Network

(b) DTN

Figure 2.2: The comparison between regular networks and DTNs

Since the challenged networks are significantly different from the regular networks, researchers have introduced a new architecture, called Delay Tolerant Network (DTN), to deal with the unique features of the challenged networks. Terrestrial Mobile Networks, Exotic Media Networks, Military Ad-Hoc Networks and Sensor/Actuator Networks are examples of typical DTNs.

2.2.1 Spray and Wait Protocol

The Spray and Wait [20] is a well-known protocol for message dissemination in DTNs. Although it is not as efficient as protocols on Internet, it works quite well in DTNs. A message is initialized with several copies, a part of which are given to others when users encounter each other. Users keep forwarding copies until they each have only one copy of the message. When a user carrying at least one copy of the message encounters the destination, he gives all his copies to the destination to complete the delivery. The Binary Spray and Wait (BSW) [20] is an optimized version of the Spray and Wait, which we use for comparison in this thesis. The user who creates the message also makes several copies of that message. He gives half of his copies to the user whom he encounters so that both he and the other user have half of the copies. The users who get copies of the message continue to give half of their copies to others until someone has only one copy to pass on to the destination.

2.2.2 Other DTN Protocols

2.2.2.1 Direct Contact Scheme

The simplest strategy for DTN is that the source holds the message until he meets the destination, which is called the direct contact scheme. So, a direct path between the source and the destination is necessary for a successful delivery. It is possible that the source never comes in contact with the destination in which case the message is not delivered.

2.2.2.2 Replica Based Protocols

The replica based protocols (e.g. [5], [9], [22], [11], and [16]) work by making several replicas of the message so that users can retransmit them upon connection establishment. The former BSW is also a kind of replica based protocols. Compared to the direct contact scheme, the replica-based protocols make it easy for messages to be delivered. But, they require more resources than the direct contact scheme because they need more memory to store the replicas. Therefore, making a reasonable decision on the message replication is the key to success of these kinds of protocols.

2.2.2.3 Knowledge-Based Protocols

Different from the former replica protocols which require no knowledge about the topology of the network, the users in knowledge-based protocols try to evaluate their own view of the topology of the network so that they can make better forwarding decisions, e.g. [24], [6] and [12]. However, the topology changes so frequently that it is hard for users to have an accurate topology.

2.2.2.4 Coding Based Protocols

Authors in [10] and [1] have suggested the approach that introduces coding techniques into the routing protocols. Instead of making a few replicas, coding based protocols encrypt data to make a large number of message blocks. If the destination receives a part of the blocks, he can decrypt the message.

2.3 Location-Based Services

The Location-Based Services (LBS) use users' location information to provide services as shown in Figure 2.3. Your smartphone can detect your coordinate and send it to LBS application server which is responsible for providing service based on the coordinate. The point of interest and the location advertisement are familiar instances of LBS. The LBS providers collect users' location information to provide service, which makes LBS providers a significant target of attack. When attackers access the databases of the LBS providers, they can learn all sensitive information in the servers. So, the LBS providers should be viewed as semi-trusted, which might expose information in front of vicious attackers.

Users face risks of information breach when they access a semi-trusted LBS provider because anyone who has access to data in LBSs is able to steal and misuse LBS users' location-privacy. Considering that LBSs rely on location-aware computing, it is unavoidable to leak users' location from LBSs. Therefore, balancing "these two competing aims of location privacy and location awareness" [2] is always a challenge.

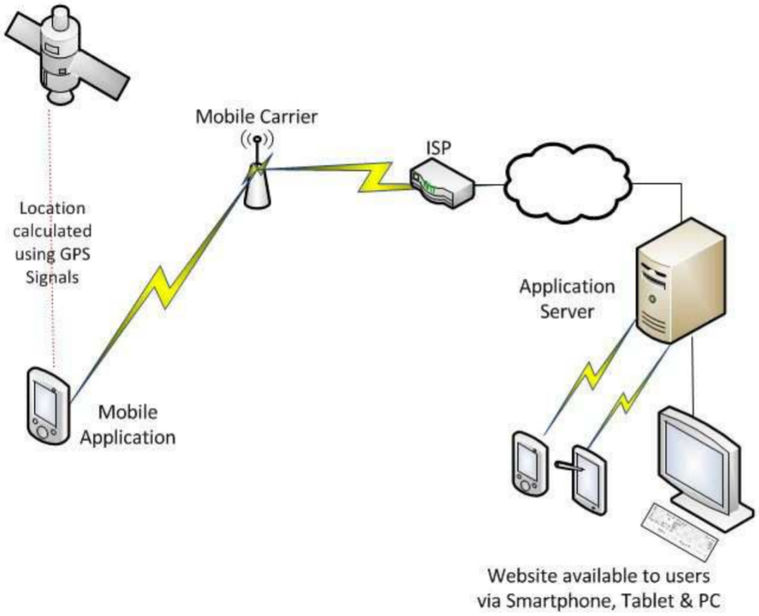

Mobile Carrier
ISP
Location calculated using GPS Signals
Application Server
Mobile Application
Website available to users via Smartphone, Tablet & PC

Figure 2.3: LBS [18]

2.4 Social Networks

Social networks which contain social interactions and personal relationships are used in location-privacy protection so that users can determine who can or cannot be trusted. In [23], the relationship of a pair of people (e.g., user i and user j) is described as a pair of directed relationship strength. The directed relationship strength from the user i to the user j can be denoted by RS^{ij}. We should notice that RS^{ij} might not be equal to RS^{ji}, because the user i might like the user j very much while the user j just views the user i as an acquaintance. Based on the relationship strength, it is possible to estimate the relational tie between two people and predict whether they are friends or strangers.

The relationship strength is compared with a threshold; if the relationship strength is higher than the threshold then the two people are friends, otherwise they are strangers. In the protocols we mention in the thesis, researchers always assume that friends are trustful. That is reasonable because you might feel more secure when your friends forward your message instead of a stranger.

2.5 Location-Privacy Protocols

The basic idea of most location-privacy protocols is hiding the original requester behind a set of users so that attackers can hardly infer the identity of the original requester. In other words, when a user sends a query, his identity cannot be inferred based on the information in the query easily. For example, the original requester can use a pseudonym instead of his real name or use an obfuscated location to send his queries, so that attackers cannot tell the difference between two sets of queries.

The location-privacy protocols are considered centralized and distributed based on whether they require any infrastructure for operation.

2.5.1 Centralized Protocols

Centralized protocols always need some infrastructure for the obfuscation process. The infrastructure could be any equipment which acts as an anonymization server. When the user wants to send a query to the LBS server, he sends the query to the anonymization server. The anonymization server changes the identity and the location in the query, which is called anonymization or obfuscation. Then, the anonymization server forwards the modified query to the LBS server, so that the attacker who attacks the LBS server can hardly learn the identity of the original requester. The LBS server can send the reply message to the anonymization server which is responsible for forwarding the reply message back to the original requester. In this way, the original requester gets the information he needs, while avoiding from being located by the attackers.

The advantage of a centralized protocol is that the anonymization server has more resources and information than users in the network, which enable the server to achieve better anonymization performance and provide sufficient protection for users. For example, if the anonymization server knows locations of all users in the network, it can modify the location and identity in the queries to the most suitable, with which the LBS server can provide acceptable results while the original requester is not exposed.

There are two major shortcomings of these kinds of protocols: i) it is hard to deploy infrastructure in some areas, ii) since the anonymization server knows users' location, it is also a risk for the users.

2.5.1.1 Single Server

Authors in [14] use a central anonymity server through which the mobile users can send queries to LBS. To make the communication between users and the central anonymity server trustful, users set up an encrypted connection with the anonymity server at the beginning. Users sends their encrypted queries to the anonymity server, so that the anonymity server is the only one who can learn the information in the query. The anonymity server decrypts the queries and uses a cloaking algorithm to perturb the position information in the queries. Then the anonymity server sends the modified queries to external services (e.g. LBS). This is a typical centralized protocol, which can reduce the re-identification risk for the users. Its drawback is that a continuous connection to the server is necessary for each user, which is hard to achieve in a sparse DTN. We cannot also assume that a MANET user can have a stable connection with a central anonymity server either.

In [15], researchers employ a matchmaker which is used to match users and advertisements, by which users can achieve anonymization of their identities and locations from the matchmaker. The system architecture in [15] is similar to the former [14], while authors in [15] just focus on the advertisement service instead of the "external services" in [14]. The role of the matchmaker is simply matching users and advertisements. Although the functions of the matchmaker in [15] and the central anonymity server in [14] are different, the intermediate trusted three-party servers (i.e., the matchmaker and the central anonymity server) separate users and the application server (e.g., an advertisement server, LBS server). The architecture in [15] does not require an encryption process, so the attackers can learn the information in the queries. When a large number of queries arrive at the matchmaker in a short time from different users, it helps the matchmaker mix the queries so that attackers can hardly trace the query, but the burden of the users who are near the matchmaker can be heavy.

In [21], researchers use a trusted, third-party location anonymization engine (LAE) that acts as a middle layer between mobile users and the LBS provider, in which exact locations and requests from clients are replaced by a location anonymization engine before they arrive at the LBS provider. Therefore, it appears that the suggest approach is almost like that shown in Figure 2.4. A trusted server is placed between users and the application server, which hides users' information and offers sufficient information for the application servers to provide acceptable service.

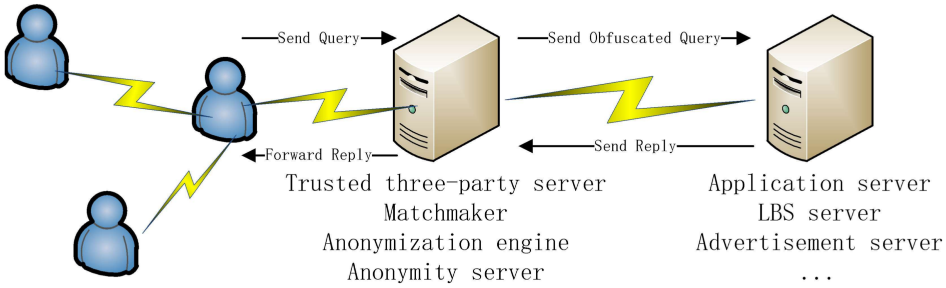


Figure 2.4: Single Anonymization Server

2.5.1.2 Multiple Servers

The above protocols always use a single anonymization server, while there are other protocols which need multiple infrastructures.

Authors in [13] propose a protocol, called Social-based PRivacy-preserving packet forwardING (SPRING), for vehicular delay-tolerant network. In their work, they employ Roadside Units (RSUs), which are a type of equipment deployed along the roadside, to assist the packet forwarding and achieve conditional privacy preservation. These RSUs are located at high social intersections, so that vehicles which pass by the RSUs send their messages to the RSUs. The RSUs have sufficient resource so that they can hold the messages for a long time, which decreases the probability of messages being dropped. The messages are forwarded to proper next-hop vehicles when the vehicles pass by the RSUs. Since messages are held by the RSUs for a period, attackers can hardly trace the messages. Besides, a large number of vehicles send many packets to these RSUs, which enables the RSUs to serve as mix servers. The advantage of SPRING is that it improves the delivery success ratio and privacy-protection performance. However, deploying the RSUs is not always feasible.

Another example is the REAL [4], in which researchers use sensor nodes which are scattered throughout the network to provide anonymized locations for users, as shown in Figure 2.5. The whole system area is partitioned into a set of aggregate locations by the sensor nodes, where there are at least k persons. To provide better location-based service, they minimize the areas of aggregate locations. When a user sends a query to the LBS, he uses the location of the sensor nodes which he belongs to, so that the attacker cannot tell the difference between the requester and the other $k-1$ users in that aggregate location. This is a typical k anonymity algorithm, whose main disadvantage is that it is difficult to deploy the sensor nodes in real-world. Besides, the mix servers and sensor nodes might be more prominent targets than the LBS providers.

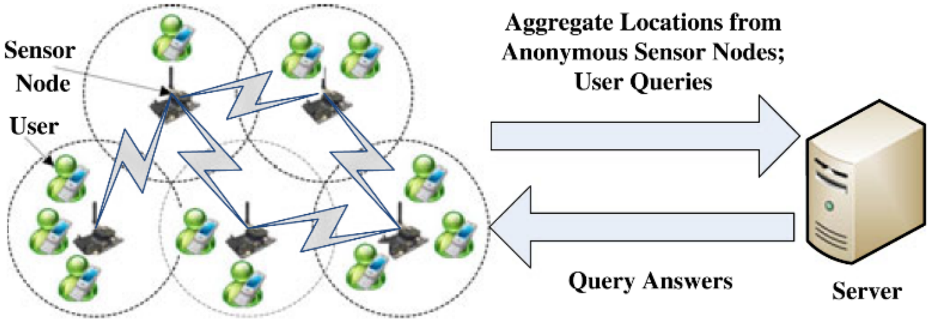

Sensor Node
User
Aggregate Locations from Anonymous Sensor Nodes; User Queries
Query Answers
Server

Figure 2.5: REAL [4]

2.5.2 Distributed Protocols

Although centralized protocols also have good performance in their delivery success ratio and privacy-protection in their own field, the MANET is a network without infrastructure. Distributed protocols are more appropriate for applications in MANET. A problem for a distributed protocol it that whether a user can trust others. Authors in [25], [26] and [27] introduce the social tie to determine whether a user is trustful.

In the distributed social based location privacy protocol (SLPD) [25], authors use two phases: the obfuscation phase and the free phase. A query from an original requester always starts in the obfuscation phase and passes through k friends. For example, the original requester sends the query to one of his friend in one hop, then the friend forwards the query to another friend. We call these friends the agents. That process repeats for exactly k times. When the k^{th} friend get the query, he switches the query to the free phase and replaces the sender identity with his own identity, and then sends the query to the destination (e.g., LBS) using any DTN protocol. The LBS then sends the reply to all k friends who are responsible for forwarding the message to the original requester. In this way, attackers can only learn the identity of all k friends instead of the original requester. Since each of these k friends knows the identity of the original requester, the message can be forwarded to the original requestor. Therefore, the attacker can hardly learn the identity of the original requester. The disadvantage of the protocol is that it is hard to encounter a friend in the network, which can decrease the success ratio for the queries. In the protocol, Hybrid and Social-aware Location-Privacy in Opportunistic mobile social networks (HSLPO) [26], authors try to improve the delivery performance by using a stochastic model which uses a Markov model for location predication. The major process is similar to that in SLPD, but an agent can forward the query to a user who is not a friend of the original requester if the user has more chance to deliver the query and a trust value is larger than a threshold. In other words, an agent continuously searches his surrounding to find the original requester's friends. If the agent cannot find the original requester's friend but his own friend, he checks whether his friend has more chance to deliver the query than him using the Markov model. If his friend is more suitable for forwarding, he sends the query to his friend. The performance of HSLPO depends on the Markov model, that is, whether it can predict users' movement accurately. In real-world, the movement model for people is more complicated than that used in experiments.

Another protocol, called Location Privacy-Aware Forwarding (LPAF) [27], also attempts to improve the performance of SLPD. Both LPAF and SLPD are similar, but LPAF just adds more friends to the protocol. When an agent cannot find a close friend (i.e., their trust value is high), he will try to find other general friends (i.e., their trust values is lower than the close friends). That might be a safety tradeoff, because some ineligible users in SLPD can be chosen as friends based on the additional criteria imposed by LPAF.

In fact, both the HSLPO and LPAF do not successfully address the problem of finding friends as agents. Although we use the friends of friends or set the threshold for friends low, there are only a few users who can be chosen as agents. In LPAF, the identity of the requester is even exposed to someone who is not very trustful.

Chapter 3

Multi-Hop Location-Privacy Protection

3.1 System Model

Our network architecture consists of two main entities: Users and LBS Providers (LBSPs). Due to the introduction of the social network, the users' social information can be used in obfuscation forwarding process. Based on available information in the social network, the relationship between two users can be considered as friends or strangers. The user who makes a query to an LBSP will be called the original requester while the others are called intermediate users. LBSPs are located in fixed locations and their coordinates are known by all users when users join the network. Attackers are assumed to be able to access LBSPs, and attempt to locate original requesters. We assume that the LBSPs are semi-trusted and the strangers are un-trusted. We also assume that both entities have sufficient resources, like computational capability, storage and battery power.

Since two friends could be a pair of multi-hop neighbors, users can leverage Optimized Link State Routing Protocol [29] to seek friends continuously after entering the network, so that they can recognize each other and make contact in time. When a user carries an obfuscation phase query, he might send the query to a multi-hop friend through several strangers. In this case, a secure communication is necessary for between them, so that they must send the query encrypted to prevent strangers from learning anything about the query. Each user obtains a pair of asymmetric keys (public and secret key) before he joins the network from a certificate authority using well-regarded techniques, like in [28]. Whenever a user detects a new friend, he sends a request to the friend asking for his public key. In this way, a user can get his friends' public key when they encounter each other. Even though several strangers can be active in the obfuscation phase, the queries can still be securely sent to the user's friend.

The relationship strength is often "a hidden effect of nodal profile similarities" [30]. Let $SV_{i,j}$ denote a value of relationship strength which user i determines whether user j is an acceptable friend based on the relationship strength. For every pair of users (i and j), we assume that there is an $SV_{i,j}$. If $SV_{i,j}$ is bigger than a specific friend threshold T_{min}, set by the original requester, user j is considered as a friend of user i; otherwise, it will be treated as a stranger. The notations used in this paper and their meanings are shown in Table 3.1.

3.2 Details of MHLPP

MHLPP aims to protect the original requester's (N_0's) location-privacy using an obfuscation path. In other words, a query q which needs to be obfuscated must go through a series of friends after it leaves N_0. The whole process includes two parts: the obfuscation phase and the free phase. In the former phase, q is only transmitted among friends, until it is sent to an area called "$obfuscationarea$". At the end of that phase the last friend N_f replaces all N_0's information by its own and forwards q with an arbitrary DTN forwarding protocol, like the one suggested in [31]. In this case, what attackers can learn from the

Table 3.1: MHLPP Symbols

Parameter	Meanings
N_0	the original requester
N_i	if $i > 0$, it denotes the friend chosen by N_{i-1}. If $i = 0$, it is N_0.
N_f	the last friend who handles the obfuscation query
N_d	the destination or the LBSP
K_i	the public key of N_i
S_i	the secret key of N_i
q	a query of N_0
r_q	the requirement for the query q
msg	a message contains q and r_q
$Emsg_i$	the encrypted msg using K_i
S_{id}	the original requester's identity
D_{id}	the destination's identity
L_s	the location of N_0 when it sends the query to N_1
R_p	the inner radius of the obfuscation area
R_s	the external radius of the obfuscation area
T_{min}	the social value bound for friends
C_{max}	the extra path limit in each obfuscation forward

database in LBSP is N_f's information, so they can hardly infer the original requester's identity and location based on that information. The free phase starts when the query q is forwarded by N_f and ends when it reaches the LBSP.

Because N_f is the only identity the LBSP knows, the LBSP has no choice other than replying to the last friend N_f when it receives the obfuscated query. The reply can be delivered with an arbitrary DTN routing protocol as the free phase query does. The friend N_f should remember who is the real destination (N_0) of this reply, then he transmits it to N_0. In this way, N_0 is able to send a query q to an LBSP while not exposing his own information.

The obfuscation phase of a query q starts when the query leaves the original requester N_0. When a user is holding an obfuscation phase query, he starts sensing connected friends continuously, which enables it to communicate with one-hop or multi-hop neighbor friends. Even though users in the mobile network use OLSR protocol [29] to detect others automatically, they do not communicate with their friend unless they have a requirement to send an obfuscation query. Also, they do not ask their friends for public keys. Therefore, carrying an obfuscation phase query requires a user to execute MHLPP algorithm.

When N_0 finds the first available friend N_1, he asks N_1 for a public key K_1, which will enable him to encrypt his query using K_1. That prevent others, e.g. strangers, from learning information in the message msg that that contains both q and r_q. r_q is N_0's requirement for q, which is always sent with the query q and remains constant until the end of the obfuscation phase (we discuss this in part C). Friends who get the query can infer q's obfuscation area based on parameters R_p, R_s and L_s in r_q, which is a ring with inner radius R_p, external radius R_s and center L_s. Before N_0 sends the query q to his friend, he initializes parameter L_s to his current location and encrypts msg using K_1 to get $Emsg_1$ which is what N_0 sends to N_1.

The destination of $Emsg_1$ is N_1, which is a plaintext in $Emsg_1$, so that other intermediate users (strangers) can help N_0 forward $Emsg_1$ to N_1. In this step, strangers transmit $Emsg_1$ using the OLSR protocol if N_1 is a multi-hop neighbor of N_0. Strangers learn nothing other than the identity of N_1, because both q and r_q are encrypted. They cannot help attackers locate N_0 because they do not know S_{id} (the identity of N_0) and D_{id} (the LBSP) included in r_q.

When N_1 receives $Emsg_1$, he decrypts it with its secret key S_1 to get q and r_q. If q is already in the obfuscation area defined in r_q (i.e., it is already in the ring), the query q finishes its the obfuscation phase. Then N_1 replaces all information of N_0 with his own. For example, the S_{id} is replaced with N_1. If a location is necessary for the LBS, N_1 uses his own current location and records this change in his memory before initiating the free phase. A free phase query can then be forwarded to the destination (i.e. LBSP).

If q is still in the obfuscation phase (not in the ring), N_1 performs similar actions just as N_0 expect modifying r_q. Another difference is that instead of finding friend randomly,

N_1 would seek for a friend who is nearer in the obfuscation area, and so will the following friend.

The detailed algorithm is explained in Algorithm 3-1 where N_x is a neighbor of N_i who is carrying the query q. Function "*DealWithQuery*" is responsible for dealing with a query. For N_0, he generates the query q and its requirement r_q. If N_i receives $Emsg_i$, he decrypts it with his own secret key S_i. If q finished its obfuscation phase at N_i, q will be required to be forwarded in free phase immediately. Otherwise, q needs to be processed in the obfuscation process. Both q and r_q are stored in N_i until they are sent to the next friend. N_i starts detecting friends continuously if and only if N_i carries one or more obfuscation phase queries.

When N_i detects a new neighbor N_x (one-hop or multi-hop neighbor), he follows steps in "*WhenEncounterUser*". For an expired query, N_i simply drops it. If the query q is already inside its obfuscation area, N_i switches it to the free phase. If q stays in the obfuscation phase and N_x is an available friend, N_i encrypts both q and r_q using N_x's public key K_x to get an encrypted message $Emsg_x$. Then N_i forwards $Emsg_x$ to N_x and stops sensing friends after $Emsg_x$ departs from it.

When we mention that N_i switches a query q to the free phase, N_i actually replaces N_0's information with its own one in q to get q^* and records this replacement in its storage, then N_i uses the Spray and Wait protocol [31] to forward q^* in plaintext. That allows N_i to hide N_0's identity and forward a reply from LBSP to N_0.

3.3 Requirement parameters

In the obfuscation phase, r_q is always in msg so that friends who get msg can make decisions (e.g. selections of friends) based on it.

All parameters (i.e., S_{id}, D_{id}, R_p, R_s, L_s ,T_{min} and C_{max}) in r_q are given by N_0 before

Algorithm 1 The Obfuscation Phase in MHLPP

```
procedure DealWithQuery
    if the current user is $N_0$ then
        $generate\,(q, r_q)$ by himself
    else
        if the current user is $N_i$, $i > 0$ then
            $(q, r_q) \leftarrow Decrypt_{S_i}\,(Emsg_i)$
    if $q$ can switch to the free phase based on $r_q$ then
        $SwitchFree\,(q)$
    else
        $msg \leftarrow (q, r_q)$
        Store $msg$
        Start sensing friends
procedure WhenEncounterUser($N_x$)
    $q \leftarrow$ get query from $msg$, $r_q \leftarrow$ get requirement from $msg$
    if $q$ timeout then
        remove $msg$ return
    if $q$ is eligible to switch into the free phase then
        $SwitchFree\,(q)$ return
    if $SV_{i,x}$ is bigger than $T_{min}$ in $r_q$ then
        if it is the first hop of $q$ then
            assign the current location to $L_s$ of $r_q$
        $Emsg_x \leftarrow Encrypt_{S_x}\,(q, r_q)$
        forward $Emsg_x$ to $N_x$
        remove $msg$ from memory
        stop sensing friends
procedure SwitchFree($q$)
    $q^* \leftarrow$ replace $q$'s requester-information ($N_0$) by $N_i$
    record $(q, q^*)$
    forward $q^*$ with DTN protocols
```

$Emsg_1$ leaves N_0. Parameters S_{id} and D_{id} record the identities of N_0 and the destination (LBSP) N_d, based on which last friend N_f is able to send the query freely to D_{id} (N_d) and forward the reply to S_{id} (N_0).

The obfuscation area is a ring with an inner radius R_p and an external radius R_s. As shown in Figure 3.1-a, the obfuscation area is actually the grey area "a". Obfuscation area must guarantee both the original requester's location-privacy and location awareness. In other words, the value of R_p should be big enough, so that there are sufficient users in the inner circle (with a radius R_p). At the same time, R_s should be small enough so that the LBSP can provide a service, acceptable to N_0.

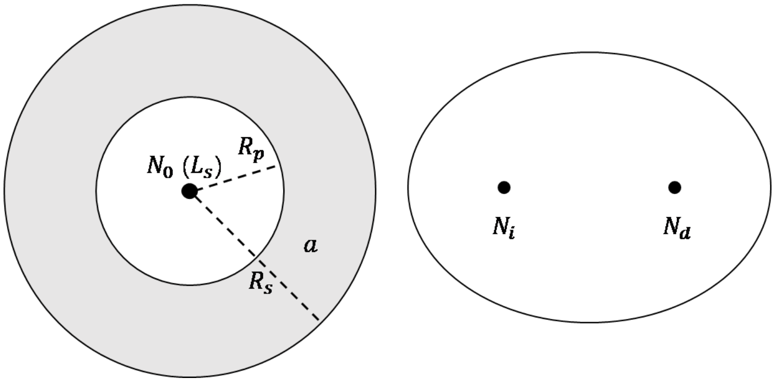


Figure 3.1: The selection of the next friend

A user N_x can be chosen by N_i as a friend for q if and only if $SV_{i,x}$ is bigger than the threshold T_{min}. The original requester N_0 can set various values for his queries based on their importance. If T_{min} is large, there would be fewer friends for any users in the network which reduces the query success rate to a certain extent, as a result. We assume that the original requester can balance the level of privacy and the success ratio.

Most DTN routing protocols aim to deliver queries through the shortest path, while MHLPP pays more attention to security in its obfuscation phase. Consequently, the obfuscation process in MHLPP results in a longer path from the original requester N_0 to the destination N_d. To limit the length of the path, we introduce parameter C_{max} which is the maximum extra path (e.g., the difference of the length and the distance between the N_i

and LBSP) we can tolerate. For any friend N_i who gets a C_{max} from r_q, if he selects N_x as the next friend, the extra path should not be longer than C_{max}. Let's denote the optimal path from user m to n by $Dis\,(m, n)$, and the extra path from N_i to N_d through N_x by $C_{i,x,d}$. Then $C_{i,x,d}$ can be defined as follow.

$$C_{i,x,d} = Dis(i, x) + Dis(x, d) - Dis(i, d) \tag{3.1}$$

$C_{i,x,d}$ must be a value smaller than C_{max}. If $Dis\,(m, n)$ is the straight-line distance between point m and n, then next friend N_x should be in an ellipse E_C with focus points N_i and N_d. Let's denote the coordinate of N_i by $\left(-\frac{d}{2}, 0\right)$ and the coordinate of N_d by $\left(\frac{d}{2}, 0\right)$. Then, the equation of the ellipse E_C is

$$\frac{x^2}{(d + C_{\max})^2} + \frac{y^2}{2d \cdot C_{\max} + C_{\max}^2} = \frac{1}{4} \tag{3.2}$$

As shown in Figure 3.1-a, L_s is the center of the ring while R_p and R_s are the inner and external radii, respectively. The query q switches to the free phase when it enters the ring area.

As shown in Figure 3.1-b, N_i who is carrying obfuscation queries should choose his next friend N_x in the ellipse, which avoids the query q going through an unacceptably long path.

In conclusion, a query q starts at the center and moves inside the ring, until it reaches the obfuscation area. The point N_i in Figure 3.1-b should be inside the ring, and the point N_d might be anywhere. As a result, a user N_i who is carrying an obfuscation query detects a friend continuously who has a larger distance from L_s and inside an ellipse. If there is a friend like that, N_i sends the query to that friend N_x.

3.4 Privacy Analysis

We assume that attackers can achieve all information in LBSPs know. Obviously, they can know the identity of the last friend N_f who replaces N_0's information with his own. It is possible for the attackers to locate N_f with little cost. For example, if N_0stops moving after sending the query q, it is reasonable for N_f to believe that N_0 is in a ring centered at the location of itself with radii R_p and R_s. In other words, the distance between N_f and N_0 should be in a range between R_p and R_s. If attackers find all users who satisfy this condition, the original requester might be among these users with high probability. Then, a success ratio $P_{r_p r_s}$ to locate N_0 can be measured by a conditional probability

$$P_{r_p r_s} = p_{r_p r_s} \cdot \frac{1}{m_{r_p r_s}} \tag{3.3}$$

where $P_{r_p r_s}$ is the probability that N_0 is in the ring (i.e., the distance between N_0 and N_f is larger than R_p and smaller than R_s). Here, $m_{r_p r_s}$ is the number of users who are in the ring. Attackers locate N_0 successfully if and only if N_0 is in the ring, at the same time, attackers pick the correct one from all $m_{r_p r_s}$ users at that area.

In the worst case, attackers know exact values R_s and R_p. Then, the Eqn. (3.3) becomes

$$P_{R_p R_s} = p_{_{R_p R_s}} \cdot \frac{1}{m_{_{R_p R_s}}} \tag{3.4}$$

where $P_{r_p r_s}$ is the probability that N_0 is on the ring (i.e., the distance between N_0 and N_f is larger than R_p and smaller than R_s). $m_{r_p r_s}$ is the number of users who are in the ring. Since parameters (e.g., R_p and R_s) in r_q are kept secret among trusted friends in our system model, attackers can hardly get the actual values of those parameters.

3.5 Complexity discussion

In order to guarantee secure communications among friends, encryption is introduced in our protocol. In the obfuscation phase, the query is transmitted along friends, i.e. N_0, N_1, N_2, ..., N_f. When the query is sent from N_i to N_{i+1}, a pair of encryption and decryption is needed, so the number of such pairs T_{en} should be equal to f.

T_{en} grows with both T_{min} (threshold used to decide friend relationship) and inner radius R_p. Essentially, it is the number of friends participating in transmitting a query q in its obfuscation phase that influences T_{en}. Given a smaller T_{min}, a user carrying the obfuscation phase query has more chances to encounter more friends in a certain area. A larger R_p also leads to a bigger area inside the ring, so that there are more friends in this area. We evaluate the number of encryptions and decryptions in our simulation. Figure 3.2 shows the average number of encryptions (T_{en}) with different friend thresholds T_{min} and inner radius R_p. We observe that T_{en} increases steadily as we increase T_{min} and R_p.

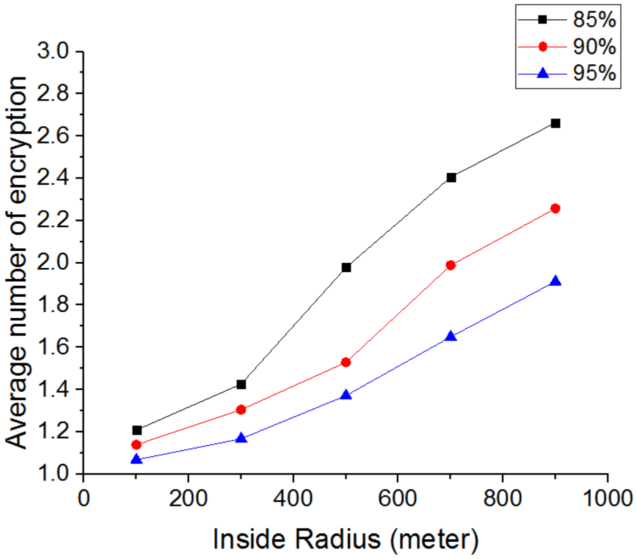


Figure 3.2: The number of encryption with various inner radius

It is evident that encryption and decryption process in MHLPP results in an extra cost in both energy and computational resources. However, the number of encryptions and

Table 3.2: MHLPP Experiment Parameters

Parameter	Value
Simulation Time	10 minutes
Map Size (W x H)	4500 m x 3400 m
Total number of users	126
Pedestrians/ Cars	84/42
Communication Area Radius	10m – 90 m
Pedestrian Speed	1.8-5.4 Km/h
Car Speed	10-50 Km/h

decryptions is quite low (below 3) which reasonable based on the experiment.

3.6 Performance Analysis

We use the map of Helsinki in our simulator to evaluate MHLPP. It is also compared against the known protocol, Hybrid and Social-aware Location-Privacy in Opportunistic mobile social networks (HSLPO). The simulation parameters are shown in Table 3.2. All pedestrians and cars are users in MHLPP. These users are moving on the map along streets continuously. There is an LBSP fixed at a random location on the map. For each user, we give him random social values between 0% and 100%, each corresponding to all other users. Each value has the same probability, so we can compute the expected number of friends of a user. For example, if we are given a privacy threshold (T_{min}) of 85%, then there might be 15% (100%-85%) users who are friends of a certain user.

As shown in Table 3.2, there are 126 users in the map. For each of them, say user i, we give him 126 random SV values, which denotes the relationship strength between him and other users, so that there are 126^2 SVs in our simulation. The SVs are between 0 and 100. As a result, if the T_{min} is equal to 85, the average number of friends of each user should be 18.9 (=$126 \times (100 - 85)/100$).

Users are placed at random locations at the beginning of each experiment. We choose another random point for each user, so that he can move back and forth along streets between that point and the point his starts at. The user speed depends on its type (pedestrians or cars) and set randomly. All queries have a 10-minute timeout. The queries which are expired before they reach the LBSP (the destination) are considered to be failed in our success ratio statistics.

Figs. 3-5 compare performances between HSLPO and MHLPP for different values of k, communication radius and privacy threshold (T_{min} in MHLPP). The k is the privacy-level requirement in HSLPO. Both HSLPO and MHLPP have different criteria in which a query can switch to the free phase. To make them comparable, we create a new parameter, called *obfuscation distance*. If a query leaves N_0 at location L_a and switches to the free phase at N_f whose location is L_b, then the obfuscation distance is the straight-line distance between L_a and L_b. We test the obfuscation distances of HSLPO with different parameters, and then we set the inner radius of MHLPP to those values. The query success ratio is the ratio of delivered queries to the total number of queries. The number of hops (h) is the number of intermediate users between N_0 and the destination (LBSP). We count the number of users surrounding the last friend in a specific range, which is k times the communication radius. We calculate the entropy using the reciprocal of the number of surrounding users.

3.6.1 Query success ratio

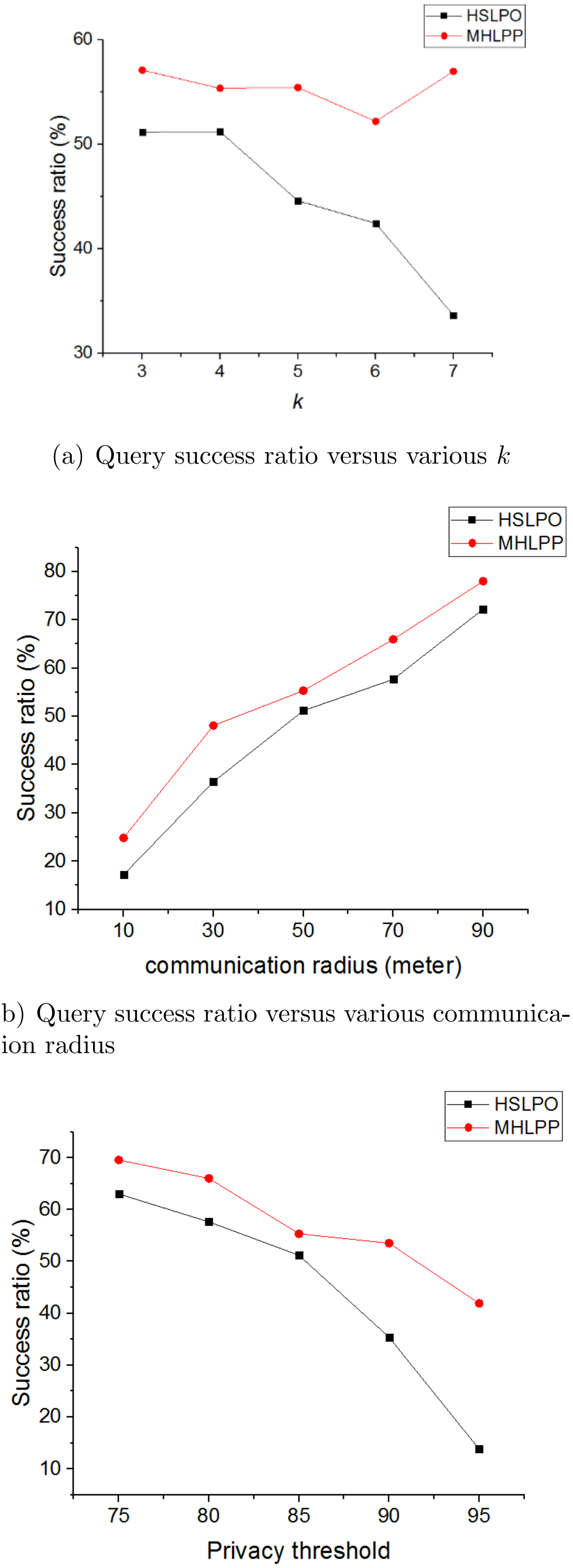
The query success ratio is the percentage of delivered queries among a number of attempts. Based on the timeout value in Table II, a query is delivered successfully, if it arrives at the LBSP (the destination) before the timeout; otherwise it fails. We use the query success ratio to evaluate the delivery performance of MHLPP.

As shown in Figure 3.3(a), the success ratio in MHLPP is always higher than that in HSLPO. As the value of k increases, HSLPO success ratio drops sharply while MHLPP remains stable. This is because the larger k is, the harder it is for HSLPO to find enough friends in a limited time. The lack of friends has less impact on MHLPP. We observe that the success ratio of MHLPP rises when $k = 7$. That is because it depends on the inner radius which is equal to the obfuscation distance of HSLPO. The obfuscation distance decreases when $k = 7$, because most of the queries which complete their obfuscation phase have a short obfuscation distance. In Figure 3.3-b, both HSLPO and MSLPP values increase and have the same trend when given a larger communication radius. The reason is that the communication radius effects the free phase more than the obfuscation phase for both. As shown in Figure 3.3-c, higher privacy threshold leads to lower success ratio in two algorithms. Its impact on HSLPO is more intense than that on MHLPP, which is the most important characteristic of MHLPP. MHLPP has a better performance than HSLPO especially when there are fewer friends in the network. MHLPP can transmit messages with the help from strangers in its obfuscation phase while HSLPO cannot.

3.6.2 Number of Hops

We count the number of hops it takes for queries to be delivered successfully and calculate the average. Every user who takes part in the delivery process is considered in the hop count. We introduce this criterion to measure the routing path length of the algorithm. MHLPP is more sensitive to the probability that a user encounters a friend than HSLPO is. The reason is that MHLPP aims to reach a certain distance rather than taking certain number of hops. In other words, MHLPP continues sending queries to other friends until queries enter their obfuscation area. In this process, MHLPP takes every chance to forward queries. If it is hard for MHLPP to find friends, it can also take the queries to obfuscation areas with fewer friends. Therefore, the probability that users encounter friends has less impact on the performance of MHLPP. The result of this experiment is shown in Figure 3.4.

In Figure 3.4-a, the number of hops in HSLPO is affected by parameter k obviously. Especially, the first k hops forwarders must be friends, which makes it hard for HSLPO to



(a) Query success ratio versus various *k*

(b) Query success ratio versus various communication radius

(c) Query success ratio versus various privacy thresholds

Figure 3.3: The Query Success Ratio Comparison Between HSLPO and MHLPP

have queries to be forwarded successfully. Both protocols have similar number of hops in their free phases, but HSLPO has exactly k hops in its obfuscation phase, while MHLPP can have fewer than k hops. In figure 4b, the number of hops in MHLPP grows with the communication radius obviously, while it does not change a lot in HSLPO, because only successful delivery queries are counted in the statistics. Given a large communication radius, MHLPP has a much higher success ratio for delivered queries as it can connect more friends. In Figure 3.4-c, the value of MHLPP drops for higher privacy thresholds. The reason is the same as Figure 3.4-b. For HSLPO, no matter how hard it is to find a friend, it attempts to find exactly k friends. However, if it is too hard to find a friend, MHLPP's friend can carry the query while moving and complete the obfuscation process. For higher privacy thresholds (i.e. resulting in fewer friends), MHLPP chooses to carry queries other than finding friends. That results in a drop in the number of hops.

3.6.3 Security

Since the principles with that two protocols protect original requesters are different, we evaluate the probability that attackers locate the original requester if the distance between him and the last friend is smaller than a some value r, which is equal to k times the communication radius. Since the last friend reveals himself to the LBSP, attackers might locate him accurately. We assume that attackers know the privacy parameter k. In the worst case, the distance between the original requester and the last friend is smaller than r when attackers start to locate the original requester. That gives attackers a chance to locate the original requester. We count all users who are inside the radius r of the last friend, and the original requester is one of them. For example, if there are m users in the area, the probability should be $\frac{1}{m}$. Figure 5 compares the entropy E of both HSLPO and

MHLPP. We use the following formula for computing entropy:

$$E = -\frac{1}{m} log_2 \left(\frac{1}{m} \right)$$

From Figure 3.5-a, we observe that MHLPP has very small (about 0.04) increase in entropy compared to HSLPO. When the original requester is in the circle centered at the last friend, HSLPO is a little more secure than MHLPP but not by very much. That is because HSLPO always switches to the free phase when the last friend encounters the previous friend, so the previous friend must in the circle. MHLPP does not have this condition. Both graphs in Figure 3.5-a and Figure 3.5-b have the same trend. The curve of MHLPP is also a little higher than that of HSLPO, while two curves almost meet when the communication radius is small or large. Given a small radius, the entropy of both protocols are small. When the radius is large, we can ignore the effect of the previous friend mentioned about for Figure 3.5-a as there are so many users in the circle. From Figure 3.5-c, since the circle neither expands or shrinks, we observe that the two curves exhibit similar behavior. The values of HSLPO are always lower than the correlated values of MHLPP. However, as we observed in the experiment, the last friend is always hundreds of meters away from the requester.

Chapter 4

Appointment Card Protocol

4.1 System Model

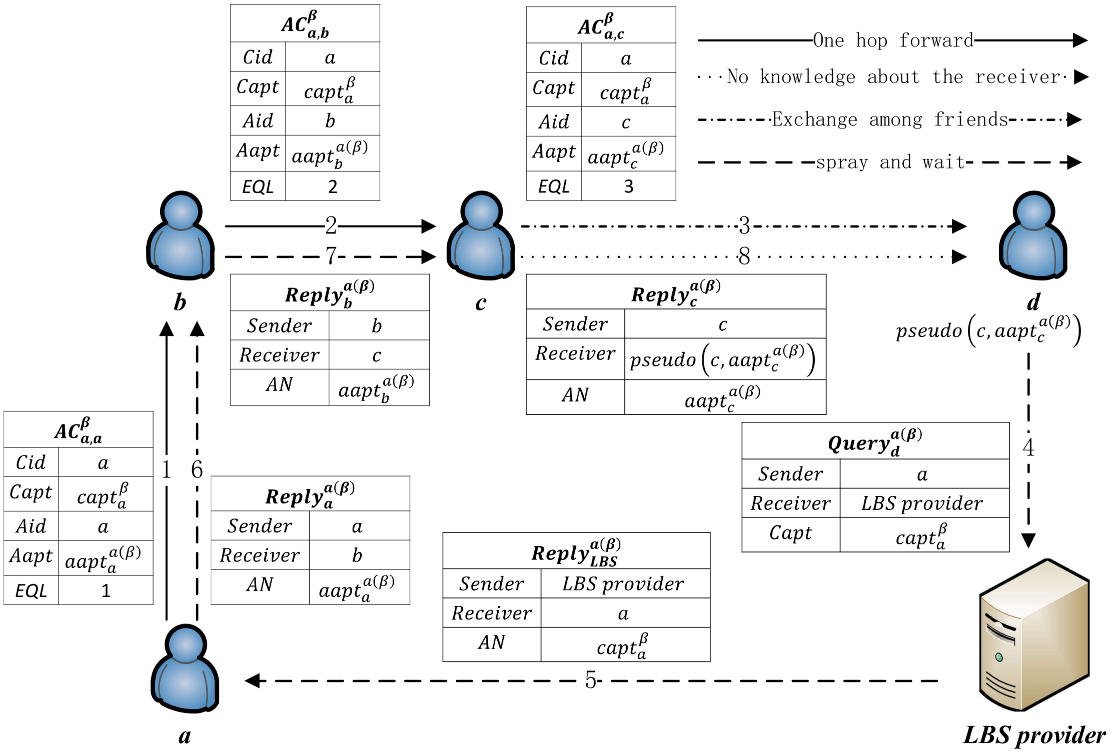
The network architecture consists of two main classes of entities: Users and Location-Based Service Providers (LBSPs). Users are mobile and communicate with others within a certain range, i.e., the communication range of their portable devices. For a given user, other users in the social network are either strangers or his friends whom he can detect when they are in his communication range. Let $RS_{i,j}$ denote the relationship strength between user i and j. If $RS_{i,j}$ is larger than a specific threshold, FT_{min}, user j is considered a friend of i. LBSPs, which provide location-based services to the users are fixed and not part of the social network. We assume that the only information which is necessary for the LBSP is a location from the original requester, but the original requester should still give an identity to the LBSP so that the LBSP can reply to that identity.

We consider external attacker capable of eavesdropping on limited traffic in the network. We assume that the attacker can access the database of LBSPs, so that he can learn everything recorded in LBSP's memory, including user identities and locations. The attacker launches an inference attack on each user who uses the LBS in an attempt to learn user's private information based on location and context in the queries. Therefore, the key to protecting location-privacy is degrading the relationship between the user identity and the location provided by him so that the attacker can hardly infer the identity of the original requester by the known information.

We propose a protocol, called Appointment Card Protocol (ACP), to protect the identity and location-privacy of the original requester by providing other users' identity (agents), which can be any user in the network so that ACP can have a large anonymity set. The friends of the original requester separate the agents and the original requester so that the agents have no knowledge about the original requester.

4.2 Appointment Card Protocol Overview

Our proposed ACP protects original requesters when they are served by LBSPs. A user (Agt_1) generates his own Appointment Cards (ACs) containing his own identity called Cid and a unique number called $Capt$(a number generated by the creator of AC). The ACs are exchanged when two users encounter each other. When the original requester sends a query, he chooses an AC and sends the query using the identity Agt_1 of the first agent which is in the AC. The LBSP replies to Agt_1 when it receives the query. Agt_1 is the one who has generated the AC and the first agent of the AC. Agt_1 then forwards the reply to the next agent (Agt_2) who already has received AC from him, and so on until the reply reaches the last agent. The last agent is responsible for forwarding it to the original requester.



$AC^{\beta}_{a,b}$	
Cid	a
$Capt$	$capt_a^{\beta}$
Aid	b
$Aapt$	$aapt_b^{a(\beta)}$
EQL	2

$AC^{\beta}_{a,c}$	
Cid	a
$Capt$	$capt_a^{\beta}$
Aid	c
$Aapt$	$aapt_c^{a(\beta)}$
EQL	3

$Reply_b^{a(\beta)}$	
$Sender$	b
$Receiver$	c
AN	$aapt_b^{a(\beta)}$

$Reply_c^{a(\beta)}$	
$Sender$	c
$Receiver$	$pseudo\left(c, aapt_c^{a(\beta)}\right)$
AN	$aapt_c^{a(\beta)}$

$AC^{\beta}_{a,a}$	
Cid	a
$Capt$	$capt_a^{\beta}$
Aid	a
$Aapt$	$aapt_a^{a(\beta)}$
EQL	1

$Reply_a^{a(\beta)}$	
$Sender$	a
$Receiver$	b
AN	$aapt_a^{a(\beta)}$

$Query_d^{a(\beta)}$	
$Sender$	a
$Receiver$	$LBS\ provider$
$Capt$	$capt_a^{\beta}$

$Reply_{LBS}^{a(\beta)}$	
$Sender$	$LBS\ provider$
$Receiver$	a
AN	$capt_a^{\beta}$

Figure 4.1: Example of ACP Message Exchange

The Figure 4.1 is an example of the execution of the ACP protocol. Explanations of the symbols in the figure are shown in Table 4.1. These symbols and the figure are used throughout the chapter to help us describe the protocol. For simplicity, we will omit some superscripts and subscripts from these symbols in the following sections when there is no ambiguity. The whole process can be considered as the following parts: 1) exchanging cards among all users who are called agents (i.e., 1 and 2), 2) exchanging cards among friends (i.e., 3), 3) sending query using information of appointment cards (i.e., 4), 4) forwarding the reply among agents (i.e., 5, 6 and 7), and 5) relaying to the original requester (i.e., 8).

4.3 Appointment Card

Since the original requester cannot use his own identity to communicate with the LBSP, he must use others' identity (Agt_1) to send queries, so that the LBSP can reply to the original requester through Agt_1. Appointment cards make it possible for the agents to forward the reply to the original requester one by one. In other words, the appointment card indicates a path through which the original requestor can get its reply.

In Figure 4.1, the users a, b and c are the agents of the appointment card AC_a^β (i.e., $Agt_1^{a(\beta)}$, $Agt_2^{a(\beta)}$and $Agt_3^{a(\beta)}$). These agents are strangers, so the attackers can hardly infer c from the identity of a. At the same time, c is in the original requester d's social tie (i.e., c is d's friend or his friends' friend, so on), and he is the only one who knows how to reach d. Therefore, it is hard for attackers to infer the identity of d from the identity of a.

Notice that c receives a's appointment card (i.e. AC_a^β) from a stranger b who knows the information of AC_a^β and the identity of the next agent c, so that it is unsafe for c to use that appointment card. In other words, the appointment card cannot be used until c exchanges it with another user (e.g., the user d) who trusts c. The appointment card is called a *ready appointment card* (*ready AC*) after it leaves the last agent (i.e., the user c), or it is called the *distributing appointment card* (simply *distributing AC*). It is obvious that *distributing AC*s are transmitted among agents who can be strangers, while a user can only get *ready AC*s from one of his friends.

To make users carry a similar number of ready ACs, ready ACs are also exchanged between friends. As a result, the last agent is not sure whether the user who gets the ready AC from him is the original requester. We introduce a pseudonym mechanism, which enables the last agent to forward the reply to an unknown original requester.

All users in the network are responsible for generating their respective ACs, and they are called the creators of their own ACs. The creator records his own identity (Cid) and a unique number ($Capt$) on his AC. When users exchange ACs, they modify Aid (the agent

ID) and Aapt (the agent's appointment number) in the AC to enable the next agent to identify who is the predecessor. Entries of AC are shown in Table 4.2.

4.4 AC Life Cycle

The life cycle of an AC starts when it is generated by its creator. The first k (see Table 4.3) agents add their identities into its Exchange Queue (EQ) (see Table 4.2) before exchanging it, which increases the length (EQL) of the EQ. When the AC's EQL reaches k, it is eligible to be used in a query and is called a *ready* AC. When an AC is used in a query, it is marked as an *used AC* by the original requester who uses the AC. No matter what state (*distributing*, *ready* or *used*) an AC is in, it can expire, as shown in the Figure 4.2. An AC starts at the *distributing* state. If the length of its EQ reaches k, it is switched to the *ready* state. It can timeout in all states. If an AC is used once, it is switched to the *used* state. A *used* AC can also be used in other queries but cannot be given to anyone.

4.5 System Parameters

All the system parameters are shown in Table 4.3.

4.5.1 Obfuscation Distance

The obfuscation distance k is the number of exchange before an AC is switched to the *ready* state. In other words, an AC must be exchanged k times before it becomes a *ready* AC.

As shown in Figure 4.3, an AC is exchanged along Agt_1, Agt_2, ..., Agt_k. Since those agents are strangers, the only relationship between two adjacent agents is that they encounter each other somewhere. The relationship between Agt_1 and Agt_k becomes weaker when we increase k. In other words, attackers can hardly infer the identity of Agt_k when he only knows the identity of Agt_1, and his difficulty increases with the increase of parameter k. As a result, it is hard for attackers to infer the original requester, even though Agt_k is in the social tie of the original requester. Since the reply message must go through a series of agents, a long obfuscation distance also lengthens the path of the reply message. Therefore, a large k makes the original requester safer, while making it harder and longer for the reply to be delivered.

4.5.2 Friends Obfuscation Distance

Since Agt_{k-1} is a stranger for Agt_k, it is possible that Agt_{k-1} is exactly the attacker. We assume that the attacker knows that Agt_k is in the social tie of the original requester. The attacker can assume that Agt_k is a close friend of the original requester, so the identity of Agt_k gives the attacker a good tip to infer the original requester. A solution to prevent the agent Agt_{k-1} from learning the original requester easily is that the last m agents are friends, as shown in Figure 4.4. Here, m is called *friends obfuscation distance*, and the last m agents are trusted agents.

It is true that Agt_i is a friend of Agt_{i+1}, $i > k-m$, but two nonadjacent trusted agents (e.g., Agt_i and Agt_{i+2}) might have a weak relationship. Since there are at least m trusted agents between the stranger Agt_{k-m} and the original requester, Agt_{k-m} can hardly infer the identity of the original requester based on information he learns. When $m = k$, all ACs must be exchanged between friends only. When $m = 1$, the last agent is the only one who is in social tie of the original requester, which is the case in our example.

However, friends encounter each other rarely, so having a large m increases the difficulty of distributing ACs. In this work, we assign m to 1.

4.5.3 Distributing Segment

We again use the example in Figure 4.1. U_a generates 3 ACs (i.e. AC_a^1, AC_a^2 and AC_a^3). It exchanges these ACs with U_b, then U_b exchanges them with U_c and so on. At last, all of them reach the original requester U_d. In this case, U_d has several ACs whose paths (the list of agents) are the same, so U_d has no choice but to choose one. In other words, since using these ACs results in the same reply path, U_d is can simply choose an AC based on specific requirement it might have (e.g. the number of possible encounters of an agent).

If agents exchange ACs with different users, the above problem will not happen. We use the system parameter distributing segment, Seg, to avoid giving all ACs to the same user. Each user maintains Seg number of *Distributing AC Lists* (DLs). It puts received *distributing* ACs in one of those DLs randomly. If Agt_i exchanges ACs with another user, Agt_i selects one of his DLs, and only ACs in that DL will be exchanged with that user.

For example, we assign $Seg = 2$ so that each user has two DLs (i.e., $DL1$ and $DL2$). When U_b receives $AC_{a,a}^1$, $AC_{a,a}^2$ and $AC_{a,a}^3$ from U_a, U_b may put $AC_{a,a}^1$ and $AC_{a,a}^2$ in its $DL1$, while $AC_{a,a}^3$ in its $DL2$. If U_b encounters U_c, U_b can give either $AC_{a,b}^1$ and $AC_{a,b}^2$ or only $AC_{a,b}^3$ to U_c. In this way, we separate ACs generated by the same creator.

4.5.4 Generating Period

Since ACs could expire, users must generate new ACs continuously. We use GP to denote the speed of generating new ACs per user, that is, each user generates a new AC every GP seconds.

4.5.5 AC Timeout

Let AT denote the timeout of ACs. An AC expires AT seconds after it is generated. When an AC expires, all agents delete the AC information from their memory.

4.5.6 Avoiding Time

If a user gets an AC at time t_i, he cannot get that AC again before $t_i + \tau$, where τ is parameter avoiding time. The reason for this is described in Section 4.6.2.

4.6 Protocol Details

4.6.1 Generating Appointment Cards

Maintaining a certain number of ACs in the network is a prerequisite for users for sending queries. Considering that ACs can expire, users must generate ACs continuously. The user who generates an AC is called the creator of the AC, at the same time, he is also the first agent of the AC. ACs are generated based on two principles: fairness and continuity.

ACs impose burden on agents. In fact, agents are unlikely to benefit from relaying messages to the original requester because they need to allocate memory to save ACs' information, and they consume energy to forward replies. To prevent users who generate more ACs than others being overloaded, some fairness mechanism must be in place to ensure that all users generate a fairly equal number of ACs.

We assume that the AC timeout (i.e. AT) is 30 minutes, and every user generates 100 ACs at the beginning and generates no AC until the 30^{th} minute. Since all agents remove the information of expired ACs from their memory, a reply can hardly be delivered when the AC it uses is timed out (expired). As a result, it is hard for a user to select an appropriate AC for his query at the 29^{th} minute when all ACs only have 1 ($= 30 - 29$) more minute, because his reply may be lost even though his query is delivered successfully. Therefore, the generating strategy should be a steady and sustainable.

In our protocol, each user generates a new AC every GP second. Since an AC has an AT-seconds timeout, there are about $\delta = \frac{AT}{GP}$ appointment cards hold by each user in the network. In other words, each user maintains about δ ACs which is generated by himself, and we meet the fairness criteria. Since users generate ACs continuously, ACs have various timeouts so that it is likely for a user to pick an AC who does not expire in a long time (at least longer than his queries and replies).

4.6.2 Exchange Distributing Appointment Cards

Users exchange their *distributing* ACs as frequently as possible, so that a *distributing* AC can be switched to a *ready* one quickly. Still, there are some other conditions which should be satisfied when exchanging a *distributing* AC.

We assume that two users, e.g. U_A and U_B, are walking together for a some time. Also assume that U_A generates an AC (e.g., AC_A) at t_0 and exchanges it to U_B, then AC_A is given back to A and so on. The distributing phase of the AC only costs a few seconds, because U_A and U_B are the only agents involved. This defeats the purpose of exchanging ACs, because it is easy for an attacker to infer the last agent who may be U_A or U_B from the first agent U_A.

We use a parameter *avoiding time,* τ, to optimize the agent selection strategy. If a user gets an AC at time t_i, he cannot get that AC again before $t_i + \tau$. For example, U_B receives a certain AC AC_A from U_A at t_0, then U_B sends it to a user C (U_C) who can also send it to others. If a user carrying AC_A encounters U_B before $t_0 + \tau$, he cannot send AC_A to U_B. Therefore, if the parameter τ is larger than or equal to the parameter AT, there is no repeated agent in an AC's EQ. In other words, an AC never reaches an agent twice when τ is equal to AT. In this thesis, we assign τ to AT.

Besides, we should also avoid ACs having the same sequence of agents, as we mention in the distributing segment subsection. Now we propose the strategy of exchanging distributing ACs.

Let us take a pair of users Alice and Bob as an example. If Alice encounters another user Bob, Bob tells Alice whether he trusts her (she is viewed as a friend by Bob). Alice picks one of her distributing AC lists (e.g. $DL1$). Alice traverses all ACs in $DL1$, and distributing ACs which satisfy the following two conditions before exchanging them with Bob.

1. If the length of the AC's EQ (i.e. EQL) is not shorter than $k - m$, then Alice must be a friend of Bob.
2. Bob was not carrying the AC in the recent τ time interval.

When Alice sends a distributing AC to Bob, she adds her identity and current time to the AC's EQ. Besides, Alice must modify the AC's *Aid* to her own identity and its *Aapt* to a new one. The information in Table 4.4 is recorded as *relay table* in her memory. We should notice that the first agent (i.e., the creator) does not have a $Aapt_{old}$, because he is precisely the one who has generated the AC, in which case the $Aapt_{old}$ indicates his own identity. When Bob gets those ACs, he puts each one of the received *distributing* ACs to his *DL*s respectively and randomly. If an AC whose EQL is already equal to k, the AC must be switched to a *ready* one when Bob gets it. Bob puts ready ACs in his ready AC list instead of distributing AC lists.

Table 4.4: Relay Table Entries

Name of entries	descriptions
$Aapt_{old}$	The *Aapt* generated by the previous agent
Aid_{old}	The identity of the previous agent
$Aapt_{new}$	The new *Aapt* (generated by himself)
ID_{nxt}	The identity of the next agent
EQL	The length of the AC's EQ (should larger than 0)
AT	The time when the AC timeout.

4.6.3 Exchange Ready Appointment Cards

Users ask for ready ACs only from their friends, which ensures that the information of the ready ACs held by a user is not exposed to strangers. The strategy of exchanging ready ACs should also meet some fairness criteria. That is, the number of each user's ready ACs should be more or less equal.

Let us consider two friends Alice and Bob as an example. Alice has 20 ACs, while Bob has 10 ACs. When they encounter each other, it is reasonable that they both give half of their ready ACs to the other. As a result, they both will have half of the total (**??**) ACs. However, that strategy does not work all the time as explained below.

The problem becomes more complex in the following condition. Alice is a friend of Bob, while Bob is not a friend of Alice. In other words, Bob trusts Alice, but Alice does not trust Bob. We assume that Alice is a trusted but suspicious girl so that many users trust her while she trusts few users while Bob is opposite. When users encounter Alice, they ask for ready ACs from her, but Alice rarely asks for ready ACs. Therefore, Alice carries few ready ACs, while Bob carries many ready ACs. When Alice and Bob encounter each other, it is Bob who is asking for ready ACs instead of Alice. To make the strategy fair and efficient, users must compare the number of their own ready ACs and the number of ready ACs carried by the other user when two users are exchanging ready ACs.

When Bob encounters Alice, Bob tells Alice the number of his ready ACs (NR_{Bob}) and whether he wants Alice's ready ACs (if he trusts Alice). If Alice learns that Bob needs her ready ACs, she compares the number of her ready ACs (NR_{Alice}) and NR_{Bob}. If $NR_{Bob} \geq NR_{Alice}$, Alice gives no ready AC to Bob; otherwise, she gives $\frac{NR_{Alice}-NR_{Bob}}{2}$ to Bob. In this way, ready ACs do not concentrate in a group of users who trust many users.

The process of exchanging ready ACs is much simpler than that for distributing ACs. Users do not modify any information in the ready ACs including $Aapt$ and Aid, so that ready ACs do not change after they leave the last (k^{th}) agent.

The algorithm of exchanging ACs when a user U_i encounters another user U_j is shown in Algorithm 4-1. The two users tell the other whether they are friends at the very beginning of their encounter. After they both know their relationship, they start to exchange their distributing ACs. When they both finish receiving the distributing ACs from the other, they continue to exchange their ready ACs at the same time. The process ends when they finish exchanging their ready ACs (or they do not need to exchange ready ACs).

4.6.4 Sending Queries

For an original requester who want to send queries to the LBSP, his query must be delivered to the LBSP while the LBSP cannot learn the identity of him. Besides, the reply message from the LBSP must be delivered back to the original requester. The basic idea is that the original requester sends his query using another user's identity to the LBSP, so that LBSP can reply to that user who is responsible to forward the reply to the original requester. That user is the first agent (Agt_1) of the AC which is used by the queries of the original requester.

In order to enable LBSP to reply to Agt_1, the original requester's query includes an sender identity Agt_1, which is equal to the Cid in the AC. Since Agt_1 needs a Capt to identify the AC used by the query (and the reply), the Capt in the AC is also in the query. The network can deliver that query to the destination LBSP easily with any DTN protocols.

We use the example in Figure 4.1. The original requester U_d has an AC (i.e., $AC^{\beta}_{a,c}$) whose creator is U_a. When U_d uses it to send his query $Query^{a(\beta)}_{d}$, the sender identity is a and the $Capt$ of the query is the $Capt$ of $AC^{\beta}_{a,c}$, as shown in Figure 4.5.

The AC is marked as used when the query is ready to be sent, so that the AC cannot be exchanged to other users. To get the reply, the original requester also uses a pseudonym, which is described later. The algorithm that a user U_i sends a query is shown in Algorithm

4-2.

4.6.5 Sending Replies

4.6.5.1 The LBSP part

When the LBSP received the query, it learns that the sender's identity is the first agent instead of the original requester, which protects the location privacy of the original requester. The LBSP reply to the first agent using the information in the query.

Figure 4.6: Constitute Replies

In Figure 4.6, the LBS provider reply to the sender U_a, when it receives $Query_d^{a(\beta)}$. The $Capt$ of the query is also included in the reply, which enables U_a to identify the AC used in the query and the reply.

4.6.6 The First Agent

When the first agent (i.e. Agt_1) gets the reply from the LBSP, he learns the AN in the reply. He searches his reply table to get the information of the AC used in the query and reply. If the AC does not expire, there must be a corresponding entry in his reply table, as shown in Table 4.5. Agt_1 gets an entry where the Aid_{old} and $Aapt_{old}$ are equal to his identity and the AN of the reply. As a result, he learns the identity of the next agent (i.e. Agt_2) from the ID_{nxt} of the entry, so Agt_1 forwards the reply to Agt_2. The AN of the reply is replaced with the $Aapt_{new}$ in the entry by Agt_1, which enables Agc_2 to identify the AC in Agt_2's reply table.

Table 4.5: Reply Table Entries Of The First Agent

Name of entries	value
Aid_{old}	The user's own identity
$Aapt_{old}$	The $Capt$ of the AC used in the reply (query)
ID_{nxt}	The identity of the second agent
$Aapt_{new}$	The $Aapt$ given to the second agent by the user.
EQL	1
AT	The time when the AC timeout.

For the example in Figure 4.1, the first agent is U_a. When he receives $Reply_{LBS}^{\mathrm{a}(\beta)}$, he learns that it is a reply from the LBS and the $Capt$ of the AC is $capt_a^{\beta}$. He searches his reply table for an entry whose Aid_{old} is equal to a and $Aapt_{old}$ is equal to $capt_a^{\beta}$. As shown in Figure 4.7, the ID_{nxt} in the entry is equal to b so he modifies the receiver of the reply message to U_b. U_a also modifies the AN in the reply message to $aapt_a^{a(\beta)}$ which enables U_b identifies the AC in his reply table, because the $Aapt_{new}$ in the entry is equal to $aapt_a^{a(\beta)}$.

Figure 4.7: The Reply of The First Agent

4.6.7 Intermediate Agents

The process of forwarding replies in the intermediate agents (the second to the $(k-1)^{th}$ one) is similar with that of the first agent. We take the second agent as an example. When the second agent receives the reply forwarded by the first one, he learns the sender's identity (i.e. the first agent) and the AN from the reply. Since he gets the AC which is used in the reply (query) from the previous agent, there must be an entry, whose Aid_{old} is equal to the previous agent's identity and the $Aapt_{old}$ is the value of $Aapt$ in the AC when he gets it, in his reply table, as shown in Table 4.6.

Table 4.6: Reply Table Entries Of The Second Agent

Name of entries	value
Aid_{old}	The identity of the previous agent (i.e. the first agent)
$Aapt_{old}$	The *Aapt* given by the previous agent.
ID_{nxt}	The identity of the next agent (e.g. the third agent)
$Aapt_{new}$	The *Aapt* given to the next agent by the user.
EQL	2
AT	The time when the AC timeout.

For the example in Figure 4.1, the second agent is U_b. When he receives $Reply_a^{a(\beta)}$, he learns that it is U_a who forwards the reply message $Reply_a^{a(\beta)}$ and the $Aapt$ of the AC is $aapt_a^{a(\beta)}$. He searches his reply table for an entry whose Aid_{old} is equal to a and $Aapt_{old}$ is equal to $aapt_a^{a(\beta)}$. As shown in Figure 4.8, since the ID_{nxt} in the entry is equal to c, he modifies the receiver of the reply message to U_c. U_b also modifies the AN in the reply message to $aapt_b^{a(\beta)}$ which enables U_c identifies the AC in his reply table, because the $Aapt_{new}$ in the entry is equal to $aapt_b^{a(\beta)}$.

Figure 4.8: The Reply of the Second Agent

For each agent Agt_i, where $2 \leq i \leq k-1$, he searches his reply table for a correlative entry, when he receives a reply. The Aid_{old} and $Aapt_{old}$ in the entry should be equal to the sender's identity and the AN in the reply. The identity of the next agent Agt_{i+1} is ID_{nxt}. Agc_i also assign $Aapt_{new}$ to the AN in the reply to help Agt_{i+1} search Agt_{i+1}'s reply table.

4.6.8 The Last Agent

The last agent also searches for a reply table entry based on the reply, while he cannot get the identity of the next agent. The reply table entry of the last agent is shown in Table 4.7.

The last agent uses the same way to look up an entry in his reply table as the previous agents. When he finds that the EQL is equal to k, he notices that he is the last agent. Then he is responsible to forward the reply to the original requester instead of another agent. He uses his identity and $Aapt_{new}$ in his reply table entry to generates a pseudonym $pseudo\,(Agc_k, Aapt_{new})$, where the function $pseudo\,(id, Aapt)$ is a public pseudonym generating function which everyone in the network knows, including the original requester.

When the original requester sends his query, he also gets the same pseudonym $pseudo\,(Agc_k, Aapt_{new})$ using the pseudonym generating function. Note that he can get parameters from the AC. Agc_k and $Aapt_{new}$ are equal to the Aid and $Aapt$ in the AC. He uses that pseudonym as his identity before he gets the reply.

When users deliver the reply from the last agent, they are looking for a user whose identity is that pseudonym. At last, the original requester gets the reply, because he is the only user who uses the pseudonym as his identity.

For the example in Figure 4.1, the last agent is U_c. When he receives $Reply_b^{a(\beta)}$, he learns that it is U_b who forwards the reply message $Reply_b^{a(\beta)}$ and the $Aapt$ of the AC is $aapt_b^{a(\beta)}$. He searches his reply table for an entry whose Aid_{old} is equal to b and $Aapt_{old}$ is equal to $aapt_b^{a(\beta)}$. Since the EQL in the entry is equal to 3 (i.e. k), he recognizes that he is the last agent. U_c calculates the pseudonym $psd_c^{a(\beta)} = pseudo\left(c, aapt_c^{a(\beta)}\right)$ then forward the reply to $psd_c^{a(\beta)}$. The original requester d gets the identity c and the $aapt_c^{a(\beta)}$ from the AC he uses, so that he uses the pseudonym $psd_c^{a(\beta)}$ as his identity. As a result, U_d get the reply from U_c. The AN of the reply is also assigned to $aapt_c^{a(\beta)}$ to avoid identical pseudonyms. The process is shown in Figure 4.9.

4.7 Appointment Number

The Appointment Number (AN) including *Capt* and *Aapt* is significant information in the AC. We explain the rules of generating them in detail and talk about the effect of ACs' timeout mechanism in this section.

4.7.1 Creator Appointment Number

The Creator Appointment Number (*Capt*) is a number which is used to identify ACs generated by the same creator. In other words, if two ACs are generated by the same creator and they do not expire, their *Capt* must be different. Therefore, the first agent (i.e., the creator) cannot find two entries which have the same $Aapt_{old}$ in his reply table, if their Aid_{old} are both his own identity.

4.7.2 Agent Appointment Number

The Agent appointment number (*Aapt*) is a number used to identify appointment cards who have the same agent. Agents generate a new *Aapt*s for appointment cards before they exchange those cards to others. In other words, an agent gives any appointment card passing on by him a unique *Aapt*, which helps the next agent identify the appointment card in his reply table. Since the *Aapt* is unique, an agent who is not the first one cannot find two entries which have the same pair of Aid_{old} and $Aapt_{old}$, neither.

4.7.3 Timeout

Agents delete the entries, which contains the information of expired ACs, from his reply table. Consequently, agents cannot forward replies using ACs which expire before this very moment. That is the reason why an original requester must choose a AC which expires after his query and reply timeout. While the timeout mechanism has more advantage than disadvantage.

Since users are moving, the distance between agents and the original requester might be too large after a long time. As a result, it is hard for agents to forward the reply messages back to the original requester, so that the original requester takes a higher risk when he uses an AC which is generated for a long time. Then these kind of ACs might not be used after a period, while it cost agents a few memories to save the ACs' information in their reply table. Therefore, all users remove the information of expired ACs to save their memories.

We should also notice that an unexpired AC and an expired one might have the same *Capt* or *Aapt*. Because no agent keeps the record of the expired AC, so that the duplication cannot confuse agents.

4.8 Example

In the following example, we use 4 users (i.e., Alice, Bob, Charlie, David and Elizabeth) to show the process of the ACP. We should notice that David trusts Charlie (Charlie does not trust David), Elizabeth trusts David (David does not trust Elizabeth). Users do not trust others except the about two pairs. The parameters are shown in Table 4.8.

Table 4.8: Example Parameters

Parameters	explanation
$\boldsymbol{k}$	3
$\boldsymbol{m}$	1
$\boldsymbol{Seg}$	2
$\boldsymbol{GP}$	10 minutes
$\boldsymbol{AT}$	80 minutes
$\boldsymbol{\tau}$	80 minutes (equal to AT)

We abstract the event that users encounter each other as a table, as shown in Table 4.9. The table list the time when two users encounter each other. For example, Bob and Charlie encounter each other at the third minute. After two users meet, they separate in one minute.

Table 4.9: Example Event

User 1	User 2	Time (minute)
Elizabeth	Bob	t_1
Bob	Charlie	t_2
Charlie	Elizabeth	t_3
Elizabeth	Alice	t_4
Bob	Charlie	t_5
Alice	Charlie	t_6
Charlie	David	t_7
David	Elizabeth	t_8

We assume that these users do not encounter any user for a long time, so that they each has 8 ($= \frac{80min}{10min}$) ACs generated by themselves. We also assume that no AC expires in our 8-minutes example. Since the parameter *Seg* is equal to 2, each user has 2 *Distributing AC Lists* (*DLs*). The initial state of our example is shown in Table 4.10. To make the example simple and clear, we do not place ACs in DLs randomly, ACs are always placed in the DL alternately. For example, if we put the first AC in DL1, then the second AC must be in the DL2. Users also obey the rule when they receive ACs from other users.

Table 4.10: Example Initial State

User Identity	DL1	DL2	RL
Alice (A)	$\mathrm{AC}_{\mathrm{A}}^{1[0]}, \mathrm{AC}_{\mathrm{A}}^{3[0]}, \mathrm{AC}_{\mathrm{A}}^{5[0]}, \mathrm{AC}_{\mathrm{A}}^{7[0]}$	$\mathrm{AC}_{\mathrm{A}}^{2[0]}, \mathrm{AC}_{\mathrm{A}}^{4[0]}, \mathrm{AC}_{\mathrm{A}}^{6[0]}, \mathrm{AC}_{\mathrm{A}}^{8[0]}$	$\emptyset$
Bob (B)	$\mathrm{AC}_{\mathrm{B}}^{1[0]}, \mathrm{AC}_{\mathrm{B}}^{3[0]}, \mathrm{AC}_{\mathrm{B}}^{5[0]}, \mathrm{AC}_{\mathrm{B}}^{7[0]}$	$\mathrm{AC}_{\mathrm{B}}^{2[0]}, \mathrm{AC}_{\mathrm{B}}^{4[0]}, \mathrm{AC}_{\mathrm{B}}^{6[0]}, \mathrm{AC}_{\mathrm{B}}^{8[0]}$	$\emptyset$
Charlie (C)	$\mathrm{AC}_{\mathrm{C}}^{1[0]}, \mathrm{AC}_{\mathrm{C}}^{3[0]}, \mathrm{AC}_{\mathrm{C}}^{5[0]}, \mathrm{AC}_{\mathrm{C}}^{7[0]}$	$\mathrm{AC}_{\mathrm{C}}^{2[0]}, \mathrm{AC}_{\mathrm{C}}^{4[0]}, \mathrm{AC}_{\mathrm{C}}^{6[0]}, \mathrm{AC}_{\mathrm{C}}^{8[0]}$	$\emptyset$
David (D)	$\mathrm{AC}_{\mathrm{D}}^{1[0]}, \mathrm{AC}_{\mathrm{D}}^{3[0]}, \mathrm{AC}_{\mathrm{D}}^{5[0]}, \mathrm{AC}_{\mathrm{D}}^{7[0]}$	$\mathrm{AC}_{\mathrm{D}}^{2[0]}, \mathrm{AC}_{\mathrm{D}}^{4[0]}, \mathrm{AC}_{\mathrm{D}}^{6[0]}, \mathrm{AC}_{\mathrm{D}}^{8[0]}$	$\emptyset$
Elizabeth (E)	$\mathrm{AC}_{\mathrm{E}}^{1[0]}, \mathrm{AC}_{\mathrm{E}}^{3[0]}, \mathrm{AC}_{\mathrm{E}}^{5[0]}, \mathrm{AC}_{\mathrm{E}}^{7[0]}$	$\mathrm{AC}_{\mathrm{E}}^{2[0]}, \mathrm{AC}_{\mathrm{E}}^{4[0]}, \mathrm{AC}_{\mathrm{E}}^{6[0]}, \mathrm{AC}_{\mathrm{E}}^{8[0]}$	$\emptyset$

The explanation of symbols used in table are shown in the former Table 4.1. To make the symbol clear, we add the EQL of the AC on its symbol. For example, if the EQL of $AC_{A,X}^{1}$ is equal to 2, it is marked as $AC_{A,X}^{1[2]}$.

4.8.1 Exchanging Appointment Cards

To make it easier, users do not pick DLs randomly as we described in the previous sections when they exchange ACs. Instead, when U_x encounters U_y, U_x always picks his DL1 and U_y always picks his DL2.

The tables in current section show the entire Distributing AC Lists (i.e. DL1 and DL2) and entire relay tables (RLs) of each user. We use the initial letter to denote each user in symbols. For example, D denotes David. In the reply tables, we color old entries with darker color to high light new ones. The structure of the AC-list tables is shown in Table 4.11, where we mark the picked DL with a darker color.

Table 4.11: User X and Y's AC Lists

X	DL1	Before	X's distributing AC list 1 before their exchange.
		After	X's distributing AC list 1 after their exchange.
	DL2	Before	X's distributing AC list 2 before their exchange.
		After	X's distributing AC list 2 after their exchange.
	RL	Before	X's ready AC list before their exchange.
		After	X's ready AC list after their exchange.
Y	DL1	Before	Y's distributing AC list 1 before their exchange.
		After	Y's distributing AC list 1 after their exchange.
	DL2	Before	Y's distributing AC list 2 before their exchange.
		After	Y's distributing AC list 2 after their exchange.
	RL	Before	Y's ready AC list before their exchange.
		After	Y's ready AC list after their exchange.

The process of exchanging AC is shown step by step as follows.

1. Elizabeth encounters Bob

At the first minute, Elizabeth encounters Bob. Elizabeth picks her DL1, and Bob picks his DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.12 and Table 4.13.

Table 4.12: Elizabeth and Bob's AC Lists At Time 1

Elizabeth	DL1	Before	$AC_E^{1[0]}, AC_E^{3[0]}, AC_E^{5[0]}, AC_E^{7[0]}$
		After	$AC_{B,B}^{2[1]}, AC_{B,B}^{6[1]}$
	DL2	Before	$AC_E^{2[0]}, AC_E^{4[0]}, AC_E^{6[0]}, AC_E^{8[0]}$
		After	$AC_E^{2[0]}, AC_E^{4[0]}, AC_E^{6[0]}, AC_E^{8[0]}, AC_{B,B}^{4[1]}, AC_{B,B}^{8[1]}$
	RL	Before	$\emptyset$
		After	$\emptyset$
Bob	DL1	Before	$AC_B^{1[0]}, AC_B^{3[0]}, AC_B^{5[0]}, AC_B^{7[0]}$
		After	$AC_B^{1[0]}, AC_B^{3[0]}, AC_B^{5[0]}, AC_B^{7[0]}, AC_{E,E}^{1[1]}, AC_{E,E}^{5[1]}$
	DL2	Before	$AC_B^{2[0]}, AC_B^{4[0]}, AC_B^{6[0]}, AC_B^{8[0]}$
		After	$AC_{E,E}^{3[1]}, AC_{E,E}^{7[1]}$
	RL	Before	$\emptyset$
		After	$\emptyset$

Table 4.13: Elizabeth and Bob's Relay Table At Time t$_1$

Elizabeth					Bob				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
E	$capt_E^1$	B	$aapt_E^{E(1)}$	1	B	B	$capt_B^2$	E	$aapt_B^{B(2)}$
E	$capt_E^3$	B	$aapt_E^{E(3)}$	1	B	B	$capt_B^4$	E	$aapt_B^{B(4)}$
E	$capt_E^5$	B	$aapt_E^{E(5)}$	1	B	B	$capt_B^6$	E	$aapt_B^{B(6)}$
E	$capt_E^7$	B	$aapt_E^{E(7)}$	1	B	B	$capt_B^8$	E	$aapt_B^{B(8)}$

1. Bob encounters Charlie

At time t$_2$, Bob encounters Charlie. Bob picks his DL1, and Charlie picks his DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.14 and Table 4.15.

Table 4.14: Bob and Charlie's AC Lists At Time t_2

Bob	DL1	Before	$AC_{B}^{1[0]}, AC_{B}^{3[0]}, AC_{B}^{5[0]}, AC_{B}^{7[0]}, AC_{E,E}^{1[1]}, AC_{E,E}^{5[1]}$
		After	$AC_{C,C}^{2[1]}, AC_{C,C}^{6[1]}$
	DL2	Before	$AC_{E,E}^{3[1]}, AC_{E,E}^{7[1]}$
		After	$AC_{E,E}^{3[1]}, AC_{E,E}^{7[1]}, AC_{C,C}^{4[1]}, AC_{C,C}^{8[1]}$
	RL	Before	$\emptyset$
		After	$\emptyset$
Charlie	DL1	Before	$AC_{C}^{1[0]}, AC_{C}^{3[0]}, AC_{C}^{5[0]}, AC_{C}^{7[0]}$
		After	$AC_{C}^{1[0]}, AC_{C}^{3[0]}, AC_{C}^{5[0]}, AC_{C}^{7[0]}, AC_{B,B}^{1[1]}, AC_{B,B}^{5[1]}, AC_{E,B}^{1[2]}$
	DL2	Before	$AC_{C}^{2[0]}, AC_{C}^{4[0]}, AC_{C}^{6[0]}, AC_{C}^{8[0]}$
		After	$AC_{B,B}^{3[1]}, AC_{B,B}^{7[1]}, AC_{E,B}^{5[2]}$
	RL	Before	$\emptyset$
		After	$\emptyset$

Table 4.15: Bob and Charlie's Relay Table At Time t_2

Bob					Charlie				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
B	$capt_B^2$	E	$aapt_B^{B(2)}$	1	C	$capt_C^2$	B	$aapt_C^{C(2)}$	1
B	$capt_B^4$	E	$aapt_B^{B(4)}$	1	C	$capt_C^4$	B	$aapt_C^{C(4)}$	1
B	$capt_B^6$	E	$aapt_B^{B(6)}$	1	C	$capt_C^6$	B	$aapt_C^{C(6)}$	1
B	$capt_B^8$	E	$aapt_B^{B(8)}$	1	C	$capt_C^8$	B	$aapt_C^{C(8)}$	1
B	$capt_B^1$	C	$aapt_B^{B(1)}$	1					
B	$capt_B^3$	C	$aapt_B^{B(3)}$	1					
B	$capt_B^5$	C	$aapt_B^{B(5)}$	1					
B	$capt_B^7$	C	$aapt_B^{B(7)}$	1					
E	$aapt_E^{E(1)}$	C	$aapt_B^{E(1)}$	2					
E	$aapt_E^{E(5)}$	C	$aapt_B^{E(5)}$	2					

1. Charlie encounters Elizabeth

At time t_3, Charlie encounters Elizabeth. Charlie picks his DL1, and Elizabeth picks her DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.16 and Table 4.17.

We should notice that Charlie does not give $AC_{E,B}^{1[2]}$ to Elizabeth, because Elizabeth does not trust Charlie. The EQL of the $AC_{E,B}^{1[2]}$ has already reached 2, which is equal to $k-m = 3-1$. As mentioned in 4.6.2, these kinds of ACs are only sent to users who trust the current agent. Therefore, Charlie cannot give $AC_{E,B}^{1[2]}$ to Elizabeth when she does not trust him.

Table 4.16: Charlie and Elizabeth's AC Lists At Time t$_3$

Charlie	DL1	Before	$AC_{C}^{1[0]}, AC_{C}^{3[0]}, AC_{C}^{5[0]}, AC_{C}^{7[0]}, AC_{B,B}^{1[1]}, AC_{B,B}^{5[1]}, AC_{E,B}^{1[2]}$
		After	$AC_{E,B}^{1[2]}, AC_{E,E}^{2[1]}, AC_{E,E}^{6[1]}, AC_{B,E}^{4[2]}$
	DL2	Before	$AC_{B,B}^{3[1]}, AC_{B,B}^{7[1]}, AC_{E,B}^{5[2]}$
		After	$AC_{B,B}^{3[1]}, AC_{B,B}^{7[1]}, AC_{E,B}^{5[2]}, AC_{E,E}^{4[1]}, AC_{E,E}^{8[1]}, AC_{B,E}^{8[2]}$
	RL	Before	$\emptyset$
		After	$\emptyset$
Elizabeth	DL1	Before	$AC_{B,B}^{2[1]}, AC_{B,B}^{6[1]}$
		After	$AC_{B,B}^{2[1]}, AC_{B,B}^{6[1]}, AC_{C,C}^{1[1]}, AC_{C,C}^{5[1]}, AC_{B,C}^{1[2]}$
	DL2	Before	$AC_{E}^{2[0]}, AC_{E}^{4[0]}, AC_{E}^{6[0]}, AC_{E}^{8[0]}, AC_{B,B}^{4[1]}, AC_{B,B}^{8[1]}$
		After	$AC_{C,C}^{3[1]}, AC_{C,C}^{7[1]}, AC_{B,C}^{5[2]}$
	RL	Before	$\emptyset$
		After	$\emptyset$

Table 4.17: Charlie and Elizabeth's Relay Table At Time t$_3$

Charlie					Elizabeth				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
C	$capt_{C}^{2}$	B	$aapt_{C}^{C(2)}$	1	E	$capt_{E}^{1}$	B	$aapt_{E}^{E(1)}$	1
C	$capt_{C}^{4}$	B	$aapt_{C}^{C(4)}$	1	E	$capt_{E}^{3}$	B	$aapt_{E}^{E(3)}$	1
C	$capt_{C}^{6}$	B	$aapt_{C}^{C(6)}$	1	E	$capt_{E}^{5}$	B	$aapt_{E}^{E(5)}$	1
C	$capt_{C}^{8}$	B	$aapt_{C}^{C(8)}$	1	E	$capt_{E}^{7}$	B	$aapt_{E}^{E(7)}$	1
C	$capt_{C}^{1}$	E	$aapt_{C}^{C(1)}$	1	E	$capt_{E}^{2}$	C	$aapt_{E}^{E(2)}$	1
C	$capt_{C}^{3}$	E	$aapt_{C}^{C(3)}$	1	E	$capt_{E}^{4}$	C	$aapt_{E}^{E(4)}$	1
C	$capt_{C}^{5}$	E	$aapt_{C}^{C(5)}$	1	E	$capt_{E}^{6}$	C	$aapt_{E}^{E(6)}$	1
C	$capt_{C}^{7}$	E	$aapt_{C}^{C(7)}$	1	E	$capt_{E}^{8}$	C	$aapt_{E}^{E(8)}$	1
B	$aapt_{B}^{B(1)}$	E	$aapt_{C}^{B(1)}$	2	B	$aapt_{B}^{B(4)}$	C	$aapt_{E}^{B(4)}$	2
B	$aapt_{B}^{B(5)}$	E	$aapt_{C}^{B(5)}$	2	B	$aapt_{B}^{B(8)}$	C	$aapt_{E}^{B(8)}$	2

1. Elizabeth encounters Alice

At time t$_4$, Elizabeth encounters Alice. Elizabeth picks her DL1, and Alice picks her DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.18 and Table 4.19.

Elizabeth does not give $AC^{1[2]}_{B,C}$ to Alice for the same reason as the previous Charlie, that is, Alice does not trust Elizabeth.

Table 4.18: Elizabeth and Alice's AC Lists At Time t$_4$

Elizabeth	DL1	Before	$AC^{2[1]}_{B,B}, AC^{6[1]}_{B,B}, AC^{1[1]}_{C,C}, AC^{5[1]}_{C,C}, AC^{1[2]}_{B,C}$
		After	$AC^{1[2]}_{B,C}, AC^{2[1]}_{A,A}, AC^{6[1]}_{A,A}$
	DL2	Before	$AC^{3[1]}_{C,C}, AC^{7[1]}_{C,C}, AC^{5[2]}_{B,C}$
		After	$AC^{3[1]}_{C,C}, AC^{7[1]}_{C,C}, AC^{5[2]}_{B,C}, AC^{4[1]}_{A,A}, AC^{8[1]}_{A,A}$
	RL	Before	$\emptyset$
		After	$\emptyset$
Alice	DL1	Before	$AC^{1[0]}_{A}, AC^{3[0]}_{A}, AC^{5[0]}_{A}, AC^{7[0]}_{A}$
		After	$AC^{1[0]}_{A}, AC^{3[0]}_{A}, AC^{5[0]}_{A}, AC^{7[0]}_{A}, AC^{2[2]}_{B,E}, AC^{1[2]}_{C,E}$
	DL2	Before	$AC^{2[0]}_{A}, AC^{4[0]}_{A}, AC^{6[0]}_{A}, AC^{8[0]}_{A}$
		After	$AC^{6[2]}_{B,E}, AC^{5[2]}_{C,E}$
	RL	Before	$\emptyset$
		After	$\emptyset$

Table 4.19: Elizabeth and Alice's Relay Table At Time t$_4$

Elizabeth					Alice				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
E	$capt_E^1$	B	$aapt_E^{E(1)}$	1	A	$capt_A^2$	E	$aapt_A^{A(2)}$	1
E	$capt_E^3$	B	$aapt_E^{E(3)}$	1	A	$capt_A^4$	E	$aapt_A^{A(4)}$	1
E	$capt_E^5$	B	$aapt_E^{E(5)}$	1	A	$capt_A^6$	E	$aapt_A^{A(6)}$	1
E	$capt_E^7$	B	$aapt_E^{E(7)}$	1	A	$capt_A^8$	E	$aapt_A^{A(8)}$	1
E	$capt_E^2$	C	$aapt_E^{E(2)}$	1					
E	$capt_E^4$	C	$aapt_E^{E(4)}$	1					
E	$capt_E^6$	C	$aapt_E^{E(6)}$	1					
E	$capt_E^8$	C	$aapt_E^{E(8)}$	1					
B	$aapt_B^{B(4)}$	C	$aapt_E^{B(4)}$	2					
B	$aapt_B^{B(8)}$	C	$aapt_E^{B(8)}$	2					
B	$aapt_B^{B(2)}$	A	$aapt_E^{B(2)}$	2					
B	$aapt_B^{B(6)}$	A	$aapt_E^{B(6)}$	2					
C	$aapt_C^{C(1)}$	A	$aapt_E^{C(1)}$	2					
C	$aapt_C^{C(5)}$	A	$aapt_E^{C(5)}$	2					

1. Bob encounters Charlie

At time t$_5$, Bob encounters Charlie. Bob picks his DL1, and Charlie picks his DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.20 and Table 4.21.

Charlie does not give $AC_{E,B}^{5[2]}$ and $AC_{B,E}^{8[2]}$ to Bob, because Bob does not trust Charlie. Since we set the avoiding time equal to AT, a distributing AC cannot be given back to previous agents. As a result, $AC_{C,C}^{2[1]}$, $AC_{C,C}^{6[1]}$, $AC_{B,B}^{3[1]}$ and $AC_{B,B}^{7[1]}$ are not exchanged by Charlie and Bob.

Table 4.20: Bob and Charlie's AC Lists At Time t$_5$

Bob	DL1	Before	$AC^{2[1]}_{C,C}, AC^{6[1]}_{C,C}$
		After	$AC^{2[1]}_{C,C}, AC^{6[1]}_{C,C}, AC^{4[2]}_{E,C}$
	DL2	Before	$AC^{3[1]}_{E,E}, AC^{7[1]}_{E,E}, AC^{4[1]}_{C,C}, AC^{8[1]}_{C,C}$
		After	$AC^{3[1]}_{E,E}, AC^{7[1]}_{E,E}, AC^{4[1]}_{C,C}, AC^{8[1]}_{C,C}, AC^{8[2]}_{E,C}$
	RL	Before	$\emptyset$
		After	$\emptyset$
Charlie	DL1	Before	$AC^{1[2]}_{E,B}, AC^{2[1]}_{E,E}, AC^{6[1]}_{E,E}, AC^{4[2]}_{B,E}$
		After	$AC^{1[2]}_{E,B}, AC^{2[1]}_{E,E}, AC^{6[1]}_{E,E}, AC^{4[2]}_{B,E}$
	DL2	Before	$\mathrm{AC}^{3[1]}_{\mathrm{B,B}}, \mathrm{AC}^{7[1]}_{\mathrm{B,B}}, \mathrm{AC}^{5[2]}_{\mathrm{E,B}}, \mathrm{AC}^{4[1]}_{\mathrm{E,E}}, \mathrm{AC}^{8[1]}_{\mathrm{E,E}}, \mathrm{AC}^{8[2]}_{\mathrm{B,E}}$
		After	$AC^{3[1]}_{B,B}, AC^{7[1]}_{B,B}, AC^{5[2]}_{E,B}, AC^{8[2]}_{B,E}$
	RL	Before	$\emptyset$
		After	$\emptyset$

Table 4.21: Bob and Charlie's Relay Table At Time t_5

Bob					Charlie				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{old}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{old}$	EQL
B	$capt_B^2$	E	$aapt_B^{B(2)}$	1	C	$capt_C^2$	B	$aapt_C^{C(2)}$	1
B	$capt_B^4$	E	$aapt_B^{B(4)}$	1	C	$capt_C^4$	B	$aapt_C^{C(4)}$	1
B	$capt_B^6$	E	$aapt_B^{B(6)}$	1	C	$capt_C^6$	B	$aapt_C^{C(6)}$	1
B	$capt_B^8$	E	$aapt_B^{B(8)}$	1	C	$capt_C^8$	B	$aapt_C^{C(8)}$	1
B	$capt_B^1$	C	$aapt_B^{B(1)}$	1	C	$capt_C^1$	E	$aapt_C^{C(1)}$	1
B	$capt_B^3$	C	$aapt_B^{B(3)}$	1	C	$capt_C^3$	E	$aapt_C^{C(3)}$	1
B	$capt_B^5$	C	$aapt_B^{B(5)}$	1	C	$capt_C^5$	E	$aapt_C^{C(5)}$	1
B	$capt_B^7$	C	$aapt_B^{B(7)}$	1	C	$capt_C^7$	E	$aapt_C^{C(7)}$	1
E	$aapt_E^{E(1)}$	C	$aapt_B^{E(1)}$	2	B	$aapt_B^{B(1)}$	E	$aapt_C^{B(1)}$	2
E	$aapt_E^{E(5)}$	C	$aapt_B^{E(5)}$	2	B	$aapt_B^{B(5)}$	E	$aapt_C^{B(5)}$	2
					E	$aapt_E^{E(4)}$	B	$aapt_C^{E(4)}$	2
					E	$aapt_E^{E(8)}$	B	$aapt_C^{E(8)}$	2

1. Alice encounters Charlie

At time t_6, Alice encounters Charlie. Alice picks her DL1, and Charlie picks his DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.22 and Table 4.23.

Table 4.22: Alice and Charlie's AC Lists At Time t_6

Bob	DL1	Before	$AC_{A}^{1[0]}, AC_{A}^{3[0]}, AC_{A}^{5[0]}, AC_{A}^{7[0]}, AC_{B,E}^{2[2]}, AC_{C,E}^{1[2]}$
		After	$AC_{B,E}^{2[2]}, AC_{C,E}^{1[2]}, AC_{B,C}^{3[2]}$
	DL2	Before	$AC_{B,E}^{6[2]}, AC_{C,E}^{5[2]}$
		After	$AC_{B,E}^{6[2]}, AC_{C,E}^{5[2]}, AC_{B,C}^{7[2]}$
	RL	Before	$\emptyset$
		After	$\emptyset$
Charlie	DL1	Before	$AC_{E,B}^{1[2]}, AC_{E,E}^{2[1]}, AC_{E,E}^{6[1]}, AC_{B,E}^{4[2]}$
		After	$AC_{E,B}^{1[2]}, AC_{E,E}^{2[1]}, AC_{E,E}^{6[1]}, AC_{B,E}^{4[2]}, AC_{A,A}^{1[1]}, AC_{A,A}^{5[1]}$
	DL2	Before	$AC_{B,B}^{3[1]}, AC_{B,B}^{7[1]}, AC_{E,B}^{5[2]}, AC_{B,E}^{8[2]}$
		After	$AC_{E,B}^{5[2]}, AC_{B,E}^{8[2]}, AC_{A,A}^{3[1]}, AC_{A,A}^{7[1]}$
	RL	Before	$\emptyset$
		After	$\emptyset$

Table 4.23: Alice and Charlie's Relay Table At Time t$_6$

Alice					Charlie				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
A	$capt_A^2$	E	$aapt_A^{A(2)}$	1	C	$capt_C^2$	B	$aapt_C^{C(2)}$	1
A	$capt_A^4$	E	$aapt_A^{A(4)}$	1	C	$capt_C^4$	B	$aapt_C^{C(4)}$	1
A	$capt_A^6$	E	$aapt_A^{A(6)}$	1	C	$capt_C^6$	B	$aapt_C^{C(6)}$	1
A	$capt_A^8$	E	$aapt_A^{A(8)}$	1	C	$capt_C^8$	B	$aapt_C^{C(8)}$	1
A	$capt_A^1$	C	$aapt_A^{A(1)}$	1	C	$capt_C^1$	E	$aapt_C^{C(1)}$	1
A	$capt_A^3$	C	$aapt_A^{A(3)}$	1	C	$capt_C^3$	E	$aapt_C^{C(3)}$	1
A	$capt_A^5$	C	$aapt_A^{A(5)}$	1	C	$capt_C^5$	E	$aapt_C^{C(5)}$	1
A	$capt_A^7$	C	$aapt_A^{A(7)}$	1	C	$capt_C^7$	E	$aapt_C^{C(7)}$	1
					B	$aapt_B^{B(1)}$	E	$aapt_C^{B(1)}$	2
					B	$aapt_B^{B(5)}$	E	$aapt_C^{B(5)}$	2
					E	$aapt_E^{E(4)}$	B	$aapt_C^{E(4)}$	2
					E	$aapt_E^{E(8)}$	B	$aapt_C^{E(8)}$	2
					B	$aapt_B^{B(3)}$	A	$aapt_C^{B(3)}$	2
					B	$aapt_B^{B(7)}$	A	$aapt_C^{B(7)}$	2

Since Alice and Charlie do not trust each other, $AC_{B,E}^{2[2]}$, $AC_{C,E}^{1[2]}$, $AC_{E,B}^{5[2]}$ and $AC_{B,E}^{8[2]}$ are not exchanged by the two users.

1. Charlie encounters David

At time t$_7$, Charlie encounters David. Charlie pick his DL1, and David picks his DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.24 and Table 4.25.

Because David trust Charlie, Charlie can give $AC_{E,B}^{1[2]}$ and $AC_{B,E}^{4[2]}$ to David. When David receives $AC_{E,C}^{1[3]}$ and $AC_{B,C}^{4[3]}$, he notices that the EQLs of the two ACs reach k so that he puts these two ACs in his ready list instead of distributing lists.

Table 4.24: Charlie and David's AC Lists At Time t$_7$

Charlie	DL1	Before	$AC^{1[2]}_{E,B}, AC^{2[1]}_{E,E}, AC^{6[1]}_{E,E}, AC^{4[2]}_{B,E}, AC^{1[1]}_{A,A}, AC^{5[1]}_{A,A}$
		After	$AC^{2[1]}_{D,D}, AC^{6[1]}_{D,D}$
	DL2	Before	$AC^{5[2]}_{E,B}, AC^{8[2]}_{B,E}, AC^{3[1]}_{A,A}, AC^{7[1]}_{A,A}$
		After	$AC^{5[2]}_{E,B}, AC^{8[2]}_{B,E}, AC^{3[1]}_{A,A}, AC^{7[1]}_{A,A}, AC^{4[1]}_{D,D}, AC^{8[1]}_{D,D}$
	RL	Before	$\emptyset$
		After	$\emptyset$
David	DL1	Before	$AC^{1[0]}_{D}, AC^{3[0]}_{D}, AC^{5[0]}_{D}, AC^{7[0]}_{D}$
		After	$AC^{1[0]}_{D}, AC^{3[0]}_{D}, AC^{5[0]}_{D}, AC^{7[0]}_{D}, AC^{2[2]}_{E,C}, AC^{1[2]}_{A,C}$
	DL2	Before	$AC^{2[0]}_{D}, AC^{4[0]}_{D}, AC^{6[0]}_{D}, AC^{8[0]}_{D}$
		After	$AC^{6[2]}_{E,C}, AC^{5[2]}_{A,C}$
	RL	Before	$\emptyset$
		After	$\mathrm{AC}^{1[3]}_{\mathrm{E,C}}, \mathrm{AC}^{4[3]}_{\mathrm{B,C}}$

Table 4.25: Charlie and David's Relay Table At Time t$_7$

Charlie					David				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
C	$capt_C^2$	B	$aapt_C^{C(2)}$	1	D	$capt_D^2$	C	$aapt_D^{D(2)}$	1
C	$capt_C^4$	B	$aapt_C^{C(4)}$	1	D	$capt_D^4$	C	$aapt_D^{D(4)}$	1
C	$capt_C^6$	B	$aapt_C^{C(6)}$	1	D	$capt_D^6$	C	$aapt_D^{D(6)}$	1
C	$capt_C^8$	B	$aapt_C^{C(8)}$	1	D	$capt_D^8$	C	$aapt_D^{D(8)}$	1
C	$capt_C^1$	E	$aapt_C^{C(1)}$	1					
C	$capt_C^3$	E	$aapt_C^{C(3)}$	1					
C	$capt_C^5$	E	$aapt_C^{C(5)}$	1					
C	$capt_C^7$	E	$aapt_C^{C(7)}$	1					
B	$aapt_B^{B(1)}$	E	$aapt_C^{B(1)}$	2					
B	$aapt_B^{B(5)}$	E	$aapt_C^{B(5)}$	2					
E	$aapt_E^{E(4)}$	B	$aapt_C^{E(4)}$	2					
E	$aapt_E^{E(8)}$	B	$aapt_C^{E(8)}$	2					
B	$aapt_B^{B(3)}$	A	$aapt_C^{B(3)}$	2					
B	$aapt_B^{B(7)}$	A	$aapt_C^{B(7)}$	2					
B	$aapt_B^{E(1)}$	VOID	$aapt_C^{E(1)}$	3					
E	$aapt_E^{E(2)}$	D	$aapt_C^{E(2)}$	2					
E	$aapt_E^{E(6)}$	D	$aapt_C^{E(6)}$	2					
E	$aapt_E^{B(4)}$	VOID	$aapt_C^{B(4)}$	3					
A	$aapt_A^{A(1)}$	D	$aapt_C^{A(1)}$	2					
A	$aapt_A^{A(5)}$	D	$aapt_C^{A(5)}$	2					

1. David encounters Elizabeth

At time t_8, David encounters Elizabeth. David picks his DL1, and Elizabeth picks her DL2. They give distributing ACs in their picked DLs to each other and update their reply tables, as shown in Table 4.26 and Table 4.27.

After David gives Elizabeth the distributing ACs, Elizabeth gets a ready AC $AC^{1[3]}_{A,D}$. Then Elizabeth has 1 ready AC, while David has 2 ready ACs, so that David has only one more ready ACs than Elizabeth. That is the reason why David does not give his own ready ACs to Elizabeth, even though Elizabeth trusts David.

We assume that Elizabeth does not get $AC^{1[3]}_{A,D}$, so that she has no ready AC. Then David should give her a ready AC from his own two randomly.

Table 4.26: David and Elizabeth's AC Lists At Time t_8

David	DL1	Before	$AC^{1[0]}_{D}, AC^{3[0]}_{D}, AC^{5[0]}_{D}, AC^{7[0]}_{D}, AC^{2[2]}_{E,C}, AC^{1[2]}_{A,C}$
		After	$AC^{2[2]}_{E,C}, AC^{3[2]}_{C,E}, AC^{4[2]}_{A,E}$
	DL2	Before	$AC^{6[2]}_{E,C}, AC^{5[2]}_{A,C}$
		After	$AC^{6[2]}_{E,C}, AC^{5[2]}_{A,C}, AC^{7[2]}_{C,E}, AC^{8[2]}_{A,E}$
	RL	Before	$\mathrm{AC}^{1[3]}_{\mathrm{E,C}}, \mathrm{AC}^{4[3]}_{\mathrm{B,C}}$
		After	$\mathrm{AC}^{1[3]}_{\mathrm{E,C}}, \mathrm{AC}^{4[3]}_{\mathrm{B,C}}$
Elizabeth	DL1	Before	$AC^{1[2]}_{B,C}, AC^{2[1]}_{A,A}, AC^{6[1]}_{A,A}$
		After	$AC^{1[2]}_{B,C}, AC^{2[1]}_{A,A}, AC^{6[1]}_{A,A}, AC^{1[1]}_{D,D}, AC^{5[1]}_{D,D}$
	DL2	Before	$AC^{3[1]}_{C,C}, AC^{7[1]}_{C,C}, AC^{5[2]}_{B,C}, AC^{4[1]}_{A,A}, AC^{8[1]}_{A,A}$
		After	$AC^{5[2]}_{B,C}, AC^{3[1]}_{D,D}, AC^{7[1]}_{D,D}$
	RL	Before	$\emptyset$
		After	$\mathrm{AC}^{1[3]}_{\mathrm{A,D}}$

Table 4.27: David and Elizabeth's Relay Table At Time t_8

David					Elizabeth				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL	Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
D	$capt_D^2$	C	$aapt_D^{D(2)}$	1	E	$capt_E^1$	B	$aapt_E^{E(1)}$	1
D	$capt_D^4$	C	$aapt_D^{D(4)}$	1	E	$capt_E^3$	B	$aapt_E^{E(3)}$	1
D	$capt_D^6$	C	$aapt_D^{D(6)}$	1	E	$capt_E^5$	B	$aapt_E^{E(5)}$	1
D	$capt_D^8$	C	$aapt_D^{D(8)}$	1	E	$capt_E^7$	B	$aapt_E^{E(7)}$	1
D	$capt_D^1$	E	$aapt_D^{D(1)}$	1	E	$capt_E^2$	C	$aapt_E^{E(2)}$	1
D	$capt_D^3$	E	$aapt_D^{D(3)}$	1	E	$capt_E^4$	C	$aapt_E^{E(4)}$	1
D	$capt_D^5$	E	$aapt_D^{D(5)}$	1	E	$capt_E^6$	C	$aapt_E^{E(6)}$	1
D	$capt_D^7$	E	$aapt_D^{D(7)}$	1	E	$capt_E^8$	C	$aapt_E^{E(8)}$	1
C	$aapt_C^{A(1)}$	VOID	$aapt_D^{A(1)}$	3	B	$aapt_B^{B(4)}$	C	$aapt_E^{B(4)}$	2
					B	$aapt_B^{B(8)}$	C	$aapt_E^{B(8)}$	2
					B	$aapt_B^{B(2)}$	A	$aapt_E^{B(2)}$	2
					B	$aapt_B^{B(6)}$	A	$aapt_E^{B(6)}$	2
					C	$aapt_C^{C(1)}$	A	$aapt_E^{C(1)}$	2
					C	$aapt_C^{C(5)}$	A	$aapt_E^{C(5)}$	2
					C	$aapt_C^{C(3)}$	D	$aapt_E^{C(3)}$	2
					C	$aapt_C^{C(7)}$	D	$aapt_E^{C(7)}$	2
					A	$aapt_A^{A(4)}$	D	$aapt_E^{A(4)}$	2
					A	$aapt_A^{A(8)}$	D	$aapt_E^{A(8)}$	2

4.8.2 Queries and Replies

We still use the previous example and assume that the delivery of queries and replies cost little time so that no AC expires in the period.

We assume that David wants to send a query to the LBS. He chooses a AC (e.g. $AC_{E,C}^{1[3]}$) from his two ACs (i.e. $AC_{E,C}^{1[3]}$ and $AC_{B,C}^{4[3]}$) randomly. If we review the previous tables, we learn that $AC_{E,C}^{1[3]}$ is generated by Elizabeth and has 3 agents (i.e., Elizabeth, Bob and Charlie). When David uses the AC to send his query, the sender identity of the query is Elizabeth. The creator appointment number (i.e. $capt_E^1$) is also included in the query. Then David uses the identity of the last agent (i.e. Charlie) and the agent appointment number (i.e. $aapt_C^{E(1)}$) to generate a pseudonym $psd_c^{E(1)} = \text{pseudo}\left(Charlie, aapt_c^{\mathrm{E}(1)}\right)$, which is used as a temporary identity of David before he receives the reply. Besides, $AC_{E,C}^{1[3]}$ cannot be given to any user until it expires. The process is shown in Figure 4.10.

Figure 4.10: David's Query

The LBS sends a reply message to Elizabeth based on the sender identity of the query. The creator appointment number (i.e. $capt_E^1$) is also included in the reply. If an attacker learns these messages, he believes that it is Elizabeth who sends the query.

When Elizabeth receives the reply message, she searches her reply table for the AC used in the reply message, as shown in Table 4.27. Since the message is sent by the LBS, the Aid_{old} should equal to her own identity, and the $Aapt_{old}$ should equal to the creator appointment number. Then she must find an entry like Table 4.28, which enables Elizabeth to modify the reply message and forward to Bob, as shown in Figure 4.11.

Table 4.28: Elizabeth's Reply Table Entry

Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
E	$capt_E^1$	B	$aapt_E^{E(1)}$	1

Figure 4.11: Elizabeth Forwards A Reply

If Bob receives the reply message successfully, he checks his reply table as shown in Table 4.21. He finds an entry whose Aid_{old} is equal to Elizabeth and $Aapt_{old}$ is equal to $aapt_E^{E(1)}$, as shown in Table 4.29. In the same way as Elizabeth, Bob forwards reply message to Charlie, as shown in Figure 4.12.

Table 4.29: Bob's Reply Table Entry

Bob				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{old}$	EQL
E	$aapt_{E}^{E(1)}$	C	$aapt_{B}^{E(1)}$	2

Figure 4.12: Bob Forwards A Reply

When the Charlie receives the reply message from Bob, Charlie learns that he is the last agent based on his reply table as shown in Table 4.25 and Table 4.30, because the EQL is equal to 3 and the ID_{nxt} is void.

Table 4.30: Charlie's Reply Table Entry

Charlie				
Aid_{old}	$Aapt_{old}$	ID_{nxt}	$Aapt_{new}$	EQL
B	$aapt_{B}^{E(1)}$	VOID	$aapt_{C}^{E(1)}$	3

To forward to the unknown original requester (i.e. David), Charlie generates a pseudonym $psd_{c}^{E(1)} = \text{pseudo}\left(Charlie, aapt_{c}^{\mathrm{E}(1)}\right)$ using his own identity and $Aapt_{new}$. We should notice that the pseudonym is exactly the temporary identity of David. Then David modifies the reply message and forwards it, as shown in Figure 4.13.

Figure 4.13: Charlie Forwards A Reply

When a user who carries $Reply_{C}^{E(1)}$ encounters David, the user forwards the reply to him, so that David gets his reply.

4.9 Experiment

We used the map of Helsinki in our simulator to evaluate performance of the proposed ACP. It is also compared against Binary Spray and Wait (BSW) protocol [32], distributed social based location privacy protocol (SLPD) [17] and our Multi-Hop Location-Privacy Protection (MHLPP). We simulated continuous movement of users along streets on the map with one LBSP, fixed at a random location on the map.

For each user, we associate a random social value between 0 - 100%, each corresponding to other users. Since each social value is assigned with equal probability, we can compute the expected number of friends of a user. If a user whose social value is larger than 85% is called a friend and there are n users in the network, there are n?85% friends.

The Shortest Path Map-Based Movement (SPMBM) [36] is used in our experiment. For each experiment, we give the simulator a random seed so that it can generate pseudo-random number based on the seed. Therefore, all the factors including users' speed and locations are the same if two experiments have the same random seed. All four protocols are tested using the same set of random seeds.

Before each experiment, the simulator runs for 800 seconds (simulator time). Then we pick 100 users out of 126 users randomly, and each of them send a query to the LBSP. Tests last for about 20 minutes (simulator time).

4.9.1 Average Query Success Ratio

The query success ratio is the percentage of delivered queries among a number of attempts. Since users sending 100 queries in each experiment totally, if s queries are delivered to the LBSP at time t, the query success ratio of time t is s%.

As shown in Figure 4.14, we compare the average query success ratio of the four protocols with 5 kinds of communication radius (10, 30, 50, 70 and 90 meters). We observe that the ACP and the BSW get a high query success ratio, while the MHLPP and the SLPD are lower than the former two protocols. The BSW is the highest one, because it is a no-privacy protocol. The ACP is just a little lower than BSW, because the query delivery process of the ACP is almost the same as that of BSW. Since users of ACP must wait for available appointment cards, they cost more time to initial their queries. But the ACP and the BSW are in the same level, comparing to the other two protocols. The MHLPP and the SLPD need to find friend to obfuscate their queries, which baffles their delivery process.

The experiment results when we test 20 minutes is shown in Figure 4.15. Comparing to the average query success ratio at 10 minutes, the MHLPP and the SLPD achieve much higher success ratio after 20 minutes than at 10 minutes. That is because it cost them so much time in their obfuscation phases when they need to find friends. In fact, some of queries of the MHLPP and the SLPD still do not finish their obfuscation phase at 20 minutes.

The communication radius can influence the success ratio. In most of cases, the success ratio rises when we increase the communication radius, and its influence is especially obvious under 50 meters. A large communication radius makes it easily for users to encounter others, which is good for them to forward queries. However, a user who is so far away from the destination does not want the intermediates of his query encounters many users nearby. Because all users who carries copies of that query are near the sender instead of the destination, which decreases their query success ratio. Therefore, when the communication radius reaches 70 meters, the success ratios almost stay at the same level.

In Figure 4.16, we observe that the ACP and the BSW have better convergence speed than the MHLPP and the SLPD. In other words, the former two protocols have a faster speed to approach the 100% query success ratio than the later two. At the very beginning, the ACP even has a little higher success ratio than the BSW. Because the ACP users need ready appointment card, and most of the users get their first ready appointment cards at places where there are many users. Therefore, users rarely generate query near the edges of the map at the beginning, which facilitates their queries delivery process. For example, if a user generates his query at the edge of the map, the copies of his query will be sent to users who is also at the edge and it takes more time for them to deliver the query. If the user does not generate his query until he arrives a place nearer center, the copies of his query will be carried by the users in the center with higher probability.

4.9.2 Average Reply Success Ratio

When the LBSP receives a query, it sends a reply to the requester. If the reply arrives the original requester before the test ends, we view it as success, otherwise, the reply is failed. There are two reasons for the failure of replies: 1) the query is not delivered to the LBSP successfully; 2) the query costs too much time so that the reply has no time to be delivered; 3) The route of the reply is too long. Since there are 100 queries in each experiment, the number of replies should be equal to 100.

In Figure 4.17, the BSW has a significant and reasonable higher success ratio than all other protocols, because it is a no-privacy protocol. The ACP is higher than the MHLPP and the SLPD, but its advantage is not as large as that in the query process. In fact, the reply process of the MHLPP and the SLPD are simpler than that of the ACP, but the ACP saves so much time in its query process that it earns a better reply success ratio than the other two.

4.9.3 Total Number of Query Relays

The query delivery processes of all the four protocols use the BSW protocol, and the BSW makes copies for queries and gives half of copies to any users it encounters. That is a significant cost for the network, so we use the number of query relaying (QR) to evaluate that cost. The QR is initialized as zero at the beginning of the test. When a user relay a (or several) copy (copies) of a query, we increase QR by 1. For example, in the SLPD, there are two phases: the obfuscation phase and the free phase. In the obfuscation phase, a query is forwarded among one-hop friends for k times. After that, it is forwarded by the BSW protocol. The BSW protocol makes c copies for the query and gives half of the copies to any encountering users. Then QR should be about $k + c$. Since a user gives all its copies to the destination if he encounters the destination at once, the QR may be smaller. The smaller that number is, the smaller cost of the network is. Since there are 100 queries, we divide QR by 100 to get an average value.

In Figure 4.18, we compare QR with four protocols. We observe that all the four protocols are at a similar level, the BSW and the ACP is a little lower than the other two. For the ACP and the BSW, they deliver queries so fast that users who carries more than one copies give all their copies to the destination before they send these copies to different users separately. As a result, many copies have no chance to be forwarded, which decreases the cost. While the MHLPP and the SLPD have obfuscation phases, the queries start to be delivered freely (in a BSW way) at a random place where might be so far way from the destination, so that almost all copies can be forwarded respectively.

The communication radius effects the total number of the forwarding queries, especially for the MHLPP and the SLPD. Those two protocols can finish their obfuscation phase more quickly which a larger communication radius, so that more queries can be forwarded freely (in the BSW way), which makes their QR larger.

4.9.4 Memory Cost

We count the number of queries carried by each user to evaluate the memory cost of the four protocols. Several copies of a query are counted for only once.

In the Figure 4.19, we compare the number of queries per user with the four protocols at 20 minutes. We observe that the BSW is the highest in most of the cases and the ACP always stays at a similar level as the BSW. The data of other two protocols (the MHLPP and the SLPD) increase as the communication radius. The MHLPP even excesses the BSW when the communication radius is 90 meters. The reason is that quite a number of the BSW and the ACP users forward their copies to the destination so that there is no copy with them at 20 minutes, while the rest of them cannot forward their copies to the destination even given a large communication radius. While the number of the other two protocol's free phase queries is significantly influenced by the communication radius. The more queries are in the free phase, the more copies are in the network.

The Figure 4.20 shows the average number of queries which are carrying by users when the communication radius is 10, 50 and 90 meters. The curves of the ACP and the BSW rise sharply at the beginning and then become flat, while those of the MHLPP and the SLPD rise smoothly and continuously.

1. When communication radius is equal to 10 meters

1. When communication radius is equal to 30 meters

1. When communication radius is equal to 90 meters

Figure 4.20: Average Number of Carried Queries Per User

4.9.5 Distributing Appointment Cards

Exchanging appointment cards is a feature of the ACP, which imports burden into the network. We count the number of exchanging appointment cards per minute to evaluate the extra cost of the ACP.

In Figure 4.22, we count the total number of exchanging ACs in the whole network. For example, if a user Alice encounters another user Bob, the total of exchanging ACs processes increases by one when Alice exchanges any ACs to Bob. We count the number of those exchanging processes occur per minute. As shown in the figure, the exchanging processes do not occur frequently, but about 2 times per minutes. Since the size of an appointment card is a small, it does not cost the network many resources. At the same time, users can get many appointment cards to help them send queries, as shown in Figure 4.21. The number of ready ACs per user is raising smoothly and steadily.

Chapter 5

OMSN routing protocol simulator

The Opportunistic Networking Environment (ONE) [36] is a simulator designed for evaluating DTN routing and application protocols. Researchers can import real-world map into the simulator so that hosts walk along streets and roads in the simulator. The movement model of hosts can be defined by researchers, which enables researchers to evaluate the performance of their protocols in different scenarios. The User Interface of the ONE is also friendly, where researchers can observe the movement of all hosts. When a round of experiment ends, ONE outputs the results to a defined file. The simulator can also execute in console mode, which allows a higher experiment speed.

The ONE is an excellent simulator for DTN research, while it has its own weakness. The way that ONE estimates the distance between two hosts is not thorough, which may lead to inaccurate results. When a host broadcasts a message, only the hosts which are in its communication radius should receive the message. If the simulator cannot give the accurate set of hosts which can receive the message, the experiment result is not reliable. Although we can avoid this problem by setting parameters carefully, this problem cannot be completely ignored. Besides, some movement models need to calculate shortest path several times, which may require several seconds. Even though that is not a long time, it can slow down the experiment speed significantly as the total time of a round of experiment is about 10-20 seconds.

In this chapter, we introduce two major algorithms used in our new ORS simulator. The vertex reduce algorithm is used for simplifying the input of the all-pair shortest path algorithm, so that the simulator can initialize quickly. The finding nodes algorithm is used to find users which are in the communication radius of a given user.

5.1 Vertex Reduction Algorithm

Movement of network nodes is a significant factor in the performance of DTNs. To evaluate the performance of different DTN protocols, researchers have presented different kinds of movement models, like in [33]. Since shortest path algorithm is an essential and time-cost part of creating many movement models, it is an important aspect to optimize the time-cost for simulators, calculating the all-pairs shortest paths in the entire map.

The best known non-negative edge weight undirected map all-pairs shortest path algorithm [34] has a complexity $O\left(n^2 \log n\right)$, which is still expensive when n is huge. For most real-world city maps, there are thousands of points in a single square kilometer. Since a ten-square kilometers map is reasonable for a DTN protocol evaluation, a simulator must cut down the time spent in calculating all-pairs shortest paths to speed up the experiment.

Real-world map developers often use short straight lines to present curves, like in [35], so that a several-meters curves may contain tens of points. It is obvious that we do not need to calculate shortest path for every one of these points. In this chapter, we present an Vertex Reduction Algorithm (VRA) to reduce the number of points before the use of the shortest path algorithm. The basic idea is that VRA removes all points whose degrees are less than 3 from the map, while keeping the result of all-pairs shortest path algorithm correct.

The rest of this chapter is organized as follows. Section 5.1.1 presents some basic definitions and lemmas. The process of VRA is described in Section 5.1.2 and Section 5.1.3.

5.1.1 Ignorable Vertex and Reserved Vertex

Vertices in the graph are considered ignorable and/or reserved (definition follows). In each iteration of VRA, we remove the ignorable vertex from the graph, while the reserved vertices compose a new graph.

Definition 1: The vertex whose degree is larger than 2 is the *reserved* vertex (R).

Definition 2: The vertex whose degree is smaller than or equal to 2 is called the *ignorable* vertex (G).

Definition 3: If two reserved vertices (e.g. R_1, R_2) are connected by a sequence of ignorable vertices (e.g. $G_1, G_2, \cdots, G_n$), then the sequence from R_1 to R_2 (i.e., $R_1, G_1, G_2, \cdots, G_n, R_2$) is a *line-segment* (LS).

A reserved vertex can belong to different line-segments, while an ignorable vertex belongs to a unique line-segment. Both the reserved vertex and the ignorable vertex are called vertices (V)

Definition 4: The shortest route between two vertices inside a line segment is called the *inner shortest path* (SPI).

Let $\mathrm{SPI}\left(V_i, V_j\right)$ denote the inner shortest path between two vertices (i.e. V_i, V_j) inside a line segment.

Lemma 1: If two vertices (i.e. V_i, V_j; $i < j$) are in the same line-segment, the shortest path between them is

$$\mathrm{SP}\left(V_i, V_j\right) = \min\left\{SPI\left(V_i, V_j\right), SPI\left(R_1, V_i\right) + SPI\left(V_j, R_2\right) + SP\left(R_1, R_2\right)\right\}$$

Since an all-pairs SPI has a complexity equal to or smaller than $O\left(n\right)$ and the n here is much fewer than the number of points on the entire map, the time-cost for SPI is ignorable comparing to the time-cost of the entire map all-pairs shortest path calculation.

Lemma 2: We assume that there are two different line-segments (i.e., $\mathrm{L}S_i = \{R_{i1}, G_{i1}, G_{i2}, \cdots, G_{in}, R_{i2}\}$ and $\mathrm{L}S_j = \{R_{j1}, G_{j1}, G_{j2}, \cdots, G_{jm}, R_{j2}\}$). We pick a vertex V_i from LS_i and a vertex V_j from LS_j, then the shortest path between V_i and V_j is

$$SP\left(V_i, V_j\right) = min\left\{SP\left(V_i, R_{i1}, R_{j1}, V_j\right), SP\left(V_i, R_{i2}, R_{j1}, V_j\right), SP\left(V_i, R_{i1}, R_{j2}, V_j\right), SP\left(V_i, R_{i2}, R_{j2}, V_j\right)\right\}$$

where $\mathrm{SP}\left(V_i, R_{ia}, R_{jb}, V_j\right) = \mathrm{SP}\left(V_i, R_{ia}\right) + \mathrm{SP}\left(R_{ia}, R_{jb}\right) + \mathrm{SP}\left(R_{jb}, V_j\right)$. Here V_i and R_{ia} are in the same line-segment, so we can use Lemma 1 to calculate $\mathrm{SP}\left(V_i, R_{ia}\right)$, so does $\mathrm{SP}\left(R_{jb}, V_j\right)$. The $SP\left(R_{ia}, R_{jb}\right)$ is the only part we need to calculate using all-pair shortest path algorithms. We should notice that R_{ia} and R_{jb} could be the same vertex, but it does not make any difference to the lemma.

5.1.2 Vertex Reduction

VRA iteratively removes ignorable vertices from the graph until only reserved vertices remain. In each iteration, we remove all ignorable vertices from the graph but keep the edges. If there is more than one route between a pair of reserved vertices, we keep the shortest one and remove others.

5.1.2.1 Remove Ignorable Vertices

Since the degree of ignorable vertices is no more than 2, an ignorable vertex has only 0, 1 or 2 neighbours. In the case of 0 or 1 neighbour, we simply delete the ignorable vertex; if it has two neighbours, we connect its two neighbours before it is removed, as shown in Figure 5.1. When we remove an ignorable vertex, the weight of the line which connects its two neighbours is equal to the sum of its recent two lines' weights (i.e., $W_l + W_r$). After we remove all ignorable vertices in a line-segment, the two reserved vertices at the end of the line-segment are connected with a line directly, whose weight is the sum of all the intermediate ones' (i.e., $W_1 + W_2 + W_3 + W_4$).

5.1.2.2 Remove Redundant Connections

After we remove all ignorable vertices, all the reserved vertices are connected directly. However, it is possible that there are several connections between a pair of reserved vertices. Since the shortest route between a pair of neighbouring vertices makes other longer ones redundant, we remove all routes except the shortest one, as shown in Figure 5.2. We assume that W_2 is the shortest line among three connections, when W_1 and W_3 are removed.

5.1.2.3 Iterations

The process is shown in Algorithm 5-1. The input of the algorithm is the original entire map, which is called $Graph_0$. In the i^{th} iteration, VRA makes a copy of $Graph_i$ as $Graph_{i+1}$. If $i > 0$, the $Graph_i$ is the output of the previous (i.e., $(i-1)^{th}$) iteration. VRA removes all ignorable vertices in the $Graph_{i+1}$ as described in Section 5.1.2.1 and remove all unnecessary routes between reserved vertices as described in Section 5.1.2.2. In other words, $Graph_i$ gets smaller and smaller as i increases, because we always remove ignorable vertices from them. If $Graph_i$ is equal to $Graph_{i+1}$, which means that the i^{th} iteration makes no modification on the graph, the algorithm ends.

5.1.3 Assembling Vertices

- () () - () () We assume that the vertex reduction process stops at the i^{th} iteration. Then, the $Graph_i$ is the input of an all-pairs shortest path algorithm. Let $SP_i\left(R_u, R_v\right)$ denote the shortest path from R_u to R_v in $Graph_i$. We can infer the all-pairs shortest path of

Graph_{i-1} based on Graph_i with Lemma 1 and Lemma 2. The algorithm ends when we get the result of $Graph_0$. The algorithm is shown in Algorithm 5-2.

We start from the $Graph_{i-1}$ and assemble all intermediate graphs, until we get the result of $Graph_0$. For any intermediate $Graph_j$, we already have the all-pairs shortest path result of its reserved vertices in the previous iteration (i.e. $Graph_{j+1}$). Then, we calculate the shortest path from any ignorable vertex to all others. When the two vertices are in the same line-segment, we use Lemma 1 to calculate their shortest path. The Lemma 1 includes four parts: $SPI\left(V_i, V_j\right)$, $SPI\left(R_1, V_i\right)$, $SPI\left(V_j, R_2\right)$ and $SP\left(R_1, R_2\right)$. Since we already have the result of $SP\left(R_1, R_2\right)$ in the previous iteration, we just need to calculate the remaining three SPI parts, which is easy and has fewer nodes. If the two vertices are in different line-segments, we use Lemma 2 to calculate the shortest path. We already have all the SP parts from the previous iteration, so we just need to deal with their respective SPI parts.

5.2 Finding Nodes in a Range

We assume that there are n nodes in a planar map (size: $h?w$). Those nodes keep moving on the map slower than a speed threshold (S). If the distance between two nodes a and b is smaller than a radius threshold (r), they are considered connected. We want to calculate all-pairs connections for all nodes on the map periodically.

If we calculate these connections directly, the complexity is $\mathrm{O}\left(\frac{n^2}{2}\right)$. We assume that $r \ll min\left\{h, w\right\}$, and nodes are distributed on the entire map symmetrically. Then, the expected number of nodes which has a connection with a node (v) is $\frac{n}{h?w}?\pi r^2 \ll n$. If we traverse all other $(n-1)$ nodes of v, it will be a waste of time.

We propose an algorithm to optimize the calculation of all-pairs connections. We draw grids whose size is $l?l$ on the map. When we calculate connections of the node v, only nodes in surrounding grids will be traversed.

5.2.1 Static Nodes

If nodes on the map are stationary, the radius threshold r is the only factor of the selection of the grid size ($l?l$). The length l of the side of a grid should be equal to r, as shown in Figure 5.3. For any node v in the dark grid, all nodes whose distance between them is equal to or shorter than r should be in the grey or dark grid. Therefore, if we want to get all connected nodes to node v, we simply need to traverse all nodes in the grey and dark grids instead of all nodes in the entire map. In this case, the expected number of nodes which we need to traverse is $\frac{9n?r^2}{h?w}$ for checking connections for a node v. If we calculate connections for all nodes, we simply need to get the distances of about $\frac{9n^2?r^2}{2?h?w}$ pairs of nodes, instead of $\frac{n^2}{2}$ pairs.

5.2.2 Mobile Nodes

The problem is a bit more complicated if nodes are moving. Assuming that we want to calculate all-pairs connections at t_1, first, we draw grids at time t_0, so that we know which grid is a node in at t_0. Instead of t_0, we want to get connections at t_1. In this case, the length l might not equal to the radius threshold r.

Suppose that we have two nodes u and v, and their distance is d_1 at t_1. Since the maximum speed of nodes is S, their distance d_0 at t_0 satisfies

$$max\left\{0, d_1 - 2\delta\right\} \leq d_0 \leq d_1 + 2\delta$$

where $\delta = S\left|t_1 - t_0\right|$. When $d_1 \leq r$, these two nodes are connected at t_1, then d_0 must satisfy $0 \leq d_0 \leq r + 2\delta$. If they are in the same or adjacent grids at t_0, as shown in Figure 5.4. We have $l = r + 2\delta$, so that $\forall d_0 \in [0, r + 2\delta]$: $d_0 \leq l$. In other words, there is no grid between two nodes at t_0.

5.2.3 Complexity

The complexity of drawing grids is $O(n)$, while that of checking connections is O $\left(\frac{9n^2?l^2}{2?h?w}\right)$, so the checking part is much more expensive than the drawing part. Since $l = r + 2\delta = r + 2S\left|t_1 - t_0\right|$, the checking part achieves its best performance when $t_1 - t_0 = 0$, then its complexity is O $\left(\frac{9n^2?r^2}{2?h?w}\right)$. That is equal to the complexity when nodes are stationary.

Chapter 6

Conclusion and Future Work

In this chapter, we will summarize our findings and discuss the future directions of the proposed work in the previous chapters yet to be explored.

6.1 Summary of Work Done

In this thesis, we proposed and analyzed two new location-privacy preserving protocols, namely MSLPP and ACP, besides developing a new OMSN simulator, called ORS, as a platform for evaluating the proposed protocols and comparing them with their counterparts.

Our first protocol, MSLPP, is a distributed location-privacy preserving protocol using social-relationship and encryption. Simulation results show that it has a better performance on delivery success ratio and provides an acceptable obfuscation compared to its counterpart HSLPO which is an improvement of their earlier scheme SLPD. Compared to HSLPO, our scheme MHLPP achieves an in increase in success ratio between 6% and 11% for the same range of parameters considered by the authors of SLPD. The success ratios of both HSLPO and MHLPP decline when the privacy threshold is increased, while the success ratio of HSLPO is always lower than that of MHLPP and decreases more sharply than that of MHLPP. Although HSLPO is slightly more secure than MHLPP based on our experiments which means the original requester has a lower rate to be located, but the entropy difference is quite small and negligible. Encryption and decryption to enhance security are added cost for MHLPP which is acceptable. Our experiment (using the map of Helsinki) presents that in a social network with 126 users, the average number of times of a query is encrypted and decrypted is less than 3 when the privacy threshold is considered as 85.

Our second location privacy-preserving protocol, ACP, also uses social-relationship. It facilitates the obfuscation process by continuously exchanging appointment cards among users so that the original requester does not communicate with any of his friends when he sends a query. Simulation results show that it has a better performance with respect to the query-delivery success ratio and provides an acceptable obfuscation compared to its counterparts. Although ACP requires more time to forward the reply, the total time required for query delivery and reply delivery is still less than that for its counterparts SLPD and MHLPP. The major disadvantage is that the ACP must exchange ACs continuously, which can consume network resources. However, that cost is low if users do not send too many queries.

To facilitate our experimentation, we have developed a simulator, called ORS, which is more suitable for OMSN simulation than the existing simulator, ONE. The ORS is more reliable and efficient than ONE for simulating opportunistic social networks.

6.2 Future Work

In MHLPP, we just considered the relationship strength between two connected people in the social network; in other words, an agent only gives the message to his direct friend. In most of the cases, the friends of your friends can also be trusted and be considered. If these friends can participate in forwarding queries, agents can have more choice in the obfuscation phase.

In ACP, the sizes of agents' relay tables are significantly affected by the number of appointment cards passing by them. If we can decrease the number of appointment cards in the network, the burden of agents can also decline. For example, a single appointment card can be shared among friends instead of being used by only one user.

References

[1] L. J. Chen, C. H. Yu, T. Sun, Y. C. Chen, and H. H. Chu. A hybrid routing approach for opportunistic networks. In *the 2006 SIGCOMM workshop on Challenged networks (CHANTS)*, pages 213–220, September 2006.

[2] M. Duckham. Moving forward: Location privacy and location awareness. In *the 3rd ACM International Workshop on Security and Privacy in GIS and LBS*. ACM, November 2010.

[3] K. Fall. A delay-tolerant network architecture for challenged internets. In *the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 27–34, 2003.

[4] C. Chow J. Zhang. Real: A reciprocal protocol for location privacy in wireless sensor networks. *IEEE Transactions on Dependable and Secure Computing*, 12(4):458–471, October 2014.

[5] E. P. Jones and P. A. Ward. Routing strategies for delay-tolerant networks. *ACM CCR*, 2006.

[6] E. P. C. Jones, L. Li, and J. K. Schmidtke. Practical routing in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 6(8):943 – 959, June 2007.

[7] A. Keränen, J. Ott, and T. Kärkkäinen. The one simulator for dtn protocol evaluation.

In *the 2nd International Conference on Simulation Tools and Techniques*. ICST, March 2009.

[8] John Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, August 2009.

[9] U. Lee, S. Y. Oh, K. W. Lee, and M. Gerla. Relay cast: scalable multicast routing in delay tolerant networks. In *IEEE International Conference on Network Protocols*, pages 218–227, October 2008.

[10] Y. Liao, K. Tan, Z. Zhang, and L. Gao. Estimation based erasure coding routing in delay tolerant networks. In *the 2006 international conference on Wireless communications and mobile computing (IWCMC)*, pages 557–562, July 2006.

[11] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, July 2003.

[12] C. Liu and J. Wu. Scalable routing in delay tolerant networks. In *the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 51–60. ACM, 2007.

[13] R. Lu, X. Lin, and X. Shen. Spring: A social-based privacy-preserving packet forwarding protocol for vehicular delay tolerant networks. In *2010 Proceedings IEEE INFOCOM*. IEEE, March 2010.

[14] D. Grunwald M. Gruteser. Anonymous usage of location-based services through spatial and temporal cloaking. In *the 1st International Conference on Mobile Systems, Applications and Services*, pages 31–42, May 2003.

[15] M. Mano and Y. Ishikawa. Anonymizing user location and profile information for

privacy-aware mobile services. In *the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, pages 68–75. ACM, November 2010.

[16] C. Mascolo, S. Hailes, L. Lymberopoulos, G. P. Picco, P. Costa, G. Blair, P. Okanda, T. Sivaharan, W. Fritsche, M. Karl, M. A. Rónai, K. Fodor, and A. Boulis. Reconfigurable ubiquitous networked embedded systems. Technical report, January 2005.

[17] Pietiläinen and A.K. *Opportunistic Mobile Social Networks at Work*. PhD thesis, Université Pierre et Marie Curie, Paris, 2010.

[18] E. Reddy, S. Kumar, N. Rollings, and R. Chandra. Mobile application for dengue fever monitoring and tracking via gps: Case study for fiji. Technical Report TR-04-2015, Mar 2015.

[19] J. Schiller and A. Voisard. *Location-Based Services*. Morgan Kaufmann, 2004.

[20] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *the ACM SIGCOMM Workshop on Delay-tolerant Networking*, pages 252–259. ACM, August 2005.

[21] L. Liu T. Wang. Privacy-aware mobile services over road networks. *the VLDB Endowment*, 2(1):1042–1053, August 2009.

[22] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, April 2000.

[23] R. Xiang, J. Neville, and M. Rogati. Modeling relationship strength in online social networks. In *the 19th International Conference on World Wide Web*, pages 644–654. ACM, April 2010.

[24] X. Xue, J. Li, Y. Cao, and J. Fang. Advanced prophet routing in delay tolerant network. In *IEEE International Conference on Communication Software and Networks*. IEEE, 2009.

[25] S. Zakhary and M. Radenkovic. Utilizing social links for location privacy in opportunistic delay-tolerant networks. In *International Conference on Communications (ICC)*. IEEE, June 2012.

[26] S. Zakhary, M. Radenkovic, and A. Benslimane. The quest for location-privacy in opportunistic mobile social networks. In *Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, July 2013.

[27] S. Zakhary, M. Radenkovic, and A. Benslimane. Efficient location privacy-aware forwarding in opportunistic mobile networks. *IEEE Transactions on Vehicular Technology*, 63(2):893–906, February 2014.

[28] M. M. Zanjireh and H. Larijani. A survey on centralised and distributed clustering routing algorithms for wsns. In *Vehicular Technology Conference (VTC Spring)*. IEEE, May 2015.