Privacy aware P2P Friend Locator

Ove Andersen
xcalibur@cs.aau.dk

Aalborg University
Department of Computer Science
Selma Lagerløfs Vej 300, DK-9220 Aalborg Ø
Denmark

Abstract

A friend locator is a location-based service that is used to detect proximity of when two users are within a user defined distance of each other. When such a service is to be developed for mobile users, two main topics needs to be taken into account. First of all, privacy is a very essential element. The users must not risk revealing personal information about their locations and whereabouts. Second, communication costs is still a very essential topic, when considering mobile devices connected to the Internet.

To meet these two general requirements, this paper presents a peer-to-peer location-based service, that is optimized both for privacy and for minimizing communication costs. Three main techniques are employed to reach this goal, namely cloaked regions, dynamic shifted circles and secure multi-party computations. This has ended up with a solution, where a user only communicates with the friends it wishes to detect proximity with, and all a user knows about a friend is, how for away the friend is, with some degree of imprecision. The user does not know in which direction the friend is located, thus it is hard to guess where the friend is located. To reduce communication costs, buffer zones for each user is generated, that allow the user to move freely around within this zone without having to perform any location updates. The solution is flexible, due to proximity distance and precision can be adjusted for each pair of friends. The design has been implemented in a prototype, and tests have been performed to show that it is communicational wise effective.

1 Introduction

Positioning technology today is a widely used term and is used in many different contexts and applications. When talking about positioning capabilities, people often think of the GPS system, but in fact, many other technologies are used, like Bluetooth or Wi-Fi. In common for these technologies are, that the receivers are getting continuously smaller and cheaper, thus they are implemented in even more and smaller devices. Today a chip, which is fully capable of receiving and translating signals from GPS satellites, does not need to measure mere than 3.6x2.4x0.6 mm. in size and use as little as 15 mW when in use[1]. At this size, it is no wonder why such technologies are being featured in a broad range of mobile products like PDA's, cell phones, smart phones, and so on.

Communicating with the world, while on the move, is a trend that has been going on for some time. It has been popular to use mail and surf the Internet while being away from a permanent location, but due to high speed mobile communication, like 3G and the enhancements of this, an entire new marked has been opened. Devices, such as Apples IPhone and devices using Google's software platform, Android, are intended to be connected to the Internet all the time, and other platforms are following in the same direction. Features, such as streaming video and music, using widgets for fetching weather forecasts, stock rates, or latest news are examples of such services, that usually makes use of an Internet connection and has become very popular among users of mobile devices.

When we are in possession of both a positioning technology and we have the possibility of communicating with the world, everything needed for a Location-Based Service (LBS) is present. A LBS is a service that, using a user's location, can provide convenient information to the user while on the move. A user on the move, or a mobile user, might want to know where the nearest shopping mall is, how far away the next gas station is, or if any other users are within a certain distance. For a user, a LBS is a simple service, that

[1] Specifications of the NXP Semiconductor GNS7560 GPS chip

can provide exactly such information wherever they are and whenever they need it.

Many different LBS' exists for dealing with various types of problems, but not much work has been done when considering proximity detection between friends. Some projects do though exist, for example Google Latitude [3] which gives the ability of telling other where you currently are located and [24], which can tell if two friends are within some proximity distance. Both projects makes use of a server, to which the users send some information about their location.

On the more technical side, a LBS' can be categorized into two different main types of architectures; Peer-to-peer and centralized. If a service is centralized, then a centralized server is used as some kind of gathering point for the users, At some times, at least, they will communicate with a server that is used to provide and/or keep track of some information. This server could for example keep track of all the users using a service and could tell whether they are close to any other users they might know. The advantage of this type of service is, the data could be stored a central place, heavy calculations could be put onto the server, and it might also help reducing the communication costs of a LBS. A great disadvantage of the centralized method is, it can be hard to guarantee the trustworthiness of the server. If the server gets corrupted by an adversary, everything it receives could be disclosed. This might be the location of the users, account information, or who are friends of a user. Thus, when working with centralized LBS' a lot of work has to be put into guaranteeing the privacy of the users sensitive information, they might share with the service.

The other type of architecture, without a centralized server, is called a peer-to-peer solution, P2P. P2P denotes, that all communication takes place in between the users of the LBS only. No external servers are involved, thus the problems of guaranteeing the trustworthiness of the centralized model is not present here. All data is kept on the users themselves, and each user decides which other users they trust their sensitive data to. One disadvantage of this architecture is, that distributing data between several users can be more expensive communicational wise. Also, there is no centralized data storage, thus this method is only usable for some LBS'. E.g. it would probably not be the preferred architecture for a LBS that generate statistics over multiple users patterns, or when the next bus leaves from the station. For such kinds of services a centralized data storage is probably preferred.

When considering the application of getting to know when a user is within proximity, both architectures are usable. The P2P way has the advantage, that each user decides which other users it trusts and will share information about its location with. When two users trust each other, the are said to be friends, and when two friends wants to know whether they are within proximity, they do not need to involve anybody else about this friendship. But even though the users are friends and they trust each other, they do not necessarily want to disclose everything about their location to each other. Maybe a users device could get compromised or stolen, and the friends should not be in risk, just because they were friend of an unlucky user. Thus privacy also needs to be taken into account in a P2P LBS.

Many solutions for preserving privacy in LBS' exists, but most concerns are related to centralized LBS'. Either they concerns querying public data (such as a gas station, which does not risk anything by revealing its location) or they concern hiding the users location from the centralized server itself. To solve this problem, using a P2P architecture, an algorithm is presented, that preserves the privacy of all the users, while two friends still gets to know when they are within proximity. The strength of this algorithm is, that even though one user gets malicious, it cannot get to know where its friends are located, and it would have no impact on the performance or quality of the service. Because the solution is meant to be deployed on mobile devices, communication costs must be considered as significant. If communication costs was not an issue, the friends could calculate the distance between them all the time, and they would know exactly when they were within proximity. Thus a method for reducing the number of messages is needed for the system to be practically usable.

The motivation for this paper is to find a way of making a P2P solution for a friend locator service, that addresses the problems of both preserving privacy and reducing communication costs. By preserving privacy is meant, that as little information as possible about a user should be revealed to the other users. The information, that a user want to hide in this context is its exact location.

In Related Work, section 2, three different topics are discussed, and what other work has been done in these areas. First of all, location privacy in LBS' are introduced, where especially cloaking techniques are touched. Second, proximity detection is discussed, where several techniques are introduced and is described why they are not directly applicable for this work. At last the topic of secure multi-part computations is introduced, which could help improve privacy of this service, by hiding in which direction the friends are located.

In this paper, a P2P solution is presented, that informs two friends when they are within proximity. All communication goes on directly between the users, and no external servers are used at all. For the users to preserve privacy, they use a location within a cloaked region instead for their exact location, along with methods for hiding their actual location. The only data known to a friend of a user is, how far away they are from each other, without revealing in which direction. To optimize communication costs, users will be moving within buffer zones, where they do not need to do any location updates. In order to prove that the service works and is efficient, experiments will be performed on a model of the system.

Section 2 will cover what work has been done, related to this project, and Section 3 will define the problem definition and introduce some notations for later use. Section 4 describes the ideas and theory for how the problems will be solved, and Section 5 will show how the system is to be practically designed. Section 6 will cover what experiments have been performed on the implemented model and explain the results, while at last Section 7 will conclude what has been achieved with this paper.

2 Related Work

For this work, three sets of studies are relevant; location privacy, proximity detection, and secure multi-party computation. The studies of location privacy is the study of how privacy can be obtained in a system, where some sensitive information must be shared. Proximity detection is used to reduce the communication costs of the system and determine whether two users are within proximity or not, and secure multi-party computation is the study which is used to perform calculations on private data between several users.

Location Privacy When revealing information about a persons location to other services or people, location privacy is a very essential subject. According to [12, 18, 23], GPS devices have been used for stalking people in many cases. This ranges from surveiling elderly people to stalking ex-girlfriends, and even the Police are known to use GPS devices for stalking people. The seriousness of GPS stalking can be seen by the usage of GPS in police actions in the USA now is being questioned in court [11].

Many spatial cloaking algorithms have been proposed, such as [5, 6, 8, 10, 15, 16, 19, 26]. The general idea of a cloaking algorithm is, to hide a user's location from other users of a service, or from the service itself. The approaches differs in the way they do this. One way is to obfuscates a user's location among other users, say the cloaking region for a user contains k other users. This approach is called $k - anonymity$ and is widely used in many approaches, but the disadvantage of this method is, that for a user to know how large the cloaking region must be, it needs to know where at least k other users are located. Therefore this is often done in a secure third-party server, called an anonymizer, which is not available in a entirely P2P solution. Therefore the simplest way a user can hide it's exact location is, to generate a cloaking region of a desired size, and use a random location within this region as its location. Thus the other users does not know where the user is exactly located, only to some degree of uncertainty.

Proximity detection Proximity detection is the ability of tracking users movements relevant to each other and reducing communication costs by reducing the number of location updates a user needs to do. This is done by making some rules for when a user must submit its location and when it is not needed. Say, if the users is moving within a very small area, this might not be interesting for some applications, and therefore the user does not need to perform a location update, thus the communication cost has been reduces.

Many studies have been performed on proximity and separation detection among mobile users in 2D space, [1, 13, 20, 21]. When considering mobile users, proximity detection is especially relevant due to that mobile clients are often reduced by low bandwidth and high communication price.

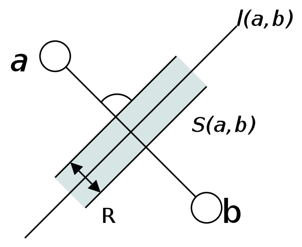

Figure 1. The strips strategy

The strips strategy, proposed in [1, 13], is an algorithm that generates a strip between two users to reduce communication costs. This is practically done, shown in Figure 1, by calculating the Euclidean distance, denoted —*a*-*b*—, between between the two users, *a* and *b*. In the middle between the two users, $\epsilon(a, b)$, a strip is made $S(a, b)$ with a center axis $I(a, b)$ and the width R. As long as none of the users enters or crosses the strip, $S(a, b)$, no location updates will be performed between these two users. The width R of the strip is a buffer zone, which is needed to ensure that the users cannot stand next to each other, being in proximity, on each side of the strip without knowing.

Another approach is the dynamic centered circles strategy, proposed in [13, 20], which generates a circle around each user, and as long as the user is moving within this circle, no updates are being performed. As shown in 2, user A, B, and C have reported their location to a centralized server. The actual locations of the users are marked by x and the center of their circles marked by a ring and each user generates a new circle only, when they get outside their own. In this figure, user A has crossed its circle and has just generated a new circle. This is done with respect to the circles that the other users are currently using, and they will not know anything about the user now generates a new circle. The circle is optimized to be the largest possible, but it has to be p away from the other users circles, to ensure

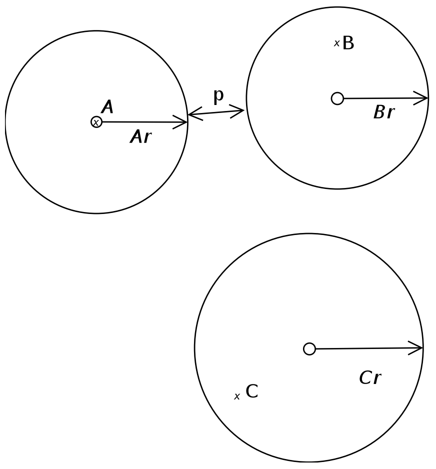

Figure 2. The dynamic centered circles strategy

that proximity will be detected, if the users gets closer than p. Therefore, it is user B's circle, that limits the size of A's new circle.

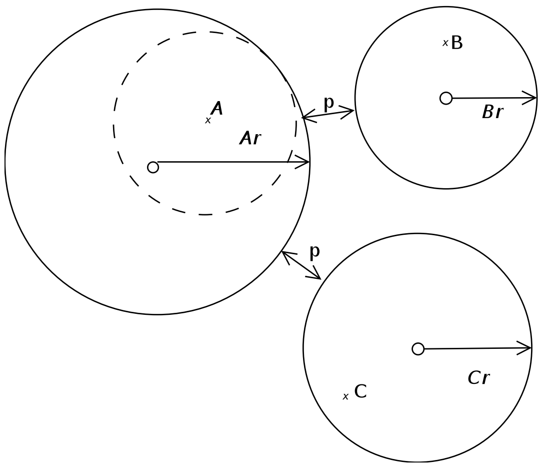

Figure 3. The dynamic shifted circles strategy

An extension of the dynamic centered circles strategy is the dynamic shifted circles strategy, [13], which shifts away the center of the circle from the users location. This is done to optimize the size of the circle, as shown in Figure 3, where A's center of its newly generated circle is shifted away from A's actual location. The centered circle is showed by a dashed line, and this must entirely be covered by the shifted circle. Now the circle is optimized to be only p away from both B and C, and the area the circle covers is improved.

The dynamic centered circles and dynamic shifted circles strategy are both intended to work with a centralized server. This is due to, that each user only has one circle, and to generate a users circle, some information of all the other users are needed. If this is to be done on the users side, then the user must know all the other users circles. This strips algorithm is on the other hand calculated between each pair of users, and this can be done fairly easy on the client. The downside of this method is, that if this is implemented in a P2P system, each users must know in which direction all the other users are along with how far away they are. So actually each user must know the precise location of the other users according to its own location. This is not optimal if location privacy is a concern in the system.

Secure Multi-party Computation Secure Multi-party Computation (SMC) is a study initially started in 1982 by Andrew C. Yao, when he introduced the millionaire problem in [27]. In general terms, the millionaire problem describes, how two millionaires, Alice and Bob, can find out who is richer without revealing their wealth to each other. So SMC is in this case used to calculate and reveal which is richest and ensure that neither Alice nor Bob gets to know anything about the others wealth.

In general SMC can be said to be the computational replacement of a trusted third party. Alice and Bob could have told a trusted person their wealth and the person could then have done the computations and revealed the result. SMC has since 1982 been extended by many, [2, 7, 9, 14, 25, 28], to be usable for a lot of different areas.

In the area of geometry, SMC has also found its place. When talking about privacy in LBS, several studies have been performed which makes use of SMC when working with objects in 2D spaces. [25] has proposed a solution that can detect collisions between two moving circles. This could be a usable solution in a system where two friends wants to know when they are within proximity. The problems is that this solution is not optimized for communication and therefore it might not be the optimal solution for a mobile service.

Other studies, such as [2, 14], considers the relations between different shapes in a geometric 2D space, such as the relation between a private point and a circle area, the relation between a private point and an ellipse area, the relation between private sets of points, and most interesting, the distance between two private points. The distance between two private points, proposed by [14], is interesting, because, if two users only knows the distance between them, they do not know in which direction each other is located.

This is depicted in figure 4, where user u only knows, that user v is $\epsilon(u, v)$ away, but not in which direction. This is depicted, by four example locations of v, but actually, v could be anywhere on the arc of the circle. This will help improving privacy, because it helps hiding the other users locations.

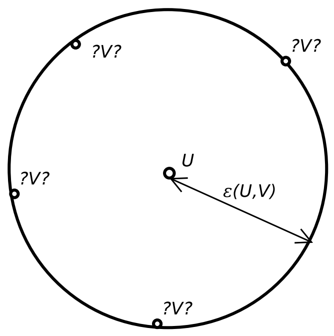

Figure 4. What u knows about v's location when direction is hidden.

3 Problem Definition

For this work, a large number of mobile users are assumed present. These users must all be equipped with some positioning technology and have have the ability of exchanging data in between each other. The solution is completely P2P and does not require or make use of any kind of third party server.

For all the users, each pair of users can either be mutual friends or not friends at all, i.e. if some user u is friend with user v then user v is also friend with user u, but user u cannot be friend with user v if v is not a friend of user u. When two users are friends, it overall means that both users wants to be informed when another is within some proximity distance. More specific it also means that they trust each other and they are willing to reveal some sensitive information about their location to each other, thus they must have some trust to each other.

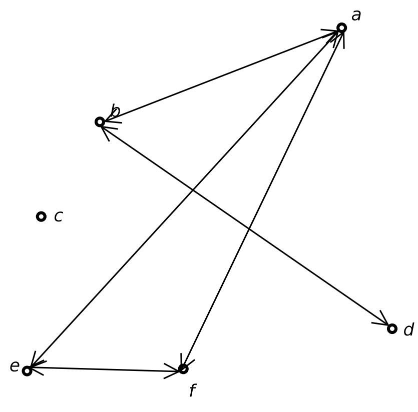

Figure 5. An overview of a sample system, with users a through f and their friendships.

Figure 5 shows an example system with the users a-f and how they are related in friendships. Each line indicates that there is a friendship between a pair of users, and the arrows indicates that the friendship is mutual. It is displayed, that user a is mutual friend with the users b, e, and f, but that does not mean that the friends of a user has to be friends with each other too. It is also possible, that a users might be using the system, but none of its friends are available, shown by user c.

For each pair of friends, say user a and b, both users must have agreed on a mutual distance for when they wants to be informed of proximity. The mutual proximity distance is denoted as $prox(a,b)$ and the actual Euclidean distance between the two users is denoted as $dist(a,b)$. To introduce some freedom in the service, β is used as a buffer variable. This will help the service to reduce the number of proximity messages that might emerge, if two users are stepping in and out of proximity. Therefore, the following rules are applicable to user a:

1. If $dist(a,b) \leq prox(a,b)$, user a will be notified it is within proximity of user b, and vise versa.
2. If $prox(a,b) \leq dist(a,b) < prox(a,b) + \beta$, user a might be notified it is within proximity of user b, and vise versa.
3. If $prox(a,b) + \beta \leq dist(a,b)$, user a will be notified it is not within proximity of user b, and vise versa.

To make the system as flexible as possible, the β variable is not constant, but can be different between each pair of friends. This means, if a pair of friends requires absolute precision, they can eliminate β and they will know exactly when they are within proximity. Then the following rules are applicable for user a:

1. If $dist(a,b) \leq prox(a,b)$, user a will be notified it is within proximity of user b, and vise versa.
2. If $dist(a,b) > prox(a,b)$, user a will be notified it is not within proximity of user b, and vise versa.

For the service to be practically usable and desired by users, some requirements must be set. First of all, the service must include some privacy features, that can help guaranteeing the privacy of the users. Second, the users does not want to reveal anything to other users of they system they do not know. This is fairly easy to obtain, due to the P2P architecture, which makes it possible for the user itself to decide which users it wants to communicate with. When concerning the friends of a user, the user might not want to disclose everything about its location either, thus some techniques for preserving some location privacy must be adapted or developed. When talking about a mobile service, some optimizations might be necessary. Both computational capabilities and communicational wise the devices might be limited. Thus the service must be effective and not overcome the capabilities of the device, but since users might have different requirements to precision, the precision must be variable between each pair of friends, along with the desired proximity distance..

Using these properties, it is possible to develop a system, that is both effective, optimized for mobile terminals, preserves user's location privacy, and is flexible, due to the possibility of changing resolution of precision and desired proximity distance.

4 The friend locator idea

Motivated by having a friend finder solution for mobile users, without the security issues of having a third-party host, a P2P solution is developed. First of all the general idea is introduced and some characteristics of the system is described. Then a method for hiding a user's location within a cloaked region is described, a method for improving the privacy of a user is discussed and at last, proximity detection along with communication optimization is presented. Table 1 covers some general notations used in this section, and the following properties applies to the system and the solution:

System: The systems exists of mobile users only. No external hosts or services are available.

Friends: The system exists of many users. If two users want to detect proximity between them, they must be mutual friends, like u and v.

Communication: All pair of mutual friends will communicate directly between each other. If two users are not friends, they will not communicate at all.

Proximity: A pair of friends will want to know when they are within some defined mutual proximity distance of each other.

Privacy: Due to privacy issues, none of the friends want to reveal their actual location, only how far away from each other they are to some degree of precision.

Flexibility: The uncertainty distance for when two friends might be within proximity can be changed. This will balance precision of the service against communication costs.

Throughout the entire section, two friends (users with mutual friendship) are used as example, namely user u and v. All users are moving around on a 2D map of the world covered by a homogeneous X, Y co-ordinate system.

Notation	Meaning
u	A user in the system. When using u as a location, u's exact location is used.
$C(u)$	User u's private cloaking region. This region is used to pick a random location for a user to use instead of a user's actual location.
$C_r(u)$	The private radius of user u's cloaking region, $C(u)$. The center must be in the user's actual location, u.
u_p	User u's submitted location, which must be within the cloaking region, $C(u)$.
$B(u)$	User u's private buffer zone. This zone is an area, where the user can move within without having to submit any updates.
$B_r(u)$	The private radius of user u's buffer zone, $B(u)$. The center must be in a user's submission point, u_p
$\epsilon(a, b)$	The Euclidean distance between two locations. A location can either be a user's exact location, u, or a users submitted cloaked location, u_p.
$\theta(u, v)$	The distance for when the two friends u and v wants to be informed of proximity.
α	The precision of proximity detection.

Table 1. Table of Notations

4.1 Hiding location using cloaking regions

When two friends want to find out, whether they are within proximity or not, they will need to find out how far away from each other they are. This could easily be done, as depicted in Figure 6, by one of the two friends, say v, submits its exact location to u. Then u would be able to calculate the exact distance between the two users, and they would know if they are within proximity. However, if they want to gain some privacy, this method is not very fit to use. Nothing is hidden from the friends and all friends of v would at any time be aware of its location.

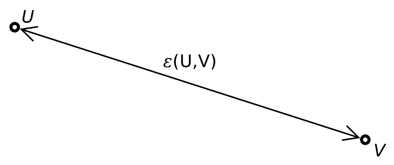

Figure 6. The simplest solution

Figure 7 shows how v could use an imprecise location point, when u wants to calculate the distance to v. Instead of v submitting its exact location, it could submit any point within a private cloaking region. The cloaking region,

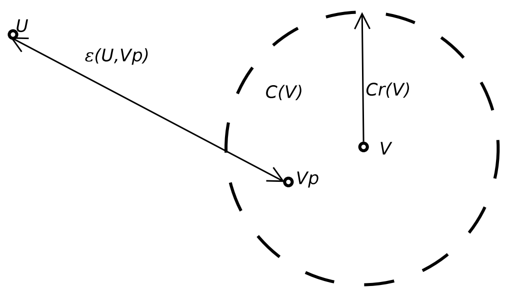

Figure 7. User v uses v_p as location instead of v

$C(v)$, is a private region with a radius $C_r(v)$, which is only known to v. This cloaking region is used to choose a random imprecise location, instead of using v's exact location, to be submitted to u. This random location is described as v_p and could be located anywhere within $C(v)$ with radius $C_r(v)$ and the center in v's exact location. The advantage of this secret cloaking region is, that u now does not know v's exact location, but only v_p. And since u does not know anything about v's $C(v)$, u cannot for sure know where v_p is located compared to v, thus v's locations has been cloaked.

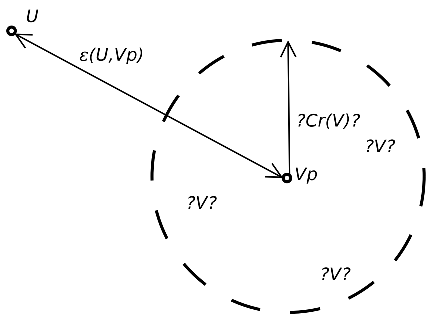

Figure 8. What u knows from what v reveals

An example of what u knows of user v's location is depicted in Figure 8. Both users have generated a cloaked region, i.e. $Cr(u)$, Since v has submitted v_p to u, it can only know how far away, with some uncertainty v is. The figure shows what variables u knows. Variables unknown to u is surrounded by ?, like $?C_r(v)?$, and areas unknown to u are surrounded by a dashed line. Since $?C_r(v)?$ is private to v, u cannot know how large $C(v)$ is nor where the center is located, therefore v could be anywhere. This is depicted by three example $?v?$, as examples of where v could be. Since u has used its exact location to calculate the distance, $\epsilon(u, v_p)$, it knows approximately how far away v is, but it can never know the precise distance, due to v_p, which is the downside of this method. The price for some privacy is the risk of inaccurate results, when users get close to proximity. This do of cause depend on the size of the cloaked region, $Cr(v)$ - the larger cloaked region, the worse the guarantees gets.

4.2 Hidden direction using secure multi-party computations

Even though a cloaked regions hides a users exact location within an area, friends of the user still have a good idea about where the user is located. The only parameter required for the system to be usable is, that the friends must know something about how far away they are from each other. Therefore, if it is possible to hide everything except for what is required for the service to run, privacy would be optimized for the job.

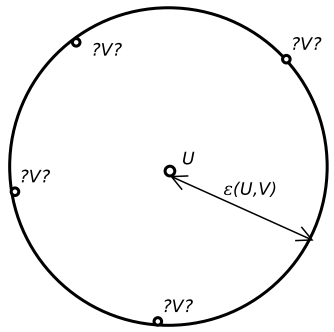

Figure 9. What u knows about v's location when direction is hidden.

Secure multi-party computations (SMC) is a way of doing computations on multiple sets of sensitive data and getting a result without revealing anything about the sensitive data to the different parties. In [14], a method for calculating the distance between two private locations have been developed. That means, say u and v, can find out how far away from each other they are, without revealing anything about their actual locations. This is shown in Figure 9. Here user u and v's exact locations are used, and the figure shows what user u knows of user v's location, namely the distance between u and v, $\epsilon(u, v)$. Therefore u only knows that v is located somewhere on the arc of the circle, but not in which direction. This is depicted by four possible locations, $?v?$, for v.

The algorithm proposed in [14] calculates the distance between two private points and it is functional and effective in terms of messages, therefore this will be used as a basis for hiding the users direction from friends. But due to the SMC, it is a bit more communication wise expensive, so when u has calculated the distance to v, u could send this to v instead of letting v do the same calculations.

If this SMC approach is being applied, then each user will not know in which direction the friends are, but it will know exactly how far away they are. This might be a problem, because if a user compares this arc, where a friend is located on, to a map, there might be very few possible locations where the friend actually could be located. This is

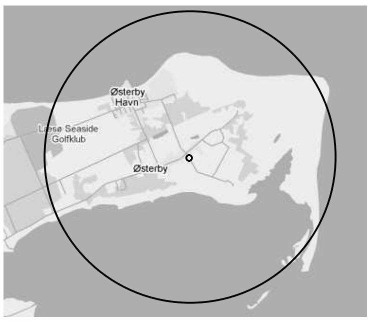

Figure 10. Example of where a friend could be located, compared to a user.

shown in Figure 10, where a user knows that a friend is on the arc of the circle somewhere. Most of the arc is located in the water, thus there is a great possibility that the friend is in this small band that is located on land, and it might be easy to guess the friends exact location. To help this problem, the cloaking region comes in handy. This will help hiding the friends actual location and introduce some uncertainty.

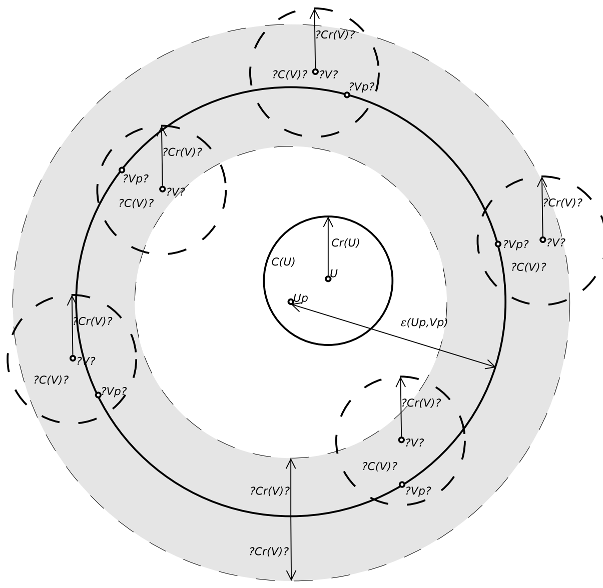

Figure 11. What u knows about v's location when using cloaking region along with hidden direction.

To ensure privacy of the two friends, neither u nor v uses their exact location for the SMC calculation, but instead u_p and v_p, as shown in Figure 11. These locations are picked randomly within their respectively cloaking regions, i.e. $Cr(u)$. Now the distance between the two submission points is known to the two friends, and it is calculated to be $\epsilon(u_p, v_p)$, and u only knows, that v_p is somewhere on the arc of the depicted full line circle. Five example locations of v_p is shown, with examples of how the entire cloaking region of v, $Cr(v)$ could be. All unknown variables to u is again surrounded by ? and unknown areas surrounded by dashed lines. Thus, the only thing u know of v's location is, that v must be within the gray area, which is size $2{*}?C_r(v)?$ - and because $?C_r(v)?$ is unknown to u, the width of the gray area is also unknown to u.

4.3 Proximity detection and bandwidth reduction

When two users are moving in the system, some techniques needs to be deployed to detect whether they are within proximity or not. To ensure this, we need to detect both proximity and separation of each pair of friends. Proximity detection is used to detect when two users get within some user defined proximity distance, $\theta(u, v)$, and separation detection is needed to detect when the two users get further away from each other than $\theta(u, v)$. Another subject that needs to be taken into account is bandwidth optimizations. When two friends, u and v, are moving around on a 2D map, the distance between them, $\epsilon(u, v)$, will be constantly changing. It could be tempting to continuously calculate the distance between the two users and check if they get within proximity. But this approach would be very expensive, both computation and communication wise. Each time one of the users would update it's location, the $\epsilon(u, v)$ from before would be obsolete, and they would need to calculate a new $\epsilon(u, v)$ to ensure proximity or separation.

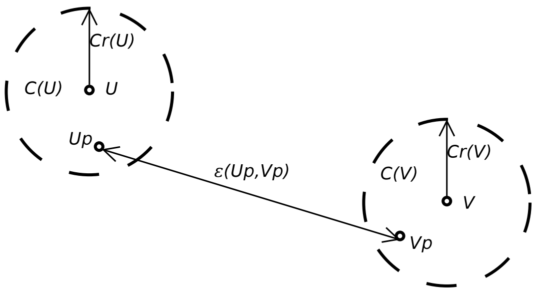

Figure 12. Both users finds a private submission point and calculates the distance between these.

For these problems, a modified version of the Dynamic Shifted Circles (DSC) strategy will be used. DSC is first of all used to create a buffer zone for a user. Normally DSC shifts away the center of the circle from the users exact location to optimize the size of the circle. But here, we shift away the center of the circle to hide the user's exact location from the other users. If use use user u as example, it would first have a cloaking region, $C(u)$, and somewhere random within this cloaking region it will pick a location as a submission points, u_p, to be used instead of u's exact location. u's buffer zone would be $B(u)$, and the center of this, would be the shifted location, u_p, and u would be able to move freely around within this buffer zone without having to worry about doing any proximity calculations. As long as u is within this zone, it will not get within proximity. This will greatly reduce the communication costs, while it can also be used to detect proximity.

Rule no.	Rule and meaning, using u as example
#1	$C_r(u) < \theta(u, v)$ The radius of the cloaking region must be lower than the desired proximity distance, and u's location must be the center of the circle. If the size of the radius violates Rule #4, it must be decreased.
#2	$C(u) \subset B(u)$ The cloaking region must be covered entirely by the buffer zone, thus u's actual location will also be covered by the buffer zone.

Table 2. General rules for proximity detection.

First of all, u finds a location for submission, u_p, within it's cloaking region $C(u)$, and the same does v. This is shown in Figure 12 which shows an overview of the system with the two users. The submission locations are found by generating a cloaking region, which, due to the Rule #1 of Table 2, must have a radius lower than the proximity distance, and the center of the cloaking regions must be the users actual location. Each user chooses a random location within this circle and they calculate the distance between the two submission locations, $\epsilon(u_p, v_p)$, using the SMC method, hence they have no idea in which direction the other user is located.

Three different modes are possible for a pair of friends to be in, and they are described separately. These modes defines, if the friends are within no chance of proximity, Section 4.3.1, might be within proximity, Section 4.3.2, or certainly are within proximity, Section 4.3.3.

4.3.1 No chance of proximity

When there is no chance for the two friends to be within proximity, the rules of Table 3 are used. Using Figure 13, we can show, the smallest distance between the two friends submission points, u_p and v_p, which is possible before there is a chance of proximity. Rule #1 states, that the maximum size of the radius of the cloaking regions must be lower than the distance of proximity, thus, if both $C_r(u)$ and $C_r(v)$ is greater than $\theta(u, v)$, and the users are not within proximity, then $\epsilon(u_p, v_p) > \theta(u, v) + 2 * \theta(u, v)$, or shorter $\epsilon(u_p, v_p) \geq 3 * \theta(u, v)$.

Rule no.	Rule and meaning, using u as example
#3	$\epsilon(u_p, v_p) > 3 * \theta(u, v)$ The distance between u_p and v_p must always be greater than three times the desired proximity, to guarantee no chance of proximity.
#4	$B_r(u) > \epsilon(u_p, v_p)/2 - \theta(u, v)/2$ The radius of the buffer zone is half the distance between u_p and v_p minus half the proximity distance. When both users use this rule, they cannot get within proximity without overstepping the buffer zone. The center of the circle must be in the submitted location point, u_p.

Table 3. Rules for when users are not within chance of proximity.

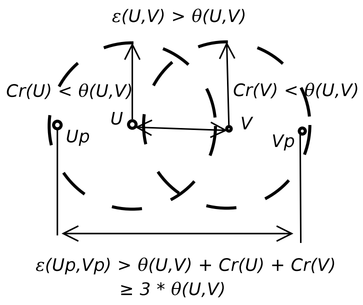

Figure 13. The smallest distance between two submission points without chance of proximity.

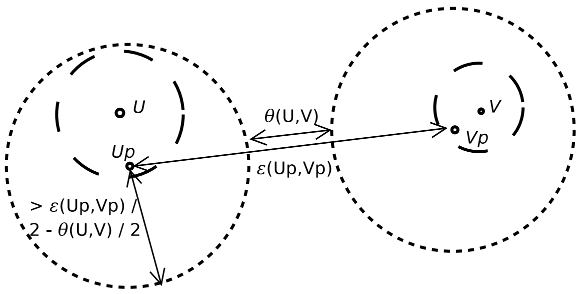

Figure 14. The size of the buffer zone.

If there is no chance of proximity, each user, here u, will generate a buffer zone, $B(u)$. Here u can move freely around within, without risking getting within proximity, thus there is no reason for doing any location updates either. Figure 14 shows the buffer zones as dotted circles, while the dashed circles are the cloaking regions. The size of the buffer zone is given by Rule #4, which states $B_r(u) > (\epsilon(u_p, v_p)/2 - \theta(u,v)/2)$. This means, that the buffer zone shall cover the largest possible area, without risking the users might get within proximity. This is done with concern to Rule #2. Since we know from Rule #1, the radius of the cloaking region must be lower than the proximity distance and the center must be in the user's exact location, we now know, that u_p can be up to $\theta(u,v)$ closer to v_p and v than u, thus the submission points can always be the closest point between two users. Rule #2 then tells us, that the cloaking region must be covered entirely by the buffer zone. Therefore, if we ensure there is $\theta(u,v)$ distance between $B(u)$ and $B(v)$, we can then guarantee, that there is no risk of the two users getting within proximity when they are moving within their respective buffer zones. If the buffer zones cannot be created without violating Rule #21, the user needs to reduce its cloaking region and start over, by recalculating $\epsilon(u_p, v_p)$.

4.3.2 Maybe proximity

If Rule #3 cannot be satisfied, then there is a chance that the users are within proximity, and they switch to this mode. Figure 13 shows the smallest distance between two users without there is any chance of being within proximity, while Figure 15 shows, the greatest distance between u and v possible, if Rule #3 is not satisfied. If this is the case we know, $\epsilon(u_p, v_p) \leq 3 * \theta(u,v)$. u can be up to $C_r(u)$ away from u_p, and the same applies to v. Using the rules in Table 4, we can now decide when users are within proximity or not. If Rule #3 is not satisfied, but Rule #5 is, the the distance between u and v must be $5 * \theta(u,v) > \epsilon(u,v) > \theta(u,v) + \theta(u,v)/\alpha * 2$. In this case, the users are close to each other, but not within proximity, and using random locations within their cloaked regions as submissions locations would introduce too great imprecision, thus Rule #8. Rule #6 says, that if there is no chance that, using precise submission points, the users will be within maybe proximity, there is no need to be in this mode. Therefore they will have to go back to the no proximity mode and use the rules from Table 3 instead.

Instead of using u_p and v_p for submission, both now use their exact location, u and v. This guarantees exact precision and reduces the risk of false positive/negative proximity detections. In this mode, a new method is used when users are testing for proximity. Figure 16 shows how this is done practically. First, the two users calculates the distance between them as always, using SMC. Then they divide the

Rule no.	Rule and meaning, using u as example
#5	$epsilon(u,v) > \theta(u,v) + \theta(u,v)/\alpha * 2$ For the users to be within chance of proximity, but not within proximity, they must be greater than the proximity distance, plus some precision.
#6	$\epsilon(u,v) \leq 5 * \theta(u,v)$ For the users to stay within the chance of proximity mode, the distance between the two friends must be less than five times the desired proximity distance. If this is not the case, the rules from Table 4 will be used.
#7	$B_r(u) > \epsilon(u,v) - (\theta(u,v) + \theta(u,v)/\alpha * 2)/2$ The radius of the buffer zone, when within chance of proximity, is half the distance between u and v minus half the proximity distance. When both users use this rule, they cannot get within proximity without overstepping the buffer zone. The center of the circle must be in the users actual location, u.
#8	$u_p = u$ When within chance of proximity, no cloaking region is used. This for ensuring precision of the distance when this is small.

Table 4. Special rules for when within chance of proximity.

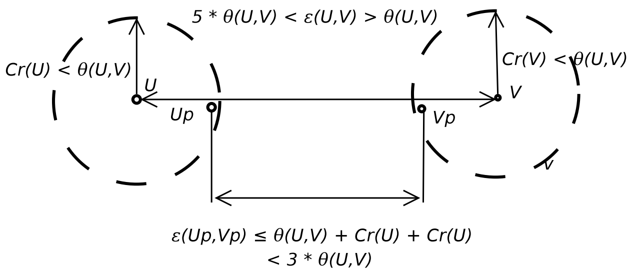
5 * θ(U,V) < ε(U,V) > θ(U,V)
Cr(U) < θ(U,V)
Cr(V) < θ(U,V)
U
V
Up
Vp
ε(Up,Vp) ≤ θ(U,V) + Cr(U) + Cr(U)
< 3 * θ(U,V)

Figure 15. The greatest distance between two submission points while violating Rule #3.

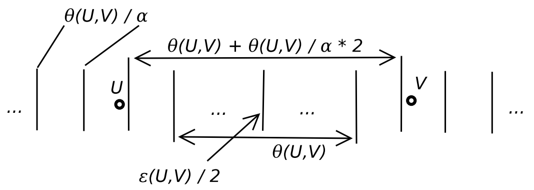

Figure 16. The method of detecting proximity between two users.

space between them into zones using α as precision. α decides how many zones the proximity distance, $\theta(u,v)$, is to be divided into.

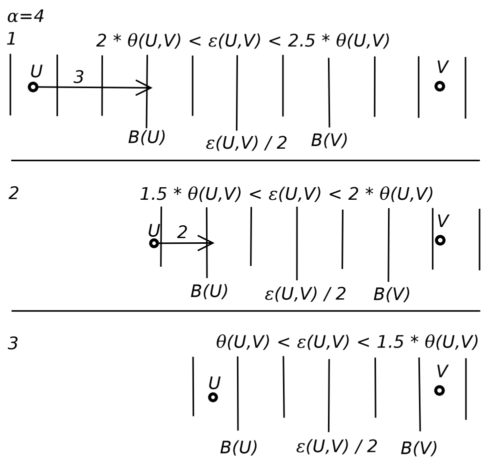
α=4
1
2 * θ(U,V) < ε(U,V) < 2.5 * θ(U,V)
U
3
V
B(U)
ε(U,V) / 2
B(V)
2
1.5 * θ(U,V) < ε(U,V) < 2 * θ(U,V)
U
2
V
B(U)
ε(U,V) / 2
B(V)
3
θ(U,V) < ε(U,V) < 1.5 * θ(U,V)
U
V
B(U)
ε(U,V) / 2
B(V)

Figure 17. An example of user u moving within proximity of user v.

If the distance between the two users is greater than $\theta(u,v) + \theta(u,v)/\alpha * 2$, they are not within proximity, and both users calculates the radius of their buffer zones using Rule #7 instead of Rule #4. $\theta(u,v) + \theta(u,v)/\alpha * 2$ is the smallest distance between two users possible before the two users are within proximity, and then they are within the two closest zones before proximity, as shown in Figure 16.

If the distance is smaller than $\theta(u,v) + \theta(u,v)/\alpha * 2$, they are very close to each other, and proximity will be notified both users. An example of this is showed in Figure 17, where two users has gotten within close distance of each other. v is standing still, for simplicity, and u is moving towards v. Precision is set to $\alpha = 4$, which means that, the size of each zone is $\theta(u,v)/4$. For the users to be within proximity, the distance then has to be $\epsilon(u,v) \leq \theta(u,v) + \theta(u,v) * 4 * 2(\theta(u,v) * 1.5)$. If e.g. α was set to $\alpha = 1000$, the distance would have to be $\epsilon(u,v) \leq \theta(u,v) + \theta(u,v)/1000 * 2(\theta(u,v) * 1.002)$, hence better precision. The first step shows, the two users are between 2 and 2.5 times $\theta(u,v)$ away from each other. User u then moves towards v, crosses three zones and when it oversteps its buffer zone, they will recalculate the distance between them. Then the users are between 1.5 and 2 times $\theta(u,v)$ away from each other, and a new set of zones is calculated. Both users also calculate a new buffer zone, and this time user u crosses two zones before overstepping its buffer zone. When recalculating again, both users find out they are within within $1.5 * \theta(u,v)$, thus they are within proximity.

The reason for, why the algorithm does not continue narrowing down to exact proximity, but uses α instead, is because the buffer zone would only get smaller for each time u oversteps it. u cannot cross over any zones anymore, and next time u oversteps, the buffer zone would be half the size of what it is now, thus the number of recalculations would increase heavily until proximity will occur. Therefore it is possible to increase α if improved proximity is desired, but this at the costs of more calculations, because the buffer zone can get smaller.

4.3.3 Proximity

#9	$epsilon(u,v) \leq \theta(u,v) + \theta(u,v)/\alpha * 2$ For the users to be within proximity they must violate Rule #3 and greater than the proximity distance, plus some precision.
#10	$B_r(u) = (\theta(u,v) + \theta(u,v)/\alpha * 2 - \epsilon(u,v))/2$ The radius of the buffer zone, when users are within proximity, is optimized for as large an area as possible, without any of the users can move outside of proximity without overstepping their buffer zones.

Table 5. Special rules for when within proximity.

When two users are within proximity, they must obey Rule #9 from Table 5 and they use a third method of calculating their buffer zones. This is shown in Figure 18, where u and v are depicted with their buffer zones, dotted circles, and proximity zones, dashed circles. Their proximity zones are constantly changing, while the users are moving, thus we need to generate some buffer zones, that will detect when there is a risk they get outside each other proximity zones. The maximum distance between them must never be more than $\theta(u,v) + \theta(u,v)/\alpha * 2$, and each user must take into account, that both users could be moving away from each other, thus $B_r(u) = (\theta(u,v) + \theta(u,v)/\alpha * 2 - \epsilon(u,v))/2$, also given by Rule #10. Then we have taken into account that the users are located $\epsilon(u,v)$ away from each other, and by dividing with 2, they are able to move into opposite directions, and if they get outside proximity, we would know it due to one or both users would leave their buffer zones. If the users get outside proximity, they will use the methods of maybe within proximity for calculating buffer zones.

5 System design

In this section, the friend finder approach will be designed with all the techniques from Section 4. The system will be a P2P system, hence only a client is needed, and all communication will go on directly in between these.

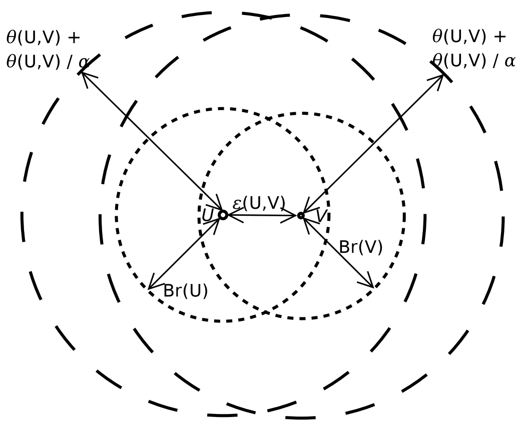

Figure 18. The users buffer zones, when within proximity.

Each user will be able to have an undefined number of friends, which it can detect proximity between. The proximity distance between each pair of friends can be different, but the proximity distance between two mutual friends must be agreed, so the distance is equal both way, thus $\theta(u, v) = \theta(v, u)$. The precision of proximity, described as α is, just like the proximity distance, agreed between each pair of friends. Thus, the precision requirements can be different between different pair of friends.

5.1 The client design

The client software will be defined by using handler algorithms, which will be raised on different events. An event could be when a remote request from a friend is received, when a user joins the system, when proximity is possible, etc. The basic handlers of the system are:

Startup When a client joins the system, the following handler is raised. This connects to the other friends, which are within the system, and initializes proximity tracking between them.

LocationChanged When the location of a client is changed, this handler is raised. This will compute if any boundaries are crossed or any other actions should be taken.

MessageReceived(msg, $args$) A handler executed, when a message is received from another client. This handler decides what should be done from the type of message defined by msg with arguments $args$, is received. The types of valid messages are presented in Table 6.

Each user contains some general information, such as a list of friends, which the user wishes to be informed of, when it is within proximity with. These data is presented in Table 7. The user needs to keep some information on each friend, along with some information concerning proximity calculation between that friend, such as a desired proximity distance and precision, last distance, and so on. All the information, each user contains about a relationship with a friend is described in Table 8.

Message Type	Args	Description
M_{SMC}	...	Calculates the distance between two users submitted locations. This is done using SMC, and requires a user identification along with a used proximity. This is seen as a black box and requires several messages both ways, therefore the arguments are not presented.
M_{dist}	u, $dist$, prx	A user sends to a friend its own id, the distance between them, and information of how close to each other they are or if they are within proximity.

Table 6. Types of messages and their arguments

Symbol	Meaning
loc	The users current location.
FS	A list of all the friends of a user.
$.x/.y$	The coordinates of a location, i.e. $loc.x$ is the x-coordinate of loc.

Table 7. Table describing what data a user generally always holds.

Calculating the distance between two private points, using SMC, has been done before in [14]. Their Protocol 1 does exactly this. The algorithm is effective and efficient. If we have two users, u and v, and u wants to calculate the distance to v, u will have to send, in total, 4 messages to v, and v will have to send 3 messages back, plus two times of 1-out-of-n oblivious transfers. 1-out-of-n oblivious transfers have been reduced to two rounds by [17] and [22]. Therefore, the total number of messages used, when getting the distance between two users, using SMC would be $4 + 3 + 4 = 11$.

Algorithm 2 describes how SMC is used in this work, and starts out by selecting whether to use the exact location or a cloaked location and then initiates the SMC algorithm by sending the initial call to the friend. Algorithm 3 is the event handler for when a friend initiates a distance calcu-

Symbol	Meaning
$prox$	The desired proximity distance.
$prec$	The desired precision level.
$dist_{old}$	The distance between a user and a friend at last location update.
loc_{old}	The exact location of the user at last location update.
loc_{tmp}	A temporary location variable for remembering what location was used at last distance calculation.
loc_{p_old}	The submission point used at last location update.
C_{r_old}	The radius of the cloaking region used at last location update.
B_{r_old}	The radius of the buffer zone used at last location update.
$proximity$	Whether two friends are within proximity, or close to each other. 0 is no chance of proximity. 1 is within chance of proximity, but not within proximity, and 2 is within proximity.

Table 8. Table describing what a user knows of a friend.

```
Data: $\mathbf{F_s} \subseteq \mathbf{C}, \forall F \in \mathbf{F_s}\ u \in F.\mathbf{F_s}$ - $U$'s list of
      friends. A friendship must be mutual.
$CLR : F \mapsto \mathbb{N}$ - a function, that returns a usable
radius for the cloaking region. Concerns Rule #2 in
Table 2.
$CL : F \mapsto P$ - a function, that returns a random
location within the cloaking region, depending on the
settings of the friend given as input.
$BR : F \mapsto \mathbb{N}$ - a function, that returns a radius of a
possible buffer zone, depending of the settings of a
friend. Concerns Rule #4, #7, and #10 of Table 3, 4,
and 5 respectively.
$SMC : F \times U \mapsto \mathbb{N}$ - a function, that, using SMC,
calculates the distance between two friends and
returns this. Described in Algorithm 2 and 3.
$DIST : P \times P \mapsto \mathbb{N}$ - a function, that returns the
Euclidean distance between two points.
```

Algorithm 1: A users local data

```
1 SMC($F$)
2 if $F.proximity = 0$ then
3     $x \leftarrow F.loc_{p\_old}.x$;
4     $y \leftarrow F.loc_{p\_old}.y$;
5 else
6     $x \leftarrow F.loc_{old}.x$;
7     $y \leftarrow F.loc_{old}.y$;
8 return the distance calculated by the SMC algorithm,
  issued by sending $M_{SMC}(U, F.proximity, ...)$ to $F$,
  where U is the identification of the user itself and
  using $x$ and $y$ as location.;
```

Algorithm 2: The secure multi-party computation algorithm.

```
1 MessageReceived($M_{SMC}, F, prx, ...$)
2 $F.loc_{tmp} \leftarrow loc$;
3 if $prx = 0$ then
4     $x \leftarrow F.loc_{p\_old}.x$;
5     $y \leftarrow F.loc_{p\_old}.y$;
6 else
7     $x \leftarrow F.loc_{old}.x$;
8     $y \leftarrow F.loc_{old}.y$;
9 Calculates the distance using SMC and the coordinates
  $x$ and $y$.;
```

Algorithm 3: The event handler for when receiving a request for calculating the distance using SMC.

lation with a user. The user saves its actual location into a temporary variables and finds out which coordinates to use from whether proximity is possible or not, line 2-7. The rest of the SMC can be seen as a black box, and at the end, the user, who initialized the calculations holds the distance between the two users.

Algorithm 4 describes what happens when a user is joining the service. Line 2-3 makes the algorithm iterate through all the friends of a user, which are online. Since this is a startup event, we reset proximity at line 4. Line 5-16 is a loop, which at line 6-8 fetches a new cloaking region and saves the information about this, along with the current location, for this specific friend. Line 9 uses the SMC function, to calculate the distance between the two users. We know from Section 4.3, that if the distance between two submitted locations is smaller than, or even to, 3 times the desired proximity distance, there is a chance that the users might be within proximity. This is checked for at line 10, and if this is the case, the precise distance between the two users is calculated at line 12. If the users are within proximity, it will be caught by line 13. At last a buffer zone is generated at line 15, which is dependent on whether a user is within proximity, chance of proximity or not within chance of proximity. Thus, this is the last line in the loop.

All the lines, 5-16, will be repeated, as long as there is no chance that the user can be within proximity and if

```
1  **Startup**()
2  **foreach** $F \in \mathbf{F_s}$ **do**
3      **if** $F$ *is online* **then**
4          $F.proximity \leftarrow 0$;
5          **repeat**
6              $F.loc_{old} \leftarrow loc$;
7              $F.C_{r\_old} \leftarrow CLR(F)$;
8              $F.loc_{p\_old} \leftarrow CL(F)$;
9              $F.dist_{old} \leftarrow SMC(F)$;
10             **if** $F.dist_{old} \leq F.prox * 3$ **then**
11                 $F.proximity = 1$;
12                 $F.dist_{old} \leftarrow SMC(F)$;
13                 **if** $F.dist_{old} \leq F.prox + F.prox/F.prec * 2$ **then**
14                     $F.proximity \leftarrow 2$;
15             $F.B_{r\_old} \leftarrow BR(F)$;
16         **until** $F.B_{r\_old} \geq DIST(F.loc_{p\_old}, F.loc_{old}) + F.C_{r\_old}$ *OR* $F.proximity > 0$;
17         send to $F$ $M_{dist}(U, F.dist_{old}, F.proximity)$;
```

Algorithm 4: The event handler for starting up.

the cloaking region is not fully contained within the buffer zone. When this is not the case, the new calculated distance is sent to the friend, line 17, which then is aware of the new distance between the two friends.

```
1  **MessageReceived**($M_{dist}, F, dist, prx$)
2  **if** $F$ *is online* **then**
3      $F.dist_{old} \leftarrow dist$;
4      $F.proximity \leftarrow prx$;
5      $F.B_{r\_old} \leftarrow BR(F)$;
6      **if** $F.proximity = 0$ *AND* $F.B_{r\_old} < DIST(F.loc_{p\_old}, F.loc_{old}) + F.C_{r\_old}$ **then**
7          **while** $F.B_{r\_old} < DIST(F.loc_{p\_old}, F.loc_{old}) + F.C_{r\_old}$ **do**
8              $F.C_{r\_old} \leftarrow CLR(F)$;
9              $F.loc_{p\_old} \leftarrow CL(F)$;
10             $F.dist_{old} \leftarrow SMC(F)$;
11             $F.B_{r\_old} \leftarrow BR(F)$;
12         send to $F$ $M_{dist}(U, F.dist_{old}, F.proximity)$;
```

Algorithm 5: The event handler for when receiving a new distance from a friend.

Algorithm 5 is used when a message is received from a friend, that tells the distance between the two users have changed. Three arguments are given, besides the description of the message. First is the identification of the friend, second is the new distance between them, and third is at what proximity level the distance is calculated. First, at line 2, it is checked whether the user is still online. If this is the case, the new distance is saved along with the proximity level, and a new buffer zone is created. Line 6 is checking if there is no chance of proximity and if the cloaking region is completely contained within the buffer zone, and line 7 loops as long as the cloaking region is not entirely contained by the buffer zone. If this is the case, line 8-11 are generating a new cloaking and buffer zone, and after looping, the new distance will be sent to the friend at line 12.

When a location for a user is changed, described in Algorithm 6, for each friend online where the user has moved outside its buffer zone, line 2-3, some calculations must be done. First of all, the new location is saved, line 4, and if the two friends were not within chance of proximity, line 7-13 is repeated until either a valid buffer zone is generated or if there is a chance of proximity. If there is a chance of proximity, or the users before were in chance of, or within, proximity, line 15, the actual distance is calculated at line 16. If there, for sure, are no chance, that the users could get within chance of proximity, using any submission point, line 17, then the users were in chance of proximity at an earlier proximity calculation. Thus we tell, that proximity is not a possibility, and a new cloaking and buffer zone is calculated repeatedly, until the cloaking region is contained within the buffer zone, line 19-24. I instead the user is within chance of proximity or within proximity, a buffer zone for this is calculated at line 25-30, and at last the information about the new distance is sent at line 31.

6 Experimental Results

In order to show that this idea is practically usable, and performs well, the design from Section 5 has been implemented in a Java application, that can simulate a number of parallel clients. Due to the implementation is multi threaded, race conditions might occur. Therefore the results might vary a little bit if the same test is being run several times. Therefore, each test has been run 10 times, and the average have been calculated. All that is needed for the clients to run is a predefined set of users, their friendship relations and settings and a set of locations with timestamps for each user. The location data used for the experiments is generated using [4] and the included map of Oldenburg city in Germany. This location generator creates a datafile, that contains a number of users and their location history, combined with timestamps, while they have been simulated

```
LocationChanged()
foreach $F \in \mathbf{F_s}$ do
    if $F$ is online AND (($F.proximity = 0$ AND $DIST(F.loc_{p\_old}, loc) > F.B_{r\_old}$) OR ($F.proximity > 0$ AND $DIST(F.loc_{old}, loc) > F.B_{r\_old}$)) then
        $F.loc_{old} \leftarrow loc$;
        if $F.proximity = 0$ then
            repeat
                $F.C_{r\_old} \leftarrow CLR(F)$;
                $F.loc_{p\_old} \leftarrow CL(F)$;
                $F.dist_{old} \leftarrow SMC(F)$;
                if $F.dist_{old} \leq F.prox * 3$ then
                    $F.proximity \leftarrow 1$;
                else
                    $F.B_{r\_old} \leftarrow BR(F)$;
            until $F.B_{r\_old} \geq DIST(F.loc_{p\_old}, F.loc_{old}) + F.C_{r\_old}$ OR $F.proximity > 0$;
        if $F.proximity > 0$ then
            $F.dist_{old} \leftarrow SMC(F)$;
            if $F.dist_{old} > F.prox * 5$ then
                $F.proximity \leftarrow 0$;
                repeat
                    $F.C_{r\_old} \leftarrow CLR(F)$;
                    $F.loc_{p\_old} \leftarrow CL(F)$;
                    $F.dist_{old} \leftarrow SMC(F)$;
                    $F.B_{r\_old} \leftarrow BR(F)$;
                until $F.B_{r\_old} \geq DIST(F.loc_{p\_old}, F.loc_{old}) + F.C_{r\_old}$;
            else if $F.dist_{old} > F.prox * (1 + 2/F.prec)$ then
                $F.proximity \leftarrow 1$;
                $F.B_{r\_old} \leftarrow BR(F)$;
            else
                $F.proximity \leftarrow 2$;
                $F.B_{r\_old} \leftarrow BR(F)$;
        send to $F$ $M_{dist}(U, F.dist_{old}, F.proximity)$;
```

Algorithm 6: The event handler for starting up.

to be moving around within Oldenburg. The coordinates of Oldenburg are fictional and cannot be mapped directly back to the real world, but the x-coordinates ranges from 0 to about 22.000 while the y-coordinates ranges from 0 to about 32.000.

Since the part concerning SMC is seen as a black box to the system, this is not implemented, but the number of times SMC is done is calculated, and this can be multiplied with the number of messages a SMC implementation actual takes. This was calculated in Section 5 to be 11 messages each time SMC has to be performed. When counting messages, the number of messages is how many messages have been sent in between all the users. Thus, if user u sends a message to user v, this is counted as one message, even though both u has to send it, and v has to receive it.

Each experiment will be introduced by describing what the purpose of the experiment is and what settings have been used. Then the results will be presented and at last discussed why the results looks like they do and if the outcome was as expected.

6.1 Impact of Precision

The precision variable, α, that defines how many zones the proximity distance will be divided into when users might be within proximity, is an interesting parameter. This balances the trade off between the number of messages sent and the required level of precision. This is only used while users might be within proximity, and have no impact on the performance, while users are far away from each other.

The setting of this experiment is 5 users, that all are mutual friends. That means that 10 mutual friendships exist, and each user has 1000 locations to submit over a timespan of 1000 time units. Proximity distance between two users is set to 500, and the only parameter chanced is the precision variable.

Precision (α)	Messages	Proximities
1	5281	19
2	5491	15
3	5534	15
4	5541	15
6	5546	15
8	5576	15
16	5595	15

Table 9. Number of messages and proximities with different precision requirements.

The output data from the experiment is shown in Table 9 and the relation between messages and number of proximities can be seen from these results. is depicted in Figure 19. This shows, that there is a very clear relation between the number of messages sent and the number of proximities. The high number of proximities, when using low precision, is because there is a high risk of false positives, due to imprecision. When there is a large risk of false positives, the number of messages will also be reduced because they do not have to keep calculating ever smaller buffer zones, to get the exact point of when proximity will occur. From the graph it is also clear, that when proximities get to a steady level, the number of messages used will only increase slightly, even though precision is continuously increased.

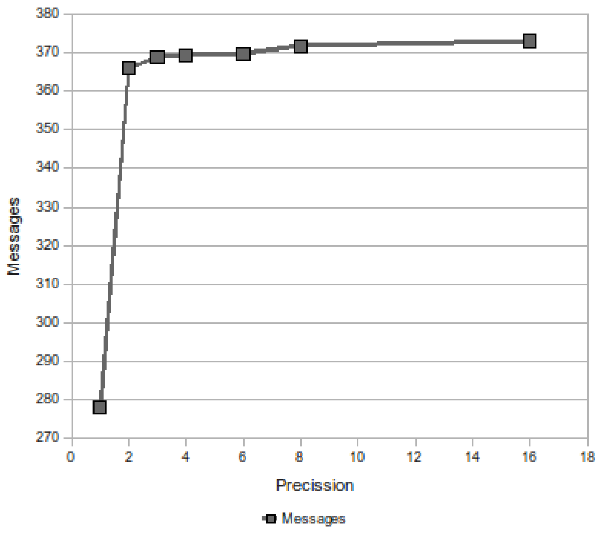

Figure 19. Graph of how precision impacts on the relation between messages and proximities, calculated from Table 9

In this case, when the α is raised to more that 2, the number of proximities are the same. This might seems a bit weird, because the precision variable might have been expected to be larger before exact precision was found. This is related to the data generated. The generator generates a location for each time stamp, and the distance a user could have moved between two timestamps might be too large for greater precision to have impact. If, on the other hand, the system was used in a context, where the location was updated constantly, the precision would keep improving the results, due to the precision of the locations.

6.2 Impact of proximity distance

The number of messages sent between two friends is a way to measure the communication costs of the service. As seen from the experiment of the impact of precision, the desired precision have some impact on the number of messages. But also the desired proximity distance between two friends will have some impact. This is due to, the methods used when users are not within proximity, perhaps within proximity or certainly within proximity are different, and this experiment will show whether it is an advantage to have a large or small proximity distance.

The setting of this experiment is again 5 users, that all are mutual friends, thus 10 mutual friendships exists. Also each user have 1000 locations over 1000 time units and the precision used at this experiment is 8. The only parameter changed is the proximity distance of each friend pair.

Proximity distance	Messages	Proximities
25	4268	1
50	4637	4
75	4765	5
100	4935	6
150	5098	11
200	5291	13
250	5615	14
300	5898	14
350	6197	14
400	6367	14
450	6766	15
500	7008	15

Table 10. Number of messages and proximities with different proximity distances.

As seen by the results, presented in Table 10, it is clear, that the desired proximity distance have a great impact on the number of messages sent between each pair of friends. This is depicted in Figure 20, where it is visible, that the larger proximity distance desired between two friends, the greater the number of messages will be. Also the number of proximities grows, due to the larger area. This is obvious, because when a larger proximity distance is chosen, there is a greater chance that two friends will be within proximity.

The results is as expected. When two friends are far away from each other, they are using large buffer zones, and only seldom needs to update their location. When two friends gets closer to the proximity distance, or even within proximity, their buffer zones are getting much smaller, thus they are doing more updates.

6.3 Other solutions

To prove that this solution actually is an improvement over direct P2P, and that the cost of using the techniques deployed by this solutions are acceptable, some tests will be done to show this. The experiments includes five different types of solutions

Full solution: The full solution presented in this paper, using SMC and cloaking regions.

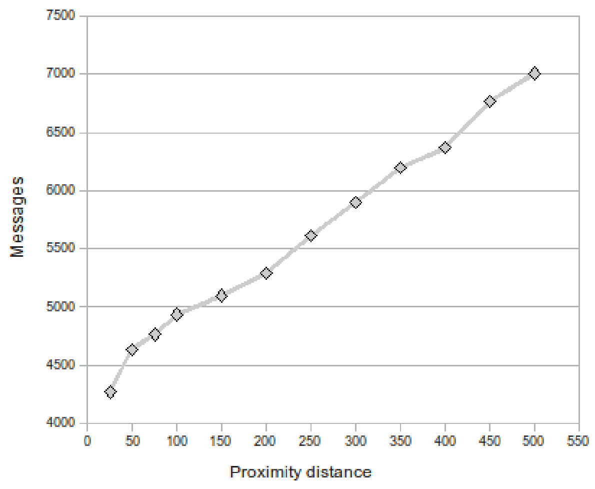

Figure 20. Graph of calculated from the results of Table 10

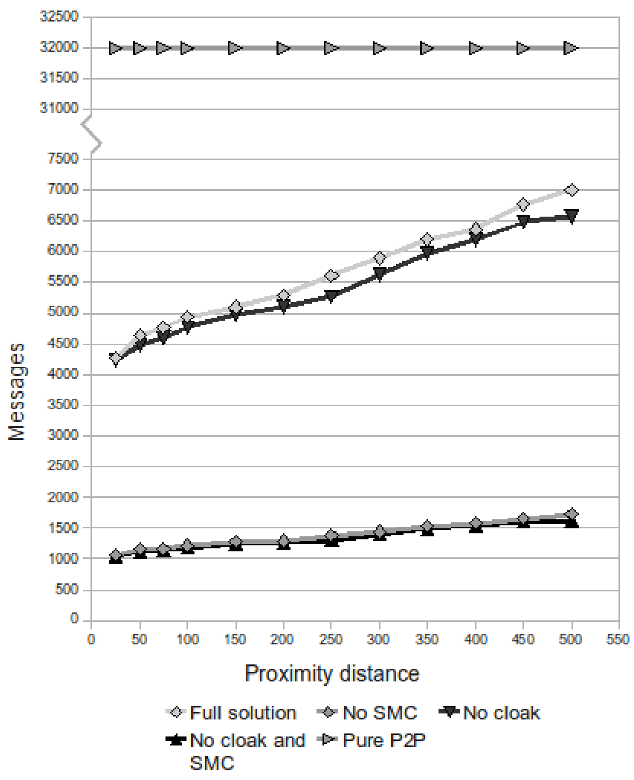

Figure 21. Graph comparing different types of solutions at different proximity distances

No SMC: The solution presented in this paper, but with SMC disabled.

No cloak: The solution presented in this paper, but with cloaked regions disabled.

No cloak and SMC: The solution presented in this paper, but with both cloaked regions and SMC disabled.

Pure P2P: A simulated pure P2P solution, where users will have to do updates each time they change their locations.

Again, the settings of this experiment is 5 users, all mutual friends, and each user have 1000 locations over 1000 time units. The precision used at this experiment is 8 and the proximity distance is changing, to show how the different solutions will behave with different proximity distances.

Figure 21 shows the number of messages that each solutions must send to do the task. It is clear, that the pure P2P solution is very ineffective. This is because it does not make use of any communication optimization techniques, thus it needs to do updates each time a user changes its location. The solution presented in this paper sends as little as $1/7$ of the messages parsed by the pure P2P solution. The most effective is the solution without cloaking regions and without SMC, but what is interesting is, that cloaking regions does not have that much impact on the number of messages. SMC does, on the other hand, require a lot of extra messages, and becomes the primary cost of this solution. About $3/4$ of the messages sent is at the cost of SMC. But when considering the significant improvement in privacy, by using SMC, this is acceptable.

These results are as expected, and shows that the techniques deployed by this solution to gain privacy comes at a price, but still improves the cost a lot, compared to a pure P2P solution.

7 Conclusion

In this paper, a P2P solution for detecting proximity and separation between mobile clients have been developed by using some different techniques. Three general strategies are used to develop this solution. Location privacy is used to hide a user's exact location within a cloaking region. Proximity detection is used to reduce the number of location updates and is archived by using a custom variant of the dynamic shifted circles strategy. At last, SMC is employed to improve security by only letting a user know how far away a friend is, not in which direction. These three techniques combined, results in a secure LBS that is optimized for communication. The solution is also flexible in the way, that precision and proximity distance parameters can be changed between each pair of friends, such that they individually can adjust for best desired settings.

A model of the solution has been implemented and tests have been performed to prove the efficiency. The experiments show the impact of the precision parameter does not have a great impact on the number of messages compared to the size of the proximity distance. Also it is proved, that SMC comes at a price, but still the solution is far better than using a pure P2P solution, and when considering the gain in privacy SMC introduces, it is worth the price.

8 Resume

This paper concerns the issues of privacy and communication costs of a mobile location-based service, that can notify friends when they are within a defined distance of each other. Not much work has been done in the field of friend locating, and the studies that have been made often includes a centralized server as a base for the service. One problems with the centralized server model is, that privacy can be hard to guarantee. Another model is the peer-to-peer model. This does not include any server, but all communication goes on between the clients only. The disadvantage of this model is often, that it is often communicational wise very expensive. The motivation for this paper is, if it is possible to make a peer-to-peer model, that is optimized for both privacy and communication.

An idea is presented, which combines three different areas to a solution, that satisfies the goal of preserving privacy and reducing communication costs.

First of all, to hide the users exact locations, cloaked regions is introduced. Each user generates a cloaked region around its location and it picks a random location within this cloaked region. Now this random location will be used instead of the users actual location, this to hide the users exact location from the others.

Second, a technique called secure multi-party computations (SMC) is used. SMC, is, in short terms, some calculations on private date being performed between two or more users, while the users does not want to reveal anything about their private data. A classic example is the millionaire problem, where two millionaires want to know who is richer without revealing anything about their actual wealth. SMC is the alternative to a trusted middle man, that the users could tell their information to, and let him do the math. In this work, SMC is used to calculate the distance between two friends, without revealing where the other friend is located or even in which direction he is. The only thing the friends gets to know from the SMC calculations is the distance between them. For calculating the distance, each user uses a random location within their cloaked regions, mentioned above. This to improve privacy, so a friend of a user does not know exactly how far away the friend is, only to some degree of uncertainty.

Third, a custom variant of the dynamic shifted circles are used to generate buffer zones for the users. A buffer zone is a zone where a user can move freely around within, without having to worry about getting within proximity of other users. When moving around within this zone, the user does not do any location updates, and thus communication is reduced. A dynamic shifted circle, is normally a circle that is generated around a user, but the center of the circle is shifted away from the user to optimize the radius. For this work, the center of the circle is not shifted away for optimizing the radius, but because of the SMC mentioned above. This because SMC is used to calculate the distance to a friend, and SMC has been using a random location within the cloaked region, thus, the center of the dynamic shifted circle must be the random point used to calculate the distance instead of the users exact location.

To make the service flexible, two parameters are variable between each pair of friends, namely the desired proximity distance and a precision level. The proximity distance decides how close the two friends must be, before proximity is detected. The precision level is introduced to balance communication versus how precise the users must be to the proximity distance before proximity is detected. This is because, when users are close to proximity, but not within proximity, more updates will be performed, and this is more communicational wise expensive, and this variable could reduce the number of updates, at the cost of a more imprecise service.

At last, a model of the solution is implemented and test is being performed, to prove that the solution practically usable and to test how it performs. The result shows, that the number of messages rises when better precision is wanted, but not by much. Also it shows, that the larger proximity distance is used, the more communicational expensive the service gets. This again, because it is more expensive in terms of messages, when users are close to proximity, and using larger proximity distances, they more often gets within chance of proximity. At last the solution is compared with pure peer-to-peer and to itself with cloaked regions and SMC disabled. It becomes clear, that the number of messages have been reduced to as low as $1/7$ of the pure peer-to-peer solution, using these chosen settings. Also it shows, that the cost of SMC is about $3/4$ of the entire service.

It has been proven, that a peer-to-peer solution for a friend locator location-based service is possible, and that both preserving privacy and reducing communication costs is possibly at the same time. Though privacy comes at a price, but this is worth the price, when considering that the location of a user is well hidden from the friends using the service.

References

[1] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler, "Buddy tracking - efficient proximity detection among mobile friends," *Pervasive Mob. Comput.*, vol. 3, no. 5, pp. 489–511, 2007.

[2] M. J. Atallah and W. Du, "Secure multi-party computational geometry," in *WADS '01: Proceedings of the 7th International Workshop on Algorithms and Data Structures*. London, UK: Springer-Verlag, 2001, pp. 165–179.

[3] T. O. G. Blog, "See where your friends are with google latitude," February 2009. [Online]. Available: http://googleblog.blogspot.com/2009/02/see-where-your-friends-are-with-google.html

[4] T. Brinkhoff, "A framework for generating network-based moving objects," *Geoinformatica*, vol. 6, no. 2, pp. 153–180, 2002.

[5] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar, "Preserving user location privacy in mobile data management infrastructures," *6th Workshop on Privacy Enhancing Technologies pp. 393-412*, vol. 4258/2006, p. 10, 2006.

[6] C.-Y. Chow and M. F. Mokbel, "Enabling private continuous queries for revealed user locations," *Lecture Notes in Computer Science pp. 258–275*, vol. 4605/2007, p. 18, August 2007.

[7] R. Fagin, M. Naor, and P. Winkler, "Comparing information without leaking it," *Commun. ACM*, vol. 39, no. 5, pp. 77–85, 1996.

[8] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)*, vol. 4, p. 10, 2005.

[9] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1987, pp. 218–229.

[10] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*. New York, NY, USA: ACM, 2003, pp. 31–42.

[11] InformationWeek, "Court asked to disallow warrantless gps tracking," March 2009. [Online]. Available: http://www.informationweek.com/news/government/federal/showArticle.jhtml?articleID=215800542

[12] KCTV5.com, "Stalkers using gps devices to track victims," 2009. [Online]. Available: http://www.kctv5.com/news/17026521/detail.html

[13] A. Küpper and G. Treu, "Efficient proximity and separation detection among mobile targets for supporting location-based community services," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 10, no. 3, pp. 1–12, 2006.

[14] S.-D. Li and Y.-Q. Dai, "Secure two-party computational geometry," *J. Comput. Sci. Technol.*, vol. 20, no. 2, pp. 258–263, 2005.

[15] M. F. Mokbel and C.-Y. Chow, "Challenges in preserving location privacy in peer-to-peer environments," in *WAIMW '06: Proceedings of the Seventh International Conference on Web-Age Information Management Workshops*. Washington, DC, USA: IEEE Computer Society, 2006, p. 1.

[16] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: query processing for location services without compromising privacy," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 763–774.

[17] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001, pp. 448–457.

[18] A. Press, “Stalkers use gps to track victims,” February 2002. [Online]. Available: http://www.wired.com/gadgets/wireless/news/2003/02/57576

[19] P. Samarati, “Protecting respondents’ identities in microdata release,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 13, no. 6, pp. 1010–1027, 2001.

[20] G. Treu, T. Wilder, and A. Küpper, “Efficient proximity detection for location based services,” in *PROCEEDINGS OF THE 2nd WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION (WPNC05) & 1st ULTRA-WIDEBAND EXPERT TALK (UET’05)*, Hannover, Germany, Mar. 2005, pp. 165–173.

[21] ——, “Efficient proximity detection among mobile targets with dead reckoning,” in *MobiWac ’06: Proceedings of the 4th ACM international workshop on Mobility management and wireless access.* New York, NY, USA: ACM, 2006, pp. 75–83.

[22] W.-G. Tzeng, “Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters,” *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 232–240, 2004.

[23] J. Voelcker, “Stalked by satellite,” July 2006. [Online]. Available: http://www.spectrum.ieee.org/jul06/4103

[24] L. Šikšnys, J. R. Thomsen, S. Šaltenis, M. L. Yiu, and O. Andersen, “A location privacy aware friend locator,” in *SSTD*, 2009.

[25] X. Wei-jiang, J. Wei-wei, H. Liu-sheng, and Y. Yi-fei, “Privacy-preserving collision detection of two circles,” in *InfoScale ’07: Proceedings of the 2nd international conference on Scalable information systems.* ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, pp. 1–7.

[26] T. Xu and Y. Cai, “Location anonymity in continuous location-based services,” in *GIS ’07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems.* New York, NY, USA: ACM, 2007, pp. 1–8.

[27] A. C. Yao, “Protocols for secure computations,” in *SFCS ’82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science.* Washington, DC, USA: IEEE Computer Society, 1982, pp. 160–164.

[28] A. C.-C. Yao, “How to generate and exchange secrets,” in *SFCS ’86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science.* Washington, DC, USA: IEEE Computer Society, 1986, pp. 162–167.