A Geography-aware Service Overlay Network for Managing Moving Objects

Theodoros Chondrogiannis and Alex Delis
Dept. of Informatics & Telecommunications
University of Athens, 15784, Athens, Greece
{grad1105,ad}@di.uoa.gr

ABSTRACT

As the proliferation of mobile devices and positioning systems continues unabated, the need to provide more robust location-based services becomes more pressing. In this context, we examine the problem of efficiently handling queries over moving objects and propose a location-aware overlay network that helps monitoring such objects while traversing contained geographic extends. We use a triangulation structure to divide a geographic area using fixed service nodes as anchors based on their geographic position. Triangulation inherently contains each moving object within an area designated by three service nodes. We introduce a method for monitoring moving objects and we present an algorithm for processing nearest-neighbor queries while restricting the amount of resources and, subsequently, the volume of transmitted messages. Through simulation, we evaluate the suggested approach and show that our nearest-neighbor query processing method provides always accurate results while it uses invariantly a constant number of service nodes. We finally show that the average physical distance between service and roaming nodes remains limited; this yields a significant number of physical connections that avoid conventional Internet routing altogether.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; H.5.3 [**Information Interfaces and Presentations**]: Group and Organization Interfaces

Keywords

Overlay networks, Mobile Objects, Location Based Services

1. INTRODUCTION

The widespread availability of diverse wireless communication options along with GPS-enabled devices has paved the way for the realization of *Location-based Services (LBS)* whose prime objective is to aid the roaming user in real-time fashion. Applications including on-demand computer-generated road and elevation maps as well as navigational facilities for road-networks are commonplace and are often embedded in all high-end contemporary mobile devices. Although systems providing *LBSs* are mostly focused on storing and processing queries over fixed location points of interest, the next generation of services will enable more collaboration and cooperative decision making among the moving objects within a network. For instance, consider a monitoring service that helps manage a taxi-cab fleet in an urban area predominantly using cooperative means: a potential passenger in need of a taxi-cab should alert the nearest available vehicle. The *LBS* monitoring service should act in "mostly-localized" fashion and help dispatch the taxi found closest to the point of request. In doing so, the service should avoid centralized action and/or the assistance of a dispatcher. Other applications that may be similarly realized include community car-pooling, cooperative police-car actions, sharing of car/bicycle rentals, as well as search-and-rescue operations [1].

The idea of location-based query processing has its roots in *GeoCast* [6] in which the concept of integrating the physical location into the design of the Internet is proposed. In the context of the above applications, the issue that is of particular interest is that of the *continuous monitoring* of the roaming nodes. An *LBS* system must always know if not the exact, at least the approximate position of the provider of a service as well as the location of the client who requests the service. Evidently, the exact position of moving nodes within a geographic extend is frequently difficult to know for a number of reasons. In addition, such a system has to process nearest-neighbor and *top-k* type queries both quickly and accurately. For example, a police patrol-car monitoring system should inform vehicles that are found closest to the location of the incident. The system should identify these patrol cars as quickly as possible and among all candidate cars only those with the ability to respond in minimum time should receive the order to intervene.

A common approach in addressing the above problems is creating and maintaining a centrally-managed location service, which can be queried and updated by the mobile devices. A number of efforts have focused on this topic and proposed algorithms for query processing and pertinent systems. For example, [4] describes a way to efficiently store moving objects and provides algorithms for fast spatial query processing. A more refined related approach is presented in [5] where appropriate metrics to represent motion are used; the latter take into consideration a number of the recent location updates of a mobile device as well as its velocity. Nevertheless, centrally-controlled approaches suffer a number of disadvantages. First of all, the load is concentrated on a single point and centralized service may fail as it might prove unable to tackle sudden surges of hot spots. Real-time requirements imposed on such a server may necessitate pricey hardware in order to to keep the query processing time short. Moreover, a centralized system does not exploit the inherent underlying geographic topology as well as the physical location of the participating and continuously moving objects. In such a centrally-controlled system, queries are initially received by Wi-Fi networks and hotspots and subsequently are routed via ISPs to the Internet in order reach the service. If a user does not have any access to the Internet, she is unable to reach the service of this centralized system at all. This however is not the case if roamers have only sporadic access to hotspots. To address the above problems, we adopt a decentralized and self-organizing approach as our key objective is to opportunistically harness Wi-Fi access for the provision of *LBS*-services to users in motion.

We propose a geography-aware service overlay network for the monitoring and evaluation of queries over moving objects. We use a triangulation scheme for organizing fixed location service nodes that exist in our network and we exploit the properties of the triangulation to monitor large numbers of roamers in distributed fashion. The main goal of our approach is to provide answers to *nearest moving neighbor* queries while restricting the amount of resources and/or service nodes required to process such queries. By restricting the geographic extend that we search in, the average query processing time is reduced. Moreover, the average workload remains limited since the majority of the service nodes are idle until a query is submitted by a roaming node close to them. Our approach also provides a good match to the underlying geographic topology as it takes into account the actual geographic position of both service and roaming nodes. As such, the proposed scheme can be applied over an existing infrastructure such as a wireless metropolitan network (WMN). We present pertinent algorithms for node arrival and departure and through simulation we show that our approach achieves quality results while it offers a great match for the underlying geographic topology.

The paper is organized as follows: Sections 2 and 3 outline background and related work. Section 4 presents all the elements of our self-organizing triangulation-based service overlay network that helps monitor roaming nodes and evaluate nearest-moving neighbor queries. Section 5 discusses preliminary simulation results and finally, concluding remarks can be found in Section 6.

2. BACKGROUND

Our proposal is founded on a $2D$ geographic space triangulation structure. The proposed algorithms are based on geometric criteria and properties of triangulation. In this section, we outline fundamental concepts that our approach exploits.

By definition, the convex hull of a set of points $P \in \Re^2$ is the intersection of all convex sets containing the points in P and is denoted as $conv(P)$. In Figure 1, points A, B, C, D, E and F constitute the convex hull. Consequently, a triangulation of a set of all points in $P \in \Re^2$ is a set of triangles

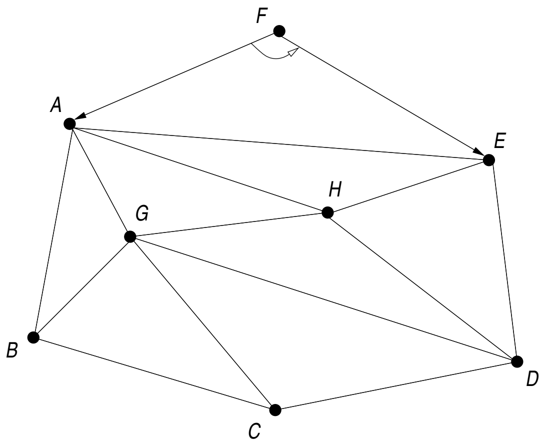

Figure 1: Sample triangulation and use of CCW predicate: F, A, E turn is counter-clockwise thus, $CCW(F, A, E) > 0$

T whose vertices are elements of P. Additionally, the union of all these triangles equals $conv(P)$ and the intersection of any pair of triangles results in one of the following: *a)* equal to $\varnothing$, *b)* equal to the common vertex of the two triangles, or *c)* equal to their common edge. The outer hull of any triangulation scheme is always equal to the convex hull of the set of points the triangulation is applied on.

Various triangulation schemes have been proposed. Each such scheme is defined by the specific criteria applied when triangles are formed. For example, the *Delaunay* triangulation maximizes the minimum angle of all angles in the triangles that ultimately make up the triangulation. In our case, we use a triangulation scheme that functions in a step-wise manner and employ no specific geometric constraint when forming new triangles. The expansion of the triangulation is *incremental* as every new point arriving in the network may form a number of new triangles. If the new point is inside the convex hull, only three new triangles are formed. More triangles may be formed if the new point is outside the convex hull and thus, the triangulation must be expanded; Section 4 presents our triangulation scheme.

For the realization of our overlay, we exploit the properties of the *Counter-Clockwise* (CCW) orientation predicate. If p_0, p_1, p_2 are points in $\Re^2$, $CCW(p_0, p_1, p_2)$ designates whether the direction of the turn defined by the three points is counter-clockwise; hence, the orientation predicate may be either positive $CCW(p_0, p_1, p_2)>0$ or negative $CCW(p_0, p_1, p_2)<0$. Should we consider points F, A and E of Fig. 1, the turn indicated by vectors $\overrightarrow{FA}$ and $\overrightarrow{FE}$ is counter-clockwise, so $CCW(F, A, E) > 0$. We exploit the CCW predicate to find whether a node is part of a convex hull. We also use the CCW predicate to define the *Ordered Finger Table* (OFT), a structure in which each node stores its neighbors in an ordered manner. We present OFT in Section 4.1. Finally, CCW is also used to examine whether a given node exists inside a given triangle.

3. RELATED WORK

The development and use of overlay networks has had much success in recent years. *Chord* [11] and *CAN* [8] are such P2P representatives that employ distributed (hash) tables (DHTs) to efficiently carry out exact queries. Their main objectives are data partitioning and searching over multiple sites. However, DHT-based systems are ineffective when it comes to spatial and/or geography-related queries.

To counter the aforementioned issue, a number of efforts have focused in the development of geographic location-aware overlays. Such a system is the *Geographic Hash Table (GHT)* [9] which stores nodes according to their geographical location. *GHT* hashes keys into geographical coordinates. Then, *GHT* stores a key-value pair at a node in the vicinity of the location to which the key of that node hashes. Other overlays that have been proposed use a different structure to organize their nodes. *Geopeer* [2] uses a *Delaunay* triangulation scheme to divide the $2D$ space into triangles. Each node is responsible only for points inside the *Delaunay*-triangle in which it participates. The above efforts although successful in handling spatial data and spatial queries over fixed-location objects, they make no provisions for managing moving objects and their respective dynamic behavior.

To the best of our knowledge, *Geogrid* [12] is a prior effort that is very related to our proposal. *Geogrid* is a decentralized and geographic-location-aware overlay network that divides two-dimensional extends into a number of rectangular regions. Each overlay node is assigned with a region and is responsible to process all requests mapped to this particular region. Additionally, the overlay node monitors all the roaming nodes inside its assigned region. In contrast, our proposal exploits triangles as they are formed by the existing fixed nodes of the underlying stationary network. We also take into account the actual position of overlay nodes something which *Geogrid* does not. Another interesting approach is *Mobieyes* [3] which is a distributed and scalable solution to continuously process queries over moving objects. *Mobieyes* uses a server to monitor the mobile devices while the devices are responsible for processing spatial queries themselves, thus, transferring a significant portion of the required computations on the mobile node. In contrast, our approach avoid imposing heavy overhead on mobile devices with computations and attempts to exploit the computational capabilities of the fixed nodes of the overlay.

4. OVERLAY NETWORK STRUCTURE

In this section, we present our overlay structure. Our system consists of two levels of nodes: the first-level is the actual overlay and all the nodes in this level are mainly static nodes with fixed location. These nodes provide the means for the cooperation among roamers and they can appear and disappear as they are envisaged to be part of a WMN. First-level nodes are organized according to our triangulation scheme (Section 4.2). A vertex in the triangulation corresponds to a static node with fixed geographic location that ultimately assists roamers to locate service provided by nearby fellow travellers. Users in motion constitute the second-level nodes of our approach. Every edge in the triangulation represents a network connection between two first-level overlay nodes. Such connection can be either provided by either direct line-of-sight radio couplings as is frequently the case in WMNs or direct connections through Internet. We term the first-level nodes as *Fixed-location Coordinating Nodes* or *FCN*s.

FCNs are responsible for handling the mobile nodes of the second-level. Each *FCN* is responsible for a number of mobile nodes that move within its vicinity. Each roamer has

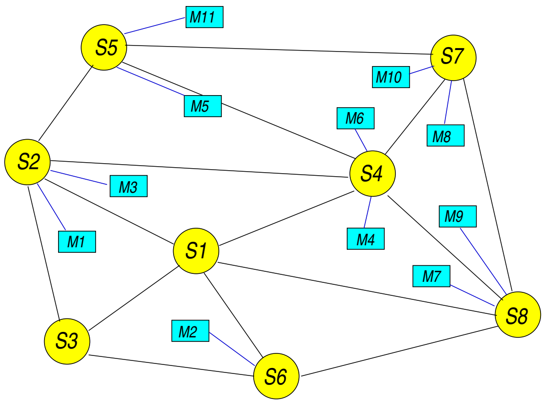

Figure 2: Triangulation enabling the overlay: *SX* are *FCNs* and *MX* represent nodes in motion.

to be within a triangle in which its "parent" or responsible *FCN* is vertex of. We assume that if a mobile node exists inside a triangle then its closest *FCN* is one of three vertices of the triangle in question. In rare occasions, this assumption does not apply. However, for maintaining simplicity in our protocol, we elect not to take into consideration such cases. In the triangulation of Fig. 2, nodes $S1$ to $S8$ are the *FCNs* of the first-level and the remaining nodes $M1$ to $M11$ correspond to users in motion.

4.1 Ordered Finger Table (OFT)

Each *FCN* maintains an *Ordered Finger Table (OFT)* that features all adjacent first-level nodes. OFT is a key component of our proposal and is designed to be an ordered cyclic list where all pertinent entries are sorted with the assistance of the CCW predicate. This sorting ensures that the orientation formed by any entry i of OFT table, the proprietor of this table along with the next OFT entry $i+1$ is counter-clockwise. For example, consider the instance of our layout in Fig. 2. $S4$ is an FCN that does not belong to the convex hull. The OFT of $S4$ contains entries referring to nodes $S1$, $S8$, $S7$, $S5$ and $S2$ in a cyclic oder. Thus, every pair of two consecutive OFT-entries form a triangle along with $S4$. This observation leads to the following definition.

Definition 1. *For every* FCN *node n located within the convex hull, all pairs of consecutive* OFT *entries in conjunction with n form adjacent triangles.*

The same condition also applies for nodes on the convex hull except one case. Since the oriented finger table is a cyclic list, then for a node on the convex hull, the next and the previous on the convex hull are neighboring entries. These two nodes cannot be connected to each other. For all other neighboring entries in the oriented finger table of a node, Definition 1 must always apply. In Figure 2, *FCN* $S6$ is on the convex hull and its OFT consists of nodes $S3$, $S8$ and $S1$. Although $S3$ and $S8$ are consecutive entries in the table, the nodes are not directly connected to each other. Notice though that both $S3$ and $S8$ are part of the convex hull. In general, we can use this fact in order to find out whether an *FCN* exists on the convex hull. If there are three or less nodes in the network, then all of them make up the convex hull. When more than three *FCNs* are present, the following definition applies:

Definition 2. *Let n_0 be a first-level node in the overlay and p_0 its geographic location. If there are more than three* FCNs *in the network and there exists a pair n_i, n_{i+1} of consecutive entries in the* OFT *table of n_0 having respective p_i, p_{i+1} locations with $CCW(p_0, p_i, p_{i+1})<0$, then n_0 is a node on the convex hull of the triangulation structure.*

These two definitions are of key importance, as they help us maintain the triangulation with minimum cost. In the following two subsection, we discuss how entries for *FCNs* are either added or removed from *OFT*s. Such operations are necessitated by the dynamic behavior that our first-level nodes demonstrate and is commonplace in WMNs.

4.2 *FCN* Node Arrival

FCN nodes may join (or depart) at any time. The main challenge is to contain all relevant update operations to a small portion of the network and keep the number of existing *FCNs* involved in at a minimum. Using a complex triangulation scheme would require costly maintenance operations and the notification of many existing *FCN*s. Instead, we opted for a simpler scheme that restricts the number of *FCN*s currently in operation that have to be notified when a new *FCN* arrives. Our approach requires only the notification of nodes that are geographically close to the newly-arrived node.

The process that takes place when a new node arrives and tries to join the network is the following: the arriving node can either broadcast a message to neighboring or those found in-line-of-sight nodes declaring its intention to join the network. The broadcasted message arrives only to *FCN*s geographically-close whereas first-level nodes far away are never involved. Here, the responding *FCN*s either contain the joining node in one of the triangles they are a vertex of, or they can infer that the new node is outside the convex-hull structure. Should the broadcast proves to be futile, a bootstrap node for the incoming *FCN* has to be notified. In this case, the bootstrap node indicates an existing *FCN* that must be contacted by the joining node so that the join procedure may commence.

An existing *FCN* selected to handle the join process adds the incoming *FCN* in its *OFT* and asks the new node to reciprocate. The new *FCN* can either be located inside a triangle or outside the convex hull. In the first case, the existing *FCN* notifies its other two counterparts in the triangle to add the incoming node to their respective *OFT*s. In the second case, the newly-arrived node has to connect with all *FCN*s on the convex hull that are visible. For every k new connections the arriving *FCN*establishes, $k-1$ new triangles are formed and the convex hull is expanded. Once the handling *FCN* has added the incoming node into its *OFT* table, it forwards the request to its neighbors that are part of the convex hull. A connection between two nodes is established if both have the other's identifier and geographic location stored in their respective *OFT*s. Algorithm 1 outlines the above process.

It is worth pointing out that existing *FCN*s that are geographically closest to an incoming node are not always selected for carrying out the joining process. This occurs due to the fact that our approach does not warrant that a handling *FCN* cannot always be the one physically closest to the new node. Nevertheless, this choice offers a pragmatic way in readily maintaining the triangulation at all times.

Algorithm 1 New Node Arrival(*FCN* node: n_{new})

```
BEGIN
$n_{new}$ broadcasts a join request
$handler \leftarrow$ findSuitableNode($n_{new}$)
connect $n_{new}$ with $handler$
if $n_{new}$ is outside triangulation then
    connect $n_{new}$ with all visible nodes
else
    find all nodes-vertices of the triangle $n_{new}$ is in
    connect $n_{new}$ with all these nodes
end if
END
```

4.3 *FCN* Node Departure

When an *FCN* departs from the network, the triangulation may not remain intact. If the departing node is not a part of the convex hull, after its departure a polygon is formed by the *FCN*s found in the finger table of the departing node. On the other hand, if the departing *FCN* belongs to the convex hull, then it is likely that the resulting hull is not convex any more. The latter has to be properly addressed. In any case, it is clear that before disconnecting, a node needs to ensure that the triangulation structure remains intact after its departure.

Firstly, let us consider the case in which the departing *FCN* is not a part of the convex hull and consequently, the *OFT* entries of the departing node form a polygon. The departing *FCN* needs to execute a polygon triangulation algorithm and compute the new edges that have to be introduced. Subsequently, the departing *FCN* notifies its counterpart to establish proper connections in pairs; this will form the missing edges. Finally, the departing node is purged from the respective *OFT*s of the *FCN*s it was connected with so far.

The situation is different when the departing *FCN* is on the convex hull. Here, the departing node has to compute the lower hull of its neighbors as this will guarantee convexity of the produced hull. After this procedure, internal polygons may be also formed. For each one of these polygons, the departing *FCN* needs to compute the missing edges. Finally, before the node ceases operation it notifies all the involved *FCN*s to properly update their respective *OFT*s. In both above cases, the exit routine is executed by the departing node.

4.4 *FCN* Node Failure

Occasionally, there are instances in which *FCN*s fail and forcefully disconnect from the network. Thus, a failing *FCN* is unable to maintain the triangulation structure and the polygon that is left behind must be rectified. In such a case, *FCN*s that are vertices of the polygon have to "realize" that their neighbor is down and self-adjust the incurring situation.

The first step in restoring the triangulation scheme is to identify the failing node. To achieve thus, every *FCN* in the network must periodically send messages to its neighbors to check whether they remain alive. Also, an *FCN* can become aware that one of its neighbors is down when it sends-out a message related to the arrival/departure of node. In any case, when a failure is discovered, a restoration process must first take place before any other operation occurs.

When a failure occurs and has been identified, each of its neighbors may know up to three entries of the OFT table in the failing node. If we designate the failing node as k in a neighbor's OFT, entries k-1 and k+1 correspond to *FCN*s that will have to be directly connected. Nevertheless such a coupling is only feasible provided that it does not violate the triangulation constraint. If the connection is established successfully then all *FCN*s involved purge the failing node from their respective *OFT*s.

The above recovery operation can never be performed by nodes in the convex hull as this might yield connections that do not comply with the triangulation constraint. As we have messages to check the liveliness of nodes referenced in *OFT*s, it is safe to assume that, at a certain point in time, all adjacent *FCN*s will either discover or be notified about the failure. The nodes will undertake incremental corrective action in the same order they discovered the failure. Subsequently, after a period of time, the triangulation will be restored successfully.

4.5 Mobile Node Management

When a mobile node enters the network, it must locate an *FCN* that will handle its request. This process is similar to the one described in Section 4.2. The suitable node is an *FCN* that contains the newly arrived mobile node in one of its triangles. If the mobile node finds itself outside the triangulated area, the suitable *FCN* is part of the convex hull which is also able to identify the mobile node as located outside the hull.

As soon as the *FCN* that will handle the join request is determined, the respective process commences. If the mobile node exists inside one of the triangles of the *FCN* in question, then the handling node checks which vertex of the triangle is closest to the geographic location from which the join request was launched. Hence, the roamer is connected to its closest *FCN* and the latter becomes the "parent" of the mobile node. If the roamer is outside the triangulation structure, then it must be connected to its closest *FCN* node on the convex hull. Thus, the handler of the join request, decides whether to forward the request to one of its two neighbors on the convex hull by comparing respective distances from the mobile node; the final recipient of the forwarding is the *FCN* which maintains the shortest distance from the mobile incoming node. In this way, only a small portion of the convex hull is checked until the geographically closest node can be located.

The departure of a mobile node from the network requires that the roamer only contacts its parent and requests its own deletion; it then can disconnect safely as the is no structure and/or additional information to be maintained.

Except for arrival and departure of a mobile node, there is an additional process that ensures the continuous monitoring of the moving nodes. As a mobile node changes its location, it has to keep its parent continuously updated on its new location. Therefore, it becomes evident that at some point in time, the mobile node may move outside of all the triangles that its parent node is a vertex of, find itself located outside the convex hull or move from outside the convex hull inside a triangle. In any case, all these location updates are monitored and the overlay reacts accordingly.

Every time a location update event is received by the parent, the corresponding *FCN* consults with an appropriate daemon that indicates whether is still the most suitable parent. This is possible as the parent know its *FCN* counterparts of triangle in which the roamer is. The performed checks are similar to those deployed when searching for the parent node upon a new mobile node arrival. As soon as a change-of-parent decision has been made, the original *FCN* hands over the care of the mobile node to the new parent. Since the location updates arrive constantly and likely with high frequency, there is only a small number of hops required for parent-switch operation to take place. In this way, we ensure that the distance between a moving node and its parent is always kept minimum.

Algorithm 2 Mobile Node Arrival($MobileNode\ mn_{new}$)

```
BEGIN
mobile node $mn_{new}$ broadcast a join request
$fcn_{par} \leftarrow findSuitableServiceNode(mn_{new})$
if $fcn_{par}$ is in convex hull then
  check neighbors of $fcn_{par}$ on the convex hull
  if $\exists\ fcn_n$ in convex hull AND geographically closer to $mn_{new}$ then
    forward the request to $fcn_n$
  else
    connect with $mn_{new}$
  end if
else
  check the vertices of the triangle $mn_{new}$ is in
  if $\exists$ vertex $fcn_n$ geographically closer to $mn_{new}$ then
    forward the request to $fcn_n$,
  else
    connect with $mn_{new}$
  end if
end if
END
```

4.6 Nearest–Moving Neighbor Queries

Two mobile nodes are nearest-moving neighbors if they maintain the shortest geographic distance among all participating mobile nodes. The distance between two roamers is computed using their geographic coordinates provided during each node's most recent location update. The actual current location of a mobile node though may not match the coordinates that will be used to calculate its distance from another roamer. An accurate computation of the nearest-moving neighbor cannot be guaranteed, unless additional information related to the roamer's motion pattern is exploited [5]. As our proposal gives priority on performance over accuracy, our algorithm attempts to find a solution quickly, by utilizing only a small, usually constant, number of nodes. We do not provide any guarantees that the answer to a query will be the optimal solution. Yet, in Section 5, we will show that our approach finds the optimal solution very frequently and that in general, it presents a good trade-off between quick response and accurate results.

A mobile node, submits the query to its parent *FCN* along with its current geographic location. The first step is to ensure that the query will be processed by the most suitable *FCN* in the neighborhood. Thus, a change-of-parent procedure may take place before commencing the query evaluation at the appropriate *FCN*. In our approach, we check at least three *FCN*s in order to find a solution. If the mobile node is located inside a triangle, the query is processed by the respective parent node at the time and its counterparts of the triangle. Otherwise, we conclude that the parent of

Algorithm 3 Nearest-moving neighbor(MobileNode *mn*)

```
BEGIN
$fcn_{par} \leftarrow$ current parent $FCN$ of $mn$
if $fcn_{par}$ is not the most suitable parent of $mn$ then
  initiate parent-switch procedure
  forward request to the possible parent and STOP
end if
if $m$ is inside one of the triangles of $fcn_{par}$ then
  $cand \leftarrow$ set of mobile node entries of the $FCN$s that
    constitute the triangle $mn$ is in.
else
  // $fcn$ is on the convex hull.
  $cand \leftarrow$ set of mobile node entries of the $FCN$s that
    are neighbors on the convex hull
end if
$m_{closest} \leftarrow$ node in $cand$ closest to $mn$
if $m_{closest}$ has been set then
  return $m_{closest}$ and STOP
end if
$fcn_{next} \leftarrow$ closest $FCN$ that hasn't processed the request
forward request to $fcn_{next}$
END
```

the mobile node is part of the convex hull. The query is then processed by the parent as well as the next and the previous *FCN*s on the convex hull. The answer to the query is the roaming node which is the closest to the coordinates the query originated from, among all the mobile nodes monitored by the *FCN*s involved. If a solution is not found among the three selected *FCN*s, the query is forwarded incrementally to other close-by nodes. This is a step-by-step procedure and when a solution is found it stops immediately. Finally, the solution is directly dispatched to the requesting mobile node by the last *FCN* node that was involved. The exact procedure for answering a nearest-moving neighbor query is presented in Algorithm 3

Consider again the instance of our layout in Fig. 2. Let $M8$ be the node that submits a nearest-moving neighbor query on its parent $S7$. The *FCN* nodes that will be checked in order are: $S7$, $S8$ and $S4$. $S4$ –the last node to be check– will be able to produce the solution $M10$; the latter will be dispatched to $M8$. On the other hand, if the node that submits the query is $M2$, the *FCN* nodes that will be utilized are in order the following: $S6$, $S3$, $S1$ and $S4$. $S1$, which is the last node of the triangle that will be checked. Up to this point, there is no solution to return. Thus $S1$ forwards the query to $S4$, which finds the closest neighbor $M4$ and sends the solution directly to node $M8$. The query was forwarded to an additional node as no *FCN* among the vertices of the triangle $M2$ finds itself in, could provide an answer.

5. EXPERIMENTAL EVALUATION

We evaluate our proposed system through simulation and provide preliminary results in this section. We use *SICSIM* [7], a discrete event-based simulator that can model the salient properties of the underlying network including network latency and bandwidth of links. *SICSIM* also provides us with an abstract layer in which we can define the generalized functions a node must implement. By extending this abstract layer, we have implemented geographic location-aware nodes and all their required operations. In addition, we have defined an operation that it is being invoked periodically and it is used to realize the mobile nodes that dispatch frequent updates about their location to their respective parent nodes. *SICSIM* also provides a monitoring system mechanism that enabled us to take snapshots of the network. This function is used for finding the actual geographically-closest neighbor of a given mobile node at a certain point of time.

To model the movement of the nodes, we implemented two events in the simulator: a location change event and a location update event. In the location change event, the position of the roaming node changes but its parent node is not notified. The notification occurs only at a location-update event. We designate the ratio of a location change event to a location update event as 10:1. In this way, we model the movement of roaming nodes and get discrete updates on their location while simulating their continuous movement.

In order to have a comparison reference using *SICSIM*, we implemented two additional systems that are evaluated along with our proposed overlay network. The first is based on a fully-centralized architecture. There is only one service node that monitors all mobile nodes and is also responsible for answering queries. The key characteristic of this system is that only two hops are required to answer a query. The mobile node sends the query to the server, the server processes it and answers back with the result. In this experiment, we don't take into consideration the processing time in the central server and we only focus on the time required for each hop. The second system we implement is a variation of the *Egoist* overlay [10]. Each *Egoist*-node selects a set of k neighbors by using a strategy that minimizes a local cost function. In the variation we employed, we define the cost function as the geographic distance between two nodes. The selfish neighbor selection that is used by *Egoist*, makes the particular variation match the behavioral pattern of a WMN. In our implementation, we suggest that when a new fixed location overlay node arrives, it connects with its 3 closest neighbors. We chose $k = 3$ in order to make the system as similar to ours as possible. When a mobile node arrives, it selects its closest service node and connects to it. The same happens when a location update event occurs, so that the mobile node is always connected to its closest *FCN* node. As for query processing, if the parent of a mobile node can answer then it does. Otherwise, the query is forwarded to the closest, to the mobile node, *FCN* node. Notice that a query is always forwarded to one service node at a time, similarly to our system. In this section, we shall refer to this system as *Egoist*(3).

5.1 Cost Metrics

The cost metrics that we have incorporated in our system are the following:

- *Hops per query:* the average number of transmitted messages from one node to another, in order to answer a query. A hop can be a message sent from a roaming node to a service node and vice versa, or a message sent from a *FCN* node to another one.
- *STU per query:* the average simulation time units that the system needs to evaluate a nearest neighbor query. Although we cannot relate simulation time to real time, we use STUs as a comparison metric among simulated configurations.

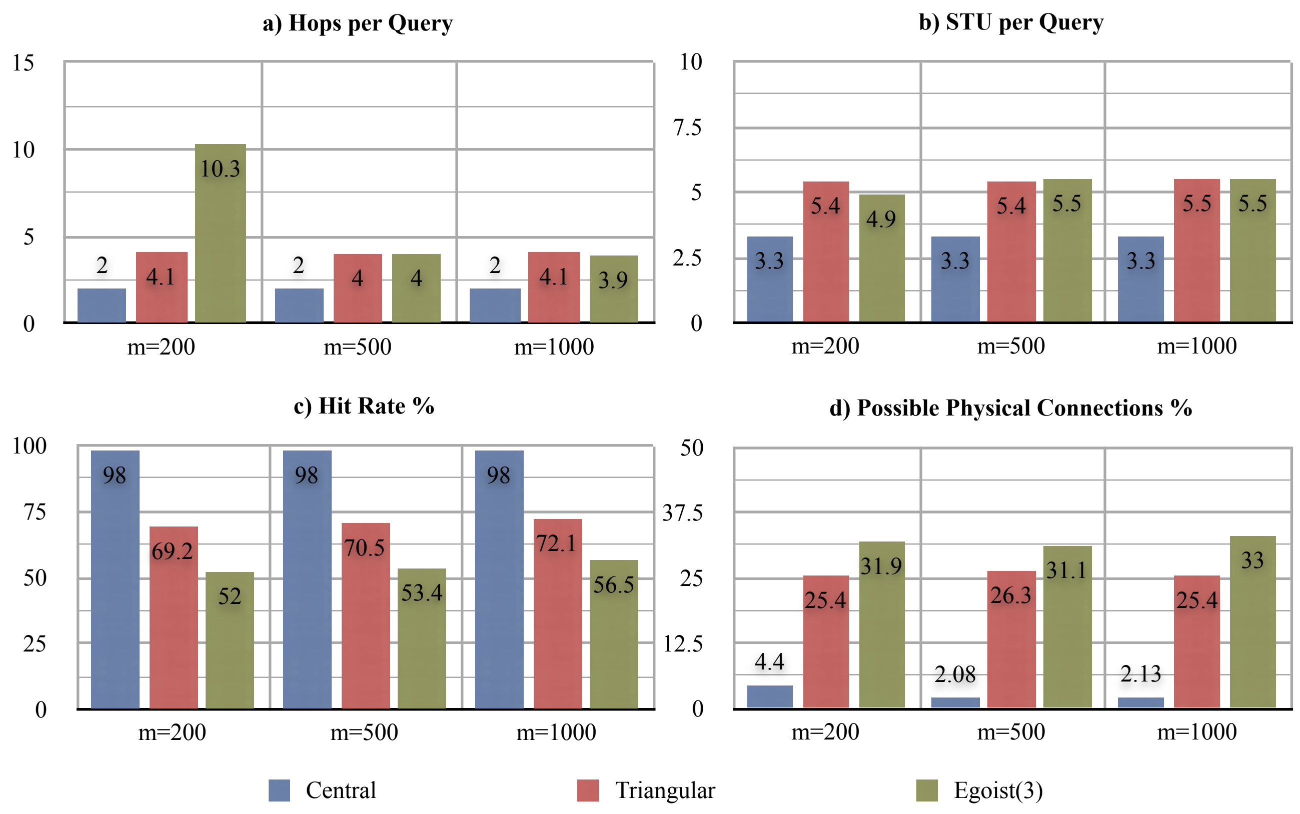

Figure 3: Visualization of the average measurements of the experimental results for 20 simulation executions.

- *Hit rate:* as we already mentioned, there are cases when the system provides an answer to a nearest neighbor query that is not the actual closest neighbor. Hit rate shows success rate of each system in finding the actual nearest neighbor at the moment the result is returned. We can find the actual closest mobile node by taking a snapshot of the network, as the simulator we use provides such function.

- *Physical Connections:* we assume that the static nodes are Wi-Fi hotspots, thus, allowing direct connections between *FCN*s and mobile nodes. This metric counts the number of connections between nodes that they could be physical. This may happen if a mobile node is within the range of the service node. We set the range of a service node to be equal to the range of an average omni-directional Wi-Fi antenna (around 800 meters). Moreover, due to the movement of the mobile nodes, the number of possible physical connections does not remain constant. Thus, the results we present show the average number of the possible physical connection in the network during the execution of a simulation.

It is worth mentioning that the hops can be also used to measure the resource utilization in each system. Since we ensure in the two overlays that a query is always forwarded to one service node at a time and cannot be forwarded to the same node twice, the following always applies.

DEFINITION 3. *If the number of hops required to process a query q is n, the number of overlay nodes utilized in order to process q is always equal to n-1.*

If a centralized structure is used, the number of hops is always equal to 2 (request/response). In the case of the overlay networks the number of hops varies. In our system, the minimum number of hops is 4 and occurs if a solution is found by using exclusively the vertices of the triangle within which the mobile node launched its query. In *Egoist(3)*, the minimum number of hops is 2 and happens when the query is answered by the parent node of the mobile node submitting the query. Finally, in both overlays the maximum number of hops that may be required is $n+1$, where n is the number of *FCN* nodes in the network.

5.2 Experimental Results

In our experimentation, we compared our system with the two systems described above. For the two overlays, we created a set of 20 fixed location service nodes distributed uniformly inside the designated area. We evaluated the 3 systems in 3 different scenarios, for 200, 500 and $1,000$ roaming nodes which are also distributed uniformly. These numbers signify how crowded the designated area is. We executed 20 simulation runs for each scenario, in which random mobile nodes submit a total of $4,000$ nearest-neighbor queries, and we computed the average values for all the metrics mentioned above. Fig. 3 depicts the outcomes of our experimentation.

Fig. 3(a) and (b) show the average hops per query and the average execution time per query for the three examined configurations. The centralized structure is clearly ahead as only two hops are required for processing a query. In contrast our approach and *Egoist(3)* require more hops. The average number of hops per query shows us the *FCN* node utilization for answering queries. Of course, in the centralized system there is only one service node and so the utilization is always 100%. But on our system and *Egoist(3)* that is not the case. The average amount of hops in our system is around 4. This means our system processed the majority of the queries submitted during the simulation process, by utilizing only three nodes. In other words, almost all queries in all scenarios were processed while the service node utilization was the minimum. For 500 and $1,000$ mobile nodes, the behavior of *Egoist(3)* is similar, but for 200 nodes, the service node utilization increases significantly. This leads us to the conclusion that our system behaves better than the *Egoist(3)* in less crowded environments.

Fig. 3(c) show the average hit rate that each of the three implemented systems achieve. The centralized structure achieves clearly a better hit rate than the two overlays. Our structure also achieves a high hit rate overcoming easily the *Egoist(3)* overlay and approaching the centralized structure, especially when the environment in which the system works is crowded. On the other hand the *Egoist(3)* overlay does not manage to achieve a high hit rate even though the geographic proximity between the nodes is strictly ensured.

Finally, Fig. 3(d) depicts the average number of possible physical connections in each case. In *Egoist(3)*, due to the fact that each mobile node always chooses to connect with its closest service node, the number of possible physical connections is the largest possible. Our system follows closely in terms of possible physical connections, while the centralized structure is way behind. Due to the nature and the organization of our approach and, since we do not use a strict triangulation scheme, it is not always possible for a roaming node to be connected with its closest *FCN* node. Yet, our system is proved to be a good match with the geographic topology of the participating service and mobile nodes.

In terms of hops, our system approaches the performance of a centralized system and the number of hops per query remains constant. We manage to achieve a high hit rate as our system finds the best possible solution three out of four times. Moreover, our structure provides a good match with the underlying topology since the number of physical connections approaches the highest possible value.

6. CONCLUSION AND FUTURE WORK

We introduced a geography-aware service overlay network for monitoring moving objects and evaluating nearest-moving neighbor queries. Our paper makes three contributions as it: *a)* presents a strategy for managing moving objects by exploiting the capabilities of a self-organizing and self-maintained structured overlay network, *b)* provides a nearest-neighbor query processing algorithms that, in the majority of cases, requires constant time to be executed and *c)* proves that the organization of the proposed system matches to a good degree the underlying geographic topology by exploiting the exact location of the overlay nodes. Our approach can support any application that requires monitoring and query processing over moving objects in wireless metropolitan networks or similar operating environments. When used over an existing infrastructure, our approach does exploit the network capabilities of the underlying topology.

We plan to extend our work by pursuing a number of directions: firstly, we will seek ways to improve the accuracy of our nearest-moving neighbor algorithm and enable efficient range queries. Secondly, we plan to investigate effective approaches for providing fault tolerance and rapid recovery. Finally, we intend on realizing a full-fledged implementation of our overlay and experiment in larger scale.

7. REFERENCES

[1] Hüseyin Akcan, Vassil Kriakov, Hervé Brönnimann, and Alex Delis. Managing cohort movement of mobile sensors via GPS-free and compass-free node localization. *Journal of Parallel and Distributed Computing*, 70(7):743–757, 2010.

[2] Filipe Araújo and Luís Rodrigues. Geopeer: A location-aware peer-to-peer system. In *Proc. of the 3rd IEEE Int. Symposium on Network Computing and Applications (NCA'04)*, pages 39–46, Washington, DC, 2004.

[3] Bugra Gedik and Ling Liu. Mobieyes: A distributed location monitoring service using moving location queries. *IEEE Transactions on Mobile Computing*, 5:1384–1402, October 2006.

[4] George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On indexing mobile objects. In *Proc. of the 18th ACM PODS*, Philadelphia, PA, 1999.

[5] Iosif Lazaridis, Kriengkrai Porkaew, and Sharad Mehrotra. Dynamic queries over mobile objects. In *Proc. of the 8th Int. EDBT Conf.*, pages 269–286, London, UK, 2002.

[6] Julio C. Navas and Tomasz Imielinski. GeoCast – geographic addressing and routing. In *Proc. of the 3rd ACM/IEEE Int. Conf. on MobiCom*, Budapest, Hungary, 1997.

[7] Amir H. Payberah, Fatemeh Rahimian, Tallat Mahmood Shafaat, Ali Ghodsi, and Seif Haridi. SICSIM: a discrete event, flow-level simulator. `http://www.sics.se/~amir/sicsim/#doc`, September 2011. Swedish Institute of Computer Science.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, pages 161–172, San Diego, CA, 2001.

[9] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. GHT: a geographic hash table for data-centric storage. In *Proc. of ACM Int. Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, 2002.

[10] Georgios Smaragdakis, Vassilis Lekakis, Nikolaos Laoutaris, Azer Bestavros, John W. Byers, and Mema Roussopoulos. EGOIST: Overlay routing using selfish neighbor selection. In *Proc. of the ACM CoNEXT Conf.*, pages 6:1–6:12, Madrid, Spain, 2008.

[11] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM Conf.*, pages 149–160, San Diego, CA, 2001. ACM.

[12] Jianjun Zhang, Gong Zhang, and Ling Liu. GeoGrid: A scalable location service network. In *Proc. of the 27th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'07)*, Washington, DC, 2007.