Disclosure-free GPS Trace Search in Smartphone Networks

Demetrios Zeinalipour-Yazti[+], Christos Laoudias[+], Maria I. Andreou[#] and Dimitrios Gunopulos[‡]

[+] University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus
[#] Open University of Cyprus, P.O. Box 24801, 1304 Nicosia, Cyprus
[‡] University of Athens, 15784 Athens, Greece
dzeina@cs.ucy.ac.cy, laoudias@ucy.ac.cy, andreou@ouc.ac.cy, dg@di.uoa.gr

***Abstract*— In this paper we present a powerful distributed framework for finding similar trajectories in a smartphone network, without disclosing the traces of participating users. Our framework, coined SmartTrace, exploits opportunistic and participatory sensing in order to quickly answer queries of the form: *"Report the users that move more similar to Q, where Q is some query trace."* SmartTrace, relies on an in-situ data storage model, where geo-location data is recorded locally on smartphones for both performance and data-disclosure reasons. SmartTrace then deploys an efficient top-K query processing algorithm that exploits distributed trajectory similarity measures, resilient to spatial and temporal noise, in order to derive the most relevant answers to Q quickly and efficiently. We assess our ideas with realistic and real workloads from Microsoft Research Asia and other sources. Our study reveals that SmartTrace computes the desired results with 74% less energy consumption and 13% faster than its centralized and decentralized counterparts. Our experimental results also confirm our analytical study.**

I. Introduction

The widespread deployment of smartphone devices featuring geo-location (e.g., AGPS, Cell tower and WLAN positioning) and other sensing capabilities (e.g., proximity, ambient light, accelerometer, camera, microphone, etc.) along with Internet connectivity through WLAN, WCDMA/UMTS(3G), HSPA(3.5G) and LTE/WiMAX(4G) networks, have brought a revolution in location-oriented mobile applications and services. IMS Research and Comscore reported over 225M smartphone sales in February 2010 (i.e., RIM, Apple, Microsoft, Google and Palm) and according to the Focal Point Group, handheld smart devices (including mobile phones and PDAs) could number 1 billion in 2010. We define a Smartphone Network as "*a set of smartphone devices that communicate in an unobtrusive manner, without explicit user interactions, in order to realize a collaborative or social task.*"

There is already a proliferation of innovative applications founded on smartphone networks. One example is opportunistic and participatory sensing [14], [3], [10], where applications can task mobile nodes in a given region to provide information about their vicinity using their sensing capabilities. Another example is road traffic delay estimation [33] using WiFi beams collected by smartphone devices rather than invoking expensive GPS acquisition. On the social side, Google Latitude[1] enables users to track the places they and their social network have visited. The given service already reports over 3M enrolled users and over 1M active users, despite the controversial privacy concerns. Similarly, mobile social networking applications like Foursquare, Gowalla and Loopt enjoy enormous success in the Smartphone community and academic efforts in this direction are also underway [31].

In this paper, we present a powerful distributed trajectory similarity search framework, coined SmartTrace, which can be utilized as a middleware service to a smartphone stack in order to promptly answer queries of the form: *"Report the users [2] that move more similar to Q, where Q is some query trace.* The notion of *similarity* captures the trajectories that differ only slightly, in the whole sequence, from the query Q. In particular, our framework is suitable for intelligent transportation systems [39], [33], social networking applications for smartphones [31], [40], habitant monitoring systems [25] and others.

An indicative query supported by our framework might be: *"Find whether there is a cycling route from the Metropolitan Museum of Art in Manhattan, through central park to the Juilliard School"*), or *"Find which Zebras moved more closely to Zebra named Abby before it got injured" [25].* There are already centralized trajectory search services such as GeoLife[3], GPS-Waypoints[4], ShareMyRoutes[5], and their academic counterparts [22], to perform this kind of querying. However, these services store user's trajectories on a centralized or cloud-like infrastructure. On the other hand, the techniques proposed in this work are decentralized and maintain the data *in-situ* (i.e., on the smartphone that generated the data). When a query emerges, we collect a set of scores from participating nodes (as opposed to collecting their location continuously) and derive the answer intelligently based on these scores only without ever unveiling the target trajectories to the query processor. While this cannot take advantage of global knowledge structures available in a centralized setting

[1] Google Latitude, 03/2011, http://www.google.com/latitude
[2] We shall use the terms *User*, *Object* and *Smartphone* interchangeably
[3] GeoLife, 03/2011, http://research.microsoft.com/en-us/projects/geolife/
[4] GPS Waypoints, 03/2011, http://www.gps-waypoints.net
[5] ShareMyRoutes.com, 03/2011, http://www.sharemyroutes.com/

(e.g., catalogs, indexes, etc.), our setting has the following important advantages:

i. Smartphones have expensive communication mediums, thus by continuously transferring massive amounts of data to the query processor can both deplete the precious smartphone battery faster, increase user-perceived delays (as shown in Section II-B and [28]), but can also quickly degrade the network health (e.g., consider the overload of AT&T's cellular infrastructure in 2009 by I-Phone users accessing data services[6].)

ii. Continuously disclosing user positional data to a central entity might compromise user privacy in serious ways (e.g., Latitude associates the user's location with a Google account and shares that info with selected friends.) This creates services that have been criticized seriously in recent years[7]. Finally, continuous data collection might not be possible as several regions are still solely relying on 2G networks and many of them do not have WiFi coverage either, thus are disconnected from the Internet for extended periods of time.

In the proposed SmartTrace framework, the tuples of each target trajectory A_i, are compared with the points of Q within some temporal and spatial window. SmartTrace, circumvents expensive and massive similarity executions by running an inexpensive linear-time (i.e., $O(|A_i|)$-time) computation on the smartphones in a pre-processing step. It then uses an iterative top-K processing algorithm in order to iteratively identify the K most similar trajectories to Q, without ever pulling the target trajectories to the centralized query processor. We validate our algorithm both analytically and empirically, showing that it offers many desirable properties that have not been studied previously. Our contributions are summarized as following:

- We propose an innovative framework for disclosure-free query processing in Smartphone Networks founded on in-situ data storage and the SmartTrace algorithm.
- We provide a theoretical analysis of our framework, using a new energy and time complexity model we define for smartphone networks. Our analysis shows that our algorithm provides high response time with low energy consumption without ever unveiling the complete trajectories to the query processor.
- We confirm our analytical study with extensive experimentation on both realistic and real datasets and the utility of our system using a prototype system in Android.

The remainder of the paper is organized as follows: Section II provides our system model and formulates the problem. Section III provides the related work of our studied problem, Section IV presents the SmartTrace framework and Section V a detailed performance analysis of our framework. Section VI, presents our experimental methodology and experimental results, while Section VII concludes the paper.

[6]"Customers Angered as iPhones Overload AT&T", Jenna Wortham, The New York Times (online), Sept. 2nd, 2009.

[7]"Google Apologizes for Buzz Privacy", David Coursey, PC World Business Center (online), Feb. 15th, 2010.

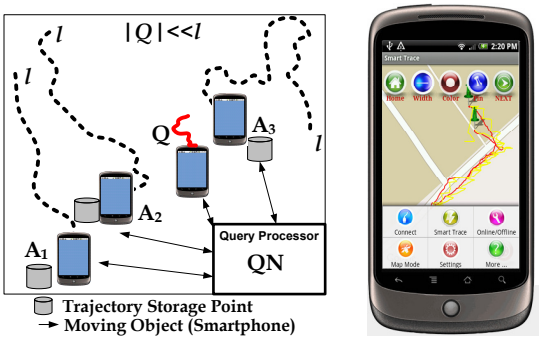

Fig. 1. **Left:** The system architecture of the SmartTrace Query Processing Framework; **Right:** Demonstration of the SmartTrace framework we have developed for Android-based smartphones [12].

II. Problem Formulation

In this section we provide the notation used throughout the paper. Specifically, we formalize our system model and also show why such a model is representative for a smartphone network using data we profiled on a real device.

A. System Model

Let $\{A_1, A_2, ..., A_m\}$ denote a set of m smartphone users moving in the xy-plane (see Figure 1 left). At each discrete time instance, object A_i ($\forall i \leq m$) generates a spatio-temporal record $A_{ij} = (t_{ij}, x_{ij}, y_{ij})$, where t_{ij} denotes A_i's temporal dimension and x_{ij}, y_{ij} A_i's two spatial dimensions. Consequently, a trajectory can be thought as a continuous sequence of l such records, i.e., $A_i = (A_{i1}, A_{i2}, \cdots, A_{il})$ (l also denoted as $|A_i|$). Here the complete trace of a trajectory A_i is stored in its entirety locally on a smartphone (e.g., on flash memory). A smartphone is a battery-operated device, thus has limited computational resources and also networking operations are expensive in terms of energy, due to MAC-layer collisions and re-transmissions. Additionally, the link is asymmetric with the uplink being much slower than the downlink (typically one order of magnitude).

Now let us consider an arbitrary snapshot similarity query $Q = (Q_1, Q_2, \cdots, Q_f)$, where $f << l$ (f also denoted as $|Q|$), which aims to uncover the K most relevant trajectories to Q, for a user-defined constant K. Q might either be initiated by a smartphone and propagated towards the querying node QN, or might be initiated at QN. Finding the similarity between Q and A_i ($i \leq m$), calls for specialized similarity measures that can operate under spatial and temporal noise. For instance, if either of the trajectories moves earlier in time or features some slight deviation in its spatial movement, then the measure must still yield high levels of similarity. Although we shall explain these similarity measures more precisely in the next section, let us for the moment assume that there is some function $FullM(Q, A_i)$, which performs the trajectory comparison between Q and A_i accurately, but at a high computational cost. $FullM(Q, A_i)$ returns a score in the range [0..1] (where 1 denotes highest similarity). In a smartphone network setup, $FullM(Q, A_i)$ can either be conducted in a centralized fashion (i.e., after transferring all

TABLE I

ENERGY PROFILING OUR HTC HERO SMARTPHONE: COMPUTATION AND NETWORK TRANSFER ARE EXPENSIVE IN TERMS OF ENERGY AND TIME.

Basic Operation on Smartphone	**Power (mW = mJ/s)**
CPU Idle (OS running)	175 mW
CPU Busy (Processing)	**369 mW**
WiFi Idle (Connected)	38 mW
WiFi Busy (Uplink 123Kbps, -58dBm)	**600 mW**
LCD Brightness (economy mode)	300 mW
Function ($len(A_i) = 100K$ 18-bytes points)	**Time**
$Transmit(A_i, server)$ (from Smartphone)	**117 seconds**
$FullM(Q, A_i)$ (on Smartphone)	**111 seconds**

m trajectories to QN), or in a decentralized fashion (i.e., after having each smartphone conducting $FullM(Q, A_i)$ locally and then identifying the desired result). As we will show in our analytical and experimental analysis, the *Centralized* approach performs really bad in terms of energy and response time but also comes at a higher privacy risk, as the users need to share their complete trajectory with QN. The *Decentralized* approach on the other hand, performs also extremely bad in terms of energy consumption as it invokes expensive trajectory comparison metrics on all smartphone participants. The SmartTrace approach we propose in this work performs well both with respect to response time and energy, but also does so without ever revealing the complete user trajectories to QN (i.e., it only returns the matched subsequence of length $|Q|$, where $|Q| << l$.)

B. Quantifying the System Model

In this subsection we present a set of real measurements we obtained from our prototype system implemented for Android smartphones. We use these measurements to explain why our system model captures precisely the intrinsic characteristics of a smartphone network. Using our system model, we will be able to analytically explain our convergence and performance properties. Our experimental platform is an Android-based HTC Hero 2.1 smartphone equipped with 802.11b/g and a Qualcomm MSM 7200A 528 MHz processor. For the evaluations that follow, we use the actual software components of our SmartTrace application implemented in JAVA [12] (see Figure 1 right), but also benchmarking tools like 3gtest [1] and PowerTutor [29].

To motivate these measurements, let us first consider the GeoLife [39], [40] dataset by Microsoft Research Asia. The dataset captures outdoor movements of 165 people at varying granularities. In particular, 95% of the dataset refers to a granularity of 1 sample every 2-5 seconds or every 5-10 meters. The average trajectory length in GeoLife is over 191K data points per user (largest: 2M data points) having respective sizes of 11,7MB (largest: 111 MB). Notice that by sampling a GPS sensor every 2 seconds for one year, and assuming no failures or downtimes on the smartphone, would yield over 15M points, occupying more than 270MB of storage. Multiplying these numbers by the thousands or millions of users enrolled to the services under discussion should quickly provide an insight into the incurred bandwidth and energy costs, shall we decided to collect them at *QN*. Now let each smartphone user hold one of these large trajectories of length l (denoted as $A_i, i \leq m$). The objective is to find the K users moving more similarly to Q, where $|Q| << l$. For instance, Q might look for *"friends that traveled from Greenwich Village, in lower Manhattan, through Little Italy and finally to the Financial district during 8:00-9:00"*. In the provided scenario, the temporal length of Q is much smaller (i.e., 1 hour) than the temporal length of the target trajectories A_i in our setting (i.e., 2 years). Also, Q is expressed at a coarser granularity than the fine-grain spatial points captured in each A_i.

For the centralized approach, which mandates the transfer of all trajectories to QN prior to query execution, we isolated the cost of uploading a GPS trajectory of 100,000 data points from the smartphone to a TCP socket server over 802.11b with an uplink of 123kbps (as measured by [1]). The given operation took us 117 seconds (i.e., almost 2 minutes!), draining over 70 Joules of precious energy. Furthermore, the centralized approach requires that each user shares its complete trajectory with the query processor, which might be unacceptable from a privacy stand-point. For the decentralized approach, which mandates the comparison of Q against A_i on every smartphone participating in the query resolution, we isolated the time and energy cost for computing $FullM(Q, A_i)$ on a single smartphone unit. This operation took us 111 seconds (again almost 2 minutes!) and amounted to over 41 Joules of energy. This happens as the trajectory similarity functions we use (i.e., LCSS presented in Section IV-B) are computationally expensive (i.e., $O(\delta \cdot l)$, where δ is the temporal window in which we conduct the search and l the length of the trajectory.) Notice that both aforementioned costs are accounted for each device participating in a query, thus the aggregated costs are much higher. Table I summarizes our findings.

Consequently, we make the following observations: i) The asymmetric download/uplink bandwidth in these environments severely hampers the massive upload of data to a server, even under trajectory compression techniques; and ii) Local processing is an expensive operation with respect to energy consumption and must be avoided whenever possible.

III. RELATED WORK

In this section we provide related research work for both spatio-temporal query processing and distributed top-K query processing, both of which lie at the foundation of SmartTrace.

Spatio-temporal queries have been an intense area of research over the years [2], with the development of efficient access methods [21], [35], [26], [36] and similarity measures, such as *Dynamic Time Warping (DTW)* [6], the *Longest Common Subsequence (LCSS)* [13], variants of L_p-norms such as *Edit Distance with Real Penalty (ERP) [23]* and *Edit Distance on Real Sequences (EDR)* [24]. These metrics have been proposed for predictive [32], historical [35] and complex spatio-temporal queries [18]. All these techniques, as well as the frameworks for spatio-temporal queries [5],

[34], [20], work in a completely centralized setting. The same applies to online trajectory searching services such as GeoLife, GPS-Waypoints, Sharemyroutes and their academic counterparts [22], which assume that user trajectories are aggregated and stored on a centralized or cloud-like infrastructure. Notice that for a centralized setting, the problem definition is considerably different, than the decentralized scenario we consider in this work, as the query processor maintains all trajectories locally and global-knowledge statistics can be maintained in local catalogs. Additionally, in a centralized setting the query processor can utilize spatial or spatio-temporal trajectory index structures, such as the R-trees (e.g., utilized in [22] and [16]), STR-trees or TB-Trees [30], in order to speed up the retrieval answers. On the other hand, in a decentralized setting none of these comes at no additional cost.

In our previous work in [37], we have already paved the way towards trajectory processing techniques in a distributed manner (i.e., without percolating each and every user geolocation to a central authority). However those were both agnostic in terms of energy and time constraints that arise in a smartphone network but also in respect to the trajectory trace disclosure issues (i.e., they assumed that the query processor can arbitrarily access the distributed trajectories). More importantly, our previous work assumed that trajectories where vertically fragmented across n distributed sites (i.e., each distributed site holds subsequences of one or more trajectories), while this work focuses on the horizontally fragmented case (i.e., each smartphone holds the complete trajectory locally.)

Top-K queries have been studied in a variety of contexts including middleware systems [15], web accessible databases [8] and stream processors [4]. An excellent survey for relational database environments appears in [19]. It has been shown in numerous studies [9], [8], [38], that top-K query processing is meaningful only if the predicate K refers to a small subset of the complete answer set (e.g., up-to 1%). For larger values of K, the query optimizer can choose to retrieve the complete answer set. The wave of centralized top-K query processing algorithms was succeeded by their distributed counterparts, namely the *TPUT* [9] algorithm and the *TJA* [38] algorithm. In all these scenarios, the queries refer to exact scores while we focus on upper bound scores that appear in the scoring table. While upper bound algorithms have also been studied in [17], those had completely different assumptions and relied on a centralized computing model.

The distributed top-K query processing problem with probabilistic guarantees, rather than exact scores, was studied in *KLEE* [27] yet the answers were again approximate. The problem of continually providing approximate top-K answers in a client-server setting was also studied in [4]. In all cases the results are approximate and continuous over a single attribute, thus operate over individual attributes (columns), while our approach is exact and operates horizontally over rows.

IV. The SmartTrace Framework

In this section we start out with the description of the SmartTrace framework presented in this work. We shall then formally prove its correctness and also provide a rigorous formal analysis of its performance and convergence properties.

A. Outline of Operation

First note that the similarity query Q is initiated by some querying node QN (or alternatively at some smartphone that propagates its Q towards QN). QN then disseminates Q to all active smartphone users in a pre-specified spatial boundary. Upon receiving Q, each candidate smartphone executes locally an inexpensive linear-time matching function. QN then collects these scores and puts together a vector of upper bounds $UB = (ub_1, \cdots, ub_m)$. We will refer to the UB-vector constructed on QN as *METADATA* and to the actual trajectories stored locally on each smartphone as *DATA*. Obviously, *DATA* is orders of magnitudes larger than *METADATA*, thus *DATA* needs to stay on the smartphone during query resolution. Our objective is to intelligently exploit the *METADATA* scores in order to identify the K highest ranked answers without pulling *DATA* to QN.

B. The SmartTrace Algorithm

The *SmartTrace* algorithm is a novel iterative algorithm for retrieving the K most similar trajectories to a query trajectory Q. Our proposed scheme performs well both with respect to response time and energy, but also does so without ever revealing the complete target trajectories to QN (i.e., it only returns the matched subsequence, if any.) Additionally, the identity of a user is not revealed (we use the notion of a screen name), unless the user user decides to do so.

Description: In step 1 of the SmartTrace algorithm (see Algorithm 1), QN instructs all m nodes to invoke the computation of the linear-time upper-bounding function $LCSS(MBE_Q, Ai)$ $(i \leq m)$. In that way it circumvents the massive deployment of the expensive similarity function $LCSS(Q, A_i)$ [37], presented next, which performs local stretching in both time and space to overcome the temporal and spatial distortions in trajectories. In particular, each node compares its local trajectory A_i to a bounding envelope of the query, i.e.,

$$LCSS(MBE_Q, A_i) = \sum_{j=1}^{|A_i|} \begin{cases} 1 & \text{if } A_i[j] \text{ within envelope} \\ 0 & \text{otherwise} \end{cases}$$

In step 2, QN retrieves all these upper bounds and adds them in descending order to a local *METADATA* vector. By doing this, QN obtains a quick summary of the trajectories similar to Q.

Steps 3 to 5 are executed iteratively until convergence. In particular, during step 3, QN adds the identities of the objects with the $\lambda + 1$ highest upper bounds to a set named S. These objects provide the first line of candidates for the answer set, as these objects have the highest $LCSS(MBE_Q, A_i)$ value. The given objects will be analyzed more carefully in the next step of the algorithm in order to determine the correct top-K set. Please notice that the objects in the S-set, do not again define the final top-K result. In particular, it is absolutely possible that some arbitrary object in the S-set with a high

Algorithm 1 : **The SmartTrace Algorithm**

Input: Query Trajectory Q, m Target Trajectories, Result Cardinality K ($K << m$), Iteration Step Increment λ.
Output: K trajectories most similar to Q.
At the query node QN:

1) **Upper Bound (UB) Computation:** Instruct each of the m smartphones to invoke a computation of the linear-time $LCSS(MBE_Q, A_i)$ ($i \leq m$).
2) **Collection of UB:** Receive the UBs of all m trajectories participating in the query and add those scores to the *METADATA* vector stored on QN. Let *METADATA* be sorted in descending order based on the UB scores.
3) **Identify Candidates:** Find the $\lambda + 1$ ($\lambda \geq K$) highest UBs in *METADATA*, and add the identities to an empty set S (denoted as the candidate set). If an element has already been added to S, during a previous iteration do not add it again.
4) **Full Computation:** Ask each element in the S-set to compute $LCSS(Q, A_i)$, in a decentralized manner, and then send back the next λ full similarity scores.
5) **Termination Condition:** If the (λ+1)-th UB is smaller than the K-th largest full match then stop; else goto step 3 in order to identify the next λ candidates.
6) **Ship Matching:** If the termination condition has been met, ship the respective matches to QN, based on some local trace disclosure policy.

$LCSS(MBE_Q, A_i)$ score has a low full score $LCSS(Q, A_i)$. Consequently, the algorithm can still not converge.

The λ parameter, mentioned previously, expresses an application-specific confidence in the *METADATA* bounds. In particular, when the *METADATA* vector contains tight bounds, then λ might be set to a small value. So this parameter defines how aggressively some application wants to determine the top-K results. It will be proven next that SmartTrace will not perform more than $O(m/\lambda)$ iterations in the worst case.

In step 4, QN asks each smartphone in the S-set, to compute the full scores (if a smartphone has been contacted in a previous iteration we do not contact it again). In particular, we ask each smartphone to locally compute $FullM(Q, A_i)$, where A_i is stored locally, and only transmit the value of $FullM(Q, A_i)$ towards QN (i.e., the decentralized way). Alternatively, we could have also fetched the trajectories of the S-set to the sink and then compute $FullM(Q, A_i)$ $\forall A_i \in S$ (i.e., the centralized way), however this would violate both the trace disclosure factor and also degrade the response time of the algorithm to a level comparable to the centralized algorithm. Notice that the fourth step of the algorithm applies only to the elements in the S-set, as opposed to all m elements so this is really much cheaper in terms of energy consumed on the smartphone as $|S| << m$.

In our case, $FullM(Q, A_i)$ is the LCSS similarity, which has been extensively used in many 1-D sequence problems, such as string matching. The 2-dimensional adaptation of LCSS using the L_∞[8] is defined as following:

Definition: *Given integers δ and ϵ, the Longest Common Sub-Sequence similarity $LCSS_{\delta,\epsilon}(A, B)$ between two sequences A and B is defined as:*

$$LCSS_{\delta,\epsilon}(A, B) =$$

$$\begin{cases} \mathbf{0}, & \text{if A or B is empty} \\ \mathbf{1 + LCSS}_{\delta,\epsilon}(\mathbf{Tail(A)}, \mathbf{Tail(B)}) & \\ \quad \text{if } |a_{x:l_1} - b_{x:l_2}| < \epsilon \text{ and} & \\ \quad\quad |a_{y:l_1} - b_{y:l_2}| < \epsilon \text{ and } |l_1 - l_2| < \delta & \\ \mathbf{max(LCSS}_{\delta,\epsilon}(\mathbf{Tail(A)}, \mathbf{B}), \mathbf{LCSS}_{\delta,\epsilon}(\mathbf{A}, \mathbf{Tail(B)})) & \\ \quad \text{otherwise} & \end{cases}$$

where δ and ϵ are application-specific parameters that allow flexible matching in the *time* (e.g., if we have two identical trajectories, but the first one moves earlier in time) and the *space* (e.g., we have two identical trajectories but the first has some slight deviation in its spatial movement) domain, respectively. LCSS deals with both aforementioned limitations of the L_p*-Norm* family of distances, because these cases are simply dropped from the matching.

In step 5, we determine whether the algorithm has reached a termination condition. In particular, we check if the (λ+1)-th highest UB is smaller than the K-th highest full matching value. If this is the case, then we can safely terminate the execution of the algorithm being sure that the correct top-K has been identified. If this condition does not hold (i.e., when the UB of an object X is larger than the K-th highest full matching value Y), then we are enforced to perform another iteration as the answer is not deterministic (i.e., either X or Y can be the K-th answer). Consequently, we increase the step increment λ so that it identifies the next λ candidates in the next round.

In the final step, which occurs only once at the very end, we might ship each matched subsequence A_i^{match} ($|A_i^{match}| << |A_i|$) to QN, which can then return it to the user. Notice, that identified nodes s_i ($i \leq K$) might choose not to share the matching or share it based on some local trace disclosure profile [11], in order to preserve k-anonymity and other higher anonymity schemes. In any case, neither QN nor the querying user will ever see the complete trajectory of participating users.

Example: Consider the example scenario of Figure 2. Assume that Q aims to find the top-2 trajectories ($K = 2$). Initially, QN sends Q to all nodes. Each object then computes an upper bound of its trajectory with respect to Q and sends this value to QN. Subsequently, QN proceeds by determining the trajectories with the highest $\lambda + 1$ *METADATA* entries, i.e., $\{A_4, A_2, A_0\}$, and adds the λ trajectories to the S-set, i.e. $S = \{A_4, A_2\}$ (steps 2-3). In step 4, QN asks the smartphones in S to compute the full matching of Q to A_i ($A_i \in S$) without unveiling their A_i. The full matching scores, which are transmitted to QN, are: $FullM(Q, A_4) = 23$ and $FullM(Q, A_2) = 22$. Since the $(\lambda + 1)$-th highest UB $(A_0, 25)$ is larger than the K-th highest full match $(A_2, 22)$,

[8]We could also use L_1 or L_2 for the recursion step.

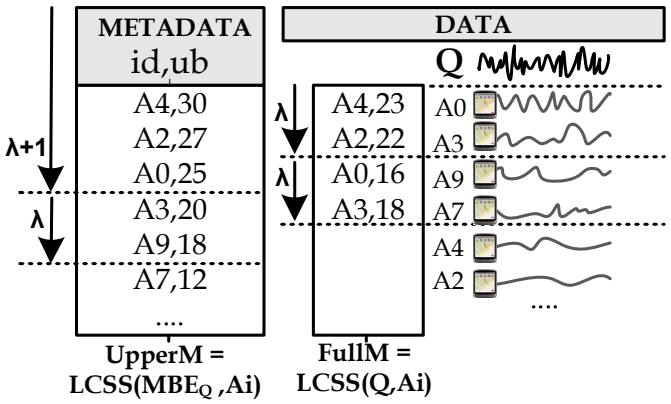

Fig. 2. Example Execution of the SmartTrace Algorithm.

the termination condition is not satisfied in the fifth step. Therefore, the second iteration of the SmartTrace algorithm is initiated to compute the next λ, ($\lambda = 2$), full matching scores: $FullM(Q, A_0) = 16$, $FullM(Q, A_3) = 18$. Now the termination has been satisfied because the new $(\lambda+1)$-th (i.e., $2\lambda + 1$) highest UB $(A_9, 18)$ is smaller than the K-th highest full match $(A_2, 22)$. Finally we tentatively might return the top-2 matched subsequences of trajectories with the highest full matches to the user (i.e., $\{(A_4^{match}, 23), (A_2^{match}, 22)\}$). Even if we return these subsequences to the querying node, it is important to mention that these are not the complete trajectories A_4 or A_2, but only the subsequences that correspond to the matching (e.g., one route out of their two year history log, given that the owner has agreed to release them.)

Theorem 1 (Correctness) *The SmartTrace algorithm always returns the most similar objects to the query trajectory Q.*

Proof: Let A denote some arbitrary object returned as an answer by the SmartTrace algorithm ($A \in Result$), and B some arbitrary object that is not among the returned results ($B \notin Result$). We want to show that $FullM(Q, B) \leq FullM(Q, A)$ always holds. Assume that $FullM(Q, B) > FullM(Q, A)$. We will show that such an assumption leads to a contradiction. In our analysis, we cover the two possible cases: i) B is not in the set of candidate objects (denoted as S-set) during the third step; and ii) B becomes part of the S-set during some arbitrary iteration of the algorithm in the third step. We will show that both cases yield a contradiction. **Case 1** ($A \in S$ and $B \notin S$): Since $B \notin S$, then $UB(Q, B) < FullM(Q, X)$, where X is the K-th highest object in the array $FullM$, by the termination condition of the algorithm (Step 5). Thus, $FullM(Q, B) \leq UB(Q, B) < FullM(Q, X) < ... < FullM(Q, A)$, because object A is in the final *Result*. Hence, it holds that $FullM(Q, B) < FullM(Q, A)$, a contradiction. **Case 2** ($A \in S$ and $B \in S$): Since both objects A and B are now part of S, the full scores of A and B are known, by step 4 of the algorithm. By the initial assumption we know that only object A belongs to the final *Result*. Thus, $FullM(Q, B) \leq FullM(Q, A)$, a contradiction □

Theorem 2 (SmartTrace λ-Convergence) *Algorithm SmartTrace performs $O(m/\lambda)$ iterations in the worst case.*

Proof: To prove the statement of the theorem we assume that all the Upper Bound values in *METADATA* are bigger than the K-th highest full matching value (i.e., the K-th highest value in $FullM$). In this case, the termination condition, in step 5, is satisfied at the very end. Consequently, SmartTrace performs O(m/λ) iterations in the worse case □

V. Performance Analysis

In this section we analytically derive the performance of the SmartTrace algorithm with respect to Time and Energy.

Cost Model: Let m smartphone users $\{s_1, \cdots, s_m\}$ participate in the execution of query Q, initiated at QN. Let the maximum length among all trajectories be denoted as l and $|Q| << l$, as explained earlier. All smartphones are connected to QN for the complete duration of Q's execution through some pre-established connection (e.g., persistent TCP socket).

We are interested in deriving analytically the *Time* ($\mathcal{T}$) and *Energy* ($\mathcal{E}$) costs for resolving Q. $\mathcal{T}$ is defined as the length of time it takes for Q to be sent to the m users plus the length of time it takes for the final top-K result to be received at QN (i.e., *the user-perceived latency for resolving* Q, formally $\mathcal{T} = MAX_{i=1}^{m}(\mathcal{T}_i)$ as the smartphones operate in parallel). On the other hand, $\mathcal{E}$ is defined as the total energy cost incurred on smartphones[9] for answering Q (i.e., the *client-perceived energy consumption*, formally $\mathcal{E} = \sum_{i=1}^{m}(\mathcal{E}_i)$). Looking at the equations from a different perspective, we notice that the total time a smartphone spends on a query, as opposed to $\mathcal{T}$, is naturally captured by $\mathcal{E}$, which is equivalent to $\sum_{i=1}^{m}(Power_i \times \mathcal{T}_i)$, where Power is measured in Watts (i.e., Volts $\times$ Amperes).

Notice that in our cost analysis we deliberately do not focus on the *Messaging* and *Bandwidth* costs, because measuring these in isolation will not expose the relevant complexities of a smartphone network environment, as explained in Section I. For instance, a protocol with a high message complexity might transmit many small-size messages, thus consuming very little bandwidth. For ease of exposition, our analysis should use the notation $\{\mathcal{E}|\mathcal{T}\}^{CPU}$, $\{\mathcal{E}|\mathcal{T}\}^{TX}$ and $\{\mathcal{E}|\mathcal{T}\}^{RX}$, to denote the energy or time cost for *processing*, *transmitting* and *receiving* one trajectory point (^{TX} and ^{RX} also capture the incurred processing costs during communication and are approximately equivalent). We will ignore any other irrelevant energy consumption costs, such as LCD and Bluetooth.

Theorem 3 (Centralized Performance): *The Centralized algorithm has an Energy and Time complexity of $O(m \cdot l \cdot \mathcal{E}^{TX})$ and $O(l \cdot \mathcal{T}^{TX})$, respectively.*

Proof (direct): In this one phase algorithm the m nodes simply send their local trajectory, of maximum length l, to QN. Consequently, each s_i requires $l \cdot \mathcal{E}_i^{TX}$ units of energy, thus the energy complexity is $O(m \cdot l \cdot \mathcal{E}^{TX})$. The time complexity

[9] QN is assumed to have no energy constraints.

is defined as $MAX_{i=1}^{m}(l \cdot \mathcal{T}_i^{TX}) \in O(l \cdot \mathcal{T}^{TX})$, as the m smartphones transmit their trajectories to QN in parallel $\square$

Theorem 4 (Decentralized Performance): *The Decentralized algorithm has an Energy and Time complexity of* $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$ *and* $O(\delta \cdot l \cdot \mathcal{T}^{CPU})$*, respectively.*

Proof (direct): In this two phase algorithm the m nodes first receive Q, which costs $m \cdot |Q| \cdot \mathcal{E}^{RX}$ energy ($|Q| << l$). Then, the m nodes spend $m \cdot \delta(|Q|+l) \cdot \mathcal{E}^{CPU}$ computations, as $LCSS(Q, A_i)$ can be computed in $O(\delta \cdot (|Q| + |A_i|))$. Finally, each of the m nodes transmits back a single scalar value (i.e., $m \cdot \mathcal{E}^{TX}$). In the second phase, QN instructs the K ($K << m$) highest ranked nodes to return their complete trajectory. This costs $K \cdot \mathcal{E}^{RX}$ for the notification and another $K \cdot l \cdot \mathcal{E}^{TX}$ for the answer. By adding up the above values in an asymptotic manner, yields a total energy complexity of $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$. The time complexity for the decentralized case is defined, somehow higher than the centralized case, as $MAX_{i=1}^{m}(\delta \cdot l \cdot \mathcal{T}_i^{CPU}) \in O(\delta \cdot l \cdot \mathcal{T}^{CPU})$, as the nodes conduct their computation and transmission to QN in parallel $\square$

Theorem 5 (SmartTrace Performance): *The SmartTrace algorithm has an Energy and Time complexity of* $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$ *and* $O(\frac{m \cdot \delta \cdot l \cdot T^{CPU}}{\lambda})$*, respectively.*

Proof (direct): In the first (1) step, each of the m nodes receives Q, which costs $m \cdot |Q| \cdot \mathcal{E}^{RX}$ ($|Q| << l$) energy. Then each s_i invokes the linear-time $LCSS(MBE_Q, A_i)$ ($i \leq m$) computation, which costs $m \cdot |Q| \cdot \mathcal{E}^{CPU}$, as $LCSS(MBE_Q, A_i)$ can be computed in $O(min(l, |Q|))$ and $|Q| << l$. Finally, each of the m nodes transmits back a single scalar value (i.e., $m \cdot \mathcal{E}^{TX}$). The second and third steps of the algorithm have no smartphone-side incurred costs. Steps 4 and 5 are executed in m/λ iterations in the worse case. In each of the iterations, we have the following costs: In step 4, the λ identified nodes execute the $LCSS(Q, A_i)$ ($i \leq m$) computation and that costs $\lambda \cdot \delta \cdot (|Q|+l) \cdot \mathcal{E}^{CPU}$, as LCSS has a time complexity of $O(\delta \cdot (l_1 + l_2))$ Yet, this cost is accrued only on a few nodes (i.e., $\lambda << m$). When this operation completes, a single scalar value is shipped from each of the λ nodes to QN costing $\lambda \cdot \mathcal{E}^{TX}$. Step 5 has again no smartphone-side incurred energy cost as it takes place on QN. Also step 6 is computed only once at the very end and costs $\lambda \cdot |Q| \cdot E^{TX}$. By adding up all aforementioned values in an asymptotic manner yields an energy complexity of $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$, as all other factors are small order and can thus be eliminated.

Similarly to the above analysis, the time complexity of SmartTrace is defined as $O(\frac{m \cdot \delta \cdot l \cdot T^{CPU}}{\lambda})$, as we are now only waiting for the slowest node during the computation of $LCSS(Q, A_i)$ and we conduct this m/λ times $\square$

VI. Experimental Evaluation

In this section we present an extensive experimental evaluation of the SmartTrace algorithm. Our experiments are conducted in trace-driven experimental mode as this allows us to scale the experiments to large real-world settings (as we lacked a large number of smartphone units). We also present the components of our prototype system [12], implemented for Android-based smart-phone devices and used in Section II-B for quantifying our system model.

A. Methodology

Datasets and Queries: We use the following two datasets:

i) *Oldenburg* [7]: This realistic dataset includes 2,000 car trajectories moving in the city of Oldenburg [7]. The average length of each trajectory is $11,731 \pm 7,193$ points, while the maximum trajectory length is 42,500 points. Each query for the above dataset is derived by adding interpolated peaks of Gaussian noise to a segment of a randomly selected trajectory in the dataset. This created variations in the shape of the queries compared to the original trajectories. Our queries have an average size of 100 spatio-temporal points. Our results are averaged over 10 queries.

ii) *GeoLife* [39], [40]: This real dataset, by Microsoft Research Asia, includes 1,100 trajectories of a human moving in the city of Beijing over a life span of two years (2007-2009). The average length of each trajectory is $190,110 \pm 126,590$ points, while the maximum trajectory length is 699,600 points. Notice that 95% of the GeoLife dataset refers to a granularity of 1 sample every 2-5 seconds or every 5-10 meters. Our queries are randomly sampled from the dataset and have an average size of 500 spatio-temporal points. Our results are again averaged over 10 runs.

Algorithms and Metrics: We compare the *SmartTrace* ($\mathcal{ST}$), *Decentralized* ($\mathcal{D}$) and *Centralized* ($\mathcal{C}$) algorithms, under a variety of settings using the datasets described earlier. Our cost metrics are: *Time* ($\mathcal{T}$) and *Energy* ($\mathcal{E}$), as documented in Section V, for varying m, K and λ parameters. Whenever we test one parameter, the complementary parameters are fixed to the following values: m to 2,000 and 1,100, respectively for the Oldenburg and the GeoLife datasets, K to 2 and λ to 30. The reader is encouraged to lookup the performance values under different settings for these parameters from the rest series (which are complementary to each other). The δ and ϵ parameters are kept constant for each dataset as those are application specific (i.e., they attempt to capture a reasonable scenario given the underlying temporal and spatial coordinates in the dataset). Varying δ and ϵ should not affect our execution scenario in any sense, as this would simply vary the matching granularity in all algorithms.

Network and Energy Model: Our communication protocol is associated with a 45 byte header (including node identifier, session identifier and other application specific parameters). In our setting, a spatio-temporal *DATA* point (18 bytes) consists of a *timestamp* that occupies 8 bytes, two 4-byte fields for the GPS coordinates and another 2 bytes for direction. In reality this overhead might be even higher (e.g., GeoLife trajectories include elevation, speed, heading direction and accuracy). However, we omit these additional attributes as they are not necessary for computing the basic edition of our algorithm that relies only on (t, x, y). Had we used them should have boosted the competitive advantage of $\mathcal{ST}$ over $\mathcal{C}$ and $\mathcal{D}$ even

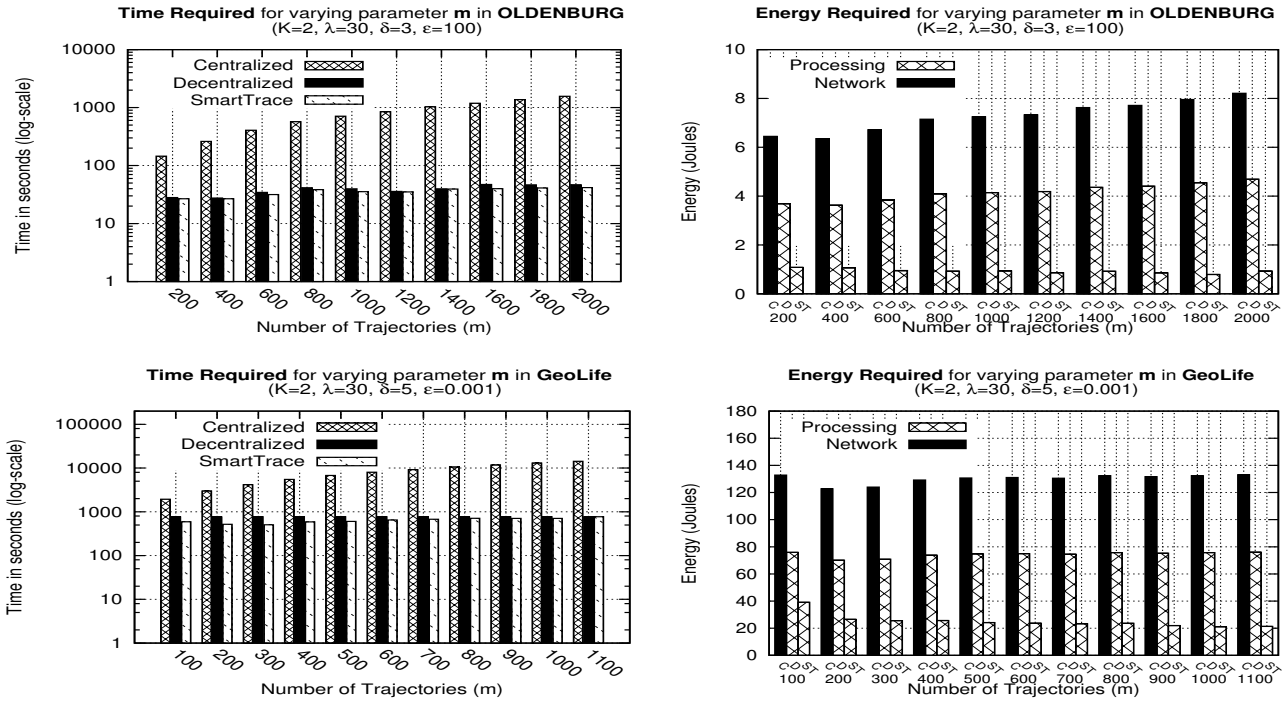

Fig. 3. **Series 1:** Varying the Number of Trajectories (m). Time and Energy results for the Oldenburg and GeoLife datasets.

more. The *METADATA* comprise only a single 4-byte real field per node for the *UB* or *FullM* matching value. QN runs on a single host that connects to the m smartphones using an 802.11b network link that has a TCP downlink of 1022kbps and a TCP uplink of 123kbps, a 237ms TCP handshake latency and application handshake latency of 493ms (as measured with [1]). Our energy profile has been derived by running SmartTrace instances using PowerTutor [29] (the results are summarized in Table I).

B. **Series 1:** *Varying the Number of Trajectories (m)*

In the first experimental series we investigate the performance and scalability of our approach with respect to m. Figure 3 presents our Time ($\mathcal{T}$, left) and Energy ($\mathcal{E}$, right) results for the Oldenburg (top) and GeoLife (bottom) datasets, respectively.

Analysis of Response Time ($\mathcal{T}$): The plots in the left column show that both the $\mathcal{ST}$ and $\mathcal{D}$ algorithms consume one order of magnitude less $\mathcal{T}$ than the $\mathcal{C}$ algorithm. In particular, for Oldenburg, we observe the following time values: $\mathcal{ST} = 35 \pm 5$ seconds, $\mathcal{D} = 38 \pm 7$ seconds and $\mathcal{C} = 810 \pm 6$ seconds; while for GeoLife, we observe the time values of $\mathcal{ST} = 644 \pm 87$, $\mathcal{D} = 769 \pm 2$ and $\mathcal{C} = 8,015 \pm 3,804$ seconds, respectively. The above results are attributed to the fact that $\mathcal{ST}$ and $\mathcal{D}$ rely mainly on local processing, i.e., $O(\frac{m \cdot \delta \cdot l \cdot \mathcal{T}^{CPU}}{\lambda})$ and $O(\delta \cdot l \cdot \mathcal{T}^{CPU})$), while $\mathcal{C}$ relies on transmission $O(l \cdot \mathcal{T}^{TX})$.

An interesting observation that arises here is that although the asymptotic time complexity of $\mathcal{D}$ looks better than the respective time complexity of $\mathcal{ST}$ (i.e., $\mathcal{ST} = m/\lambda \cdot \mathcal{D}$), our experiments unveil that $\mathcal{ST}$ is faster than $\mathcal{D}$ for the two datasets, by 3 seconds and 125 seconds, respectively. By carefully analyzing our traces we found that this is attributed to the variable length of trajectories in our datasets. In particular, $\mathcal{D}$ is always condemned to process the longest trajectory in its computation, while $\mathcal{ST}$ will process these very long traces only if they belong to the top-K result. By analyzing our executions, we found that the processed trajectories by $\mathcal{ST}$ are on average only $11,500$ and $188,000$ points, for Oldenburg and GeoLife respectively, compared $42,500$ and $699,600$ respectively, processed by $\mathcal{D}$.

One final observation is that the $\mathcal{T}$ cost for $\mathcal{C}$ increases linearly with increasing m (notice that the y-axis of the plot is in log-scale). This is attributed to the fact that as we upload more trajectories to QN, the given server needs to spend more processing to compute the LCSS similarities. Notice that this cost was not accounted for in our analytical model and time cost (i.e., $\mathcal{T}_C \in O(l \cdot \mathcal{T}^{TX})$), as the server could have computed the m arriving trajectories in parallel (i.e., on a processing cluster). Nevertheless, in our experiments this cost shows up as we execute QN on a single workstation.

Analysis of Energy Consumption ($\mathcal{E}$): While somebody might claim that the competitive advantage of $\mathcal{ST}$ over $\mathcal{D}$, with respect to $\mathcal{T}$, is not that great (i.e., 8% and 17%, respectively), the right column of Figure 3, shows that there is a 67% and 81% percent $\mathcal{E}$-competitive advantage of $\mathcal{ST}$ over $\mathcal{D}$ and $\mathcal{C}$, respectively. In particular, we observe the following energy values per smartphone: $\mathcal{ST} = 0,93J$, $\mathcal{D} = 4,15J$, and $\mathcal{C} = 7,27J$, for the Oldenburg dataset; and $\mathcal{ST} = 25J$, $\mathcal{D} = 74J$, and $\mathcal{C} = 130J$ for the GeoLife dataset. For ease of exposition, these plots refer the cost-per-smartphone while our analytical $\mathcal{E}$ bounds are accounting all m smartphone units.

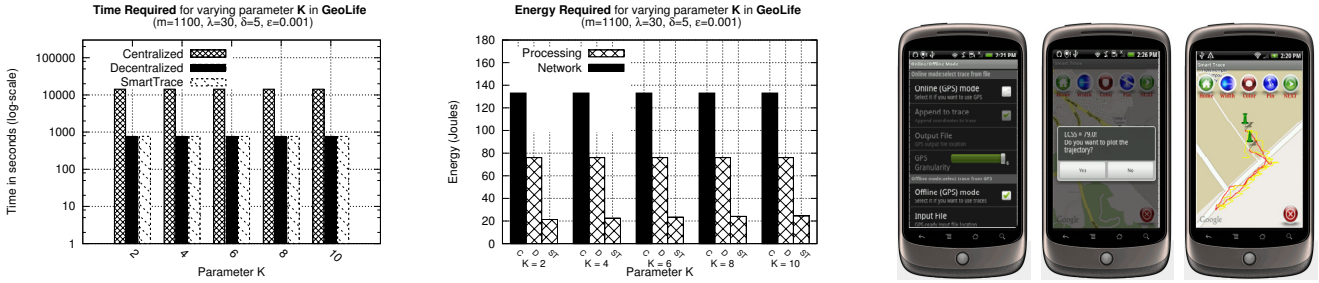

Fig. 4. **Series 2 (left):** Varying the Cardinality of the Answer-set (K); **Prototype Implementation (right)**: Screenshots of the SmartTrace GUI in Android.

Additionally, we observe that the $\mathcal{C}$ algorithm spends all its energy on network operations, i.e., $O(m \cdot l \cdot \mathcal{E}^{TX})$, while $\mathcal{ST}$ and $\mathcal{D}$ spend the bulk of their energy on smartphone-side processing operations, i.e., $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$ and $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$, respectively. In fact, the networking costs for these algorithms are as small as $2.59mJ$ and $2.29mJ$ per query (this is why they don't show up in the plots). The above result, confirms that the network overhead of the $\mathcal{ST}$ algorithm is not high, as it transmits smaller size packages as opposed to the large and monolithic packages used by $\mathcal{C}$.

C. Series 2: *Varying the Cardinality of the Answer-set (K)*

In the second experimental series, we investigate the performance and scalability of our approach with respect to K. For brevity, we will only present the results for the GeoLife dataset, but the respective Oldenburg results look very similar. We will additionally not cover the observations already discussed previously.

Figure 4 (left, middle), shows that the performance of all three algorithms is independent of K for both the $\mathcal{T}$ and $\mathcal{E}$ results. This result is confirmed by our analytical study where we have shown that all three algorithms are independent of K (see Theorem 3, 4 and 5). In particular, the $\mathcal{E}$-complexity of $\mathcal{ST}$ is $O(m \cdot \delta \cdot l \cdot \mathcal{E}^{CPU})$, while the $\mathcal{T}$-complexity of $\mathcal{ST}$ is $O(\frac{m \cdot \delta \cdot l \cdot T^{CPU}}{\lambda})$. Notice that these results hold as long as K is smaller than m (i.e., $K << m$). If K was larger, then we should certainly expect an increase in both $\mathcal{T}$ and $\mathcal{E}$. Yet, top-K queries are not designated for these types of workloads, as explained in Section III. Also these workloads are not useful in our setting as a user would be overwhelmed with many less relevant answers. For this experiment K refers to approximately 1% of m.

D. Series 3: *Varying the Iteration Step Increment (λ)*

In the last experimental series, we study how the iteration step increment λ affects the convergence of the SmartTrace algorithm. In particular, we observe the number of iterations our algorithm takes for different values of λ and for different sizes of trajectories (m).

Figure 5 (left and middle) shows our result for the Oldenburg and GeoLife datasets, respectively. The first observation is that the more aggressive λ gets, the quicker the $\mathcal{ST}$ algorithm converges. In particular, we observed that the two datasets feature an average number of iterations equal to 7.6 and 9.4, respectively. Additionally, we also observe that the number of iterations grows almost linearly by increasing m. Both aforementioned observations are explained by the result of Theorem 2, where we showed that $\mathcal{ST}$ requires O(m/λ) iterations in the worse case. Interestingly, we mention that this worse case has not happened in any of our experiments, but setting this parameter optimally through some learning phase will be a subject of future research.

Figure 5 (right) shows the $\mathcal{T}$-complexity of the $\mathcal{ST}$ algorithm, for the same executions described above (we again omit the results from Oldenburg for brevity, as they look similar). The given plot validates that $\mathcal{ST}$ is inversely proportional to λ, (i.e., Theorem 5 showed that $\mathcal{T}_{ST} \in O(\frac{m \cdot \delta \cdot l \cdot T^{CPU}}{\lambda})$.) Notice that in the given experiment λ is ranging between 1% - 5% of m, which is larger than the K value we used in this study (i.e., up-to 1%). Had we been more aggressive would certainly improve the response time but would have also consumed more energy. Although setting λ in an optimal manner would require some additional structures, configuring it to approximately 5% of m worked great for the tests we have conducted.

E. Prototype Evaluation

We have developed a prototype system that realizes the SmartTrace framework (see Figure 4, right) [12]. Our client-side software is developed around the Google Map API and its installation package (i.e., APK) has a size of 510KB. Our code is written in JAVA and consists of approximately 4,500 lines-of-code (LOC). In particular, our server-code uses $\approx$ 1,500 LOC and runs over JDK 6 and Ubuntu Linux, our smartphone-code uses $\approx$ 2,500 LOC plus $\approx$ 250 lines of XML elements that go the Manifest file (settings) and the user interface XML descriptions. In the future we plan to take the computationally- and IO-intensive tasks outside the VM by implementing them in native (C) code using the newly released Android NDK. The SmartTrace GUI allows a user to query other devices by example, plot and iterate through the responses using a variety of presentation functions as well as to configure a wide range of parameters such as δ, K, etc. The absence of a large smartphone testbed, did not allow us to focus on the performance characteristics of the SmartTrace framework in real environments. We are however in the process of setting up a testbed of 50 smartphones that will allow us to address this aspect in the future.

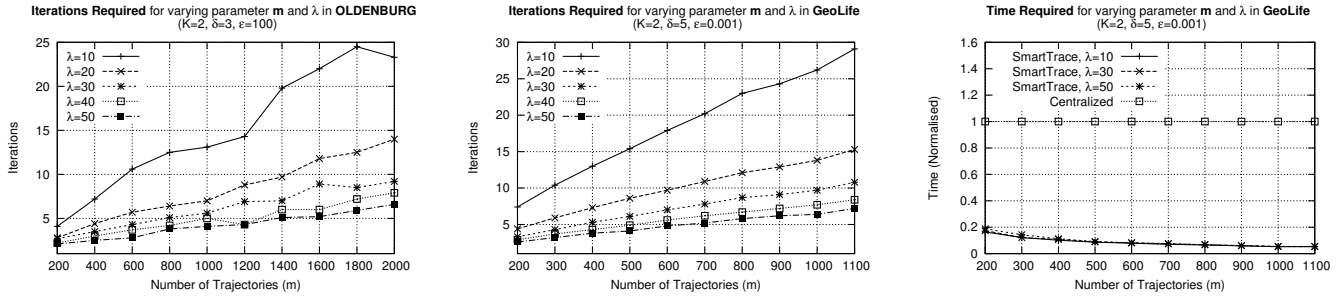

Fig. 5. **Series 3:** Varying the Iteration Step Increment (λ) in SmartTrace. Number of Iterations and Time Results for the Oldenburg and GeoLife datasets.

VII. Conclusion

This paper presents a powerful spatio-temporal similarity search framework, coined SmartTrace, which enables distributed trajectory search queries over in-situ data. We have assessed our ideas with realistic and real workloads. Our study reveals that SmartTrace computes the desired results with 74% less energy consumption and 13% faster than its fully centralized and fully decentralized counterparts. Our experimental results also confirm our extensive analytical study. In the future we plan to deploy SmartTrace over an infrastructure of 50 smartphone devices we are currently setting up and also plan to extend our study in indoor spaces.

Acknowledgments: This work was supported in part by the first author's Startup Grant, funded by the University of Cyprus, EU's FP7 CONET project (#224053), EU's FP6 Marie Curie TOK "SEARCHiN" project and EU's FP7 "SemSorGrid4Env" and "MODAP" projects.

References

[1] 3gtest Tool, `http://www.eecs.umich.edu/3gtest/`.
[2] Al-Aha K. , Snodgrass R., Soo M., "Bibliography on Spatiotemporal Databases," In SIGMOD Record, 22(1), 59–67, 1993.
[3] Azizyan M., Constandache I., Choudhury R.-R., "SurroundSense: mobile phone localization via ambience fingerprinting," In *MobiCom'09*.
[4] Babcock B., Olston C., "Distributed Top-K Monitoring", In *SIGMOD'03*
[5] Bakalov P., Hadjieleftheriou M., Tsotras V., "Time Relaxed Spatiotemporal Trajectory Joins," In *GIS*, 2005.
[6] Berndt D., Clifford J., "Using Dynamic Time Warping to Find Patterns in Time Series," In *KDD*, 1994.
[7] Brinkhoff T., "A Framework for Generating Network-Based Moving Objects," In *GeoInformatica*, 6(2), 2002.
[8] Bruno N., Gravano L., Marian A., "Evaluating Top-K Queries Over Web Accessible Databases," In *ICDE*, 2002.
[9] Cao P., Wang Z., "Efficient Top-K Query Calculation in Distributed Networks," In *PODC'04*, pp. 206-215, 2004.
[10] Campbell A., Eisenman S., Lane N., Miluzzo E., and Peterson R., "People-centric urban sensing," In *WICON*, 2006.
[11] Chow C-Y., Mokbel M.F., Aref W.G., "Casper*: Query Processing for Location Services without Compromising Privacy" In ACM TODS 34(4), pp. 1-48, 2009.
[12] Costa C., Laoudias C., Zeinalipour-Yazti D. and Gunopulos D., "SmartTrace: Finding Similar Trajectories in Smartphone Networks without Disclosing the Traces", Demo paper to appear in IEEE ICDE'11.
[13] Das G., Gunopulos D., Mannila H., "Finding Similar Time Series," In *PKDD*, 1997.
[14] Das T., Mohan P., Padmanabhan V.N., Ramjee R., Sharma A., "PRISM: platform for remote sensing using smartphones," In *MobiSys*, 2010.
[15] Fagin R., Lotem A., Naor M., "Optimal Aggregation Algorithms For Middleware," In *PODS*, 2001.
[16] Frentzos E., Gratsias K., Theodoridis Y., "Index-based Most Similar Trajectory Search," In *ICDE* 2007.
[17] Guo L., Amer-Yahia S., Ramakrishnan R., Shanmugasundaram J., Srivastava U., Vee E. "Efficient top-k processing over query-dependent functions," In *PVLDB* 1(1) 1044-1055 (2008)
[18] Hadjieleftheriou M., Kollios G., Bakalov P., Tsotras V.J., "Complex Spatio-Temporal Pattern Queries," In *VLDB*, 2005.
[19] Ilyas I.F., Beskales G., and Soliman M.A., "A Survey of Top-k Query Processing Techniques in Relational Database Systems", In *ACM Computing Surveys* 40(4), 2008.
[20] Jeung H., Lung Yiu M., Zhou X., Jensen C.S., Tao Shen H., "Discovery of convoys in trajectory databases," In *PVLDB* 1(1), 2008.
[21] Kollios G., Gunopulos D., Tsotras V.J., Delis A., Hadjieleftheriou M., "Indexing Animated Objects Using Spatiotemporal Access Methods," In *TKDE* 13(5), 2001.
[22] Chen Z., Shen H-T., Zhou X., Zheng Y., Xie X. "Searching trajectories by locations: an efficiency study," In *SIGMOD*, 2010.
[23] Chen L., Ng R.-T., "On The Marriage of Lp-norms and Edit Distance," In *VLDB*, 2004.
[24] Chen L., Ozsu T., Oria V., "Robust and Fast Similarity Search for Moving Object Trajectories," In *SIGMOD*, 2005.
[25] Liu T., Sadler C.M., Zhang P., Martonosi M., "Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet," In *MobiSys*, 2004.
[26] Ni J., Ravishankar C.V. "Indexing Spatio-Temporal Trajectories with Efficient Polynomial Approximations," In em TKDE 19(5), 2007.
[27] Michel S., Triantafillou P., Weikum G., "KLEE: A Framework for Distributed Top-K Query Algorithms," In *VLDB*, 2005.
[28] Musolesi M., Piraccini M., Fodor K., Corradi A., Campbell A.-T., "Supporting Energy-Efficient Uploading Strategies for Continuous Sensing Applications on Mobile Phones," In *PerCom*, 2010.
[29] PowerTutor Tool, `http://powertutor.org/`.
[30] Pfoser D., Jensen C. S., and Theodoridis, Y., Novel "Approaches to the Indexing of Moving Object Trajectories", In *VLDB*, 2000.
[31] Sarigol E., Riva O., Alonso G., "A tuple space for social networking on mobile phones," In *ICDE*, 2010.
[32] Tao Y., Sun J., Papadias D., "Analysis of Predictive Spatio-Temporal Queries," In *ACM TODS* 28(4), 2003.
[33] Thiagarajan A., et. al., "VTrack: Accurate, Energy-aware Road Traffic Delay Estimation using Mobile Phones," In *SenSys*, 2009.
[34] Vieira M., Bakalov P., Tsotras V., "On-Line Discovery of Flock Patterns in Spatio-Temporal Data," In *GIS*, 2009.
[35] Vlachos M., Hadjieleftheriou M., Gunopulos D., Keogh E., "Indexing multi-dimensional time-series with support for multiple distance measures," In *SIGKDD*, 2003.
[36] Xia T., Zhang D., Kanoulas E., Du Y., "On Computing Top-t Most Influential Spatial Sites," In *VLDB*, 2005.
[37] Zeinalipour-Yazti D., Lin S., Gunopulos D., "Distributed Spatio-Temporal Similarity Search," In *CIKM*, 2006.
[38] Zeinalipour-Yazti D. et. al., "Finding the K highest-ranked answers in a distributed network," In ComNet 53(9), 1431-1449, 2009.
[39] Zheng Y., Liu L., Wang L., Xie X., "Learning transportation mode from raw gps data for geographic applications on the web," In *WWW'08*.
[40] Zheng Y., Zhang L., Xie X., Ma W.-Y., "Mining interesting locations and travel sequences from gps trajectories," In *WWW'09*.