IoT Data Prefetching in Indoor Navigation SOAs

ANDREAS KONSTANTINIDIS, University of Cyprus, Cyprus and Frederick University, Cyprus
PANAGIOTIS IRAKLEOUS and ZACHARIAS GEORGIOU, University of Cyprus, Cyprus
DEMETRIOS ZEINALIPOUR-YAZTI, MPI for Informatics, Germany and University of Cyprus, Cyprus
PANOS K. CHRYSANTHIS, University of Pittsburgh, USA

10

Internet-based Indoor Navigation Service-Oriented Architectures (IIN-SOA) organize signals collected by IoT-based devices to enable a wide range of novel applications indoors, where people spend 80–90% of their time. In this article, we study the problem of *prefetching (or hoarding) the most important IoT data from an IIN-SOA to a mobile device, without knowing its user's destination during navigation.* Our proposed *Grap (Graph Prefetching)* framework structurally analyzes building topologies to identify important areas that become virtual targets to an online heuristic search algorithm we developed. We tested Grap with datasets from a real IIN-SOA and found it to be impressively accurate.

CCS Concepts: • **Information systems → Data management systems**; **Mobile information processing systems**; • **Human-centered computing → Ubiquitous and mobile computing systems and tools**;

Additional Key Words and Phrases: Internet-of-things, indoor navigation, mobile prefetching

ACM Reference format:
Andreas Konstantinidis, Panagiotis Irakleous, Zacharias Georgiou, Demetrios Zeinalipour-Yazti, and Panos K. Chrysanthis. 2018. IoT Data Prefetching in Indoor Navigation SOAs. *ACM Trans. Internet Technol.* 19, 1, Article 10 (November 2018), 21 pages.
https://doi.org/10.1145/3177777

1 INTRODUCTION

Internet of Things (IoT) refers to a large number of physical devices being connected to the Internet that are able to see, hear, think, perform tasks as well as communicate with each other using open protocols [1, 2, 15, 30]. IoT enables the development of smart applications in important domains, such as transportation, health care, industrial automation, emergency response and business, having significant impact on the quality of people's life and the growth of the world's economy and security [1]. Studies showed that a typical family in the developed world owns about 5–10 internet-connected devices, such as smartphones, smartTVs, smart-home devices, and the

Authors' addresses: A. Konstantinidis, University of Cyprus, Nicosia, 1678, Cyprus, Frederick University, Nicosia, 1036, Cyprus; email: akonstan@cs.ucy.ac.cy; P. Irakleous and Z. Georgiou, University of Cyprus, Nicosia, 1678, Cyprus; emails: {pirakl02, zgeorg03}@cs.ucy.ac.cy; D. Zeinalipour-Yazti, MPI for Informatics, Department 5: Databases and Information Systems, Campus E1 4, Saarbrücken, 66123, Germany, Univ. of Cyprus, Nicosia, 1678, Cyprus; email: dzeinali@mpi-inf.mpg.de; P. K. Chrysanthis, University of Pittsburgh, Pittsburgh, PA, 15260; email: panos@cs.pitt.edu.

1533-5399/2018/11-ART10 $15.00
https://doi.org/10.1145/3177777

like, and according to Gartner[1] it is expected that this number will increase to more than 500 smart devices by 2022. To realize this potential growth, emerging technologies, innovations, and service applications need to grow proportionally to match market demands and user needs [10].

The omni-present availability of sensor-rich smartphones along with the fact that people spend 80–90% of their time in indoor environments has recently boosted an interest around the so-called *Internet-based Indoor Navigation Service-Oriented Architectures* (IIN-SOA) [32]. These comprise indoor models, such as floor-maps and points-of-interest (POIs), along with IoT-based raw data, such as wireless, light, and magnetic signals, used to localize users. There are numerous IIN-SOAs ([32] provides a taxonomy), including *Skyhook, Google Indoor Maps, Infsoft, Indoo.rs, IndoorAtlas,* and our in-house *Anyplace* IIN-SOA.[2] There is a wide range of domain-specific IIN-SOAs, in domains such as in-building guidance and navigation, inventory management, marketing, and elderly support through ambient and assisted living [28]. Collectively, these are expected to improve location awareness providing thus smart answers to a variety of smart transportation, smart houses, and smart cities scenarios and having a remarkable role on the evolution of intelligent decision-making that can improve they quality of our lives. For example, consider a smart IIN-SOA for elderly support that provides instant emergency notifications to caregivers when elderly people leave their bed at night without returning (thus being in need of help) or a smart IIN-SOA for an airport that can predict future traffic patterns in the terminals and allocate resources accordingly.

A major problem with collected IoT data in IIN-SOA is that this data changes very *dynamically*, requiring users to continuously synchronize their state with the IIN-SOA to enjoy an accurate localization service. For instance, consider a hypothetical scenario related to the US Library of Congress, where a *user u* aims to interactively carry out content-based search, exploration, and navigation (i.e., the user is interactively exploring the space in a *"targetless"* manner). The oblivious solution is to provide a traditional IIN-SOA *(s)* that can perform the queries in the cloud. Unfortunately, internet connectivity in indoor spaces is *intermittent* due to inadequate Wi-Fi coverage, blockage of 3G/LTE signals, and so on. As such, *u* cannot reach *s* on an ongoing basis to refine upcoming search and navigation targets as these emerge. An alternative to cloud-based search is to hoard the complete IoT data on the mobile app of user (i.e., by caching it *a priori*). Unfortunately, IoT data is *massive* and *dynamic*, making complete hoarding a resource-wasteful, time-consuming, and error-prone solution, due to outdated data. *Clearly, there is a need to strike a balance between these two alternative solutions, the solution of no hoarding, and the solution of full hoarding.*

In this article, we study the problem of *prefetching (or hoarding) the most important IoT data blocks from an IIN-SOA s to a mobile user u, without knowing the target of u during navigation.* Such a prefetching functionality is of paramount importance for the maintenance of a reliable IIN-SOA, because mobile devices suffer from intermittent internet connectivity. This results in high latencies for accessing the IIN-SOA and, subsequently, to location inaccuracies. Our proposed framework, named *Grap (Graph Prefetching)*, decides which pieces of the IoT data space are required by a user during navigation in two steps: (i) an offline pre-processing step, named *createDG*, during which a target building is structurally analyzed as a graph to identify important areas in a target building; (ii) an online search step, named *Graph-Distance A*-based* (GDA) algorithm, during which these multiple "virtual" targets are iteratively explored using domain-specific indoor heuristics. Grap results in an intelligent prefetching service that hoards spatial indoor context on the mobile device of *u* whenever *u* has network connectivity. In cases of intermittent connectivity, *u* localizes itself from its local hoarded data.

[1]Sept. 08, 2014, Gartner Inc., URL: https://goo.gl/c6VTWG.
[2]Anyplace, URL: https://anyplace.cs.ucy.ac.cy/.

A preliminary formulation of the *IoT data prefetching* problem has appeared in our previous work [12]. Our prior work was established on the assumption that historic user trajectories inside buildings were available to solve this problem efficiently. In practice, however, such user trajectories are hard to obtain due to rising privacy concerns [11] and respective legislation (e.g., EU General Data Protection Regulation). Additionally, even though the discussion and examples in our article only focus on indoor spaces, situated within a building and isolated from the outside world through some physical door, our propositions can be extended into outdoor spaces as well, which also aim to support effective offline support for the disconnected workforce. For example, even though Google Maps maintains centrally public transportation timetables, predicted traffic, satellite, or terrain tiles, labels and description of POIs, none of these are available, at the time of this work, when a user is operating in offline mode. The *IoT data prefetching* propositions of this work could therefore provide a way for a user to download the most relevant data blocks from the navigation service, providing full resolution to the available data and taking into account issues of intermittent connectivity and limited data availability. Overall, our contributions in this work are summarized as follows:

- —We propose a generalized framework, named Grap, for prefetching IoT-based location data that yields high prefetching accuracy and high localization performance under intermittent network connectivity.
- —We propose a dependency graph generation technique, named *createDG*, during which a target building is structurally analyzed in an offline manner to determine the most significant parts of a building. We also propose GDA, which is a multi-target Graph-Distance A*-based algorithm that chases multiple targets iteratively using indoor domain-specific heuristics.
- —We evaluate our design with extensive experimentation and analysis on real datasets that we obtained through an open source indoor navigation architecture (IIN-SOA) that we have developed over the years and that has won several awards for its accuracy and utility [31].
- —We show how Grap has been integrated to an open source IIN-SOA. This exercise has helped us to validate that our propositions are practical and can be implemented in a real system.

The remainder of the article is organized as follows: Section 2 provides background details and related work on IoT-based IIN-SOA, prefetching, and graph-based algorithms. Section 3 provides our system model and formulates the problem. Section 4 presents the Grap framework and its internal components. Section 5 provides an overview of our real prototype system. Section 6 presents our experimental methodology and results while Section 7 concludes the article.

2 BACKGROUND AND RELATED WORK

In this section, we provide background and related work on IoT-based IIN-SOA, prefetching, and graph-based algorithms, upon which our presented techniques are founded.

2.1 IoT-Based IIN-SOA and Anyplace

A major characteristic of IoT is the inter-connectivity of things in the network and that the IoT architecture must ensure their proper operation in both the physical and the virtual world [15]. While this can be achieved by taking into consideration the scalability, extensibility, adaptiveness, modularity, and interoperability of heterogeneous devices, things may move geographically, may need to communicate in real-time, and interact dynamically [26]. Therefore, a *Service-Oriented Architecture* (SOA) [15, 29], which treats a complex system as a set of well-defined simple objects of subsystems that can be re-used and maintained individually [29], is a good choice for IoT-demanding features.

An example of an IoT-based IIN-SOA is our own Anyplace [31], which follows an SOA design that allows plug-n-play additional modules, either for extending system capabilities—by implementing new features—or for enhancing user-experience by improving existing functionalities (e.g., map-matching and sophisticated data fusion to increase localization accuracy). The public Anyplace service has to this date supported more than 100,000 real user interactions, with many more users using its standalone installations. The Anyplace native Android application is composed of the *Navigator* and the *Logger* that can benefit from Wi-Fi fingerprinting [19, 21, 32] available under this platform. The *Logger* application enables users to record Wi-Fi readings from nearby Wi-Fi *Access Points* (APs) and upload them to our *Server* through a Web 2.0 API (in JSON). It is used by volunteers for contributing Wi-Fi data and for crowdsourcing the radiomaps (RMs) of buildings [6] (i.e., four directional fingerprinting in multiple rounds to remove noise). The *Navigator* allows users to see their current location on top of the floorplan map and navigate between POIs inside the building with high accuracy (i.e., 1.96 meters at the Microsoft Indoor Localization Competition at ACM/IEEE IPSN '14 [19]). The localization function *loc*() of Anyplace comprises the following phases: in the first offline phase, it records the so-called *Wi-Fi fingerprints*, which comprise *Received Signal Strength* (RSS) indicators of Wi-Fi APs at certain locations (*x,y*) pin-pointed on a building floor map (e.g., every few meters). In the second offline phase, the Wi-Fi fingerprints are joined into a NxM matrix, named the *Wi-Fi RadioMap*, where N is the number of unique (x, y) fingerprints and M the total number of APs. Finally, a user can compare its currently observed RSS fingerprint against the RM to find the best match, either on the server side or *in situ* at the smartphone device after downloading the whole RM by using known algorithms such as KNN or WKNN [14].

One fundamental drawback of the Anyplace's final RSS fingerprint comparison step is that users on-the-move require communicating with the Anyplace service continuously over a Wi-Fi network, which negatively affects their localization *accuracy* when there is *intermittent connectivity* [27]. The alternative of downloading to the smartphone device the massive RM (e.g., WiGLE.net had 5.4 billion unique records by November, 2017) prior to the localization may potentially lead to *high overhead time*, *waste of limited smartphone battery*, as well as *high cost* due to expensive mobile data plans. Thus, this study aims at advancing the literature with intelligent prefetching techniques that allow users to carry out accurate indoor navigation in a targetless manner using only a selective portion of the IoT-based data residing on the IIN-SOA provider, which provides high performance (time and network capacity) but also resilience to intermittent connectivity scenarios.

2.2 Prefetching and Mobile Connectivity

Caching is the process of storing data locally, so that future requests for that data can be served faster. It finds applications in the complete spectrum of the computing memory hierarchy (i.e., from low-level hardware to high-level software). *Prefetching (or hoarding)* on the other hand, is the process of *downloading* and then *storing* data locally in a cache, so that future requests for that data can be served in the event of a network failure. Prefetching was originally used in file systems [23] for optimizing the I/O operations on a disk by caching disk blocks that will be needed in the near future. File system prefetching was also adopted by network and distributed environments such as CODA [22], which is a distributed file system that provides novel features such as the disconnected operation that makes server data available in mobile computing environments when the network connection is lost. Prefetching is also used on the web for allowing browsers to pre-load frequently visited web links and content (like music and videos) to reload them faster and therefore optimize the web navigation performance [4, 20].

In mobile networks, the connectivity in indoor spaces is often intermittent, referring to the frequent disconnection of a mobile node in random time intervals. This often occurs due to the following two reasons [27]: (i) there is a gap between the coverage of two APs and thus the connectivity experienced by mobile users passing by will likely to be intermittent and (ii) because of physical obstacles as well as high mobility patterns of the mobile users. In either case, intermittent connectivity may break data connections, if the connectivity disruption between a mobile node and an AP is long enough and the available transfer rate provided is below a certain threshold.

Several techniques have been proposed to tackle the intermittent connectivity problem in mobile networks such as mobility management [16], cooperative downloading schemes [25], AP deployment algorithms [34], prefetching [9], routing [33], or combinations of those techniques [27]. Prefetching systems in mobile networks aim at hiding the frequent disconnections and/or the latency of data transfers over poor and intermittently connected environments. In particular, a prefetching system predicts what data an application will request in the future and speculatively retrieves and caches that data in anticipation of those future needs [9]. All these prior solutions are not directly applicable to our formulated problem as we both do not have access to historic user data (used for learning and future predictions) but only data capturing the structural semantics of buildings.

2.3 Graph-Based Search

Our proposed *Grap (Graph Prefetching)* framework uses preprocessed building topology graphs to search for important areas that have to be prefetched. Graph-search algorithms can be classified into: *Uninformed (blind) algorithms* and *Informed search algorithms* [18]. Uninformed algorithms have no domain knowledge of the problem state and traverse the graph by using unsophisticated approaches that may have only information about the state, the successor function, the goal test, and the path cost. Uninformed search algorithms are characterized by the order in which the nodes are visited to reach the goal-solution and include approaches such as *Random-Walk, Breadth-First Search (BFS), Depth-First Search (DFS), and Iterative Deepening*. Informed search algorithms on the other hand are characterized by a *utility* in scanning the solution space to reach the goal. These algorithms utilize some kind of an evaluation function that assesses a set of options either brute-force (e.g., BFS), stochastically (e.g., simulated annealing and hill climbing) or using some heuristic function that assesses the distance of the current solution from a target (e.g., A* heuristic).

However, AI research focuses on simple *prediction* graph-related problems by applying Machine Learning (ML) techniques. For example, predicting a sub-structure of a given graph for mobility prediction of wireless users using artificial neural networks or variants is proposed in [8]. The hard part in ML techniques is *training*, for example a neural network, with real annotated location trajectories, because such trajectories have to be collected at scale and there are rising privacy concerns behind massive location tracking of the mobile workforce [11]. In the context of this work, we designed a framework that proposes an informed-search algorithm that is independent of sensitive user-centric data and can operate solely based on structural semantics available in building plots (i.e., no user data).

3 SYSTEM MODEL AND PROBLEM FORMULATION

This section formalizes our system model, assumptions, and problem. The main symbols and notation used in the rest of the article are summarized in Table 1.

3.1 System Model

We assume a planar indoor area I containing a finite set of locations that are partially covered by a set of Wi-Fi APs $\{ap_1, ap_2, \ldots, ap_M\}$. Each ap_i has a unique ID (i.e., MAC address) that is publicly

Table 1. Key Notation and Symbols

Notation	Description
I, l	Indoor space, location inside I
ap_i, AP, M	Access point i, set of all ap_i, $\|AP\|$
s, u, U	Localization service, user requesting localization, set of all u
r, R	Timestep of current localization request, set of all prior r
V_l, V_r	Fingerprint (RSSs and ap-IDs) at location l (used in offline mapping), Fingerprint at request r
RM, RM_r^u	RadioMap mapping V_r to l, Partial RadioMap of u at r ($\|RM_r^u\| \ll \|RM\|$)
l_r^u, λ_r^u, $loc()$	Actual, estimated location of u computed by $loc()$ at r
θ	RSS threshold below which u is considered disconnected from s
A_r, T_r, C_r	Point accuracy, CPU time, and network capacity costs for request r
α, K	Best possible A_r at r, dwell time (time required to download K data blocks)

broadcast and passively received by anyone moving in the coverage of ap_i. The signal intensity at which the ID of ap_i is received at location l is termed the RSS of ap_i at l, where -110 indicates when an ap_i is out of reach. The set of RSS values measured and the ap-IDs read at a location l is termed *fingerprint* V_r at timestep r of location l.

We further assume an indoor positioning server s that has constructed beforehand an RM that is a database of offline *fingerprint* V_l measured at various locations $l \in I$. Any subset of RM rows will be denoted as *partial RM* (RM_r^u), requested by some user u at timestep r. Server s uses a localization function $loc()$ to compute from RM (or RM_r^u) an estimation $\lambda_{l'}^u$ of some new unknown location l' given the fingerprint $V_{l'}$. An RM_r^u includes a set of entries that are geographically close and surround l allow for a better estimation $\lambda_{l'}^u$ (or $\lambda_{r'}^u$). Therefore, the fingerprints are spatially grouped into equal-sized blocks at s for facilitating the localization process.

Since we assumed that the arrangement of Wi-Fi APs in I results in partial coverage and weak RSS at some locations, we can define an RSS threshold θ, below which the data transmission rate is practically zero. Such a definition will help us to articulate our analytical and experimental arguments pertinent to intermittent network connectivity. Specifically, client u with a fingerprint V_r at timestep of request r is practically offline if the maximum signal strength max_{V_r} it receives from any covering ap_i is below threshold θ, i.e., $max_{V_r} < \theta$. Formally, this is captured by the $connected_r$ definition that follows:

$$connected_r = \begin{cases} 0, & if\ \max_{V_r} \leq \theta \\ 1, & if\ \max_{V_r} > \theta \end{cases}. \tag{1}$$

The number of blocks that can be downloaded while being $connected_r$ depends on the amount of time that the user stays connected at r, denoted as the *dwell time*. For experimentation purposes, *dwell time* is configured to K blocks.

3.2 Research Goal and Metrics

Research Goal: Enable a mobile user to consecutively localize itself accurately and efficiently in an indoor environment, where connectivity is intermittent, using an IIN-SOA holding RM.

The efficiency of the proposed techniques to achieve the above research goal is measured by the following client-side metrics: (i) the *point accuracy* achieved by u while localizing, (ii) the *CPU Time* required for the localization, and (iii) the *Network Capacity* for the complete localization operation.

Definition 3.1. Point Accuracy (A_r) is the Euclidean error distance between the location estimation λ_r^u (i.e., the estimated location over the partial RM_r^u) and the actual location l_r^u of user u

(i.e., the estimated location over the whole RM) at the timestep of localization request r, given by: $A_r = |\lambda_r^u - l_r^u| + \alpha$ (α is the lower bound localization accuracy achieved by any function *loc()*).

Definition 3.2. CPU Time (T_r) is the processing time used on u for running the localization function *loc()* given a localization request r. This includes the time costs for transmitting/receiving the RM (or RM_r^u) and for executing the loc() function on the mobile device.

Definition 3.3. Network Capacity (C_r) is the total number of messages $|RM_i|$ needed on u for localization request r.

Our research goal can be expressed by the minimization of the following three objective functions:

$$\min F_A = \frac{1}{|R|}\sum_R A_r, \qquad \min F_T = \frac{1}{|R|}\sum_R T_r, \qquad \min F_C = \frac{1}{|R|}\sum_R C_r. \tag{2}$$

3.3 Baseline Approaches

Existing techniques for indoor localization using Wi-Fi fingerprinting can be categorized as follows:

(i) Server-Side Approach (SSA): In this approach, u starts out by obtaining a V_r that is shipped to s. The location estimation λ_r^u of u is computed on s executing $loc(V_r, RM)$, and transmitted back to u. In this approach, the values of the objective functions F_T and F_C are *minimum*, as we will show in both the performance analysis and evaluation sections, given that u does not carry out any computation. The drawback of this approach is that it suffers from intermittent connectivity. Particularly, in the case of successful communication, the best possible localization can be achieved by *loc()* using RM (i.e., the case where point accuracy $A_r = \alpha$). Otherwise, the location estimation of u at timestep r' (i.e., $\lambda_{r'}^u$) can only be inferred upon the last estimation computed, i.e., $A_r = |\lambda_{r'}^u - l_r^u| + \alpha$. We observe that the parameter A_r grows worse as u moves further away from l_r^u. Therefore, in the SSA approach, the objective function F_A for point accuracy *grows* worse, as θ gets smaller.

(ii) Client-Side Approach (CSA): In this approach, u downloads the whole RM from s. Assuming that the download process has completed, u can obtain a localization estimate by obtaining a V_r that is compared against RM. This approach minimizes the objective function F_A, since it achieves the best possible point accuracy $A = \alpha$ and it is not affected by intermittent connectivity. The drawback of the CSA is that the objective functions F_T and F_C are *maximum*, as we will again show in both the performance analysis and evaluation sections, given that u has to both download the complete RM but also delay its computation by going through the complete RM. Although this is a one-time cost, which might seem bearable for continuous localization, it can still be prohibitive in real-world scenarios where RM changes quickly over time or for complex building structures.

4 THE GRAP FRAMEWORK

In this section, we describe our general Graph Prefetching (Grap) framework and discuss its various parameters and techniques.

4.1 Outline of Operation

The Grap framework comprises the following conceptual steps (see Figure 1): (i) it structurally analyzes building topologies yielding a Dependency Graph (DG) that represents the probability/

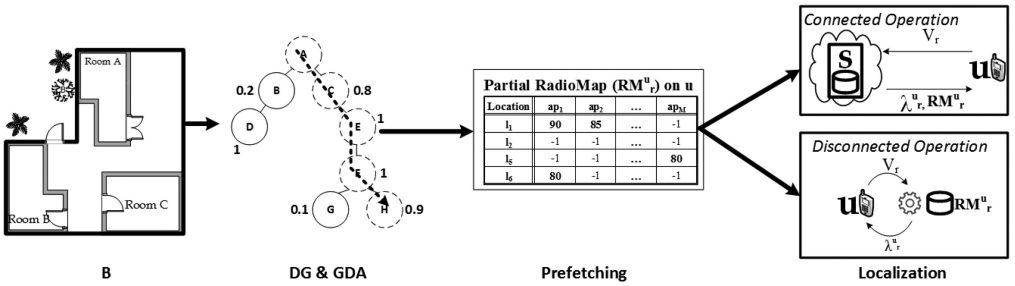


Fig. 1. The conceptual steps of the proposed Grap framework.

likeliness of users visiting other nodes in the building. This can be calculated using problem-specific information such as structural information of a buildings or even historical traces as we used in [12]; (ii) a proposed GDA algorithm then finds the nodes with the highest probability to be visited from the current location of u; (iii) the fingerprints falling within the selected nodes form a partial RM, RM_r^u; and (iv) in connected operation, u forwards its fingerprint V_r to server s and receives its location λ_r^u and RM_r^u while in disconnected operation, u calculates λ_r^u locally using only the prefetched RM_r^u. Using the above framework, Grap prefetches a small group of RM entries (*partial RM*) on u, which can aid localization at u in case it looses connection to s. In this way, it overcomes the drawbacks of both SSA and CSA discussed in the previous section.

4.2 Graph-Distance A* (GDA) Algorithm

In this section, we discuss our proposed GDA algorithm and its major components, namely the *createDG*() technique for generating the DG and the *graphSearch*() technique for selecting the best possible partial RM, RM_r^u. We also discuss how each single parameter of GDA influences its performance in terms of the three performance metrics we defined in Section 3, i.e., point accuracy A_r, CPU time T_r, and network capacity C_r.

4.2.1 Dependency Graph Generation. The *createDG*() technique summarized in Algorithm 1 constructs a DG and connects the nodes through physical transitions by consulting the actual plan $B = \langle N, E \rangle$ of the building (we ignore the associated raster graphic of the building). The N-set contains the POIs, which refer to rooms, intersections, elevators, staircases, and so on in the building as these have been provided by architects or crowdsourcers (so these are always up-to-date). The E-set contains the corridors, physical pathways, and so on, as these are aligned to floors inside a building. The DG has been proposed to represent the connectivity on indoor POIs as well as the POI's importance in a building (with respect to the probability to be visited) and the POI-to-POI distances. Formally, the DG is defined as a weighted undirected graph $DG = \langle N', E' \rangle$, where

(1) N' is the set of nodes, each corresponding to a POI along with a *self-importance* $s(n_i)$ weight indicating its probability to be visited by a random user. Formally, $N' = \forall(n_i, s(n_i))$.
(2) E' is the set of edges, each corresponding to an *edge* along with an *edge-weight* indicating the L_p*-Norm* distance between two connected nodes. Formally, $E' = \forall((n_i, n_j), d(n_i, n_j))$.

The *createDG*() technique assigns weights to all nodes (lines 3 to 12 of Algorithm 1) by estimating their self-importance in terms of how likely users pass from a particular node (a.k.a. authority hub) to reach their destination. The algorithm also uses a *damping factor* (df) to capture a probability of a random user to continue navigating at any step of the iterative computation. In particular, the importance of a node is defined iteratively and depends on the degree (*number of links*) of that

ALGORITHM 1: *createDG*(): builds the dependency graph

Input: Building plan $B = \langle N, E \rangle$: un-weighted graph, t: convergence iterations
Output: Dependency Graph $DG = \langle N', E' \rangle$

1: $nb \leftarrow neighbors(B)$ ▷ nb: set of neighbors for each node
2: $vs \leftarrow horizon(B)$ ▷ vs: horizon set for each node
3: **for all** $n_i \in N$ **do** ▷ **Step 1: Calculate Node Weight Set** $N' = \forall(n_i, s(n_i))$
4: $s(n_i) = |nb(n_i)| + |vs(n_i)|$ ▷ initialize self-importance of node n_i
5: **end for**
6: $df \leftarrow 0.85$ ▷ df: damping factor
7: **while** $t > 0$ **do**
8: **for all** $n_i \in N$ **do**
9: $s(n_i) = \frac{1-df}{|N|} + df \times \sum_{n_j \in nb(n_i)} \frac{s(n_j)}{|nb(n_j)|}$ ▷ iteratively refine self-importance of node n_i
10: **end for**
11: $t \leftarrow t - 1$ ▷ t: alternatively, iterate until $\sum_{n_i \in N}(||s(n_i)_t - s(n_i)_{t+1}||_2) < t$
12: **end while**
13: **for all** $(n_i, n_j) \in E$ **do** ▷ **Step 2: Calculate Edge Weight Set** $E' = \forall((n_i, n_j), d(n_i, n_j))$
14: $d(n_i, n_j) = \sqrt{(n_i - n_j)^2}$ ▷ d: any L_p norm distance metric (e.g., Euclidean distance)
15: **end for**

particular node as well as on how many nodes in N can be visited via that node (i.e., the so-called *horizon*) to favor nodes with low number of links but high betweenness (such as nodes that bridge two buildings or central corridors). This differentiates our approach from similar approaches like the PageRank algorithm [5], which does not use any horizon or outlook during its computation process. Finally, the *createDG*() technique assigns weights to all edges by calculating the Euclidean distance between them (lines 13 to 15 of Algorithm 1).

4.2.2 Partial Radiomap Selection. In this section, we design an A^* search approach using domain-specific indoor heuristics to explore the DG generated in the previous section to find the best nodes that u should prefetch from s.

In an intermittently connected environment where u might be in a disconnected state during the next timestep, u greedily hoards as much data as possible from s before being disconnected. An A^* search algorithm comprises an evaluation function $f(a, z) = g(a, b) + h(b, z)$, where $g(a, b)$ denotes a cost function that gives the path cost from the start node a to intermediate node b and $h(b, z)$, a heuristic function that estimates the cheapest path from b to the goal z. In our context, both $g()$ and $h()$ are calculated using the graph-distance cost G_2 [3, 17], which reflects the topological constraints and physical entities of a building, such as elevators, corridors, walls, and so on, given that the L_p-norm distances (e.g., Euclidean, Manhattan) are unsuitable [3]. For example, assume two nodes n_i, n_j, and an edge (n_i, n_j) on the graph if and only if there is a physical transition between the two nodes. The Euclidean distance L_2-norm is equal to the line segment directly connecting them. In cases where the path between n_i and n_j contains some intermediate nodes (i.e., $n_i \rightsquigarrow n_j$), then the distance between them, denoted as graph distance estimation $d(n_i \rightsquigarrow^{G_2} n_j)$, is not a direct line from n_i to n_j, but the summation of the Euclidean distances from start node n_i to n_j.

In Algorithm 2, we present the detailed steps of the GDA algorithm (i.e., also denoted as graphSearch()) executed once per timestep r. It is important to recall the fact that we aim at finding the best nodes to be prefetched in a *target-less manner* before u is disconnected from s and therefore there is *no goal state* for $h()$ to be calculated. In line 1 of Algorithm 2 we show how, in the absence of a target, the *graphSearch*() technique initially finds m *virtual targets* that represent the m most

ALGORITHM 2: GDA:=$graphSearch()$: executed at each r and generating a new RM_r^u while u is moving

Input: $DG = \langle N, E \rangle$, RM, n_u: user current location, m: number of virtual targets
Output: Partial Radiomap RM_r^u

1: $H_m \leftarrow top(sort(find(n_u, w, DG)), m)$ ▷ identify virtual targets in DG w hops away from n_u
2: $resultSet := \{\}$; $openSet := \{n_u\}$ ▷ set of nodes evaluated in GDA; set of nodes to be evaluated
3: $g(n_u, n) = 0$ ▷ **Graph Distance Cost** (from n_u to some intermediate node n)
4: $h(n, n_j) = \infty$ ▷ **Heuristic Cost** (from intermediate node n to a virtual target $n_j \in H_m$)
5: $f(n_u, n_j) = \infty$ ▷ **Total Cost** (from n_u toward one virtual target $n_j \in H_m$)
6: **while** $(connected_r)$ **do** ▷ u is connected to s at timestep r according to Equation (1)
7: $n := findMin(openSet)$ ▷ n: Next intermediate node (with minimal cost)
8: **for all** $n_i \in neighbors(\mathrm{n})$ **do** ▷ n_i: Neighbor of next intermediate node
9: **if** $n_i \notin resultSet$ **then**
10: $g(n, n_i) \leftarrow d(n \rightsquigarrow^{G_2} n_i)$ ▷ Graph distance cost of n to n_i
11: **for** $j = 1, \ldots, m$ **do**
12: $h(n_i, n_j) \leftarrow Dijkstra(n_i, n_j)$ ▷ Heuristic cost from n_i to target n_j
13: $f(n_i, n_j) = g(n, n_i) + h(n_i, n_j)$ ▷ Total cost from n_i to n_j
14: **end for**
15: $add(openSet, n_i, min(f(n_i, n_j)|j = 1, \ldots, m))$ ▷ Add n_i using its least cost toward n_j
16: **end if**
17: **end for**
18: $remove(openSet, n)$; $add(resultSet, n)$ ▷ Evaluation for node n has been completed
19: $RM_r^u \leftarrow pRM(resultSet, RM)$ ▷ Build RM_r^u incrementally and stream results to u
20: **end while**

possible destinations of u based on its current node location (n_u). The technique selects those m destinations based on their self-importance $s(n_i)$ of DG presented in the previous section. It is important to explain that these m destinations are selected within w hops from n_u so that these are not very far destinations (we use the notation w^{no} to refer to an unlimited window).

After the m virtual targets are selected and data structures are initialized, in lines 6–19 our A^* approach aims to identify the best possible targets to be explored next using two sets (an *openSet* used for nodes to be evaluated and a *closedSet* used for nodes already evaluated). In an intermittently connected environment, where u might be in a disconnected state during the next timestep, u has to greedily hoard as much data as possible from s before being disconnected. This idea is at the algorithmic level shown by line 6. In lines 7–9, we iterate over the next possible intermediate nodes one node at a time. For each explored node, we calculate $g()$ from a current location n to each neighboring node n_i and calculates $h()$ by running the Dijkstra algorithm to find the shortest path in terms of G_2 from n_i to each potential target $n_j, j = 1, \ldots, m$ in lines 10–14 of Algorithm 2. The node n_i with the lowest $f()$ is recorded in line 15. Given that the user might disconnect at any given moment of the above algorithm execution, the server s streams results back to user u as these become available (i.e., line 19). User u utilizes these updates to build RM_r^u, which will be used for localization in subsequent disconnected states.

One optimization is that in cases where w^{no} is used, the m destinations are not required to be updated among successive GDA executions (i.e., line 1). To further optimize the performance of Grap, a cache on the smartphone's internal storage (e.g., sdcard, flash memory) is used to keep previous RM_r^u. When this optimization process is utilized, the user checks if any of the locally cached RM_r^u can serve its localization request. This reduces the occasions where localization requests to s are initiated and thus, network resources are conserved.

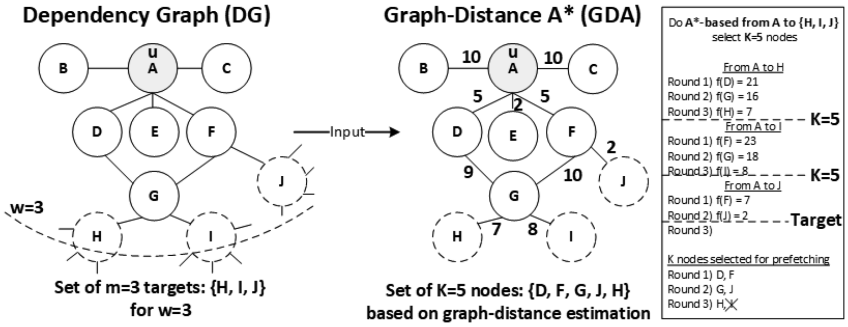


Fig. 2. Example execution of the GDA algorithm.

Example. Consider the scenario in Figure 2 where a user u is at node A and requests a RM_r^u while in a connected operation state. Grap constructs a dependency graph of 10 nodes $\{A, \ldots, J\}$ as illustrated in Figure 2 (left) by analyzing some building plan. The edges represent the physical transitions between the nodes. The *createDG()* function of Grap selects the $m = 3$ targets with the highest probability of being the destination to u within a window $w = 3$. This denotes that the targets should be at least three hops away from node A (i.e., the current location of u). Let us assume that the m most promising targets (i.e., the virtual targets) are nodes H, I, J denoted with dotted circles.

In the second stage of the Grap framework shown in Figure 2 (right), we invoke GDA *graphSearch*() to identify the best possible nodes to prefetch along the way to the m destinations. Let us assume w.l.o.g. that in our example scenario u is enough time to download $K = 5$ nodes before being disconnected from s. At the beginning, GDA expands all neighboring nodes of A and finds the best with respect to $f(n)$ toward each of the m target. From the available options, node D is the best choice towards target H with $f(D) = 21$ and node F toward targets I, J with $f(F) = 23$ and $f(F) = 7$, respectively. Both D and F are selected to be prefetched in round 1. In the second round, GDA selects node G as the best choice towards targets H, I with $f(G) = 16$ and $f(G) = 8$, respectively, and reaches target J. Therefore, nodes G and J are selected to be prefetched and the search toward target J stops. Finally, the algorithm selects nodes H and I as the final nodes to be prefetched with $f(H) = 7$ and $f(I) = 8$, but because we have already selected four nodes to be prefetched, there is space for just one more node. GDA selects node H (and discards node I) because its overall cost function $f(n)$ is better.

4.3 Performance Analysis

We analytically derive the performance of Grap with respect to the estimated Accuracy A, CPU time T and Network capacity in messages C at the client side u. We adopt a worst-case analysis as it provides a bound for all input. Our experimental evaluation in Section 6 shows that, under real datasets, our approach performs more efficiently than the projected worst case. The analysis is based on our system model and ignores any other performance costs not directly associated with the localization phase during the disconnected operation, any offline calculations performed by s (e.g., running the *createDG*() technique) or any idle time at u while s performs calculations (e.g., running the *graphSearch*() technique) since we consider these costs negligible. For ease of exposition, our analysis uses the notation $\mathcal{T}^{TX}$, $\mathcal{T}^{RX}$, and $\mathcal{T}^{P}$ to denote the computational cost needed by u for transmitting, receiving, and processing a single RM entry V_l from s.

Lemma 1. *Grap guarantees an estimated localization accuracy of at least* $A_r = max_{\forall i \in RM}(|\lambda_i^u - l_i^u|) + \alpha$, *for a user* u *at a localization request* r.

Proof. The maximum Euclidean distance error that can be provided by the Grap framework is equal to the maximum distance between any two locations in the whole RM plus the estimated point accuracy constant α. Particularly, let us assume that user u enters a building from its one end and at r requests to prefetch a partial RM, RM_r^u from s. Server s runs our *graphSearch*() technique and generates a RM_r^u composed of the fingerprints associated to a single node around the user's initial location l_r^u. Then u moves to a location $l_{r'}^u$ at the other end of the building, facing disconnections along the whole path. At that location, u runs the $loc()$ function using the prefetched RM_r^u, received from s at the very beginning, and calculates an estimated user location $\lambda_{r'}^u$ that is the farthest location available in RM with respect to $l_{r'}^u$. In this worst-case scenario, Grap provides a $A_r = |\lambda_{r'}^u - l_{r'}^u| + \alpha$. □

Similar to Lemma 1, CSA guarantees the best possible localization accuracy $A_r = \alpha$ because u has downloaded the complete RM and thus $\lambda_r^u = l_r^u$ for every r. On the other hand, SSA has a similar worst-case accuracy bound as the Grap framework given that it also does not have the complete RM.

Lemma 2. *The Grap framework has a computational cost of* $O(\mathcal{T}^{TX} + I' \cdot M \cdot \mathcal{T}^{RX} + I' \cdot \mathcal{T}^{P})$, *where* I' *is the number of RM entries retrieved from* s *and* M *the number of RM dimensions.*

Proof. During the connected operation, u sends a request for localization to s spending $\mathcal{T}^{TX}$ time. Then s responds to u with an estimated location $\lambda_r^u = (x_u, y_u)$ of size equal $\mathcal{T}^{RX}$, as well as $I' << I$ database entries, where each entry has $M + 2$ values, therefore u spends $I' \cdot M \cdot \mathcal{T}^{RX}$ time. The I' entries represent the fingerprints associated with the nodes selected by our GDA approach to be prefetched by u. Finally, during the disconnected operation, u localizes itself using the I' entries spending $I' \cdot \mathcal{T}^{P}$. We can safely assume that $I' < I$, therefore, adding all computational time costs in an asymptotic manner yields $O(\mathcal{T}^{TX} + I' \cdot M \cdot \mathcal{T}^{RX} + I' \cdot \mathcal{T}^{P})$. □

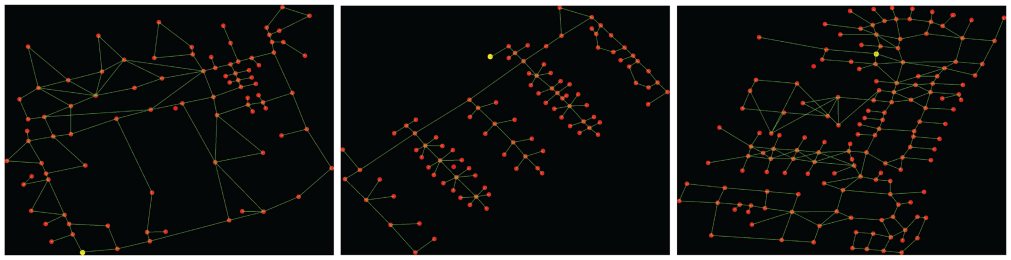
Similar to Lemma 2, CSA has a *max* computational cost $O(\mathcal{T}^{TX} + I \cdot M \cdot \mathcal{T}^{RX} + I \cdot \mathcal{T}^{P})$ and SSA a *min* computational cost $O(\mathcal{T}^{TX} + \mathcal{T}^{RX})$ for each r. The message cost for all three techniques CSA, SSA, and Grap is thus $O(I)$, $O(1)$ and $O(I')$, respectively.

5 GRAP PROTOTYPE IMPLEMENTATION

In this section, we describe the system that we implemented to evaluate the efficiency of the Grap framework and to validate that our propositions can easily be integrated in a real system. Our system comprises the Grap Evaluator and the Grap Navigator. Both components where implemented on-top of our in-house *Anyplace* IIN-SOA, which allows entities (i.e., users, companies, organizations, and so on) to realize indoor information management systems, including product search and POI navigation, on top of existing wireless network infrastructure by leveraging rich multi-sensory data available on smartphones (see Section 2.1).

5.1 Grap Evaluator

The aim of the Grap Evaluator is to allow trace-driven evaluation and visualization of the presented techniques with data available through the public Anyplace IIN-SOA. The evaluator comprises a *data connector*, a *simulator*, and a *visualizer*. The data connector connects to Anyplace and downloads IoT-data available through its open API. For this task, we had to introduce some new JSON endpoints to Anyplace. The *simulator* then implements all the algorithms along with evaluation metrics discussed in the next section. The *visualizer* shows the graph generated by various building plans (e.g., see Figure 3) but also allows tracing the presented algorithms to understand their behavior. All components were written in JAVA, compiled using JDK 8.0 and comprising ≈16,040



(a) Hotel (b) Campus (c) Mall

Fig. 3. *Datasets*. The topologies of the three datasets obtained through the Anyplace IIN-SOA JSON API. The visualization is carried out with the Grap Evaluator/Visualizer.

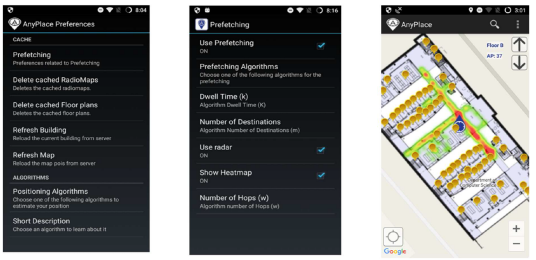


Fig. 4. The Grap Navigator (left) showing the menu for switching the prefetching feature on/off; and (center) varying the GDA parameters; (right) viewing the prefetched IoT data and user's location.

lines of- code (LOC). After concluding our trace-driven experimentation, we ported the prefetching algorithms to an android client app presented next.

5.2 Grap Navigator

The aim of the Grap Navigator was to create a proof-of-concept realization of our propositions in a real mobile indoor search, exploration and navigation tool. We particularly adapted the Anyplace navigator with options to introduce intelligent prefetching and caching. The installation package of the Grap android client we developed was only around 5MB. Overall, our code consists of approximately 34,575 LOC, including 2,010 lines of XML elements that go in the Manifest file (settings) and the user interface XML descriptions of the navigator.

Our prototype GUI in Figure 4 allows a user to select the prefetching functionality to navigate in a building without suffering by intermittent connectivity along with supplementary control features that are useful for demonstration purposes. The GUI allows a user to visualize the RM_r^u prefetched at each localization step on a map. The interface uses the actual plan of a building and overlays a heatmap that represents the signal strength values in dBm collected from nearby APs (i.e., the fingerprints) and prefetched on the user's smartphone. Figure 4 (right) shows a heatmap of the partial RM prefetched on the mobile device, which is considerably smaller in size than the fingerprints of all floors.

6 EXPERIMENTAL EVALUATION

This section presents an extensive experimental evaluation of our proposed Grap framework. We start out with the experimental methodology and setup, followed by our experimental studies.

6.1 Methodology

Real Datasets: We constructed three realistic datasets from three real IoT-data obtained through the publicly available API of our Anyplace IIN-SOA described in Section 2.1.

Hotel Pittsburgh: This dataset was collected at the Wyndham Grand Pittsburgh Downtown Hotel in Pittsburgh, PA. In particular, it consists of around 500 reference fingerprints taken from ∼308 Wi-Fi APs installed in the two floors of the hotel and neighboring buildings. The structure of the hotel is of a rectangular shape as shown in Figure 3(a), it covers around 6,500m^2 and consists of 201 POIs and 247 edges.

Campus CSUCY: This dataset was collected at the Department of Computer Science (CS), University of Cyprus. In particular, it consists of 5,000 reference fingerprints taken from ∼266 Wi-Fi APs installed on the four floors of the CS and neighboring buildings. We collected our data by walking over a path that consists of ≈2,900 locations. The structure of the CSUCY campus is of a bus-like shape as shown in Figure 3(b), it covers around 2,500m^2 and consists of 397 POIs and 440 edges.

Mall of Cyprus: This dataset was collected at the Mall of Cyprus. In particular, it consists of 800 reference fingerprints taken from ∼279 Wi-Fi APs installed on the two floors of the mall and neighboring buildings. The structure of the mall is of a mesh-like shape as shown in Figure 3(c), it covers around 18,500m^2 and consists of 214 POIs and 289 edges.

User Traces: To evaluate the scalability of our propositions, we generated realistic user traces of various scales where a user follows and localizes at pre-defined locations. The traces are designed for our evaluation study to show the performance versus accuracy tradeoff in using Grap. Particularly, the distinct locations are of fixed distance between each other (e.g., around 5 meters) and the size of traces varies from 15–30 localizations steps (i.e., a user moving in a multi-floor building and travels around 50–150m). The RM is also spatially grouped into equal-size blocks that correspond to the POIs of a particular building. Both the traces and blocks can be viewed and verified using the Grap visualizer described in Section 5.

Algorithms: We compare the proposed Grap framework with two Anyplace (no-prefetching) baseline approaches and two Grap (with prefetching) baseline approaches.

Anyplace (no Prefetching) Baseline Approaches: SSA and CSA, as described in Section 3.3, do not prefetch any localization data since the former keeps the whole RM at the server-side and the latter downloads the whole RM on the mobile device of the user u prior localization.

Grap (with Prefetching) Baseline Approaches: The proposed *GDA* approach of the *Grap* framework, as described in Section 4, the BFS approach that selects nodes to be prefetched level-by-level based on user's current location starting from the neighbor nodes before moving to the next level nodes and the *Random* (*RND*) selection approach, which selects nodes to be prefetched randomly.

Metrics: Our cost metrics are *CPU Time* (*T*), *Location Accuracy* (*A*), and *Network Capacity* (*C*) as defined in Section 3.2. The mean and standard deviation of the results is shown with error bars in all experimental studies that follow, each entailing 37 localization steps (i.e., the route length of trajectories in our experiments).

Parameters: In all experiments that follow, the simulation parameters were configured as follows: dwell time $K = 15$ (i.e., time required to download K data blocks), number of virtual targets $m = 3$, lookahead window $w = 3$, effective network threshold $\theta = -40$dBm, and localization method = *WKNN*. The influence of each of those parameters on the proposed approach is investigated individually in Experiments 2–5 (Sections 6.3 to 6.6) by fixing the rest of the parameters accordingly.

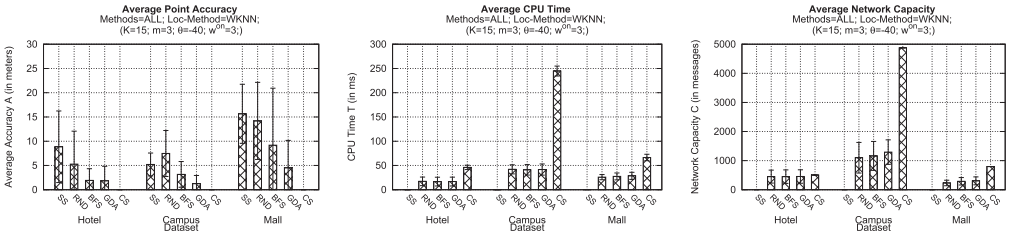


Fig. 5. Experiment 1—Performance Evaluation: GDA evaluation in terms of average point accuracy A (left), average CPU Time T (center), and network capacity C (right) in all datasets.

6.2 Experiment 1: Performance Evaluation

In the first experiment, our main target was to assess the performance objectives A, T, and C for the compared algorithms. As shown in Figure 5, the no-prefetching approaches CS and SS are the extreme cases, providing the best A and the best resources consumption, respectively, demonstrating a clear tradeoff between A versus (T and C). The proposed *GDA* approach of the *Grap* framework performs well ($<5m$) and better than both prefetching RND and BFS approaches in all datasets. In particular, GDAprovides an average $A = 1.84$m in the Hotel dataset that is an improvement of 65% w.r.t. RND and 40% w.r.t. BFS. In the Campus dataset, GDA provides an average $A = 1.26$m that is an improvement of 83% w.r.t. *RND* and 60% w.r.t. BFS. Finally, in the Mall dataset, *GDA* provides an average $A = 4.52m$ that is an improvement of 68% w.r.t. *RND* and 50% w.r.t. *BFS*. The accuracy of GDA is relatively close (i.e., 2.54m, on average) to the best possible localization A of the CS approach and provides an improvement of at least >70% in all three datasets over the SS approach. This shows that GDA calculates and prefetches an almost best set of fingerprints (partial RM, RM_r^u) and therefore it successfully overcomes the intermittent connectivity issues.

We observe that all approaches sacrifice performance (i.e., T and C) for the sake of better quality (i.e., A). The CS approach requires the highest T (118.6 msec, on average) and C (2,060 messages, on average) since it downloads the whole RM (the actual numbers are summarized in Section 6.1) and uses all fingerprints at each localization step. For the Campus dataset, which entails one order of magnitude more fingerprints, this drawback is even more apparent. The prefetching approaches including the proposed GDA require less T and C than CS, since they utilize a partial RM, and more resources than SS, which does not download any fingerprints and does not calculate its location locally at the client side. In particular, GDA requires 28msec more T and 683 more messages, on average, compared to SS. In some cases, the proposed GDA approach consumes slightly more resources than the other two prefetching techniques due to the fact that it performs more sophisticated computations (e.g., calculates the graph distance toward m destinations) to calculate the RM_r^u than the RND and BFS approaches. However, the slight increase on the resources consumption offers much better A, as discussed earlier.

6.3 Experiment 2: Dwell Time (K)

Experiment 2 examines how dwell time K influences the behavior and the performance of the proposed Grap framework. Recall that K represents the available time for downloading the fingerprints associated to K nodes of the DG and therefore represents the size of the partial RM, RM_r^u, which is downloaded by u when connected and processed by u when in a disconnected state. In particular, in this experiment we evaluate the performance of all prefetching approaches (RND, BFS, and GDA) that incorporate the K parameter in their solution in terms of A, T and C. We also include the performance of the no-prefetching (i.e., CS and SS) approaches that are independent of K for comparison purposes.

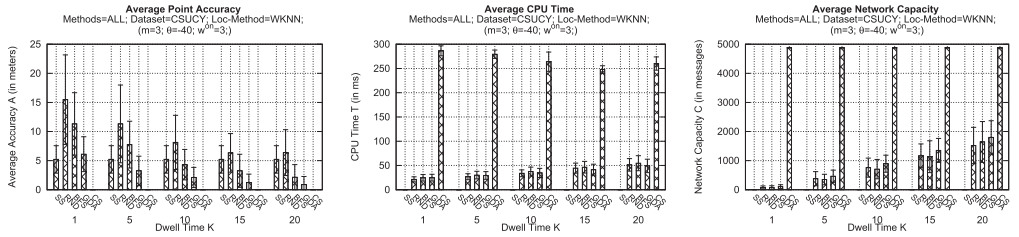


Fig. 6. Experiment 2—Dwell Time (K): examining the *GDA* accuracy (left), CPU time (center) and Network Capacity C (right) while varying the Dwell Time (K).

Figure 6 (left) shows that all prefetching approaches are positively affected by the increase of K, since high K means that there is more time to download more fingerprints for a more fine localization. However, this negatively affects the two performance metrics (i.e., T and C in Figure 6 (center) and (right)) since the increase of K results in more fingerprints to be downloaded during a connected state and a larger RM_r^u to be processed while localizing in a disconnected state. Moreover, the results in Figure 6 also show that the delicate selection of fingerprints by the proposed GDA approach overwhelms the absence of a large number of fingerprints when K is small since the provided A varies from 6m for $K = 1$ to 0.8m for $K = 20$ and therefore it is influenced less than its counterpart prefetching approaches. In particular, GDA provides an average of 75% better A than RND, around 52% than BFS and it is preferable than the no-prefetching SS approach in all cases except the extreme case for $K = 1$, where the amount of prefetched fingerprint does not suffice to outweigh the SS approach.

Note that all prefetching approaches will reach an A equal to the best possible A of the CS approach when the dwell time of a localization step is enough for downloading the whole RM. This increase of K, however, results in an increase on the resource consumption with the results of the CS approach showing the T and C needed in the worst case. The slight variations between the prefetching approaches in terms of T and C for the same K are due to the additional effort needed for calculating the K nodes, which consequently selects K different nodes with varying number of fingerprints.

Clearly, there is a tradeoff between the dwell time $K << N$ and the benefit (i.e., the A, T, and C metrics we defined). Particularly, with larger K, better A is expected on the one hand but more C and T is spent, since $K \rightarrow N$ and $I' \rightarrow I$. If user u sets $K = N$ then $I' = I$ and $RM_r^u = RM$ and u will receive the whole RM. In this case, Grap is the same as the CSA described earlier.

6.4 Experiment 3: Effective Network (θ)

Experiment 3 examines the sensitivity of the proposed GDA approach and all other approaches for various θ parameters. Recall that the θ parameter represents the effectiveness of the network with respect to a user being connected, meaning that the lower the θ of a network is, the less intermittent connectivity inside a building exists and therefore the highest the probability of user u to be connected while navigating in the building.

Figure 7 (left) shows that all prefetching approaches are positively affected by the decrease of θ since user u is more frequently connected and therefore communicates with the server s more often for enriching its RM_r^u with more nodes and consequently more fingerprints. The proposed GDA approach provides, however, the best A in all cases that increases while the θ parameter decreases due to the fact that its has more opportunities to download an efficient set of nodes and represent the actual path that the user will follow as well, as there is an increased probability to correct possible flaws at the initial calculations. In particular, GDA provides a poor A of around

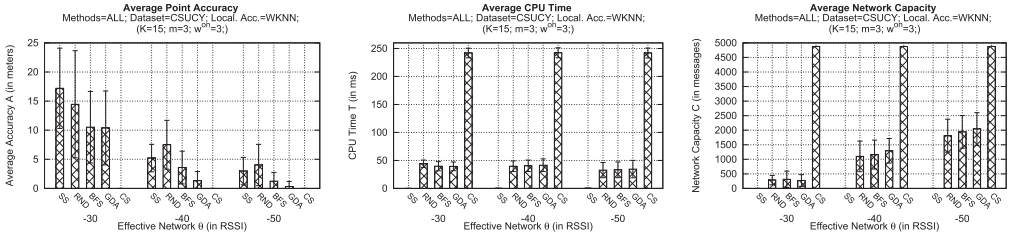


Fig. 7. Experiment 3 — Effective Network (θ): examining the *GDA* accuracy (left), CPU time (center) and Network Capacity C (right) while varying the effective network θ threshold.

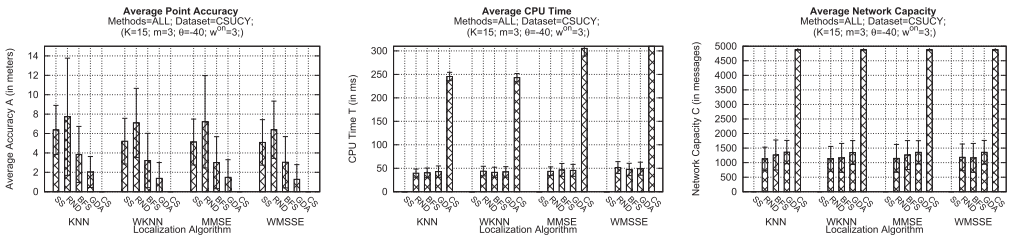


Fig. 8. Experiment 4—Localization Methods: examining the *GDA* accuracy (left), CPU time (center), and network capacity C (right) with respect to various localization methods.

10m for $\theta = -30$dBm that is 58% better than the accuracy provided by RND and 2% better than the BFS and a fine A of 0.33m for $\theta = -50dBm$ that is 75% better than the one provided by RND and 43% better than BFS. Regarding the no-prefetching approaches the SS is influenced from the network effectiveness more than any other approach, since for high θs (−30dBm) and consequently for high intermittent connectivity, u cannot communicate with s and therefore it cannot localize and navigate in the building. Therefore, SS provides worse A than both BFS and GDA in all cases, due to the fact that the latter prefetch fingerprints when connected to localize at the smartphone in cases of disconnections. On the other hand, the CS approach is independent to the θ parameter since u downloads the whole RM *a priori* and does not require any communication with s during navigation.

In terms of resource consumption, however, the CS is the worst since it utilizes maximum resources in all cases irrespective of the actual needs of the localization process. The SS approach is the best in these performance metrics since it requires the minimum resources at each localization step. The most important information, however, with respect to T and C comes from the prefetching approaches since the results show that all approaches are not influenced by the θ parameter and provide similar results in all cases. This is another major benefit of the proposed GDA approach since it provides considerable better A by utilizing similar resources compared to RND and BFS.

6.5 Experiment 4: Localization Algorithms

Experiment 4 examines the impact of various localization algorithms (i.e., K-Nearest Neighbors (KNN), Weighted-KNN (WKNN), Minimum Mean Square Error (MMSE), Weighted-MMSE (WMMSE)) available in the Anyplace IIN-SOA [32] on the performance of all approaches. Any localization technique can be used in the *loc()* step of the Grap framework for calculating the user's current location by utilizing a (partial) RM at every localization step. The results in Figure 8 show that neither the prefetching approaches nor the no-prefetching approaches are considerably

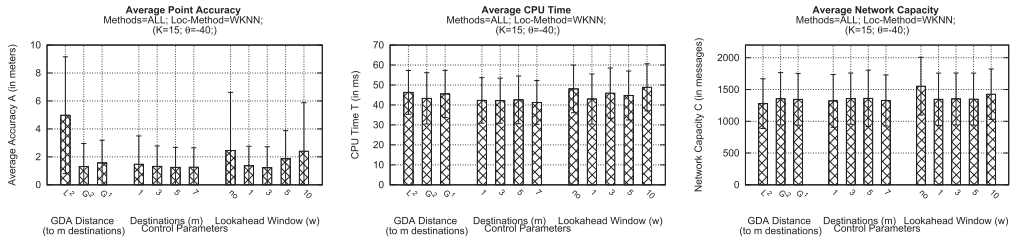


Fig. 9. Experiment 5—Control Parameter Experiments: examining the *GDA* accuracy (left), CPU time (center), and network capacity C (right) with respect to various control parameters.

influenced by the localization algorithm since they provide similar results with respect to A, T, and C in all cases. In general, all approaches provide slightly worse A when the KNN algorithm is used and utilize almost negligibly more resources. In all cases, however, the proposed GDA approach provides better results than all prefetching approaches and successfully adopts the tradeoff between A and resources consumption T and C with respect to the no-prefetching approaches, SS and CS.

6.6 Experiment 5: GDA Sensitivity Analysis

In these experiments, we examine several control parameters of the proposed GDA approach and how these parameters influence its performance in terms of A, T, and C. Recall that for the Experiments 2–4 we presented above, we already evaluated the parameters that had to do with the system configuration (e.g., effective network, dwell time, and localization algorithms), but not the algorithmic parameters of Grap that we carry out in this section.

The first control parameter experiment in Figure 9 (left) studies the GDA distance parameter as discussed in Section 4.2, which is the $h(n)$ heuristic of the evaluation function $f(n)$ of our A*-based approach that estimates the cost from node n to each of the m virtual targets. We examined three different $h(n)$ heuristics: (i) L_2, which calculates the Euclidean distance from n to m nodes, (ii) G_2, which calculates a graph distance by using the Dijkstra algorithm, and (iii) the greedy G_1 heuristic, which finds the shortest path by using the (node) weights of the dependency graph DG (i.e., the self-importance). The second control parameter experiment varies the number of virtual targets m taken into consideration at each iteration. The third control parameter experiment examines the lookahead window parameter that represents the maximum distance in number of hops between the current node and the m nodes. The w^{no} means that there is no constraint in the number of hops and therefore the *GDA* approach finds the m most popular virtual targets of the whole building.

The results of the first control parameter experiment in Figure 9 (left set of parameters at each plot—for various GDA distance heuristics) show that the proposed approach performs better when the graph distance G_2 estimation is used, since it provides about 74% better A than L_2 and 16% better accuracy than G_1, utilizing less T and similar C, at the same time. This is due to the fact that the G_2 heuristic can more easily adopt the constraints and explicit characteristics of an indoor environment and can more easily provide a more representative distance cost between two indoor locations. The variations on m of the second control parameter experiment in Figure 9 (center at each plot) show that GDA is slightly affected by this parameter and demonstrate a slight preference for an m that is neither too high (e.g., $m = 7$) nor too low (e.g., $m = 1$). This is due to the fact that a high m means that GDA will have a large number of goals and therefore it may divide the limited number of K nodes that will be selected for prefetching into many paths and subsequently may result in a BFS-like behavior. On the other hand, with a low K there is a high probability of selecting a wrong virtual target and therefore prefetch nodes along a wrong direction. Finally, the

third control parameter experiment in Figure 9 (right at each plot—variations of the lookahead window w) show that GDA prefers small lookahead window w since this means that the m virtual targets will be closer to the current location of u at each iteration and therefore it would be much more easier for *GDA* to fix a wrong decision (i.e., selecting a popular destination that is far away from the actual path that the user follows.)

6.7 Experiment 6: Real Prototype Evaluation

In the last experiment, we use our real prototype system on a Samsung Galaxy S3 (Android 4) using an Exynos 4 Quad (GT-I9300). Particularly, we follow 20 random routes of 15m long each in the CS UCY campus and we measure the accuracy and energy consumption. The energy consumption is measured using PowerTutor, which, according to [7] is 86% accurate. We utilize the same parameter settings as in the previous experiments, i.e., dwell time $K = 15$, number of virtual targets $m = 3$, lookahead window $w = 3$, effective network threshold $\theta = -40$dBm and localization method = *WKNN*. All messaging goes through the 802.11 Wi-Fi interface. Extensive simulation over multiple smartphones was outside the scope of this work, even though we refer interested readers to our prior work SmartLab [13] to assess the complex dimensions arising in testing extensively smartphone applications on multiple real smartphone devices.

With respect to average accuracy, the results are comparable to our previous results and discussions, since the accuracy varies between 3m and 10m. The energy consumption of our real prototype can be considered reasonable since it consumes around $42.64J$ on average for 30 routes, which means around $2J$ for each 15m-long indoor navigation or 0.06% of a fully charged battery that is much less than the 1.09% ($\approx 35J$) needed to request and download the Wikipedia mobile site [24].

7 CONCLUSIONS AND FUTURE WORK

In this article, we study the problem of *prefetching the most important IoT data blocks from an IIN-SOA to a mobile device, without knowing its user's destination during navigation*. Our proposed framework, named *Grap (Graph Prefetching)*, structurally analyzes in an offline phase the building topology graphs to identify important areas inside building complexes (e.g., malls, hotels, campuses). The identified "hotspots" subsequently become virtual targets to an online heuristic graph search algorithm we developed, named *Graph-Distance A*-based* (GDA). We tested our Grap framework with real datasets from our production prototype IIN-SOA, which reached over 100,000 real users, and found Grap to be impressively accurate while retaining high performance levels (i.e., CPU time, network capacity and energy consumption). Our prototype implementation validates that our propositions are pragmatic and can easily make their way into future IIN-SOA.

In the future, we plan to investigate the trade-off between the CPU/network capacity and the accuracy objectives in the context of multi-objective optimization. We also plan to extend our experimental evaluation into domain-specific field studies (e.g., involving health/hospitals, education/universities) but also deal with more realistic evaluation scenarios that would require the adaptation of the physical infrastructure behind our experiments (e.g., varying number of APs, incorporating UWB transceivers or beacons). Finally, we will also release our developed artifacts as an open-source project.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. 2015. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Comm. Surv. Tutor.* 17, 4 (2015), 2347–2376.

[2] L. Atzori, A. Iera, and G. Morabito. 2010. The internet of things: A survey. *Comput. Netw.* 54, 15 (2010), 2787–2805.

[3] C. Becker and F. Dürr. 2005. On location models for ubiquitous computing. *Personal Ubiquitous Comput.* 9, 1 (2005), 20–31.

[4] C. Bouras, A. Konidaris, and D. Kostoulas. 2004. Predictive prefetching on the web and its potential impact in the wide area. *World Wide Web* 7, 2 (2004), 143–179.

[5] S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International Conference on World Wide Web 7 (WWW7)*. Elsevier Science Publishers B. V., 107–117.

[6] G. Chatzimiloudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti. 2012. Crowdsourcing with Smartphones. *IEEE Internet Comput.* 16, 5 (2012), 36–44.

[7] M. Dong and L. Zhong. 2011. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*. ACM, 335–348.

[8] L. Ghouti, T. Sheltami, and K. Alutaibi. 2013. Mobility prediction in mobile ad hoc networks using extreme learning machines. *Procedia Computer Science* 19 (2013), 305–312.

[9] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson. 2012. Informed mobile prefetching. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*. ACM, 155–168.

[10] I.-Y. Ko, H.-G. Ko, A.-J. Molina, and J.-H. Kwon. 2016. SoIoT: Toward A user-centric IoT-based service framework. *ACM Trans. Internet Technol.* 16, 2 (2016), 8:1–8:21.

[11] A. Konstantinidis, G. Chatzimilioudis, D. Zeinalipour-Yazti, P. Mpeis, N. Pelekis, and Y. Theodoridis. 2015. Privacy-preserving indoor localization on smartphones. *IEEE Trans. Knowl. Data Eng.* 27, 11 (2015), 3042–3055.

[12] A. Konstantinidis, G. Nikolaides, G. Chatzimilioudis, G. Evagorou, D. Zeinalipour-Yazti, and P. K. Chrysanthis. 2015. Radiomap prefetching for indoor navigation in intermittently connected Wi-Fi networks. In *Proceedings of the 16th IEEE International Conference on Mobile Data Management* 1 (2015), 34–43.

[13] G. Larkou, C. Costa, P. G. Andreou, A. Konstantinidis, and D. Zeinalipour-Yazti. 2013. Managing smartphone testbeds with smartlab. In *Proceedings of the 27th USENIX Conference on Large Installation System Administration (LISA'13)*. USENIX Association, 115–132.

[14] B. Li, J. Salter, A. G. Dempster, and C. Rizos. 2006. Indoor positioning techniques based on wireless lan. *1st International Conference on Wireless Broadband and Ultra Wideband Communications*, 13–16.

[15] S. Li, L.-D. Xu, and S. Zhao. 2015. The internet of things: A survey. *Inf. Syst. Front.* 17, 2 (2015), 243–259.

[16] T. M. Lim, C. K. Yeo, F. Lee, and Q. V. Le. 2009. Tmsp: Terminal mobility support protocol. *IEEE Trans. Mobile Comput.* 8, 6 (2009), 849–863.

[17] H. Lu, C. Guo, B. Yang, and C. S. Jensen. 2016. Finding frequently visited indoor POIs using symbolic indoor tracking data. In *Proceedings of the 19th International Conference on Extending Database Technology (EDBT'16)*. 449–460.

[18] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. 2002. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing (ICS'02)*. ACM, 84–95.

[19] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen. 2015. A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks (IPSN'15)*. ACM, 178–189.

[20] S. Papastavrou, G. Samaras, P. Evripidou, and P. K. Chrysanthis. 2006. A decade of dynamic web content: A structured survey on past and present practices and future trends. *IEEE Commun. Surv. Tutor.* 8, 2 (2006), 52–60.

[21] P. Prasithsangaree, P. Krishnamurthy, and P. Chrysanthis. 2002. On indoor position location with wireless LANs. In *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications* 2 (2002), 720–724.

[22] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, D. Steere, and C. Steere. 1990. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput.* 39 (1990), 447–459.

[23] E. Shriver, C. Small, and K. A. Smith. 1999. Why does file system prefetching work? In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC'99)*. USENIX Association, 6–6.

[24] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. 2012. Who killed my battery?: Analyzing mobile browser energy consumption. In *Proceedings of the 21st International Conference on World Wide Web (WWW'12)*. ACM, 41–50.

[25] O. Trullols-Cruces, M. Fiore, and J. Barcelo-Ordinas. 2012. Cooperative download in vehicular environments. *IEEE Trans. Mobile Comput.* 11, 4 (2012), 663–678.

[26] M. Vögler, J. Schleicher, C. Inzinger, and S. Dustdar. 2016. A scalable framework for provisioning large-scale IoT deployments. *ACM Trans. Internet Technol.* 16, 2 (2016).

[27] Y. Xia and C. K. Yeo. 2014. Mobile internet access over intermittent network connectivity. *J. Netw. Comput. Appl.* 40 (2014), 126–138.

[28] J. Xiao, Z. Zhou, Y. Yi, and L. M. Ni. 2016. A survey on wireless indoor localization from the device perspective. *ACM Comput. Surv.* 49, 2 (2016), 25:1–25:31.

[29] L.-D. Xu. 2011. Enterprise systems: State-of-the-art and future trends. *IEEE Trans. Indu. Inf.* 7, 4 (2011), 1551–3203.

[30] L. Yao, Q. Z. Sheng, and S. Dustdar. 2015. Web-based management of the internet of things. *IEEE Internet Comput.* 19, 4 (2015), 60–67.
[31] D. Zeinalipour-Yazti, and C. Laoudias. 2017. The anatomy of the anyplace indoor navigation service. In *ACM SIGSPATIAL Special*, Vol. 9, ACM Press, 3–10.
[32] D. Zeinalipour-Yazti, C. Laoudias, K. Georgiou, and G. Chatzimiloudis. 2017. Internet-based indoor navigation services. *IEEE Internet Comput.* 21, 4 (2017), 54–63.
[33] Z. Zhang. 2006. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *IEEE Commun. Surv. Tutor.* 8, 1 (2006), 24–37.
[34] Z. Zheng, P. Sinha, and S. Kumar. 2009. Alpha coverage: Bounding the interconnection gap for vehicular internet access. In *Proceedings of the IEEE INFOCOM*. 2831–2835.

Received May 2017; revised November 2017; accepted December 2017