Future Generation Computer Systems 118 (2021) 14–36

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

It's about Thyme: On the design and implementation of a time-aware reactive storage system for pervasive edge computing environments

João A. Silva *, Filipe Cerqueira, Hervé Paulino, João M. Lourenço, João Leitão, Nuno Preguiça

NOVA Laboratory for Computer Science and Informatics, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa, 2829-516 Caparica, Portugal

ARTICLE INFO

Article history:
Received 20 December 2019
Received in revised form 23 October 2020
Accepted 8 December 2020
Available online 15 December 2020

Keywords:
Distributed storage
Publish/subscribe
Wireless networks
Mobile devices
Edge computing

ABSTRACT

Nowadays, smart mobile devices generate huge amounts of data in all sorts of gatherings. Much of that data has localized and ephemeral interest, but can be of great use if shared among co-located devices. However, mobile devices often experience poor connectivity, leading to availability issues if application storage and logic are fully delegated to a remote cloud infrastructure. In turn, the edge computing paradigm pushes computations and storage beyond the data center, closer to end-user devices where data is generated and consumed, enabling the execution of certain components of edge-enabled systems directly and cooperatively on edge devices. In this article, we address the challenge of supporting reliable and efficient data storage and dissemination among co-located wireless mobile devices without resorting to centralized services or network infrastructures. We propose Thyme, a novel time-aware reactive data storage system for pervasive edge computing environments, that exploits synergies between the storage substrate and the publish/subscribe paradigm. We present the design of Thyme and elaborate a three-fold evaluation, through an analytical study, and both simulation and real world experimentations, characterizing the scenarios best suited for its use. The evaluation shows that Thyme allows the notification and retrieval of relevant data with low overhead and latency, and also with low energy consumption, proving to be a practical solution in a variety of situations.

© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

We are witnessing a rapid growth of both the capabilities and amount of mobile devices worldwide [1,2]. As such, there is a wide adoption of smartphones and tablets for performing the most diverse activities, from leisure to work-related tasks. Hence, the volume of data generated by these devices, like user-generated content and sensor data, is also growing rapidly [2,3].

Much of the data generated by mobile devices in all sorts of social gatherings (like sports events, protests, festivals, or ceremonies) has *localized* and *ephemeral* interest. People in such events are usually interested in similar types of information (e.g., statistics and videos at sports events), and such interest typically diminishes over time. Thus, swift and spontaneous data storage and dissemination among neighboring mobile devices can be of great usefulness. For instance, smartphones carried by people in such gatherings can collect lots of useful data that, when shared among co-located devices, may help others discover new points of interest, enjoy videos of special moments (from multiple viewpoints), or avoid waiting lines or crowded areas in a venue.

In many situations, making information available may be of paramount importance (e.g., disaster situations [4], military scenarios [5]), or just really helpful (e.g., crowded events [6]). So, being dependent on network infrastructure access to support such use cases may be unwise, or even unfeasible, due to their potential overload or destruction. Even assuming the availability of infrastructure, transferring large amounts of data to and from the cloud can lead to network congestion, various delays, and possible monetary costs. Furthermore, in those scenarios, mobile devices often experience poor or intermittent connectivity, leading to *availability* issues if application storage and logic are fully delegated to a remote cloud infrastructure. Still, the non-negligible costs associated with network infrastructure setup (e.g., adding access points) further motivates the need to have devices interact in a device-to-device (D2D) manner, through an infrastructure-less or ad-hoc network [7]. Thus, the main question we address in this article is: *how to support reliable*

* Corresponding author.
E-mail addresses: jaa.silva@campus.fct.unl.pt (J.A. Silva), fa.cerqueira@campus.fct.unl.pt (F. Cerqueira), herve.paulino@fct.unl.pt (H. Paulino), joao.lourenco@fct.unl.pt (J.M. Lourenço), jc.leitao@fct.unl.pt (J. Leitão), nuno.preguica@fct.unl.pt (N. Preguiça).

https://doi.org/10.1016/j.future.2020.12.008
0167-739X/© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

and efficient data storage and dissemination among co-located mobile devices without resorting to centralized services and subsisting with no network infrastructure?

The extensive proliferation of mobile devices at the edge of the network, along with the increasing growth of their capabilities, offers a massive computing and storage infrastructure of (still mostly) untapped resources. Together, the ubiquitous smart mobile devices, the opportunistic gathering of users, and the growing pervasiveness of edge computing environments [8,9], have enabled novel opportunities for data storage and dissemination at the *network edge*. In fact, it is more efficient to communicate and distribute information among *nearby* devices than to use distant centralized intermediaries [10,11]. By storing data near its source (e.g., where it is generated), applications can be more responsive while *relieving* some of the load from cloud and network infrastructures, potentially also providing increased data privacy and ownership.

Allowing systems' components to actively and directly collaborate at the edge requires some form of distributed data repository as to share and disseminate information. Thus, we propose Thyme, a novel *time-aware reactive storage* system for networks of mobile devices, that exploits *synergies* between the storage substrate and the P/S communication paradigm. It fuses the storage interface with a P/S abstraction, enabling co-located mobile devices to store and disseminate data among them. Contrary to previous solutions, queries are in the form of *subscriptions* that have a specific *time scope* defining when they are active (and can even include the past). Leveraging this novel time-aware abstraction, Thyme is able to achieve robust, efficient and timely data storage, dissemination, and querying. It also allows both the notification and retrieval of desired data with low overhead and latency, using limited bandwidth and while being resilient to possible message losses and node failures.

In typical storage systems [12–14], users are required to *actively* and *explicitly* search for the desired data, following a *request/reply* interaction model. Since the kind of distributed environments we target are highly volatile and dynamic, we adopt a *reactive* and *loosely coupled* data dissemination mechanism [15]. By integrating a P/S abstraction, users (or applications) can register their interests, being subsequently notified of any data items matching those interests. This allows users to quickly discover what data exist in the system in a reactive manner, and *only* be notified about data they are interested in.

In the kind of gatherings we are addressing, individual moments are intrinsically tied by time relations (e.g., the band performing at time x in the music festival, or the second speech on a rally). Also, people are often interested in information with these associated time references (e.g., find photos of the opening band). Hence, Thyme considers *time* to be a first order dimension. Subscriptions include a time frame that defines their *active timespan*, either in the future, in the present, or in the past, effectively providing the full *time decoupling* of the P/S paradigm [15].

We present two different approaches to Thyme. The first one, Thyme-LS, follows a lightweight, yet effective, unstructured approach using local storage and query flooding. The second more intricate one, Thyme-DCS, is inspired by the fact that geographical positions have a close relation to topology in wireless networks, and follows a data-centric storage (DCS) approach [16], whereby we build a storage substrate over a geographic hash table (GHT) [17]. We implement both approaches in the ns-3 network simulator [18]. Moreover, we also address the application of Thyme to networks of real mobile devices, implementing Thyme-DCS as a library for Android devices, and developing a proof-of-concept photo sharing application on top.

Although previous systems present in the literature offer some features similar to Thyme (e.g., tuple spaces [19,20] or peer-to-peer (P2P) systems [12,14]), none provides the same overall characteristics (as we detail in Section 2). Thus, to the best of our knowledge, Thyme is the first system to provide reliable reactive storage for pervasive edge computing environments that may be effectively and efficiently used in either small, medium and large scale scenarios.

This article builds on the work presented in [21] and extends it. Here, we present an in-depth definition of the time-aware reactive storage abstraction (Section 3); and also a more comprehensive description of the design of our time-aware reactive storage system, Thyme (Section 4), and of its two different approaches, Thyme-LS (Section 5) and Thyme-DCS (Section 6). Furthermore, we leverage the work in [22] to elaborate an extensive evaluation of our proposed solution. We further describe an implementation of the Thyme-DCS approach, as a Java library, for sharing and storing data in networks of Android mobile devices (Section 7); and also the design of a photo sharing Android application atop Thyme that enables users to share and persist photos in multiple photo galleries sustained by a network of mobile devices (Section 7.4). Lastly, we report a characterization of the scenarios best suited for the use of the proposed solutions, through a three-fold evaluation: via an analytical study (Section 8), and both simulation (Section 9) and real world experimentations (Section 10). We close the article with a broad discussion of our main findings (Section 11) and our conclusions (Section 12).

In summary, the original contributions of this article are the following:

1. A detailed definition of our time-aware reactive storage model;
2. An analytical study where we derive approximate formulas for communication costs and operations complexity, and use them to compare our different approaches;
3. An extended evaluation of our approaches through simulation experimentations; and
4. An in-depth study of the energy costs in the Thyme-DCS approach through real world experimentations with Android devices.

2. Related work

Concerning work related to our proposal, we address and compare against three main categories: P/S systems, data storage and dissemination systems in general, and the particular case of tuple spaces.

2.1. Publish/subscribe

Typical P/S systems are stateless, meaning that only subscribers online at the time of publication are notified. Hence, the notion of publication persistence has not been addressed in most systems. Some approaches for wired settings exploit the concept of a persistent data repository, by means of distributed buffers [23] (allowing only to specify how many data items to request from the past when subscribing) or by integration with traditional databases [24] (without any notion of time). Apache Kafka [25], a system originated at LinkedIn, has recently gained significant popularity, clearly demonstrating the feasibility and potential of the P/S communication paradigm. However, such solutions do not consider time as a first order dimension of the P/S abstraction. Furthermore, solutions for wired scenarios cannot be easily adapted for wireless setting where connectivity is not stable.

Table 1 highlights the main aspects when comparing our proposal with other P/S systems. In the particular context of wireless settings, Chapar [27] is, as far as we know, the only persistent P/S

Table 1
Comparison with P/S systems.

	Environment	Time Assignment	Substrate
[23]	Wired	Subscriptions	REBECA [26]
[24]	Wired	–	DB
Chapar [27]	Wireless	Publications	OLSR [28]
Thyme [21]	Wireless	Pub. & Sub.	GHT/Flooding

Table 2
Comparison with data storage and dissemination systems.

	Infrastructure	Data Avail.	Substrate
Krowd [12]	Yes	No	1-hop DHT
Ephesus [29]	Yes	Yes	DHT
MobiTribe [30]	Yes	Yes	Central Server
PAN [13]	No	Yes	Prob. Quorums
Phoenix [31]	No	Yes	Simple Quorums
iTrust [32]	No	No	Random Walks
PDS [14]	No	±	ICN [33]
Thyme [21]	No	Yes	GHT/Flooding

system. However, it only assigns time to publications, which are buffered only until their lifetime expires (as a time-to-live). In Thyme, subscriptions have their time scope assigned. While publications are permanently stored, they may be deleted from the system upon request. Thus, new subscribers can always request previously published data. Moreover, Chapar is not functionally symmetric, demanding more work from broker nodes, and thus achieving poor load balancing, an aspect that has been explicitly considered in the design of Thyme.

2.2. Data storage and dissemination

Table 2 highlights the main aspects when comparing our proposal with other general data storage and dissemination systems. Krowd [12] and Ephesus [29] enable content sharing and storage among nearby mobile devices. While Krowd relies on a one-hop distributed hash table (DHT), requiring each device to know every other device in the network, Ephesus is sustained by a classical DHT. As an handicap, they both require some kind of network infrastructure for inter-device communication (e.g., an access point). Of the two, Ephesus is the only to address device mobility or failure, and data availability, via replication. In turn, Thyme supports several wireless technologies, and targets multi-hop environments using a GHT, known for being more suitable in wireless networks. It also employs several replication mechanisms to address mobility and data availability.

MobiTribe [30] is a system for content sharing on mobile devices, across the Internet. It uses a central server for content discovery, peer registration and metadata management. Data is replicated in several peers according to their interests, and it uses prefetching techniques to improve retrieval latency.

PAN [13] and Phoenix [31] are two systems for reliable storage in mobile ad-hoc networks (MANETs). PAN is an asymmetric system based on probabilistic quorums, while Phoenix uses a round-based simple quorum protocol for one-hop networks only. iTrust [32] and PDS [14] focus on data discovery and retrieval on co-located devices. iTrust is based on random walk techniques, while PDS is inspired in information-centric networking. PDS's aggressive caching policy can lead to serious storage overheads, and since data is only cached if requested, less popular data may even disappear. iTrust does not address data availability issues.

All these systems employ the *request/reply* interaction model, whereby peers have to proactively search for content. In turn, Thyme explores synergies between the P/S paradigm and the storage substrate, to provide both persistent publications and a reactive interaction model, thus allowing applications to react to new data being generated and stored. At the same time, it presents the potential to decrease network traffic, as users do not need to be constantly searching for the desired data.

Other approaches based on opportunistic and delay-tolerant networking [34–36] provide communication and content sharing in the presence of intermittent connectivity. They take advantage of opportunistic contacts between peers to allow the exchange and spread of information. This means content dissemination is best effort, i.e., information spreading depends on the availability and willingness of interested peers to carry such content. Some of these systems also provide a reactive interaction model for data retrieval. They are, however, devised for extreme environments that relax temporal restrictions to the order of hours or days, something not feasible for the kind of use cases we target.

Table 3
Comparison with tuple spaces systems.

	Purpose	Data sharing
TuCSoN [38]	Internet	Constant
LIME [19]	Federated TS	Transient
TOTA [20]	Autonomous Propagation	Rule-based
Thyme [21]	Wireless	Constant

2.3. Tuple spaces

The Linda model [37] (or tuple spaces) is an interaction paradigm for parallel computing. It provides a shared data space abstraction, i.e., a shared repository of immutable structured information, called tuples. It provides three simple operations: `in`, read and remove a tuple from the tuple space; `rd`, (non-destructively) read a tuple from the tuple space; and `out`, write a tuple into the tuple space. Systems like TuCSoN [38], LIME [19], and TOTA [20] adapted the tuple spaces model for mobile and wireless environments. Besides the model's proactive operations (for inserting, reading, and removing tuples), these systems allow actions to be performed as *reactions* to certain events. Table 3 highlights the main aspects when comparing our proposal with other tuple spaces systems.

Although reactions are similar to Thyme subscriptions, they have significant differences. First, reactions always execute on the client side, i.e., on the host that installed it, and always receive the tuple that triggered the reaction. This does not allow load balancing when executing the reactions and when matching reactions with tuples. It also has the potential to generate more traffic than actually required, because it is not possible to filter data at the source. In Thyme, subscription matching is executed by randomly selected peers that may change in each matching, thus improving load balancing and optimizing the data to be delivered to each client.

Another major difference is that tuple spaces do not differentiate between data and metadata management, i.e., everything is represented as a tuple. In Thyme, when receiving a notification, nodes only receive an object's metadata (containing a small amount of information), and only after that decide if the object is interesting enough and proceed to retrieve it. Since metadata is usually much smaller than the actual data, this strategy can considerably reduce network traffic. Also, when managing replication and mobility, metadata may require updates. Since tuples are immutable, the only way of modifying metadata is to remove and insert a new (changed) tuple, which may trigger unwanted reactions. This can be bypassed by making an intricate decomposition of the metadata into several tuples. Although this may work in small scale scenarios, it can quickly become cumbersome, and penalize performance in large scale scenarios, as targeted by Thyme.

TuCSoN [38] was designed for mobility in Internet environments and presents the notion of programmable tuple spaces (spread over Internet nodes). Tuple spaces are enhanced in that their behavior in response to agent's operations can be extended so as to embody application-specific computations. These tuples spaces are rather complex and cumbersome to reason with. Furthermore, it is not easily adaptable to dynamic wireless environments (e.g., it assumes reliable communication), and its main focus in on the programmability of the (coordination) tuple space.

LIME [19] breaks up the notion of a global tuple space, and distributes its content across multiple mobile components. When components are within range (i.e., mobile agents are on the same host or communication is available between mobile hosts), the contents of the tuple spaces held by the individual mobile components are transiently shared, forming a federated tuple space. The contents of these virtual tuple spaces evolve in time according to the current connectivity pattern. Although reactions enable tuple spaces to react to the insertion of relevant tuples, they are sensitive to hosts' connectivity, since the federated tuple space only takes into account data spaces from components within range. In several mobile and wireless environments, this approach is not sufficient to generally support distributed services or applications. Therefore, LIME was devised for small scale scenarios. In turn, Thyme leverages a lightweight flooding approach or a GHT for ensuring the best possible connectivity in large scale scenarios, and its routing schemes jointly with its replication mechanisms allow the matching of publications against subscriptions of all peers in the network.

In TOTA [20], tuples are injected and can autonomously propagate into the network according to specific rules (i.e., they are not assigned to specific tuple spaces). Each tuple is formed by: a content, the tuple data; a propagation rule, the policy by which the tuple has to be cloned and diffused across the network, and how the tuple content should change during propagation; and a maintenance rule, the policy whereby the tuple content should evolve/change due to events or time elapsing. Propagation consists in a tuple cloning itself, being stored in the local tuple space, and moving to neighbor nodes recursively. Tuples are not necessarily distributed replicas. According to their propagation and maintenance rules, they can assume different values in different nodes (expressing some kind of contextual or spatial information). In the end, unlike traditional event-based models, tuples propagation is not driven by a P/S schema, but is encoded in the tuples' propagation rule. By constantly monitoring the network local topology and the insertion of new tuples, TOTA can automatically re-propagate or modify the content of tuples as necessary conditions occur. Subscriptions only react to changes in a node's local tuple space (or from its one-hop neighbors). To achieve something similar to Thyme, data should be propagated to every network node in order for subscriptions to be matched against the data. Otherwise, some nodes would not be notified about relevant data. TOTA also requires every node to execute the matching of subscriptions against tuples, thus suffering from redundant work and poor load balancing. In contrast, Thyme allows for better load balancing, distributing the load when matching subscriptions and publications.

2.4. Others

Regarding *app stores*, there are several applications for sharing files between mobile devices, e.g., SuperBeam [39] and Xender [40] allow synchronous one-to-many data exchange. However, data is only available while its owner is online. There are still other applications [41,42] and specialized devices [43] that provide ad-hoc (multi-hop) communication among mobile devices, allowing data dissemination when network infrastructures are inaccessible.

3. Time-aware reactive storage

Typical storage systems provide a request/reply *proactive* interaction model for data retrieval, making it difficult to be aware of the available data, and requiring users to explicitly search for it. Also, in most P/S systems, publications are *transient*, i.e., once matched and disseminated, they are not further stored or processed. Thus, only subscribers online at the time of publication are notified. To overcome such shortcomings, we build strong synergies between the storage substrate and the P/S paradigm. On the one hand, the storage substrate leverages the P/S abstraction to provide a *reactive* interaction model whereby users register their interests through subscriptions and are notified as new relevant data is generated, not requiring them to be constantly searching for new data. On the other hand, the P/S abstraction takes advantage of the storage substrate to provide *persistent* publications, enabling the *time-awareness* concept and providing full time decoupling [15].

Our storage interface provides the usual data store operations: insert, retrieve, and delete. Additionally, due to its integration with the P/S abstraction, it also offers the regular P/S operations: publish, subscribe, and unsubscribe. All operations are asynchronous, receiving results through callbacks.

3.1. Inserting data

Due to the integration with a P/S abstraction, the insert operation (of the storage substrate) is *fused* with the publish operation (of the P/S system). As a result, the insertion of a data object into storage may trigger the sending of notifications to subscribers.

A *data object* is the basic unit of work and is seen as an opaque set of bytes. Every object has some associated metadata that consists in a tuple

$$\langle oid, T, s, ts^{pub}, nid \rangle$$

where:

- oid is the object identifier;
- T is a set of tags or keywords related with the object, e.g., hashtags used in social networks;
- s is a summary or a small description of the object, e.g., a thumbnail of an image or a video;
- ts^{pub} is the object insertion/publication timestamp; and
- nid is the owner's node identifier.

To avoid name collisions (among different nodes), the system-wide unique *object key* is the pair $\langle oid, nid \rangle$, composed of both the object and the owner's node identifiers.

Tags are used as topics for subscriptions, thus enabling a *topic-based* P/S system. Although topic-based addressing [44] is not as expressive as content-based systems [45], it requires far less filtering and computations, which fits our target environments populated by battery-constrained mobile devices. Nonetheless, this tagging feature provides a flexible annotation scheme, e.g., by adding the owner's node identifier to the tags of its own objects, an application can easily enable the retrieval of all the objects stored by a certain node/user.

3.2. Deleting data

To support subscriptions with a past time frame, insertions must be persistently stored within the system. Accordingly, this model also supplies an operation to remove data from storage. The delete operation removes an object from storage, making it inaccessible to future subscriptions. Note that subscriptions targeting the past will not see deleted objects, even if these were initially available in the subscription's time frame.

Taking into account a simple access control mechanism, only the owner of a data object can delete it (i.e., a node can only delete objects it inserted).

3.3. Querying data

Since we make use of the P/S abstraction, querying data means *subscribing* to the desired tags. As a response, notifications will be received for data objects matching the issued subscriptions.

With time as a first order dimension, a subscription consists in a tuple

$$\langle sid, q, ts^s, ts^e, nid \rangle$$

where:

- *sid* is the subscription identifier;
- *q* denotes the query that defines which tags are relevant;
- ts^s and ts^e are the timestamps defining when the subscription's time frame starts and expires, respectively; and
- *nid* is the subscriber's node identifier.

Unlike typical topic-based P/S systems, that only allow one topic per subscription, we support *arbitrary* propositional logic formulas where literals are tags associated with objects (e.g., '$A \& (B \mid C)$' captures objects tagged with A and at least one of B or C).

The ts^s and ts^e timestamps specify the subscription's time frame in which the subscription is active, where the special value $\perp$ represents, respectively, the times at which the system started and stopped to exist. Assuming a subscription is issued at time t: $ts^s = \perp \wedge ts^e = t$ matches events that happened before the subscription (this allows a typical search or find operation on the data store); $ts^s = t \wedge ts^e = \perp$ matches events after or concurrent with the subscription; and $ts^s = ts^e = \perp$ matches all the past and future events in the system. Notice that these parameters can also take any concrete timestamp value.

Due to the unreliable nature of our target (wireless) environments, subscribers are notified of all relevant data in a *best effort* manner. After a subscription, notifications may be triggered in two situations:

- upon an insertion, by detecting that the object being stored matches existing subscriptions; and
- upon issuing a subscription that spans into the past, by detecting that this new subscription matches previously stored objects.

Note that, to minimize the information passing through the network, notifications are sent to the respective subscribers carrying *only* the metadata of the matching objects (and not the entire data objects).

The unsubscribe operation revokes a subscription before it naturally expires after its end timestamp, ts^e.

When issuing a subscription for a popular tag, that spans into the past, the subscriber might get flooded by a large amount of notifications (i.e., an excess of past notifications), which implies lots of communication. To attenuate this problem, when subscribing for a time frame in the past, a subscriber is only notified about the n most recent objects from a total of x matching objects, with $n \leq x$. Then, if interested, a subscriber can request more of those objects, receiving the notifications in *explicitly requested batches* (similar to the concept of pagination). All subsequent matching objects will be notified as usual.

3.4. Retrieving data

Through subscriptions, users are notified *only* about data they are interested in, allowing them to discover what data exist in the system in a reactive manner. Even so, a typical search/find operation can still be done by subscribing to the desired query with timestamps $ts^s = \perp$ and $ts^e = now$ (Section 3.3).

Due to our reactive model, objects can *only* be retrieved as a response to notifications (using the received object metadata), thus revealing a relation between the subscribe and retrieve operations. So, object metadata is the only information given to subscribers for them to decide if objects are relevant enough for retrieval. Received notifications must be acted upon, and may either be discarded, trigger an immediate retrieve operation, or be stored by the application for later processing.

4. System overview

The design of a time-aware reactive storage system for pervasive edge computing environments presents a set of interesting challenges. For example, where to place data and how to find it? What are the proper trade-offs between communication and reliability? How and what data to disseminate (and when)? And in the end, how to integrate the two interfaces – storage and P/S – without losing their principal characteristics, and, at the same time, making the resulting interface easy for developers to grasp and use? THYME's design, that we present next, considers these and other issues.

4.1. Use cases

THYME can be used to build generic data dissemination services for the kind of environments we are targeting. We argue that THYME fits perfectly in scenarios where big crowds are gathered, using their mobile devices to collect data (e.g., photos, video, text) and share that same data with people in their vicinity, akin to social networks [46].

Consider, for instance, a scenario where spectators in different parts of a football stadium may share their views of the game through self-generated multimedia content. In this case, spectators would be able to see key moments of the game from multiple viewpoints, including those of the spectators in key locations or closer to the field. Offering such possibility can significantly improve user experience – something football teams are willing to invest in, if it means they will attract more fans to their stadiums [47].

In fact, this kind of augmented user experience is already being explored by some companies [47], using the venues' existing fixed communication infrastructures, which are single points of failure that may be subjected to overload conditions and other failures (e.g., power outages [6]). In turn, the pervasiveness of mobile devices and the advances in the edge computing paradigm offer the possibility to provide such enriched user experience with negligible cost for infrastructure managers, while at the same time, working to alleviate the load on those infrastructures.

4.2. System model

We consider a classical asynchronous model comprised of $\Pi = \{n_1, \ldots, n_k\}$ mobile devices (hereafter called nodes) with no mobility restrictions, other than those imposed by the venue they are in and the natural speed limits of humans[1]. Our algorithms do not assume any radio technology or routing infrastructure, being practical in several wireless settings. Nodes communicate by exchanging messages through a wireless medium (e.g., Bluetooth, Wi-Fi ad-hoc, Wi-Fi Direct [48], Wi-Fi Aware), and have no access

[1] The record for top speed achieved by a human is 12.4 m/s, by Usain Bolt, seen during the 100 meters final of the 2009 World Championships in Athletics, in Berlin.

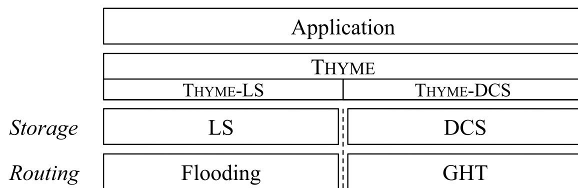


Fig. 1. System overview.

to any form of shared memory. Nonetheless, nodes should be able to establish (point-to-point) communication channels with (all) their one-hop neighbors, and thus need to have some kind of discovery mechanism in order to determine their neighbors. We also consider the classical crash-stop failure model, whereby nodes can fail by crashing but do not behave maliciously.

Data objects are considered immutable. Also, we do not consider security or access control concerns, thus only publicly shareable data is manipulated (e.g., as in social networks). Due to the unreliable nature of wireless communication mediums, Thyme notifies subscribers of all relevant data as completely and faithfully as possible, i.e., missing some notifications is permitted because applications are not expected to be mission-critical.

Each node has a globally unique identifier and can determine its geographical position, through GPS or other means [49]. Thus, nodes can be aware if they are moving or not. We also assume nodes' clocks to be synchronized (with a negligible skew). Both these assumptions are reasonable since we target mobile devices (e.g., smartphones) and nowadays even low-end devices come equipped with GPS and synchronize their clocks with the network providers, while other solutions allow device location even indoors [49].

4.3. Architecture

Since we target decentralized networks based on battery-constrained devices, load balance is a concern. As such, in Thyme, akin to (flat) P2P systems, nodes are functionally symmetric and share the same responsibilities, i.e., there are no centralized or specialized components (like P2P super-peers or P/S brokers), and each node can be a publisher, a subscriber, or both.

Thyme's design comprises three main layers, depicted in Fig. 1. The bottom layer handles message routing. The middle layer is the storage substrate. The top layer is Thyme itself, providing its interface for applications.

As illustrated in Fig. 1, we propose two different approaches for the two bottom layers (routing and storage). Thyme-LS (see Section 5) uses nodes' local storage, and query flooding, thus data objects are stored locally by their owners, while subscriptions are fully replicated in every node of the system. Its routing layer provides flooding to the entire network (using UDP broadcast), and (multi-hop) unicast using a typical ad-hoc routing protocol (e.g., DSDV [50], OLSR [28]).

In turn, Thyme-DCS (see Section 6) follows a DCS approach [16], using a simple key–value storage substrate that we built over a cell-based GHT for wireless networks [17]. Its routing layer is materialized by this GHT, called cell hash routing (CHR). As represented in Fig. 2, the physical/geographical space where the system is to be available is divided in a grid, i.e., into equally-sized square-shaped cells, and all physical nodes within a cell collaboratively act as a virtual node. Messages are addressed to geographic locations, thus routed to the cell that contains the message destination. Messages addressed to a cell are delivered to all physical nodes within the cell (similar to [51]). For instance, in Fig. 2, a message addressed to a location inside the boundaries

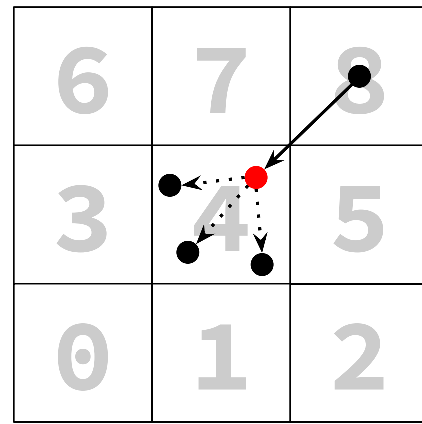


Fig. 2. Geographic hash table and its virtual nodes.

of cell 4 is received by the red node (chosen randomly by the routing protocol of the GHT; see Section 6.6.1), and then is forwarded to all the other neighbor nodes inside the cell.

The use of the GHT is two-fold: (1) cells are used to store all system data (data objects, its metadata, and subscriptions); and (2) cells are exploited to match subscriptions and objects, i.e., cells act as virtual P/S brokers.

Wireless communication mediums are known to be subject to many forms of interference, hence messages may be lost and not reach their final destination. However, as a design principle, this layer does not provide any mechanisms to recover from lost messages on the wireless medium, delegating this responsibility to the upper layers (abiding by the end-to-end argument [52]).

5. An unstructured approach: Thyme-LS

Thyme-LS employs a lightweight unstructured approach and has no extra maintenance overhead. It uses nodes' local storage, and query (in our case, subscription) flooding. Both insert and delete operations are entirely executed locally. Thus, objects are only stored by their owners. On the other hand, Thyme-LS uses *subscription flooding* as its event routing strategy (like Gryphon [45,53] and SIENA [54]). Hence, subscribe and unsubscribe operations are flooded and executed in every node, so subscriptions are fully replicated across all the system. These operations are broadcasted to all its one-hop neighbors who, then, forward the message to all their one-hop neighbors and so forth. Nodes keep track of received messages so that the ones already forwarded will not be sent again. Since every node has the complete set of all the system-wide subscriptions, the matching between objects and subscriptions is completely local.

In this approach, notifications may be triggered in two occasions:

- upon an insert operation, if that new object matches any of the node's locally stored subscriptions; and
- upon issuing a subscription (when flooding the respective message), each node that receives a subscription checks if it matches any of its locally stored objects.

Retrieve operations request the desired objects directly from their owners, using the information received in the notifications, and the multi-hop unicast communication primitive provided by the routing layer (Section 4.3).

Here, node mobility is handled in a completely transparent way by the underlying protocol used in the routing layer. Nodes can move freely, and the routing protocol takes care of all the necessary changes that come from that movement in order to

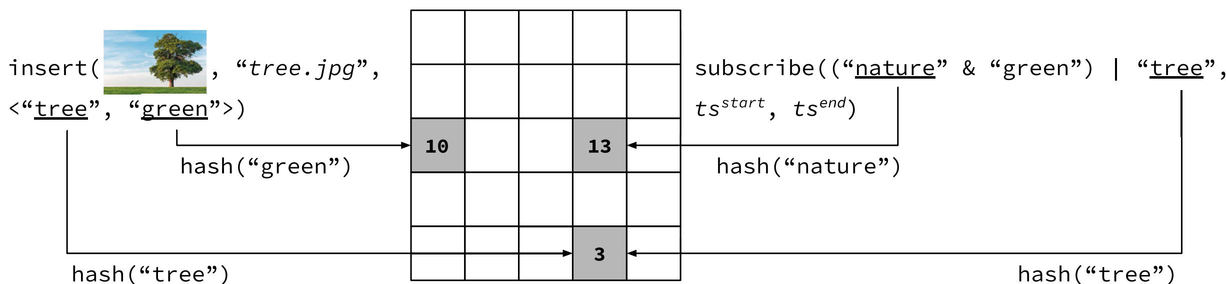


Fig. 3. Insert and subscribe operations in Thyme-DCS. The tags' hashing determines the cells responsible for managing the object metadata (cells 3 and 10) and the subscription (cells 3 and 13). If a subscription has matching tags with an object, it will also have overlapping (responsible) cells, guaranteeing the matching and sending of notifications to the subscriber.

continue to provide connectivity. Also, since objects are only stored locally by their owners, Thyme-LS does not guarantee objects' persistence once their owners fail or leave the system.

When joining the system, nodes broadcast a join request. To avoid a flooding of replies, only a few neighbors (randomly selected through a coin toss) respond back with their locally stored subscriptions. Also, to minimize collisions, replies are delayed a (configurable) random amount of time. If no replies are received after a maximum number of retries, the joining node assumes it is alone, and starts operating as normal.

6. A structured approach: Thyme-DCS

This approach leverages heavily on the notion of cell (or virtual node) conveyed by its routing layer (Section 4.3). By using geographical information, Thyme-DCS has two complementary aspects: (1) it provides topology-awareness by design; and (2) allows the inference of the location of relevant data to subscriptions, enabling access to such data using a location-aware strategy.

6.1. Inserting data

When executing an insert operation, this approach leverages on the cells conveyed by the underlying GHT. Object data and metadata are managed differently. The latter is indexed (and, thus, replicated) in all the cells resultant from hashing the object tags. The actual object is replicated in all the nodes of the owner's cell (see Section 6.2). This ensures only a small amount of data (i.e., the metadata) is sent through the network, whereas the bulk of the data is kept near its source.

Fig. 3 illustrates an insert operation of a photo with identifier "tree.jpg", and tags "tree" and "green". The cells resultant from hashing each tag are responsible for managing the object's metadata and checking if subscriptions match this object.

6.2. Replication

Since we target dynamic and pervasive edge computing environments, in order to provide data availability and tolerance to churn, this approach employs two replication mechanisms.

6.2.1. Active replication

Active replication takes advantage of the virtual nodes provided by the cell-based GHT. Upon an insertion, an object is disseminated inside the owner's cell. Onward, every node inside the cell should be able to reply to retrieve operations for that object. This ensures tolerance to churn and guarantees that stored content will remain in the system even if their owners' leave. Note that the objects' metadata is also (actively) replicated in the cells resultant from hashing the objects' tags (Section 6.1).

6.2.2. Passive replication

In turn, passive replication leverages on the nodes that already retrieved an object to provide more replicas scattered in the network, increasing data availability. At the same time, it offers a list of multiple locations from where the object may be retrieved.

6.2.3. Replication list

To enable both mechanisms, the system needs to keep track of the whereabouts of each object replica. This is achieved by listing an object's replica locations in its metadata, in what we call *replication lists*, L_{rep} (a list of pairs with node identifier and cell address—$\langle nid, cid \rangle$). Thus, in this case, the object metadata consists of a tuple

$$\langle oid, T, s, ts^{\text{pub}}, nid, \boldsymbol{L_{\text{rep}}} \rangle$$

The replication lists are bound to a (configurable) maximum number of locations, maintaining only the most recent entries. Also, a list only contains one entry for an object's active replica, representing all the nodes inside that cell (and this is a permanent entry on the list despite its recency).

Since nodes can *move*, their location may change over time. Hence, after a node stabilizes in a (new) cell, it must update its location for the passive replicas of the objects it holds (through location update messages).

6.3. Deleting data

In the delete operation, the object metadata indexed by the object tags is removed from the responsible cells. However, while active replicas are also explicitly removed, the same does not happen to passive ones. Nonetheless, since the metadata is removed (and with it, so is the replication list) they become inaccessible and thus stop working as (passive) replicas.

6.4. Querying data

Since the GHT used by Thyme-DCS only routes messages to geographical positions, there is the need to know where to send notifications, i.e., the node's address is not enough. Thus, subscriptions are extended with the location (i.e., cell address) of the subscriber node, *cid*. This information needs to be updated every time the subscriber node changes its cell. In the end, a subscription in Thyme-DCS consists of the tuple

$$\langle sid, q, ts^{\text{s}}, ts^{\text{e}}, nid, \boldsymbol{cid} \rangle$$

6.4.1. Divide and conquer

Leveraging on the fact that every propositional logic formula has an equivalent in disjunctive normal form (DNF), we employ a divide and conquer strategy of breaking the disjunction into its individual conjunctive clauses, and evaluate each one separately. For a match to occur, it suffices that one evaluates to true.

The use of DNF enables load balancing when matching objects against subscriptions, since the work can be split among different cells/nodes, each evaluating only one of the query's conjunctions. Additionally, it minimizes the amount of information transmitted to the responsible cells (by sending only a subset of the query, i.e., the relevant conjunction).

For each conjunction, we randomly select as its key one of its *positive* literals (what we call *conjunction keys*). Hashing that literal determines the cell where to send that part of the query. That cell becomes a (virtual) broker for the subscription, and the nodes in the cell are responsible for checking if objects match the subscription, and notifying the subscribers, if need be. Thus, this approach employs the *rendezvous-based event routing* approach (like Scribe [55] and Hermes [56]). Fig. 3 depicts a subscription of a query with two conjunctions. For each, one of its literals is chosen as its key, and determine which cells will become the virtual brokers for the subscription (in this case the two conjunction keys are "nature" and "tree").

For instance, assume the query

$$(A \& B \& E) \mid (A \& \neg C) \mid (D)$$

already in DNF. The disjunction is divided into its three conjunctions: (1) $A \& B \& E$; (2) $A \& \neg C$; and (3) D. Due to the restrictions already mentioned, conjunctions 2 and 3 have their keys automatically determined (literals A and D, respectively). But, any literal in conjunction 1 may be chosen to be its key.

Even that, in some cases, the conversion to DNF can lead to an exponential explosion of the formula [57], we argue that most ordinary users do not make use of complex queries nor logic operators regularly. Thus, we do not expect this to be an issue in practice.

6.4.2. Notifications

After a subscription, there are two occasions that may trigger notifications:

- upon an insertion, cells indexing the object metadata by its tags are responsible for checking if the new object matches any existing subscriptions stored locally; and
- upon a subscription, cells indexing the subscription by its conjunction keys are responsible for checking if the locally stored metadata match that new subscription.

When we break the subscription query into its multiple conjunctions, it suffices that one of the conjunctions evaluates to true, for a match to occur. But, since the conjunctions are evaluated by (probably) different cells, when different conjunctions of a same subscription both evaluate to true, the subscriber will receive duplicate notifications for the same matched object. Let us assume the query given before, $(A \& B \& E) \mid (A \& \neg C) \mid (D)$. When a match with object $\mathcal{X}$, with tags A and D, is verified, both conjunctions 2 and 3 are evaluated to true. Consequently, two notifications will be sent to the subscriber (one from each cell that verified the match). We embrace this byproduct of our divide and conquer approach in two ways. First, we treat these duplicates as a positive outcome, because this (small) redundancy provides, to some degree, tolerance to lost messages. Second, we employ a duplicate detection in the subscriber side (and drop duplicate notifications).

6.4.3. Moving subscribers

When a subscriber moves to a different cell (i.e., each time a node crosses the boundary of a cell), it must update its location for every active subscription it owns. During this situation, notifications sent to moving subscribers may never reach their destination. In such cases, the underlying routing layer returns negative acknowledgments (NACKs) for messages addressed to *individual* nodes that could not be delivered (see Section 6.6.4). NACKs are used to convey that a node is no longer in its supposed cell, which may be caused by movement, or node failure.

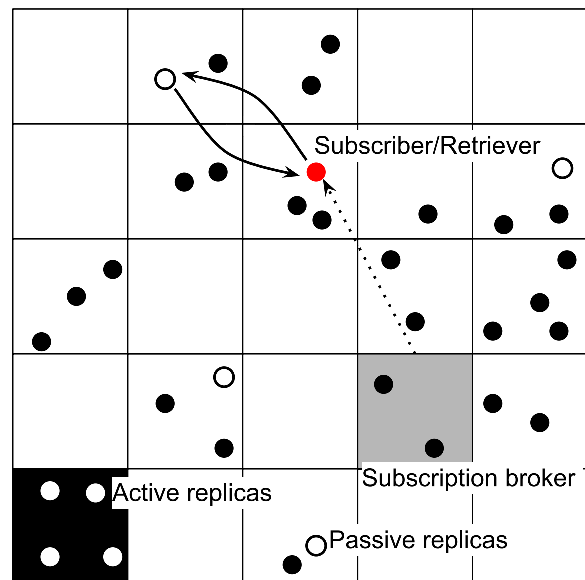


Fig. 4. Notification and retrieve operation in THYME-DCS. The dotted arrow is a notification sent to the subscriber. The other arrows represent a retrieve operation (request and reply).

Node movement will be detected through the subscriber's location update[2]. In such case, THYME can re-send the notifications that were not previously delivered. Otherwise, after a (configurable) maximum waiting time, THYME assumes the node has failed, and simply stops sending notifications. In case the node did not fail, and was just a straggler, it will have to re-issue all its subscriptions.

6.4.4. Unsubscribing

When executing this operation, unsubscribe messages are sent to the cells determined by hashing each conjunction key of this subscription, and the specified subscription is removed from storage.

6.5. Retrieving data

Retrieve operations leverage the replication mechanisms in order to optimize from where to request an object. From all the locations in the replication list (Section 6.2) received in the object metadata (with the notification), the requesting node chooses the geographically closer replica, and sends a retrieve request for the desired object, as Fig. 4 illustrates. If a negative reply is received, the requester proceeds and tries to retrieve the object from the next closest location in the replication list (until no more options are available, or a maximum number of retries is reached). As a last attempt, the cell *actively* replicating the desired object will be used (if not already tried), because it offers higher chances of success compared to every other replica.

One interesting aspect of using geographical routing is that it becomes easier for nodes to make hints on which replicas are better (i.e., closer), using the geographic distance as metric. Since geographical positions have a close relation to topology in wireless networks, we expect this approach to minimize the distance data has to travel in the network, allowing for a location-aware strategy when retrieving objects.

[2] In fact, these location updates can be merged with the updates for the passive replicas (Section 6.2.3), reducing the amount of communication needed.

6.6. Storage substrate & routing layer

The major drawbacks of a routing protocol based on a DHT for wireless networks are the mismatch between the logical and physical topologies, and the high maintenance overheads [58]. Inspired from both wired [51,59] and wireless [17,60] settings, we adopt a cell-based GHT as our routing protocol. By using geographic information, there is no mismatch between the logical and physical topologies [58]. Also, by leveraging on the control traffic of the underlying geographic routing protocol, the GHT does not add any other maintenance costs. At the same time, the cell-based approach relaxes the requirements for location accuracy, and is more robust to topology changes (requiring no action as long nodes move inside their current cells).

DHTs only provide routing, thus we implement a DCS substrate on top of this GHT, providing a simple key–value storage abstraction. Data items are named, and both their insertion and retrieval are performed using those names. To make this layer more suitable for the highly dynamic environments we target, we introduce several mechanisms and optimizations.

6.6.1. Routing

Our routing scheme is very similar to the ones used in [16,17]. Routing is done at cell-level, using a variation of the greedy perimeter stateless routing (GPSR) protocol [61]. GPSR makes greedy decisions, forwarding messages to the next neighbor geographically closer to the message destination. When such strategy is not possible, the algorithm resorts to a recovery mode that forwards messages around the voids in the network. For forwarding messages from cell to cell, we use unicast in order to take advantage of the (per hop) MAC-level retransmission mechanism. This layer provides: (1) a routing mechanism between cells; (2) routing to an individual node (in a specific cell); and (3) broadcast within the context of a single cell[3]. In our implementation, the one-hop broadcast is used as a neighbor discovery mechanism – transmitting *periodic beacons* with the node's current cell –, and as the intra-cell communication primitive. Since broadcast is not acknowledged at MAC-level, this makes it a best effort communication primitive.

6.6.2. Dynamic cell structure

It is impossible to ensure that every cell is populated. Thus, we address empty cells forcing keys to take an entire loop around those cells [16,17], stopping in the cell closest to the supposed destination (which becomes a *proxy* of the key's destination cell). This raises another problem when nodes populate previously empty cells, or leave the system and make some cell empty. So, a cell becoming empty has to deliver all its keys to its proxy cell. In turn, a cell becoming populated needs to receive its keys from its proxy cell, and also all the keys of the empty cells for which it now becomes the new proxy.

If two geographically independent clusters of cells connect at some point in time, GPSR and this proxy logic will trigger a rearrangement of the cells structure and its data. Eventually, the two clusters will merge and cell data will stabilize in its due location [16].

6.6.3. Mobility awareness

We argue that moving nodes render routing information *volatile*, thus, in sharp contrast with CHR and GPSR, our routing layer is mobility-aware: only *stationary* nodes actively participate in message routing. Since our target scenarios have mild mobility patterns (i.e., nodes do not move constantly, and some might not even move during the entire event), *only* stationary nodes form the GHT. When a node starts to move and leaves its current cell, it stops participating in the routing protocol (i.e., it stops forwarding messages). It resumes the protocol when it detects itself as being stationary, by joining the local cell. Notice that in this event, data stored by the node in the previous cell is replicated only at that cell. The data owner however, will also update its new location in the metadata of previously inserted objects (behaving as a passive replica for that content). While moving, nodes still process received periodic beacons, allowing them to keep communicating with the GHT.

[3] Although the broadcast is received by other nodes in range, the message is filtered out at the routing layer.

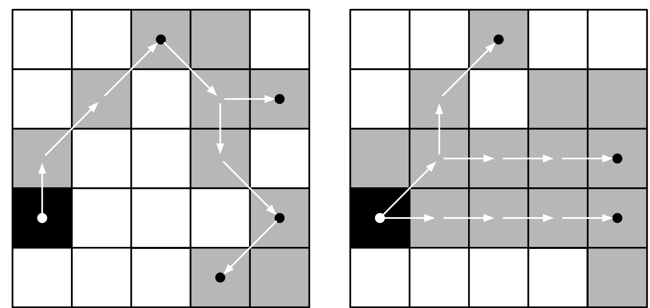


Fig. 5. Message destination aggregation working examples. The dark squares are populated cells, and black dots are the multiple message destinations.

6.6.4. Negative acknowledgments

According a typical GHT interface, nodes are not individually addressable, i.e., we only send messages to specific geographic positions (that correspond to cells in our cell-based approach). Nonetheless, we support the sending of messages to a node in a *specific cell* (e.g., send a message to node *a* in cell 12). To allow the upper layers to react to a node failure or migration from one cell to another, the routing layer replies with a NACK to a *message source*, when a message addressed to an *individual* node could not be delivered (because the node was not in the supposed cell).

6.6.5. Message destination aggregation

For messages that are to be delivered to multiple destinations (e.g., notifications), we optimized our routing scheme by only propagating a single message to those destinations, in what we call *message destination aggregation*. This message is only duplicated when strictly required, which happens when the message's next hop for different destinations is not the same (as depicted in Fig. 5). Thus, achieving a kind of tree-like routing, contributing to reduce the energy consumption and the occupancy of the wireless medium. This mechanism is more effective in sparsely populated scenarios (Fig. 5(a)), as there are less possible paths where messages can be duplicated. Contrary, in densely populated scenarios, since there are more direct paths from source to destination, this observation cannot be exploited so efficiently (Fig. 5(b)).

6.7. Joining the system

In Thyme-DCS, a node joining the system waits a configurable amount of time, listening for other nodes' periodic beacons sent by its neighbors. If, during that time, it receives a beacon sent by a neighbor in its own cell, the sender of that beacon is used as an entry point. A join request is then exchanged, and the joining node receives all the cell state in a reply. If a maximum number of retries is reached, the node assumes it is alone in the cell, and starts operating normally, i.e., the cell was empty, and is now occupied as described before.

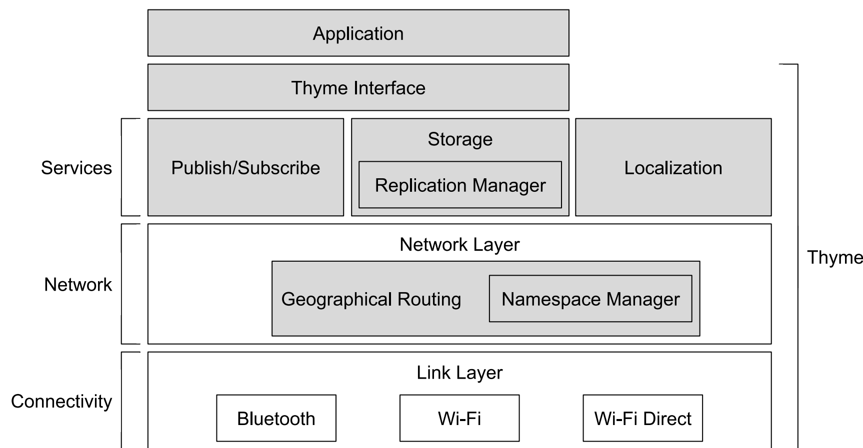


Fig. 6. THYME-DCS Android library architecture.

Table 4
Signature of the insert and subscribe operations.

```
void insert (DataObject data, Collection<Tag> tags, byte[] description, OperationHandler opHandler)
void subscribe (TagExpression query, Time start, Time end, NotificationHandler notHandler, OperationHandler opHandler)
```

data	-	data object to store
tags	-	tags associated with the data object
description	-	description of the data object to store
opHandler	-	handler for handling the success or failure of the operation
query	-	tags relevant for the subscription
start/end	-	validity time interval for the subscription
notHandler	-	handler for the reception of notifications of data objects matching the given subscription

7. Android implementation

Both THYME-LS and THYME-DCS were implemented in the ns-3 network simulator [18] to allow large-scale experiments (see Section 9). Nevertheless, to be able to experiment with our proposed solution in real world scenarios, even though in a small scale, in this section we address the application of THYME to networks of Android mobile devices. Here, we focus on the THYME-DCS approach due to its range of applicability, since it copes with mobility and churn concerns. We apply the THYME-DCS approach to real world networks of Android devices and use it in the development of a photo sharing application.

7.1. Architecture

Fig. 6 depicts the multi-layer software stack that executes at each node. We present this architecture from a top-down perspective focusing on the challenges raised by implementing THYME in real world networks of mobile devices.

THYME *Interface.* It offers the (asynchronous) operations presented in Section 3. The outcome of these operations must be dealt with through the implementation of specific handlers (i.e., callbacks). As an illustration, Table 4 presents the signatures of the insert and subscribe operations for our Java prototype.

Publish/subscribe. Separated in client and server counterparts, this module manages the match between stored objects and subscriptions, and emits the necessary notifications. The client side manages insert and subscribe operation requests, as well as the reception of notifications. In turn, the server counterpart manages the matching (and storage) of data objects and subscriptions, having into account that each data object/subscription features a namespace identifier (see Section 7.2).

Storage. Also divided into client and server counterparts, this module manages the storage of data objects and their metadata, as well as of subscriptions. The internal Replication Manager sub-module manages the active and passive replication mechanisms (Section 6.2). In this implementation, passive replication is built-in and cannot be disabled. However, active replication is optional: its default can be set to either active or inactive, and can be overridden in a per operation basis. For instance, data stored by the Publish/Subscribe module (i.e., data objects and subscriptions) always use active replication to ensure that such information is replicated inside the responsible cell. On the other hand, the dissemination of commercial advertisements may not require persistent storing. The actual active replication model may be injected in the Replication Manager, in order to support different strategies such as cell-wide replication, independently of the cell's population, or maintain a certain number of replicas, also independently of the cell's population.

Localization. Geographic routing requires nodes to be able to determine their own geographical position. For that purpose, we are currently resorting to the globally available GPS information. This option allowed the rapid prototyping of the Localization service. However, there are other possible solutions [49].

Network communication. One of the fundamental differences between the ns-3 THYME and our Android implementation is the *ad-hoc* communication, as this mechanism is not available in the targeted off-the-shelf Android devices. To overcome this limitation, all network communication in our prototype makes use of a communication library [62] developed in the context of the Hyrax research project[4] (i.e., the white components in Fig. 6). This library supports one-hop and multi-hop networks by using one or more wireless networking technologies. Currently, the following are supported: Wi-Fi, Wi-Fi Direct and Bluetooth. Regarding the

4 http://hyrax.dcc.fc.up.pt

API, the library supports synchronous and asynchronous unicast and scoped broadcast messaging.

We adapted this communication library to support the GPSR protocol [61], and support cell-level routing. Messages may be addressed to either a node or a cell. In the latter case, a random node of the target cell is chosen to either route the message to the next cell or to process the message itself, if the destination has been reached. Cell-wide communication is achieved via the scoped broadcast functionality.

7.2. Multiple namespaces

The ns-3 version of Thyme was designed with a single grid in mind (Section 4.3), being the grid defined before the system starts. This grid works as a namespace or like a directory in a file system. However, this real world implementation allows for multiple (overlapping or non-overlapping) grids/namespaces, and provides a namespace discovery mechanism. This feature allows for a two-level naming hierarchy that was flat before. For instance, in the context of the photo sharing application (see Section 7.4), this feature enables the existence of multiple photo galleries shared by different users.

To cope with this demand, an application may manage multiple instances of Thyme, each bound to its own namespace. These instances present the previously described instance-agnostic Thyme interface, but embed an internal unique identifier that will be used by all modules of the software stack, ensuring the clear separation between the data of the multiple Thyme instances.

The creation of a new namespace requires the configuration of the geographical area to be covered by the associated Thyme instance, and the name to use when advertising the instance to the network. Currently, namespaces/grids have a rectangular shape, and their configuration requires a reference point and its length in all four cardinal directions. The dimension of each cell is computed automatically and depends on the networking technology in use. In the case of Bluetooth and Wi-Fi Direct, the dimension is computed from the technology's usual communication range. In the case of Wi-Fi, the size of the cell is set by a platform configuration parameter.

7.3. Handling mobility

Device mobility impacts Thyme in several ways. First of all, it is necessary to know the device location, in order to: (1) deliver notifications; (2) send replies to previously issued requests; and (3) track of the whereabouts of passive replicas.

Secondly, it is necessary to know in which cell a device is parked so that device may contribute to the cell's responsibilities, namely storing data objects and subscriptions.

Mobility is detected by sensing the device's accelerometer. Subsequently, the node will switch to *mobility* mode as soon as it leaves its cell, and will persist in such mode until it remains stationary for a configurable time period. While it is moving, a node will not work on behalf of any cell, but will process messages addressed to itself, such as notifications. To that end, as it moves across cells, the node will have to update its subscriptions' data with its new location (Section 6.4.3).

When the system locally detects that a node is no longer moving, if the final cell is not the same as the origin, the node discards all the (meta)data kept about the origin cell, updates its location in the system (namely with respect to the passive replicas it holds), and begins working on behalf on its new cell, replicating data and answering requests.

7.4. Shared photo gallery

A practical Thyme use case is a photo sharing application to be used at social events. Thus, as a case study, we developed the Shared Photo Gallery application that allows users to share photos without requiring Internet access. The *app* can run on any device with Android 5.0 (Lollipop) or higher, without having *root* access, and works even in the absence of a communication infrastructure (when using Bluetooth or Wi-Fi Direct).

Users publish (or insert) photos with at least one tag and subscribe to the tags they are interested in, indicating a validity time frame for each subscription. This time frame may be unbounded in both ends, allowing for subscriptions to cover the event lifespan. Whenever a published photo matches one of the active subscriptions, a notification is sent with the photo's thumbnail (and a list of possible download locations). Upon reception of such notification, the user may choose to immediately start the download, postpone it, or discard the notification. Whatever the action, the user will be informed of its success or failure.

Fig. 7 depicts some of the application's screens. In Fig. 7(a) it is possible to identify four tabs: *Private* displays the private photos from the device's gallery, that can be published; *Publications* displays the photos already published by the user; *Downloads* shows the photos that were previously downloaded; *Available* displays the photos whose download has been postponed.

Also, in this figure one can see the subscription and unsubscription buttons, represented by the bell symbols. The other figures (Figs. 7(b)–7(d)) illustrate the processes of, respectively, publishing a photo, issuing a subscription, and handling the reception of a notification.

The application may interact with more than one gallery. Users may thus search and connect to active galleries on neighboring devices[5] or create their own galleries. To navigate between galleries the user has simply to access the menu in the upper right corner and select the *Switch gallery* option, which will lead to a list of the available galleries.

8. Analytical study

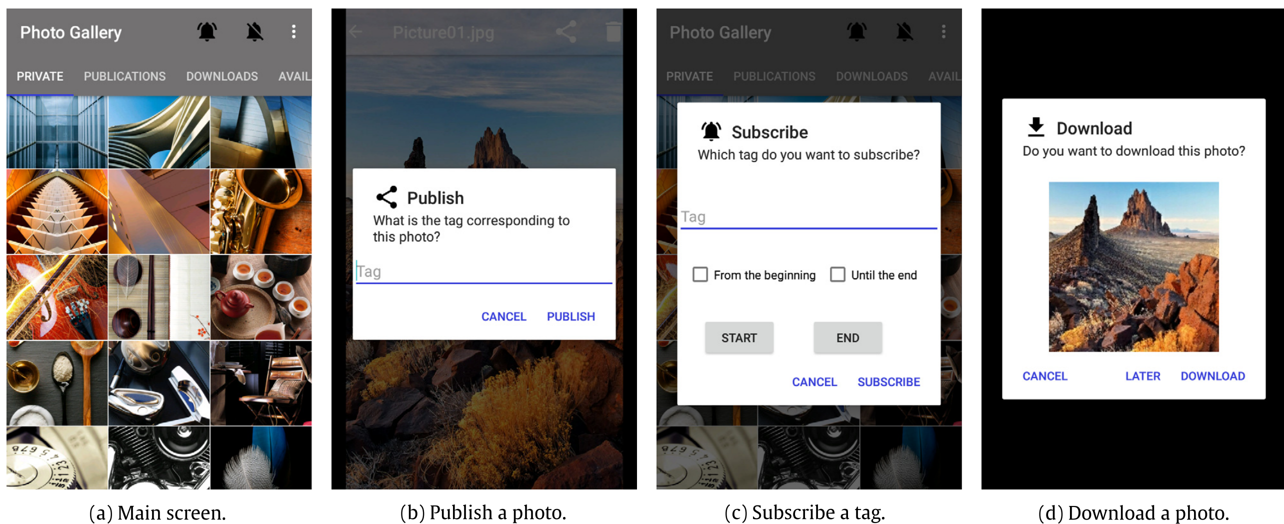
We now compare our approaches using a simple analytical model to derive approximate formulas for communication costs and operations complexity. In the following, we use the asymptotic costs of $O(n)$ message transmissions for floods and $O(\sqrt{n})$ for point-to-point routing, where n is the number of nodes in the system [16]. However, since in Thyme-DCS we cluster nodes into cells, point-to-point routing still costs $O(\sqrt{n})$ but, here, n becomes the *number of (populated) cells* in the system.

As a baseline for comparison, we devise an additional approach, Thyme-ES, using the client/server model and based on *external*, centralized storage. Storage is external in the sense that it does not belong to the nodes forming the network, i.e., it belongs to a different (server) component, known a priori by every node in the system. Objects, their metadata, and subscriptions are stored in external storage, and every operation is sent to that server to be processed (and replied back). Naturally, this server component is a single point of failure in the system, but can use any known techniques from the literature to address this issue (e.g., failover, or state machine replication [63]).

8.1. Time complexity

Operations (average) time complexity is as shown in Table 5. Delete and unsubscribe are the inverse operations of insert and

[5] Access control and security is mandatory in this environment but falls outside the scope of this article.



(a) Main screen. (b) Publish a photo. (c) Subscribe a tag. (d) Download a photo.

Fig. 7. Shared Photo Gallery Android application.

Table 5
Operations time complexity.

	ES	LS	DCS
Insert/Delete	$O(\sqrt{n})$	$O(1)$	$O(\sqrt{n})$
Retrieve	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n})$
Subscribe/Unsubscribe	$O(\sqrt{n})$	$O(n)$	$O(\sqrt{n})$

subscribe, respectively. Despite their messages carrying slightly different information, in terms of complexity, they exhibit the same behavior.

Since ES and DCS use point-to-point communication for every operation, their complexities are the same. However, while in ES, the majority of the work is executed by the external component, in DCS, the work is spread among the cells, i.e., the system nodes. Also, take into account that by clustering nodes into cells, the point-to-point routing has the potential to be more efficient. In turn, LS trades linear (un)subscribe operations for constant inserts and deletes. In all approaches, retrieve operations are executed using point-to-point communication, by using the information contained in the metadata received in the notifications and requesting the object directly to one of the available replicas.

8.2. Space complexity

Regarding space complexity, ES exhibits an extreme behavior, because the external component has to store every piece of data of the system (i.e., objects, their metadata, and subscriptions). In turn, LS sits in the middle of the spectrum, by storing objects only in their owners' storage, but fully replicating every subscription. Lastly, DCS spreads both the storage of objects, metadata and subscriptions among its cells/nodes, through hashing. In terms of storage, the LS and DCS approaches are not directly comparable. However, they both reside in the middle of the spectrum, spreading (different) parts of the system data among the nodes.

8.3. Communication costs

The communication cost structure for each approach is described by several parameters. Let D_i denote the total number of stored objects; D_r denote the number of retrieved objects; S denote the total number of issued subscriptions; and D_s denote the number of matching objects (i.e., the total number of notifications). For DCS, c is the (average) number of nodes in a cell.

We compare costs using approximations for both the total number of sent messages in the network (taking into account multi-hop routing), and the number of messages sent by a hot-spot (i.e., the *maximal* number of messages sent by *any* particular node). In this comparison, we only address the insert, retrieve, and subscribe operations, and the respective notifications. The delete and unsubscribe operations are analogous to insert and subscribe, respectively. With this setup, the approximate communication costs (total and hot-spot) are as shown in Table 6.

The formulas for ES are derived from the observation that every operation is sent to the central component to be processed, and a response is sent back to the requester. Thus, for every operation, the communication cost (of a point-to-point message) is multiplied by two. The exception are notifications that only require a message sent in one direction (from the central component to the subscriber). Naturally, the hot-spot is going to be the central component, which has to process every received message and reply accordingly. Hence, the hot-spot communication cost is the sum of all the received messages (that have to be replied) and the sent notifications.

For LS, the formulas are deduced from the facts that inserts are local and do not require communication, while subscriptions are flooded through the entire system (Section 5). Both the retrieve operation and notifications follow the same logic as the previous approach.

Lastly, the formulas for DCS are inferred taking into account its cell-based GHT approach. The insert operation requires the same steps as ES, two point-to-point messages (to the broker cell and back to the requester). However, in each of these two steps there is a scoped dissemination of the corresponding messages in the local cells (i.e., the dissemination of the metadata in the broker cell, and the dissemination of the data object in the data owner's cell; Section 6.2.1). Next, the retrieve operation requires the same steps as the other two approaches (two point-to-point messages), with an additional point-to-point message for setting a new passive replica (Section 6.2) and the corresponding scoped dissemination in the broker cell. The subscribe operation follows the same logic as the insert. However, it only has one scoped dissemination of the subscription in the broker cell. Notifications follow the same logic as the previous approaches (sending a point-to-point message directly to the notification receiver).

Here, we assume a simple scenario where inserted data objects have only one tag, and subscriptions also have only one conjunction key. In more elaborate scenarios, the (possibly variable)

Table 6
Operations communication costs.

	ES	LS	DCS
Insert	$2D_i\sqrt{n}$	$\emptyset$	$2D_i\sqrt{n} + 2D_i \cdot c$
	+	+	+
Retrieve	$2D_r\sqrt{n}$	$2D_r\sqrt{n}$	$3D_r\sqrt{n} + D_r \cdot c$
	+	+	+
Subscribe	$2S\sqrt{n}$	$S \cdot n$	$2S\sqrt{n} + S \cdot c$
	+	+	+
Notification	$D_s\sqrt{n}$	$D_s\sqrt{n}$	$D_s\sqrt{n}$
Hot-spot	$D_i + D_r + S + D_s$	$D_r + S + D_s$	$D_i + D_r + S + D_s$

number of tags will have impact in the number of messages required for some operations.

Now, we can conclude that the total message count in LS grows faster (linearly with n) than in ES and DCS. Another important conclusion is that, if $D_i \gg S$, then LS has significantly lower message count than the other two approaches. This comes from the fact that LS insert (and delete) operations execute with no communication. Naturally, if we invert that condition, ES and DCS will exhibit lower message counts than LS.

Once again, ES and DCS exhibit a similar behavior in terms of overall communication costs. However, DCS presents slightly higher costs in almost every operation, thus they have intrinsically different performance behaviors. These higher costs come from the replication mechanisms employed by DCS (Section 6.2). In an insert operation, besides the normal request/reply messages, by applying active replication (Section 6.2.1), both the object data and metadata are (actively) replicated in the owner's and responsible cells, respectively. In the retrieve operation, after obtaining the object data, the requester node passively replicates that object, thus needs to update the replication list in the object metadata (Section 6.2.3). Regarding subscribe operations, subscriptions are also actively replicated in their responsible cells.

With these extra mechanisms, naturally many operations in DCS have a slightly higher communication cost. However, while ES has an external, central component acting as a server (that is also a single point of failure of the system) and storing all the system's data, DCS spreads that load among its cells/nodes. Thus, in ES the hot-spot cost is the actual cost paid by the external server. On the other hand, the hot-spot cost in DCS is not actually paid by a single node, because that cost is shared among the different cells (and among the nodes of each cell). Even if there is only one cell, this work will be (randomly) balanced between the nodes inside it.

We can also look at these costs as the amount of work a device has to do on behalf of the system. Specific to the DCS approach, the handling of messages related to the five operations grows linearly with the number of such operations, and does not depend on the number of nodes per cell. That is, n operations require each node on the cell responsible for the target tag to process on average n messages: one node receives the initial message and then broadcasts it to its cell neighbors. The same happens on each retrieve operation: a message is sent to the target cell to indicate the existence of a new passive replica, and this information is then broadcasted within the cell. Regarding active replication, the use of such mechanism implies one broadcast on the owner's cell for each operation. So, with active replication each node in such cells processes on average one message per operation.

The only type of messages that depends on the number of nodes per cell is the ones concerning notifications. As depicted in Fig. 8, the more populated a cell is, the less work each node has to do. These only require the intervention of one node per cell, the one checking the match between an object and a subscription, and sending the notification to the corresponding subscriber. This work is also load balanced, because the object-subscription

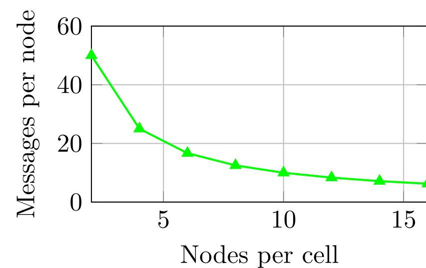


Fig. 8. Average number of messages sent per node for 100 notifications processing.

matching is not performed by the same node inside a cell. It is distributed randomly among the cell nodes (during message routing).

8.4. Discussion

If the number of inserted data objects is larger than the system size and the number of subscriptions, Thyme-LS may be preferable. However, this approach does not address data persistence in case of node failure.

On the contrary, Thyme-DCS addresses data persistence through replication. It should be preferable in cases when the network is large compared to the number of stored objects, being more worthwhile in densely populated scenarios.

9. Evaluation through simulation

Our experimental evaluation is divided in two parts: simulation and real world experiments. First, we use a network simulator (ns-3 [18]) to experiment our proposal in large-scale scenarios, and implemented our two approaches: Thyme-LS and Thyme-DCS. Secondly, we implemented the Thyme-DCS approach as an Android library, and developed a proof-of-concept photo sharing application on top, allowing experiments in a small scale scenario using real mobile devices.

This part of our evaluation focus on the simulation experiments and seeks to answer the following questions:

1. Which are the trade-offs provided by each approach of Thyme?
2. How does each approach deals with churn?
3. How do they react to node mobility?

Each data point reports the average of five randomly generated network topologies, each independently run three times, making a total of 15 runs per data point. As a baseline for comparison, we used the centralized approach, Thyme-ES, described at the beginning of Section 8.

The metrics used in this section to answer the previously defined questions are: amount of generated traffic (in bytes and in number of packets), and operations' latency and success ratio. All these metrics allow the comparative analysis of the behavior of both Thyme approaches. Since we are addressing resource-constrained mobile devices, the lower the generated traffic and the operations' latency the better, because this has direct implications in the devices' battery usage. In the end, the best approach is the one able to achieve the lowest latency and generated traffic while producing high operations' success ratios.

9.1. Implementation

We use ns-3.27 and nodes communicate through 802.11 Wi-Fi ad-hoc (using UDP). Both Thyme-ES and Thyme-LS use DSDV [50] as their routing protocol.

In Thyme-DCS, when a cell becomes empty or populated, a state transfer needs to happen between cells (Section 6.6.2). Currently, we do not implement such mechanism, thus, in our experiments, cell structure is static (i.e., populated and empty cells will remain as such throughout the experiments). This poses some limitations regarding node mobility and churn in Thyme-DCS: nodes may move freely inside a cell, but may only leave a cell if it remains populated afterwards; and nodes may only migrate to previously populated cells.

To recover from lost messages, all approaches employ a retransmission mechanism. After a configurable amount of time has passed without receiving the expected replies, the operation is retried. If a maximum number of retries is reached, the operation fails with a timeout error code.

Our code implementation of Thyme is available at https://bitbucket.org/hyrax-nova/thyme-ns3, jointly with the trace files used in the simulation experiments.

9.2. Setup and methodology

Unless stated otherwise, all parameters were left with the simulator's default values. We used Wi-Fi 802.11g configured with a constant rate manager and a data rate of 6 Mbps. The RTS/CTS threshold was configured to 1500 bytes.

In order to mimic a realistic scenario, we emulate an application similar to an online social network on top of Thyme (akin to Twitter), e.g., that could be used by fans watching matches in fan zones set up for the 2018 FIFA World Cup.

Trace files were generated with all the operations to be issued during a simulation run. For that, we crawled tweets issued during the 2016 UEFA European Championship final, between Portugal and France[6]. Tweets were used as data objects, where: the tweet id was used as the object identifier; the text was used as the object data; the timestamp was used as the object insertion time; and the hashtags were used as the object tags. The top-k most active users were chosen, and every other operation was generated from that, using exponential distributions configured with different λ values (i.e., rates).

Subscriptions were generated taking into account the tags of the inserted objects, and the top 60% of the most popular tags were used for the subscriptions' queries (for simplicity sake, each subscription subscribed to one tag chosen at random). Subscriptions were generated in two forms: time independent ($ts^s = ts^e = \perp$); and in the future ($ts^s = now$ and $ts^e = \perp$). Time independent subscriptions where generated with a probability of 60%. During the first half of the game, subscriptions were generated with a rate of three operations per user per hour, and reduced to one per user per hour for the remainder of the event.

Delete and unsubscribe operations, which are expected to be rare, were generated with a rate of 0.5 and 0.2 operations per user per hour, respectively, only during the second half of the game.

We crawled a total of three hours, starting at 20:00 2016-07-10. To make the simulation execution more lively (and to reduce the simulation total time), we compressed the three hours into ten minutes of simulated time. Since we use real tweets for trace generation, the distribution of operations in a trace file is irregular, with occasional spikes and void moments. Fig. 9 depicts an example of the distribution of operations in a trace file

[6] Using the code in https://github.com/Jefferson-Henrique/GetOldTweets-python

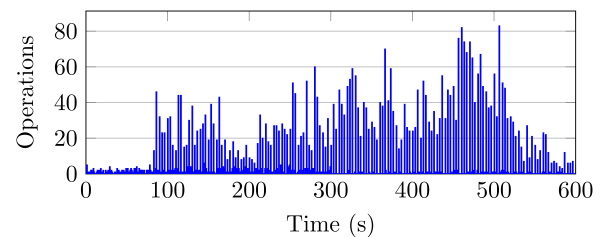


Fig. 9. Distribution of operations over time in a trace.

over time (for 100 users). The trace files used in this section are publicly available in our code repository[7].

The simulation area has a rectangular shape, where nodes were places uniformly at random, to mimic many of the venues we are targeting. For Thyme-DCS, cell size is 40x40 m, which entails a radio range of ±113 meters (roughly the range in our simulated Wi-Fi setting). In all experiments, we had an average density of two nodes per cell (in the following areas: 160 × 80 m, 240 × 120 m, 320 × 160 m, 400 × 200 m, 480 × 240 m, and 560 × 280 m, respectively, for each network size used in the plots).

Proactive routing protocols, like DSDV, require time to converge. Thus, in our simulations, the application running on the nodes only started after 30 seconds. Nodes randomly joined the system in the next 30 seconds, and operations started being issued only after that. At the end of the simulation, nodes only shutdown 60 seconds after operations stopped being issued. Thus, the total simulation time was 720 seconds. All Thyme approaches executed the same traces and used the same methodology.

Since notifications are not operations triggered by the users, we use *recall* (i.e., how many relevant items are selected) as a measure of success. However, we use the number of matching objects for a perfect execution of the trace, where operations *always* succeed and are executed *instantly*. Take into account that, if some operation fails, most likely the number of actual achieved notifications will not match the expected. For instance, if an insertion fails, all the subscriptions matching that object will not be matched against it, and notifications are reported as lost. Thus, achieving 100% recall is practically impossible for our comparison baseline.

9.3. Results

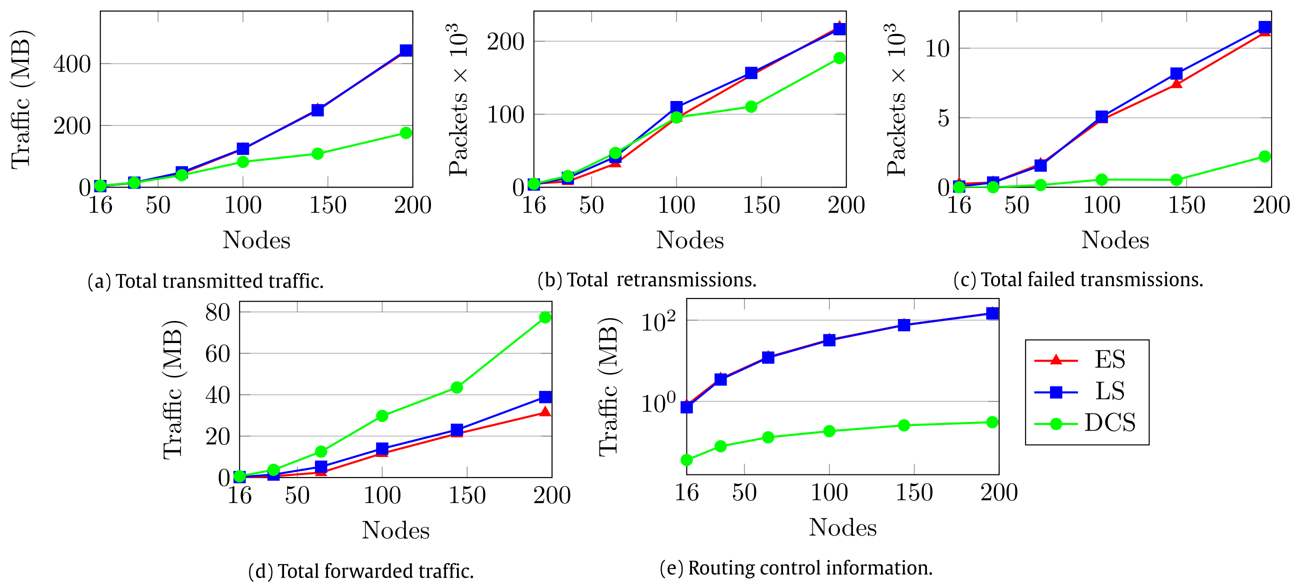
We now present the achieved results for three distinct scenarios, ranging from totally stable nodes to scenarios with faulty or mobile nodes.

9.3.1. Static and stable nodes

In Fig. 10, we can observe the impact that each approach of Thyme has on the lower layers of the network stack (and helps answering question 1). Fig. 10(a) reports the total traffic transmitted by all nodes (at the physical layer—PHY), during the simulation. ES and LS overlap and both exhibit quite an overhead. With 196 nodes, they report more than 2× the transmitted traffic of DCS. Energy is a valuable resource when targeting mobile devices. Thus, looking at these values in an energy perspective, ES and LS will spend twice the energy to do roughly the same work as DCS.

In turn, Figs. 10(b) and 10(c) depict values reported by the link layer—MAC. The former shows the total number of retransmitted packets, and the latter shows the total number of packets that

[7] https://bitbucket.org/hyrax-nova/thyme-ns3/src/master/scripts/traces/files/



(a) Total transmitted traffic. (b) Total retransmissions. (c) Total failed transmissions.

(d) Total forwarded traffic. (e) Routing control information.

Fig. 10. Lower layers metrics (static scenario).

exceeded the maximum number of retransmission attempts. The standard IEEE 802.11 Wi-Fi MAC layer implements CSMA/CA and a per hop retransmission mechanism. Thus, in some way, these figures depict the interference level observed in each approach while operating. Both ES and LS require many more retransmissions than DCS to overcome the loss of messages that is inevitable in a wireless communication medium. This can be explained by the amount of traffic generated by those two approaches. Usually, the more traffic is generated, the larger is the probability of collisions in the wireless medium, and thus more transmissions have to be retried (creating a snowball effect).

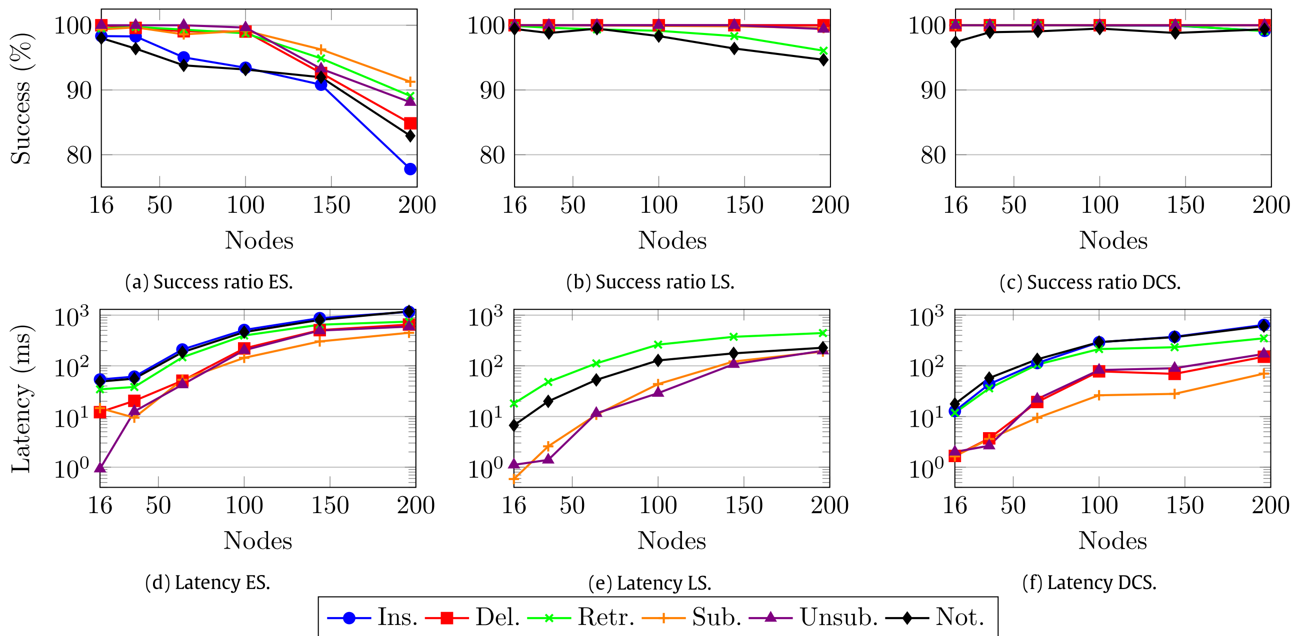
Next, Fig. 10(d) depicts the total traffic forwarded by all the nodes in the system (at the network layer—IP). In some sense, this reports the amount of work nodes have to do on behalf of the system. In this case, DCS forwards more traffic because its messages are forwarded through longer routes than ES and LS (that use DSDV). This is even more exacerbated by some peculiarities of the routing protocol used by DCS (Section 6.6.1). For instance, the fact that some messages may need to loop around voids in the network (Section 6.6.2), while DSDV computes shortest paths to every node.

Fig. 10(e) shows the total amount of control information the routing protocols transmit. Both ES and LS use DSDV, a proactive routing protocol, whereas DCS uses a geographic routing protocol (Section 6.6.1). While DSDV needs to exchange bulky routing tables to compute the shortest paths to every other node in the network, the geographic routing used by DCS routes messages using only local information (nodes only exchange very small periodic control beacons). However, messages may be routed through longer routes in geographic routing. With 196 nodes, a quarter of all the transmitted traffic of ES and LS was control traffic (notice the logarithmic scale in the y axis). These two last figures (Fig. 10(d) and 10(e)) show a clear trade-off. As more control traffic is exchanged, the routing protocols can achieve better routing paths and with that reduce the amount of forwarded traffic. However, that control traffic can correspond to a large percentage of the total network traffic (increasing the total amount of collisions).

Fig. 11 depicts application-level metrics, such as operations success ratio and latency. Regarding operations success, we verify that DCS is above 99%, except for notifications that fluctuate a little bit and have a success ratio as low as 95% (Fig. 11(c)). LS also reports high success ratio (Fig. 11(b)). Since insert and delete operations are executed locally, they always succeed. Subscribe and unsubscribe operations have above 99% success. Only retrieve operations and notifications have a very small reduction as the system grows, having 96% and 95% success, respectively, with 196 nodes. For ES (Fig. 11(a)), we see a slight decrease in the success ratio as the system grows, having as low as 78% success with 196 nodes. In every approach, notifications are a type of message that does not employ an application-level retransmission mechanism, thus they are more susceptible to interferences.

Regarding operations latency (Figs. 11(d)–11(f)), we can see that for a small number of nodes all approaches behave similarly, with ES having slightly higher latencies. As the number of nodes increases, accompanied by increased interferences (Fig. 10(c)), we verify that latencies also increase. This is caused by the need for more retransmissions. However, notifications have lower latency in LS, because the geographic routing of DCS cannot compete with the shortest paths of DSDV. Thus, showing the advantage of calculating shortest paths. On the other hand, retrieve operations in DCS have a slightly lower latency, because DCS causes overall less interferences and it employs a location-aware strategy when retrieving data (Section 6.5). In ES, the decrease in success ratio is accompanied by an increase in operation latency. This comes from the fact that the majority of operation failures happen due to timeout. Since operations have to be retried several times, naturally latency increases. Overall, timeouts may indicate a congested network, where operations consistently have to be retried several times.

In summary, Fig. 10 shows that in DCS nodes transmit much less traffic that in both LS and ES. This comes at the cost of latency, when compared to LS, as shown in Fig. 11. The centralized approach has the worst behavior when the size of the system grows, with a decreasing success ratio and latency much higher that both DCS and LS. These observations come from the fact that both ES and LS use DSDV as the underlying routing protocol. This protocol computes shortest paths to every other node in the network, and to maintain its routing tables up-to-date that information is distributed between nodes by sending full dumps infrequently and smaller incremental updates more frequently, which still represent a large overhead with respect to transmitted data. On the other hand, DCS uses GPSR as its routing protocol, which uses only local information for routing. Thus, they represent a design trade-off: the more control information is transmitted, the better routing decisions can be made.



(a) Success ratio ES. (b) Success ratio LS. (c) Success ratio DCS.

(d) Latency ES. (e) Latency LS. (f) Latency DCS.

Fig. 11. Application-level metrics (static scenario).

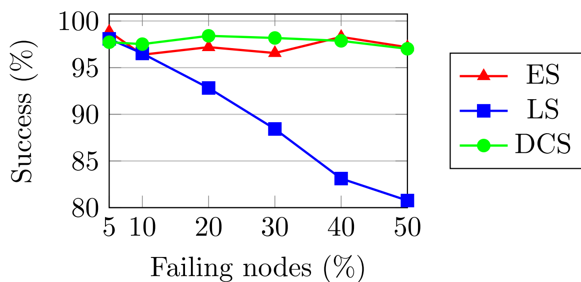
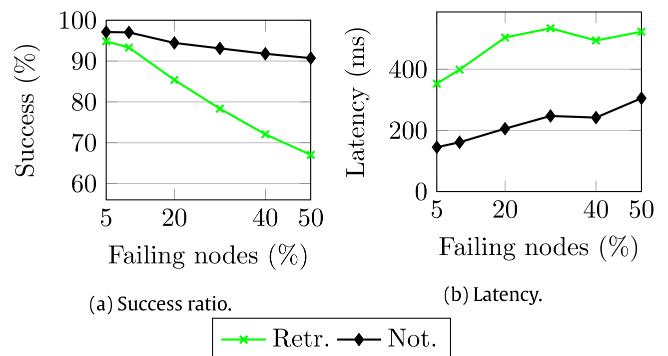


Fig. 12. Notification success ratio (crashing, 100 nodes).



(a) Success ratio. (b) Latency.

Fig. 13. Application metrics for LS (transient, 100 nodes).

9.3.2. Static but failing nodes

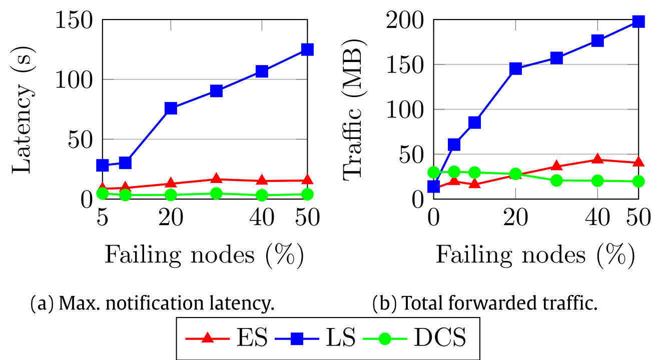
In our target scenarios, mobile devices may experience poor connectivity and/or low battery, thus these devices may fail and leave the network. As such, concerning churn, i.e., the ingress and egress of nodes in the system, we experiment with two different scenarios. We show the impact of nodes leaving the system definitely, e.g., nodes crashing. Secondly, we show the impact of nodes with intermittent failures, thus entering and leaving the system multiple times throughout the simulation. These scenarios allow to evaluate aspects regarding data availability and persistence in the presence of failures (and help us answer question 2).

Permanent failures. In this scenario, from the same trace files as before, we generated new ones where nodes are *either* publishers or subscribers, and publishers choose a random instant (between 200 and 300 seconds of the simulation, i.e., around the middle of the simulation) to leave the system abruptly.

In LS, insertions are executed locally, thus not requiring communication. However, because only the object owner stores that data, if that node fails, all the data it stores will disappear with it. Fig. 12 shows exactly that. In LS, as more nodes with relevant data fail and leave the system, the success ratio decreases because the matching between subscriptions and objects is not detected. Since DCS employs replication mechanisms (Section 6.2), even when object owners leave the system, matching still occurs. ES is not affected simply because all the system data is stored in external storage. As long as that server component does not fail, even if nodes do, data will always be available.

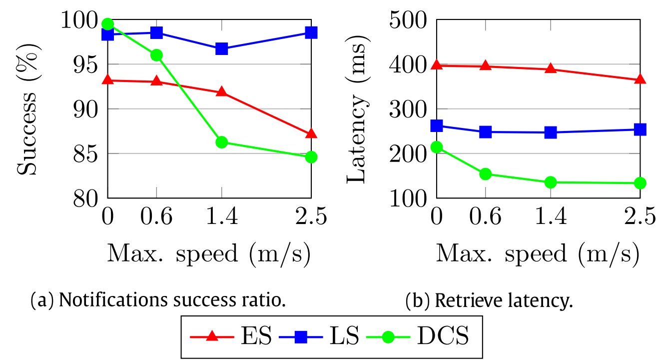
Transient failures. In this scenario, using the same trace files as in Section 9.3.1, randomly selected nodes alternate between the on and off states, during 120 and 60 seconds respectively. Nodes have a 75% probability of changing to the opposite state, otherwise they stay in the same state for an equal period of time.

With nodes entering and leaving the system frequently, retrieve and notification operations are the ones that can be more affected, specially in the LS approach. Fig. 13 presents some application metrics of the LS approach for these two operations. Since DCS employs replication mechanisms, it is little affected by the intermittent churn, with operation success ratio well above 90%, and latencies consistently between 150–300 milliseconds. Due to its central server component, ES is also little affected by the intermittent churn, with operation success ratio above 80%, and slightly higher latencies than DCS, between 400–600 milliseconds. LS, however, suffers from low success ratio in the retrieve operation (Fig. 13(a)). Although some notifications are detected, when a node tries to retrieve an object, as the amount of failing nodes increases, the probability of the data owner being off also increases. This is also accompanied by an increase in the latency of notifications (Fig. 13(b)). In LS, the matching between



(a) Max. notification latency. (b) Total forwarded traffic.

Fig. 14. Transient scenario, 100 nodes.



(a) Notifications success ratio. (b) Retrieve latency.

Fig. 15. Mobile scenario, 100 nodes, pause 120 seconds.

a node's stored objects and subscriptions that were issued when the node was off have to wait for the node to switch state and join the system (Section 5). When joining, a node receives the subscriptions issued by all the other nodes previous to it entering the system (from its neighbor nodes). Then, from all the received subscriptions, the joining node finds those (new subscriptions) of which it was unaware and checks if it has matching objects. Fig. 14(a) corroborates this. The maximum latency for DCS and ES notifications stays stable as the percentage of failing nodes increases. But, in LS, the maximum latency for a notification increases from approximately 25 to 100 seconds when the percentage of failing nodes increases from 5% to 40%.

Fig. 14(b) shows a byproduct of the retrieve operation low success ratio. With no churn, DCS forwards more traffic because its insert and delete operations require communication. However, with this kind of intermittent churn, LS forwards much more traffic than DCS. This is due to the fact that retrieve operations are retried (and fail) several times. Also, this entering and leaving of nodes from the network causes routing tables to become out of date, and thus need to exchange control information much more frequently.

9.3.3. Mobile but stable nodes

In this scenario, we use the same trace files as in Section 9.3.1, however nodes are able to move. This scenario helps us answer question 3. When moving, nodes use the random waypoint (RWP) mobility model, which interleaves pauses with movement. We argue that the plain RWP mobility model does not quite mimic the movement pattern people have in the kind of events we target. For instance, in a music concert, people do not move constantly. In fact, they do not move much during most of the time, except in intermissions. To make it better resemble our target scenarios, we made an adaptation: every time a node is about to move, it tosses a coin do decide whether to move or not. If not, the node continues in a pause moment. In this scenario, 60% of nodes are mobile, and have a moving probability of 80%.

Fig. 15(a) shows a small caveat of DCS: increasing node speed lowers the notifications success ratio. We argue this happens because every node inside a cell is supposed to have the same state and work collaboratively as one. But, the intra-cell communication primitive is the unreliable one-hop broadcast. Thus, nodes inside a cell may not receive the same messages. Mobility may create even more entropy in the cell state.

Fig. 15(b) presents a byproduct of the location-aware retrieval strategy used by DCS. While, ES and LS are required to retrieve data from a specific location, DCS might have different replicas for retrieval at its disposal, and it can choose the one closer to the requester (having the possibility of lowering latency).

10. Evaluation through real devices

This second part of our evaluation has the main goal of assessing the functionality of our Thyme Android library and of the proof-of-concept application. For this, we seek to answer the following questions:

1. Does the implemented Android Thyme-DCS library behaves as expected?
2. What is the behavior of our Android application in terms of operations' latency?
3. What about in terms of energy consumption?

Each data point reports the average of five independent runs.

The metrics used in this section to answer the previous questions are: operations' latency and energy usage. These metrics allow the analysis of each operation's behavior in the context of the Thyme-DCS approach implemented in real Android devices. Once again, since we are addressing resource-constrained mobile devices, the lower the operations' latency and the energy usage the better (having into account that these two metrics have a correlation implication between them).

10.1. Implementation

Both the Thyme-DCS library and the Shared Photo Gallery application were developed for devices with Android 5.0 (Lollipop) or higher, with no root access required.

In our Java prototype, we also do not implement the state transfer mechanism when cells become empty or populated (Section 6.6.2). So, cells must remain populated or empty throughout the experiment. Here, we also did not implement some optimizations like NACKs (Section 6.6.4) and message destination aggregation (Section 6.6.5).

Similarly to the ns-3 implementation, here operations also employ a retransmission mechanism.

10.2. Setup and methodology

We conducted a series of experiments with different scenarios trying to simulate some possible realistic use cases. In these uses cases, we test all the features provided by the Thyme-DCS library and used by the photo sharing application: devices inserted and deleted photos, subscribed (and unsubscribed) to tags in the past and future, received notifications, and retrieved available photos. Each device had a randomly assigned role (publisher or subscriber), and operations were executed in a closed loop. We used a custom profiler to collect various metrics during these experiments aiming to analyze several performance indicators, such as latency and energy consumption.

Table 7
Devices specifications.

	HTC Nexus 9	Motorola Nexus 6	LG Nexus 5X	Motorola Moto G (2nd gen.)
CPU	Dual-core 2.3 GHz	Quad-core 2.7 GHz	Hexa-core 4 × 1.4 GHz + 2 × 1.8 GHz	Quad-core 1.2 GHz
RAM	2 GB	3 GB	2 GB	1 GB
Storage	16 GB	32 GB	16 GB	8 GB
Battery	Li-Po 6700 mAh	Li-Po 3220 mAh	Li-Po 2700 mAh	Li-Ion 2070 mAh
OS	Android 7.1.1	Android 7.1.1	Android 7.1.1	Android 7.1.1
Wi-Fi	802.11 a/b/g/n/ac	802.11 a/b/g/n/ac	802.11a/b/g/n/ac	802.11b/g/n

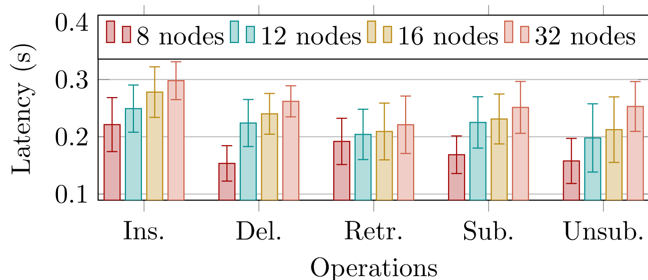


Fig. 16. Operations latency.

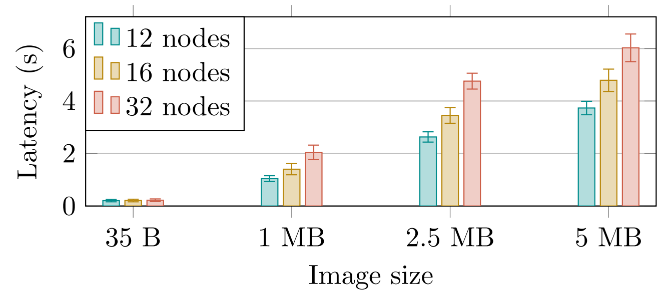


Fig. 17. Retrieve operation latency varying image size.

Unless stated otherwise, experiments were conducted using images with 35 bytes in size. This allow us to measure the operations latency and overheads taking only into account the data generated by the system. Naturally, as the size of the inserted data grows, operations latency and system overheads also grow proportionally (as we will show) with respect to the available bandwidth.

Experiments were conducted in a network of up to 32 Android mobile devices, connected to a Wi-Fi access point. The tests up to 16 devices were performed exclusively with Nexus 9 devices, while the 32 devices testbed used every type of device referred in Table 7. In all experiments, we used two cells and devices were divided equally between the two. We also used only one namespace for all devices.

10.3. Results

We now present the achieved results for the collected metrics. The error bar depicted in the plots indicates the standard deviation.

10.3.1. Functionality

With all the uses cases, we conducted experiments in order to test all the operations and features of the Thyme Android library. After extensive testing, we verified that all operations were performed successfully, including those involving the retrieval of data inserted by devices that had left the system. The use cases with churn allowed us to test the operations in scenarios with device or communication failures.

In the end, throughout all the experiments, operations were always completed successfully, proving the data persistence in the system. It should be noted that the retrieve operation's success, in the cases with churn, is guaranteed at the expense of an increase in the operation latency, i.e., if a device that left the system is selected or if the message is lost, the operation will be retried after a timeout.

10.3.2. Latency

Operations latency is an important measurement for assessing an application's usability. For that purpose, we measured the latency of all five operations (during the execution of the use cases) and present the average.

Fig. 16 depicts the latency of those operations, varying the number of mobile devices in the network. The values represent the time elapsed from the moment the action was triggered by the user in the application interface, until the reception of the operation's success reply. In general, we can consider that all the operations show acceptable response times (around 200 milliseconds on average). Insert is the operation that may take longer, depending on the size of the thumbnail sent in the metadata, which was kept particularly small (35 bytes) in our experiments. The experiment confirms that increasing the number of nodes also increases the network traffic, which in turn increases interference, reduces the available bandwidth for each device, and impacts the latency of the operation. Even so, all results are kept under 300 milliseconds, which is perfectly acceptable for an interactive application, with all the operations executed in quasi-real-time and ensuring a good user experience.

Regarding the retrieve operation, the latency depends on the size of the data item to be obtained, as shown in Fig. 17. An image with 5 MB in a network of 32 devices, takes about 6 seconds, which is acceptable considering that the operation runs in the background and the user may keep on using this or other application on the device.

In conclusion, the implementation of Thyme and the developed application meet our expectations, presenting good response times, which guarantee a good interactive experience to the user.

10.3.3. Energy consumption

When it concerns mobile devices, energy consumption is a determining factor, since devices are battery-constrained. In order to evaluate the energy consumption of our application, we used the aforementioned use cases and measured the energy consumption: (1) when issuing an operation; (2) when processing an operation request; and (3) in the maintenance of the virtual node, i.e., update the state after a new request is received for the cell.

Battery consumption was measured exclusively on the Nexus 9 devices, to avoid variations in measurements that could occur if different devices were used. The battery measurements were done automatically via a module that uses the *BatteryManager* class provided by the Android OS. After a first analysis of the results, we concluded that the energy consumption values did

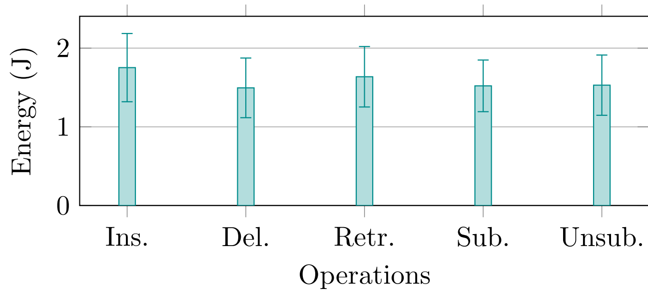


Fig. 18. Energy consumption when issuing an operation.

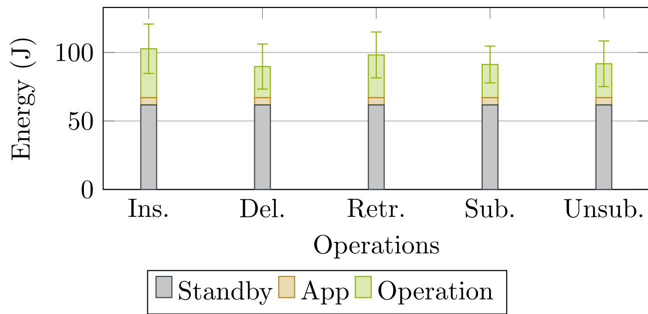


Fig. 19. Energy consumption when issuing an operation in a closed loop during one minute.

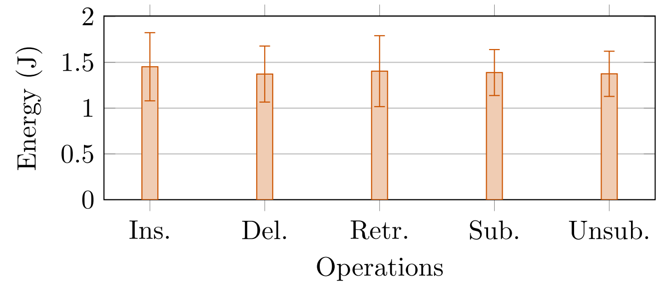


Fig. 20. Energy consumption when processing an operation request.

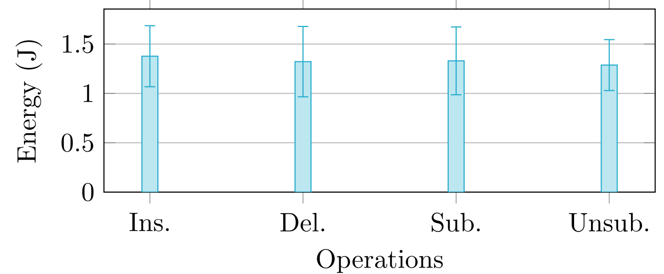


Fig. 21. Energy consumption in the cell maintenance.

not depend on the number of devices in the network, since we verified a marginal variance. Thus, we present these results as an average, independent of the number of devices in the network.

Issuing a request. Fig. 18 presents the energy consumption when issuing each operation. From this plot, we can verify that every operation uses very little energy. For instance, issuing an insert operation consumes around 2 Joule. However, it is not easy to understand the energy consumption in real terms for the average user. Thus, Fig. 19 displays a breakdown of the energy consumption of a device while executing a specific operation in a closed loop for *one minute*. This would represent a very intense scenario, however it will be useful to test the worst case in energy consumption. Standby mobile phones, i.e., only connected to the Wi-Fi router (without Internet access), consume around 61 Joule. When running the application, the battery consumption increases by 6 Joule to about 67 Joule; an increase caused by the periodic sending of cell management messages to neighboring nodes. The energy consumed by the different operations is mostly equivalent. On average, delete and (un)subscribe are the more energy friendly operations, consuming around 23 Joule, to a total of 90 Joule. On the other end, with the insert operation the total energy consumption is about 103 Joule.

Putting things more into perspective, in the Nexus 9 devices, 1% of battery corresponds to around 960 Joule. According to the presented data, scenarios with *continuous* and *intensive* use of the application for *10 minutes* (publishing photos, receiving notifications and downloading available photos) consumes roughly 950 Joules, which represents about 1% of the device's battery charge. A value we claim is quite reasonable for such intensive use.

Processing a request. Since we are talking about a collaborative and distributed system, nodes also work on behalf of each other, i.e., on behalf of the system. Naturally, processing an operation request requires a node to do some computations, and thus, spend some battery. Fig. 20 displays the energy costs involved in processing each type of operation. Looking at this plot, we can observe that energy consumption is similar among all the operations. On average, processing an operation request spends around 1.5 Joule. We argue such a low value is acceptable, even more so since work inside a cell is balanced among all the nodes in the cell. Even in a worst case scenario, by processing requests in a closed loop during 10 minutes, a node will spend, on average, about 830 Joule, which represents a consumption of less than 1% of a Nexus 9 total battery charge.

Cell maintenance. The THYME-DCS routing layer is based on a GHT (Section 6.6). In turn, the GHT is based on the notion of virtual nodes or cells (comprised by physical nodes). Thus, physical nodes inside a cell work on behalf of their cell (e.g., through active replication; Section 6.2.1). As such, the maintenance of a cell requires some computations and an increase in energy consumption of the mobile devices. When a node processes an operation request, all its cell neighbors will have to update their state accordingly. Fig. 21 shows the energy consumption of a node during the cell maintenance required for each operation. From this plot, we can verify that the energy consumption for cell maintenance is comparable to the cost of processing the operation request (presenting only a negligible decrease). This comes from that fact that when processing the specific request, these nodes do not have to send further messages (i.e., they only process the one they received). Similarly to the rationale followed before, a node processing the cell maintenance requests continuously during 10 minutes, on average would spend less than 800 Joules, which would mean an expenditure of 0.8% of the Nexus 9 total battery charge.

Total energy cost. Since mobile devices can be simultaneously issuing operations and processing operation requests, and still have to participate in their cells' maintenance, we have to account for these three energy components. Following the same worst case scenario (issuing and processing operations for 10 minutes in a closed loop), and summing these three components, we get

$$950 + 830 + 800 = 2580 \text{ Joule}$$

corresponding to around 2.7% of the Nexus 9 total battery charge. A value we argue is quite acceptable for such an *intensive* use of the application. Thus, to deplete 50% of the device's battery

charge, a client would need to keep this intensive use continuously for *more than three hours*. In a moderate usage scenario and for a device with a complete battery charge, we estimate a battery life of well over six hours.

11. Discussion

Thyme-ES presents a baseline. It has an external, centralized component where all the system's data is stored. Being a centralized server, it presents itself as a bottleneck and a single point of failure. Thus, this approach assumes it never fails, otherwise the service will become completely unavailable. If that assumption is not an issue, then Thyme-ES can be an option, but only for small scenarios. As our experiments show, increasing the number of nodes in this approach leads to an increase in operation latency and a decrease in operation success ratio, resulting from the central component being a bottleneck and all nodes compete for its resources. Note that in our experiments, the centralized component resided close to the client nodes. If that was not the case and it was located in the cloud, we would see much higher latencies.

Due to its flooding approach, Thyme-LS causes far more interferences than Thyme-DCS. This is exacerbated by the number of nodes in the system (Section 9.3.1). Churn is also a problem for Thyme-LS, because insertions are executed locally and there is no replication mechanism (Section 9.3.2). Thus, if a node fails, all the data it stored will become unavailable, representing some kind of partial failure because only that node's data is unavailable. Its flooding approach means that as the number of nodes increases, so does the amount of traffic and interferences. In summary, Thyme-LS is more suitable for smaller scenarios with no strong data availability requirements.

In turn, Thyme-DCS leverages geographic routing to employ replication, and location-aware data retrieval (Section 9.3.2). However, one-hop broadcast is unreliable by nature, thus the assumption that every node inside a cell has the same state needs to be relaxed (Section 9.3.3). Nonetheless, its cluster-based GHT deals well with the increase number of nodes in the system. So, Thyme-DCS is more suitable for larger scenarios with moderate mobility patterns and data availability requirements.

Regarding the real world experiments and our proof-of-concept Android application, results show adequate response times for interactive usage and low battery consumption. Yet, the work each node has to do on behalf of the system grows linearly with the amount of work delegated on the cell where they reside. This load can be reduced by using partial replication techniques, for instance, when the cell's population surpasses a given threshold. However, even with the use of full replication in cells, our experiments show that the application can be used during short and medium duration events with no risk of rapidly discharging the devices' battery.

In sum, these three approaches have very different characteristics. Which one is appropriate for a specific setting will depend on the conditions of the environment and the nature of the workload. Thus, we stress that Thyme-DCS is not always the approach of choice, but rather that under some conditions it is preferable. In fact, the perfect case is a system that embodies all of these approaches, and application developers can choose which to use according to the task at hand.

Overall, this time-aware reactive storage concept makes a fundamental *overhead shift*. Instead of requiring users to explicitly search for the available data, it allows users to register queries (with defined time boundaries), which are then notified as new relevant data is stored in the system—providing a reactive interaction model. As a consequence, the overhead from the stakeholders that actually benefit more from this approach – users requesting data – is reduced (compared with the explicit search approach), and is *moved* to the stakeholders that do not benefit directly from it – users that have the data and can provide it. This can work as an argument against this approach. However, we reckon that users usually do not mind sharing their resources just for a greater good (e.g., P2P file sharing), specially if they can also benefit from what the systems have to offer. Also, note that when a node obtains a data object, it becomes a new source for that same object (i.e., a passive replica). Thus, it also goes from one side to the other. That is, it goes to side of the stakeholders that contribute to the system (like a seeder in a P2P file sharing application).

Other compelling reasons are also the *volunteer computing* [64, 65] and the *crowdsourcing* [66,67] hypes. Volunteer computing uses computing resources (e.g., processing power, storage, etc.) donated by the general public to do distributed scientific computing. Systems like BOINC [68] have been proved and tested throughout the years, being used by numerous scientific projects, e.g., SETI@home [69]. Results have also shown the general public massively adheres to this kind of initiatives and is willing to share their computing resources for a "greater good" [69]. Crowdsourcing is a type of participative online activity in which an entity proposes to a group of individuals the voluntary undertaking of a task entailing mutual benefit [70]. This idea has been extensively used as a cost-effective way of harnessing the collective power of multiple individuals. Inclusively, it even changed the way of working of various sectors of the world's economy [66]. With all these aspects in mind, it makes sense to crowdsource the computing and storage power of a collection of nearby mobile devices to support a new generation of applications. Furthermore, people have shown to be receptive to the idea of harnessing the individual resources in order to make sense of the old motto "unity is strength".

12. Conclusion

In this article, we present the concept of *time-aware reactive storage*, that fuses the P/S paradigm with the storage substrate, and allows queries within a specific time scope. The insert operation of the storage substrate is merged with the publish operation of the P/S system, enabling applications to be notified as relevant data is generated and stored. Subscriptions allow propositional logic formulas as queries, and have a time frame defining when they are active. We also describe Thyme, a novel time-aware reactive storage system for pervasive edge computing environments. We detail two different approaches to Thyme: Thyme-LS follows a lightweight unstructured approach using local storage and query flooding, while Thyme-DCS employs a DCS approach using a storage substrate built over a GHT for wireless networks. In addition, we describe our implementation of the Thyme-DCS approach as an Android library and its use to develop a proof-of-concept photo sharing application. The innovative characteristics of Thyme offer a novel way for sharing and accessing data that has been previously stored, or is being generated in quasi-real-time, in a network of mobile devices.

The three parts of our evaluation are complementary to each other, showing different facets of our approaches. Overall, the evaluation shows that Thyme allows the notification and retrieval of relevant data with low overhead and latency. However, all the presented approaches display different behaviors and each may be best suited for scenarios with specific characteristics. In general, we show that the developed approaches exhibit a good performance and low energy consumption in the target environments (and under various conditions).

This work can be seen has a first step towards a data storage and dissemination system for a wide-area setting, like a campus

or a music festival. In this scenario, data will still be stored in the devices, and communication will mostly be device-to-device to offload it from the network infrastructure. There are however several open issues, of which we highlight the following. Non-contiguous spaces, such as the ones composed of multiple Wi-Fi access points: more sophisticated hashing functions and/or maybe the use of cloudlets may allow to cope with such environments. Rapidly state-changing cells: cells may be populated or not, being this state managed by the GHT. However, with high mobility, this state may change rapidly, leading to overheads due to the need of transferring data between devices, and ultimately causing some of this data to be lost, if there is no time to make the necessary backups. A hierarchical cell organization, or a partial replication approach may be possible directions. We also highlight as future work the integration of this approach with opportunistic infrastructure support [71], privacy and security concerns in this type of environments (mainly access control and trust), and tackling the issues raised by handling large data objects.

CRediT authorship contribution statement

João A. Silva: Conceptualization, Software, Validation, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Filipe Cerqueira:** Conceptualization, Software, Validation, Investigation, Data curation, Visualization. **Hervé Paulino:** Conceptualization, Software, Data curation, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition. **João M. Lourenço:** Conceptualization, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition. **João Leitão:** Conceptualization, Writing - review & editing, Funding acquisition. **Nuno Preguiça:** Conceptualization, Writing - review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by Fundação para a Ciência e a Tecnologia (FCT-MCTES) through project DeDuCe (PTDC/CCI-COM/32166/2017), NOVA LINCS UIDB/04516/2020, and grant SFRH/BD/99486/2014; and by the European Union through project LightKone (grant agreement n° 732505).

References

[1] Cisco, Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016—2021, Tech. Rep., Cisco, 2017.
[2] Cisco, Cisco Annual Internet Report (2018—2023), Tech. Rep., Cisco, 2020.
[3] Cisco, The Zettabyte Era: Trends and Analysis, Tech. Rep., Cisco, 2017.
[4] B. Manoj, A.H. Baker, Communication challenges in emergency response, Commun. ACM 50 (3) (2007) 51–53, http://dx.doi.org/10.1145/1226736.1226765, URL http://doi.acm.org/10.1145/1226736.1226765.
[5] DARPA, Creating a secure, private internet and cloud at the tactical edge, 2013, https://www.darpa.mil/news-events/2013-08-21, (Accessed: 10 May 2018).
[6] J. Erman, K. Ramakrishnan, Understanding the super-sized traffic of the super bowl, in: Proceedings of the 2013 Internet Measurement Conference, IMC '13, ACM, New York, NY, USA, 2013, pp. 353–360, http://dx.doi.org/10.1145/2504730.2504770, URL http://doi.acm.org/10.1145/2504730.2504770.
[7] Y. Yan, N.H. Tran, F.S. Bao, Gossiping along the path: A direction-biased routing scheme for wireless ad hoc networks, in: Proceedings of the 2015 IEEE Global Communications Conference, GLOBECOM '15, IEEE, 2015, pp. 1–6, http://dx.doi.org/10.1109/GLOCOM.2014.7417867.
[8] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: Vision and challenges, SIGCOMM Comput. Commun. Rev. 45 (5) (2015) 37–42, http://dx.doi.org/10.1145/2831347.2831354, URL http://doi.acm.org/10.1145/2831347.2831354.
[9] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet of Things J. 3 (5) (2016) 637–646, http://dx.doi.org/10.1109/JIOT.2016.2579198.
[10] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, M. Satyanarayanan, Quantifying the impact of edge computing on mobile applications, in: Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys '16, ACM, New York, NY, USA, 2016, pp. 5:1–5:8, http://dx.doi.org/10.1145/2967360.2967369, URL http://doi.acm.org/10.1145/2967360.2967369.
[11] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, M. Satyanarayanan, The impact of mobile multimedia applications on data center consolidation, in: Proceedings of the 2013 IEEE International Conference on Cloud Engineering, IC2E '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 166–176, http://dx.doi.org/10.1109/IC2E.2013.17.
[12] U. Drolia, N.D. Mickulicz, R. Gandhi, P. Narasimhan, Krowd: A key-value store for crowded venues, in: Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture, MobiArch '15, ACM, 2015, pp. 20–25, http://dx.doi.org/10.1145/2795381.2795388.
[13] J. Luo, J.-P. Hubaux, P.T. Eugster, PAN: Providing reliable storage in mobile ad hoc networks with probabilistic quorum systems, in: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '03, ACM, New York, NY, USA, 2003, pp. 1–12, http://dx.doi.org/10.1145/778415.778417, URL http://doi.acm.org/10.1145/778415.778417.
[14] X. Song, Y. Huang, Q. Zhou, F. Ye, Y. Yang, X. Li, Content centric peer data sharing in pervasive edge computing environments, in: K. Lee, L. Liu (Eds.), Proceedings of the 37th IEEE International Conference on Distributed Computing Systems, ICDCS '17, IEEE Computer Society, 2017, pp. 287–297, http://dx.doi.org/10.1109/ICDCS.2017.26.
[15] P.T. Eugster, P.A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv. 35 (2) (2003) 114–131, http://dx.doi.org/10.1145/857076.857078, URL http://doi.acm.org/10.1145/857076.857078.
[16] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, S. Shenker, GHT: A geographic hash table for data-centric storage, in: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, WSNA '02, ACM, New York, NY, USA, 2002, pp. 78–87, http://dx.doi.org/10.1145/570738.570750, URL http://doi.acm.org/10.1145/570738.570750.
[17] F. Araújo, L.s. Rodrigues, J. Kaiser, C. Liu, C. Mitidieri, CHR: A distributed hash table for wireless ad hoc networks, in: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS), ICDCSW '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 407–413, http://dx.doi.org/10.1109/ICDCSW.2005.48.
[18] G.F. Riley, T.R. Henderson, The ns-3 network simulator, in: K. Wehrle, M. Güneş, J. Gross (Eds.), Modeling and Tools for Network Simulation, Springer Berlin Heidelberg, 2010, pp. 15–34, http://dx.doi.org/10.1007/978-3-642-12331-3_2.
[19] G.P. Picco, A.L. Murphy, G.-C. Roman, LIME: Linda meets mobility, in: Proceedings of the 21st International Conference on Software Engineering, ICSE '99, ACM, New York, NY, USA, 1999, pp. 368–377, http://dx.doi.org/10.1145/302405.302659, URL http://doi.acm.org/10.1145/302405.302659.
[20] M. Mamei, F. Zambonelli, Programming pervasive and mobile computing applications: The TOTA approach, ACM Trans. Softw. Eng. Methodol. 18 (4) (2009) 15:1–15:56, http://dx.doi.org/10.1145/1538942.1538945, URL http://doi.acm.org/10.1145/1538942.1538945.
[21] J.A. Silva, H. Paulino, J.M. Lourenço, J. Leitão, N.M. Preguiça, Time-aware reactive storage in wireless edge environments, in: Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous '19, ACM, 2019, pp. 238–247, http://dx.doi.org/10.1145/3360774.3360828.
[22] F. Cerqueira, J.A. Silva, J.M. Lourenço, H. Paulino, Towards a persistent publish/subscribe system for networks of mobile devices, in: Proceedings of the 2nd Workshop on Middleware for Edge Clouds & Cloudlets, MECC '17, ACM, New York, NY, USA, 2017, pp. 2:1–2:6, http://dx.doi.org/10.1145/3152360.3152362, URL http://doi.acm.org/10.1145/3152360.3152362.
[23] M. Cilia, L. Fiege, C. Haul, A. Zeidler, A.P. Buchmann, Looking into the past: Enhancing mobile publish/subscribe middleware, in: Proceedings of the International Workshop on Distributed Event-Based Systems, DEBS '03, ACM, New York, NY, USA, 2003, pp. 1–8, http://dx.doi.org/10.1145/966618.966631, URL http://doi.acm.org/10.1145/966618.966631.
[24] L. Vargas, J. Bacon, K. Moody, Integrating databases with publish/subscribe, in: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS), ICDCSW '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 392–397, http://dx.doi.org/10.1109/ICDCSW.2005.79.

[25] J. Kreps, N. Narkhede, J. Rao, et al., Kafka: A distributed messaging system for log processing, in: Proceedings of the Networking Meets Databases Workshop, NetDB '11, 2011, pp. 1–7.

[26] G. Mühl, Large-Scale Content Based Publish, Subscribe Systems (Ph.D. thesis), Germany, 2002, URL http://elib.tu-darmstadt.de/diss/000274.

[27] A.R. Khakpour, I. Demeure, Chapar: A persistent overlay event system for MANETs, Mob. Netw. Appl. 15 (6) (2010) 866–875, http://dx.doi.org/10.1007/s11036-010-0238-6.

[28] T.H. Clausen, P. Jacquet, Optimized link state routing protocol (OLSR), RFC 3626 (2003) 1–75, http://dx.doi.org/10.17487/RFC3626.

[29] J.A. Silva, R. Monteiro, H. Paulino, J.M. Lourenço, Ephemeral data storage for networks of hand-held devices, in: Proceedings of the 2016 IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA '16, IEEE, 2016, pp. 1106–1113, http://dx.doi.org/10.1109/TrustCom.2016.0182.

[30] K. Thilakarathna, H. Petander, J. Mestre, A. Seneviratne, Mobitribe: Cost efficient distributed user generated content sharing on smartphones, IEEE Trans. Mobile Comput. 13 (9) (2014) 2058–2070.

[31] R.K. Panta, R. Jana, F. Cheng, Y.F.R. Chen, V.A. Vaishampayan, Phoenix: Storage using an autonomous mobile infrastructure, IEEE Trans. Parallel Distrib. Syst. 24 (9) (2013) 1863–1873, http://dx.doi.org/10.1109/TPDS.2013.84.

[32] I. Lombera, L.E. Moser, P.M. Melliar-Smith, Y.-T. Chuang, Mobile ad-hoc search and retrieval in the iTrust over Wi-Fi Direct network, in: Proceedings of the 9th International Conference on Wireless and Mobile Communications, 2013, pp. 251–258.

[33] G. Xylomenos, C.N. Ververidis, V.A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K.V. Katsaros, G.C. Polyzos, A survey of information-centric networking research, IEEE Commun. Surv. Tutorials 16 (2) (2014) 1024–1049, http://dx.doi.org/10.1109/SURV.2013.070813.00063.

[34] M. Demmer, B. Du, E. Brewer, Tierstore: A distributed filesystem for challenged networks in developing regions, in: Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST '08, USENIX Association, Berkeley, CA, USA, 2008, pp. 3:1–3:14, URL http://dl.acm.org/citation.cfm?id=1364813.1364816.

[35] J. Su, J. Scott, P. Hui, J. Crowcroft, E. De Lara, C. Diot, A. Goel, M.H. Lim, E. Upton, Haggle: Seamless networking for mobile applications, in: Proceedings of the 9th International Conference on Ubiquitous Computing, UbiComp '07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 391–408, URL http://dl.acm.org/citation.cfm?id=1771592.1771615.

[36] N.P. Palma, V. Mancuso, M.A. Marsan, Infrastructureless pervasive information sharing with COTS devices and software, in: 19th IEEE International Symposium on "a World of Wireless, Mobile and Multimedia Networks", WoWMoM '18, IEEE Computer Society, 2018, pp. 1–9, http://dx.doi.org/10.1109/WoWMoM.2018.8449733.

[37] D. Gelernter, Generative communication in linda, ACM Trans. Program. Lang. Syst. 7 (1) (1985) 80–112, http://dx.doi.org/10.1145/2363.2433, URL http://doi.acm.org/10.1145/2363.2433.

[38] A. Omicini, F. Zambonelli, Tuple centres for the coordination of internet agents, in: Proceedings of the 1999 ACM Symposium on Applied Computing, SAC '99, ACM, New York, NY, USA, 1999, pp. 183–190, http://dx.doi.org/10.1145/298151.298231, URL http://doi.acm.org/10.1145/298151.298231.

[39] LiveQoS, Superbeam, 2017, https://superbe.am/, (Accessed: 13 August 2020).

[40] Anmobi Inc., Xender, 2014, http://www.xender.com/, (Accessed: 13 August 2020).

[41] Open Garden Inc., Firechat, 2017, https://www.opengarden.com/firechat.html, (Accessed: 27 April 2018).

[42] Briar Project, Briar, 2017, https://briarproject.org/, (Accessed: 27 September 2020).

[43] goTenna Inc., Gotenna mesh, 2017, https://www.gotenna.com/, (Accessed: 05 October 2020).

[44] B. Oki, M. Pfluegl, A. Siegel, D. Skeen, The information bus: An architecture for extensible distributed systems, SIGOPS Oper. Syst. Rev. 27 (5) (1993) 58–68, http://dx.doi.org/10.1145/173668.168624, URL http://doi.acm.org/10.1145/173668.168624.

[45] M.K. Aguilera, R.E. Strom, D.C. Sturman, M. Astley, T.D. Chandra, Matching events in a content-based subscription system, in: Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '99, ACM, New York, NY, USA, 1999, pp. 53–61, http://dx.doi.org/10.1145/301308.301326, URL http://doi.acm.org/10.1145/301308.301326.

[46] K. Seada, C. Perkins, Social Networks: the Killer App for Wireless Ad Hoc Networks? Tech. Rep., Nokia Research Centre, 2006.

[47] Yinzcam Inc., Yinzcam, 2019, http://www.yinzcam.com/, (Accessed: 13 August 2020).

[48] A. Teófilo, D. Remédios, J.M. Lourenço, H. Paulino, GOCRGO and GOGO: two minimal communication topologies for wifi-direct multi-group networking, in: Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous '17, ACM, 2017, pp. 232–241, http://dx.doi.org/10.1145/3144457.3144481.

[49] A. Rai, K.K. Chintalapudi, V.N. Padmanabhan, R. Sen, Zee: Zero-effort crowdsourcing for indoor localization, in: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, Mobicom '12, ACM, New York, NY, USA, 2012, pp. 293–304, http://dx.doi.org/10.1145/2348543.2348580, URL http://doi.acm.org/10.1145/2348543.2348580.

[50] C.E. Perkins, P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, in: Proceedings of the Conference on Communications Architectures, Protocols and Applications, SIGCOMM '94, ACM, New York, NY, USA, 1994, pp. 234–244, http://dx.doi.org/10.1145/190314.190336, URL http://doi.acm.org/10.1145/190314.190336.

[51] J. Leitão, L. Rodrigues, Overnesia: A resilient overlay network for virtual super-peers, in: Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems, SRDS '14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 281–290, http://dx.doi.org/10.1109/SRDS.2014.40.

[52] J.H. Saltzer, D.P. Reed, D.D. Clark, End-to-end arguments in system design, ACM Trans. Comput. Syst. 2 (4) (1984) 277–288, http://dx.doi.org/10.1145/357401.357402, URL http://doi.acm.org/10.1145/357401.357402.

[53] G. Banavar, T.D. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, D.C. Sturman, An efficient multicast protocol for content-based publish-subscribe systems, in: Proceedings of the 19th International Conference on Distributed Computing Systems, ICDCS '99, IEEE Computer Society, 1999, pp. 262–272, http://dx.doi.org/10.1109/ICDCS.1999.776528.

[54] A. Carzaniga, D.S. Rosenblum, A.L. Wolf, Design and evaluation of a wide-area event notification service, ACM Trans. Comput. Syst. 19 (3) (2001) 332–383, http://dx.doi.org/10.1145/380749.380767.

[55] M. Castro, P. Druschel, A. Kermarrec, A.I.T. Rowstron, Scribe: a large-scale and decentralized application-level multicast infrastructure, IEEE J. Sel. Areas Commun. 20 (8) (2002) 1489–1499, http://dx.doi.org/10.1109/JSAC.2002.803069.

[56] P.R. Pietzuch, J. Bacon, Hermes: A distributed event-based middleware architecture, in: Proceedings of the 22nd International Conference on Distributed Computing Systems, Workshops ICDCSW '02, IEEE Computer Society, 2002, pp. 611–618, http://dx.doi.org/10.1109/ICDCSW.2002.1030837.

[57] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, second ed., Cambridge University Press, 2002, http://dx.doi.org/10.1017/CBO9780511809088.

[58] T. Zahn, J. Schiller, MADPastry: A DHT substrate for practicably sized MANETs, in: Proceedings of the 5th Workshop on Applications and Services in Wireless Networks, ASWN '05, 2005.

[59] J. Paiva, J. Leitão, L.E.T. Rodrigues, Rollerchain: A DHT for efficient replication, in: Proceedings of the 12th IEEE International Symposium on Network Computing and Applications, NCA '13, IEEE Computer Society, 2013, pp. 17–24, http://dx.doi.org/10.1109/NCA.2013.29.

[60] K. Seada, A. Helmy, Rendezvous regions: a scalable architecture for service location and data-centric storage in large-scale wireless networks, in: Proceedings of the 18th International Parallel and Distributed Processing Symposium, IPDPS '04, IEEE Computer Society, 2004, http://dx.doi.org/10.1109/IPDPS.2004.1303252.

[61] B. Karp, H.T. Kung, GPSR: Greedy perimeter stateless routing for wireless networks, in: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, MobiCom '00, ACM, New York, NY, USA, 2000, pp. 243–254, http://dx.doi.org/10.1145/345910.345953, URL http://doi.acm.org/10.1145/345910.345953.

[62] J.a. Rodrigues, E.R.B. Marques, L.M.B. Lopes, F. Silva, Towards a middleware for mobile edge-cloud applications, in: Proceedings of the 2nd Workshop on Middleware for Edge Clouds & Cloudlets, MECC '17, ACM, New York, NY, USA, 2017, pp. 1:1–1:6, http://dx.doi.org/10.1145/3152360.3152361, URL http://doi.acm.org/10.1145/3152360.3152361.

[63] A.N. Bessani, J. Sousa, E.A.P. Alchieri, State machine replication for the masses with BFT-SMART, in: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN '14, IEEE Computer Society, 2014, pp. 355–362, http://dx.doi.org/10.1109/DSN.2014.43.

[64] L.F.G. Sarmenta, Bayanihan: Web-based volunteer computing using java, in: Proceedings of the Second International Conference on Worldwide Computing and Its Applications, WWCA '98, Springer-Verlag, London, UK, UK, 1998, pp. 444–461, URL http://dl.acm.org/citation.cfm?id=645966.674584.

[65] D.P. Anderson, G. Fedak, The computational and storage potential of volunteer computing, in: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, CCGRID '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 73–80, http://dx.doi.org/10.1109/CCGRID.2006.101.

[66] J. Howe, The rise of crowdsourcing, Wired Mag. 14 (6) (2006) 1–4.
[67] A. Doan, R. Ramakrishnan, A.Y. Halevy, Crowdsourcing systems on the world-wide web, Commun. ACM 54 (4) (2011) 86–96, http://dx.doi.org/10.1145/1924421.1924442, URL http://doi.acm.org/10.1145/1924421.1924442.
[68] D.P. Anderson, BOINC: A system for public-resource computing and storage, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 4–10, http://dx.doi.org/10.1109/GRID.2004.14.
[69] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, SETI@Home: An experiment in public-resource computing, Commun. ACM 45 (11) (2002) 56–61, http://dx.doi.org/10.1145/581571.581573, URL http://doi.acm.org/10.1145/581571.581573.
[70] E. Estellés-Arolas, F.G. Ladrón-De-Guevara, Towards an integrated crowdsourcing definition, J. Inf. Sci. 38 (2) (2012) 189–200, http://dx.doi.org/10.1177/0165551512437638.
[71] J.A. Silva, J. Leitão, N. Preguiça, J.M. Lourenço, H. Paulino, Towards the opportunistic combination of mobile ad-hoc networks with infrastructure access, in: Proceedings of the 1st Workshop on Middleware for Edge Clouds & Cloudlets, MECC '16, ACM, New York, NY, USA, 2016, pp. 3:1–3:6, http://dx.doi.org/10.1145/3017116.3022873, URL http://doi.acm.org/10.1145/3017116.3022873.

João A. Silva received his B.Sc. and M.Sc. degrees in computer engineering from the NOVA School of Science and Technology, NOVA University Lisbon (FCT/UNL), in 2011 and 2013, respectively. He is currently working towards his Ph.D. degree at FCT/UNL, and is a student researcher at the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS). His research interests include edge computing, and data storage and dissemination, with special emphasis on mobile and wireless environments.

Filipe Cerqueira received his M.Sc. degree in computer engineering from the NOVA School of Science and Technology, NOVA University Lisbon (FCT/UNL), in 2017. He is currently working as a Software Engineer at Axians. His research interests include publish/subscribe systems and data persistence in mobile environments.

Hervé Paulino, Ph.D., is an Associate Professor at the Computer Science Department of the NOVA University Lisbon, and a member of the NOVA LINCS research center. He received his Ph.D. in computer science from the NOVA University Lisbon in 2006, in the area of mobile agent computing. Currently, his research interests center on availability in large-scale systems (with particular focus on edge systems) and on the parallel programming of heterogeneous systems.

João M. Lourenço is an Associate Professor at the Computer Science Department of the NOVA School of Science and Technology of NOVA University Lisbon, and a founding member of the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS). He received his Ph.D. from the NOVA University Lisbon in 2004, with a thesis on debugging of distributed programs. Currently his primary research interests include in-memory data management for parallel and large-scale computing systems, testing and debugging of concurrent programs, and edge/fog computing.

João Leitão is an Assistant Professor at the Computer Science Department of the NOVA School of Science and Technology of NOVA University Lisbon (FCT/UNL), and a member of the NOVA LINCS research center. He received his Ph.D. in information systems and computer engineering from the Instituto Superior Técnico (IST/UL), in 2012. His research interests are focused on the design of fault-tolerant and efficient large-scale systems, with particular interest in geo-distributed, cloud computing, and peer-to-peer systems. He is a member of the ACM and IEEE.

Nuno Preguiça is an Associate Professor with Habilitation at the Computer Science Department of the NOVA School of Science and Technology of NOVA University Lisbon (FCT/UNL), and leads the Computer Systems group at the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS). The broad aim of his research is to allow efficient and correct data sharing among geo-distributed users. He has participated in a number of national and EU projects. He co-invented CRDTs and received a Google Research Award in 2009 for his work on solutions for cloud data management.