Computer Networks 107 (2016) 178–193

Contents lists available at ScienceDirect

Computer Networks

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

A middleware for opportunistic content distribution

Ólafur Helgason, Sylvia T. Kouyoumdjieva*, Ljubica Pajević, Emre A. Yavuz, Gunnar Karlsson

School of Electrical Engineering and ACCESS Linnaeus Center, KTH Royal Institute of Technology, Stockholm, Sweden

ARTICLE INFO

Article history:
Received 27 August 2015
Revised 13 April 2016
Accepted 31 May 2016
Available online 1 June 2016

Keywords:
Opportunistic communication
Middleware
Wireless ad-hoc networking

ABSTRACT

In this work we present a middleware architecture for a mobile peer-to-peer content distribution system. Our architecture allows wireless content dissemination between mobile nodes without relying on infrastructure support. In addition, it supports the dissemination of contents between the wireless ad-hoc domain and the wired Internet. In the ad-hoc domain, contents are exchanged opportunistically when nodes are in communication range. Applications access the service of our platform through a publish/subscribe interface and therefore do not have to deal with low-level opportunistic networking issues such as beaconing, service discovery, connection setup and matching or soliciting contents. Our middleware consists of three key components. A *content structure* that facilitates dividing contents into logical topics and allows efficient matching of content lookups and downloading under sporadic node connectivity. A *solicitation protocol* that allows nodes to solicit content meta-information in order to discover contents available at a neighboring node and to download content entries disjointedly from different nodes. An *API* that allows applications to access the system services through a publish/subscribe interface. In this work we present the design and implementation of our middleware and describe a set of applications that use the services provided by our middleware. We also assess the performance of the system using our Android implementation as well as a simulation implementation for large-scale evaluation.

1. Introduction

Multimedia usage has spread from personal computers and Internet into people's palms as mobile phones have become smart platforms for digital content. Due to the popularity of these devices, contents are frequently being produced and accessed by users on the move. As a result, cellular data traffic is growing at a rapid pace and it is predicted to double roughly every year in the near future [1]. Matching this growth with a corresponding capacity increase in the wireless infrastructure networks is a significant challenge. This evolution calls for new architectures for disseminating contents to mobile users.

Our work considers content-centric networking in the context of mobile wireless networks. The main focus is on opportunistic distribution of contents where mobile nodes exchange content items when in direct communication range. This communication mode enables dissemination of contents between mobile nodes without relying on infrastructure, which can be beneficial in cases when infrastructure may be: (1) absent as in rural or developing regions; (2) overloaded due to the aforementioned traffic demand; (3) broken as in the case of a natural disaster; (4) unavailable or expensive to use due to the data plan subscription of the user, for instance while roaming; or (5) censored or limited to certain services or contents.

In this paper we propose *Fram*[1], a middleware architecture that allows applications on mobile devices to share contents. The devices can utilize connections with access points when in range, and may distribute contents opportunistically among mobile nodes otherwise. Contents are structured to facilitate efficient lookup and downloading under disruptive node connectivity. This requires rethinking of networking basics: While existing network architectures focus on addressing *nodes* and forwarding of packets between such nodes, our system aims at addressing and disseminating *contents*. Hence, instead of relying on end-to-end semantics between a requesting client and a provider, our dissemination mechanism relies on opportunistic content forwarding without any routing substrate; contents are routed implicitly through the combination of a receiver-driven solicitation protocol and the actual node mobility. As a result, sophisticated multi-hop communication protocols, where for example routes have to be built up and maintained, are not necessary. Consequently, the architecture does not assume a traditional network layer.

* Corresponding author.
E-mail addresses: olafurr@kth.se (Ó. Helgason), stkou@kth.se (S.T. Kouyoumdjieva), ljubica@kth.se (L. Pajević), emreya@kth.se (E.A. Yavuz), gk@kth.se (G. Karlsson).

[1] *Forward* in Swedish.

Despite years of research there are not many mobile ad-hoc systems that have been deployed and many of the devised protocols and mechanisms have not seen practical use. We believe that the end-to-end connectivity approach, adopted from wired networks into traditional mobile ad-hoc networks (MANETs), is one of the main reasons for lack of success and that the looser connectivity paradigm advocated by opportunistic and delay-tolerant networks has greater potential to succeed. Embracing mobility as an information carrier and incorporating connectivity disruptions into the system design, as opposed to treating it as an exceptional error state, avoids much of the complexity required for trying to maintain end-to-end communication paths in a mobile environment. We acknowledge that some types of applications may be difficult to support with opportunistic networking, in particular applications with tight delay constraints such as real-time audio/video conversations and streaming.

With its content-based routing and addressing, our system can be seen as a publish/subscribe system that decouples the communicating entities from the contents and thus it inherently allows asynchronous communication and leverages looser delay constraints. With respect to content availability, scaling comes naturally as popular contents are likely to be available at many nodes in the system. It is particularly well-suited for data-centric applications and distributing contents that are popular and tolerant to modest delay such as conducting local quizzes or surveys, audio or video broadcasting and on-site networking or dating-profile exchange. The proposed architecture also promotes openness: anybody who wishes to publish or retrieve contents is free to do so. We therefore believe that the system has the necessary features to stimulate organic user growth, which has previously led to the success of many systems and services. The architecture is inspired by podcasting and BitTorrent. Our operating scenario is however radically different than what is experienced on the wired network since our architecture has to cope with sporadic contacts, none or limited end-to-end connectivity and short contact durations. Although a previous feasibility study for content distribution among pedestrians in such environments shows promising results [2,3], there are still many challenges that need to be addressed and solved by an actual system design and this is the focus of the current work.

The rest of the paper is organized as follows. In Section 2 we describe the design of our system. Section 3 describes our middleware implementation for the Android platform along with a set of implemented applications. In Section 4 we evaluate the performance of our system using both the Android implementation and a simulator. Section 5 discusses related work and Section 6 summarizes our findings and concludes the paper.

2. System design

In this section we present the design of our system. The system supports content distribution in a wireless ad-hoc network with opportunistic node contacts, as shown in Fig. 1. Contents can be generated by nodes in the Internet domain as well as by mobile devices in the ad-hoc domain. The Internet and ad-hoc domains are linked by gateways that assist in disseminating contents between domains and perform any necessary translations or proxy services. The focus of our work is on the wireless ad-hoc domain but we emphasize that our design allows for seamless distribution of contents between the wired Internet and the wireless ad-hoc domain.

2.1. Service overview

Our design imposes a hierarchical structure on contents based on the publish/subscribe paradigm [4]. Nodes publish *entries* on *feed* channels. A *feed* channel logically groups together related contents and it consists of a number of *entries* that contain, among other fields, the actual data object of interest, which we refer to as an *enclosure*. Nodes express interest in feeds by subscribing to them and the system then implements the delivery of published entries to feed subscribers.

In the ad-hoc domain content disseminates via a solicitation protocol in which a node solicits entries for one or more feeds from a peer (a peer node can be either a mobile device or a proxy gateway, for instance an access point, that acts as a bridge between the ad-hoc and Internet domains). Feeds and entries contain a number of meta-information fields such as a globally unique ID, author, date and time of last update. The meta-information is primarily used to facilitate searching, filtering and unique matching of contents. The content structure in the system thus allows for ease of searching and a higher hit rate of content queries than if they were made for individual unstructured contents.

The publish/subscribe paradigm is well-suited for content-centric networking and it has characteristics that are highly attractive for opportunistic networks with intermittent connectivity. In particular, it decouples publishers and subscribers such that a subscriber neither needs to know who the publisher of the content is, nor connect to it. Successful delivery of content is not dependent on the original publisher being up and running; as long as the content is available in the network, either at other subscribers or at caching nodes, a subscriber has a chance of obtaining it. This decoupling also facilitates an asynchronous communication model with loose delay constraints that helps to cope with the dynamic network topology. Finally, this form of mediated publish/subscribe does not rely on particular nodes, e.g. rendezvous nodes, which facilitates a decentralized implementation that is mandatory in the wireless ad-hoc domain and highly desirable in the Internet domain for performance, scalability and fault-tolerance.

2.2. Content structure

Content addressing and organization adopts and extends the content structure of the Atom Syndication Format [5]. This format has primarily been used for publishing web-feeds and podcasts on the Internet. The content structure is however quite generic and allows for more use-cases than what has commonly been implemented. It also maps nicely to the publish/subscribe semantics of our system. We note that the content structure is not bound only to the Atom format, and can easily be migrated to other formats such as JavaScript Object Notation (JSON). The listings in Fig. 2 show a sample content structure presented in the Atom and JSON formats. We further provide a detailed description of the content structure with respect to the Atom format. Contents are grouped into *feeds*. A feed is an unlimited series of *entries* that contain the actual data objects of interest. Each feed can have multiple entries published at different times by different nodes. Both feeds and entries have associated meta-data. Each feed must contain a permanent globally unique ID assigned by the creator, a title and a timestamp that indicates the latest activity on the feed, i.e. the most recent update of any locally discovered entry. A feed can also contain other optional meta-information such as author, subtitle and category. Similarly, each entry must contain a globally unique ID, a title and a release timestamp. The feed and entry identifiers are URIs (Uniform Resource Identifiers) which facilitates flexible naming and allows a variety of existing naming mechanisms to be used or new ones to be applied. An entry can optionally have a range of other elements including one or more *enclosures*. An enclosure is a single file attachment and would typically be an audio, video, or text file. To transfer enclosures efficiently over the opportunistic contacts, we divide the enclosures into *chunks*, small data units of fixed size, which can be exchanged with higher probability

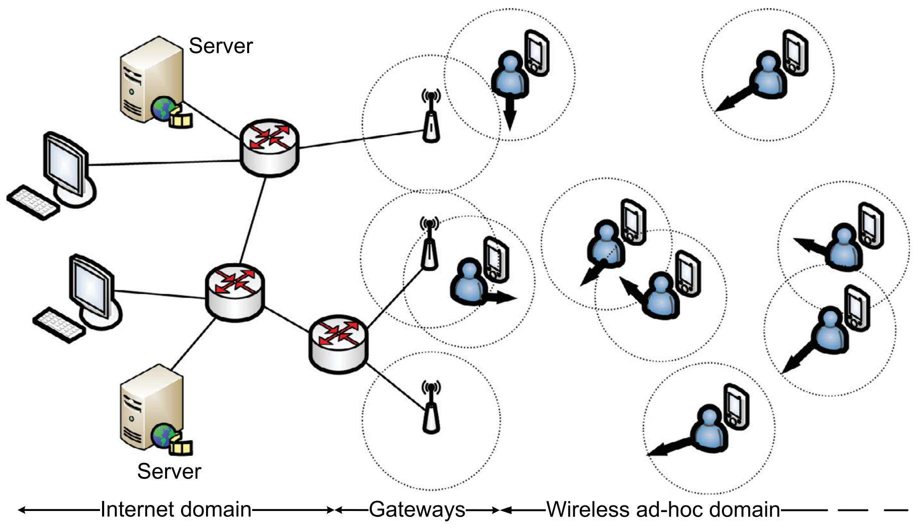


Fig. 1. The system supports content distribution in the fixed Internet and in a wireless ad-hoc network with opportunistic node contacts.

```
<feed>
   <title>Adhocpoets.org</title>
   <id>feed:adhocpoets.org</id>
   <updated>2015-06-01T18:30:02Z</updated>
   <entry>
      <title>Steinn Steinarr - Time and the Water</title>
      <id>tag:steinn.steinarr@adhocpoets.org,
         2015-06-01:Time_and_the_Water.mp3</id>
      <updated>2015-06-01T18:30:02Z</updated>
      <link rel="enclosure" title="Time_and_the_Water.mp3"
         type="audio/mpeg" length="1378129" />
   </entry>
</feed>
```

```
{
  "feed" : {
    "name" : "Adhocpoets.org",
    "uri" : "http://adhocpoets.org/feed",
    "updated" : "2015-06-01T18:30:02Z",
    "entry" : {
      "name" : "Steinn Steinarr - Time and the Water",
      "uri" : "http://adhocpoets.org/feed/entry",
      "updated" : "2015-06-01T18:30:02Z",
      "enclosure" : {
        "name" : "Time_and_the_Water.mp3",
        "uri" : "http://adhocpoets.org/feed/entry/tw",
        "type" : "audio/mpeg",
        "length" : 1378129
      }
    }
  }
}
```

Fig. 2. An example of a data structure based on the Atom format (above) and the JSON format (below). Contents are grouped into *feeds*, and each feed consists of one or more *entries* that contain actual data objects of interest. An entry may in turn include a file attachment in the form of an *enclosure*.

during a single radio contact of limited duration. Chunks allow the downloading of a previously incompletely downloaded entry to be resumed from the same node or any other node that has the entry or parts of it. They are indexed starting from 1 and we extend the Atom format to include chunk information for enclosures. If a chunk is only partially received from a peer (e.g. due to lost connection), it is discarded.

2.3. Middleware for wireless ad-hoc domain

Nodes in the wireless ad-hoc domain are generally mobile devices, equipped with a radio that can operate in ad-hoc mode to establish contacts with other mobile nodes in range. When two nodes are within communication range they associate and exchange contents according to a solicitation protocol. Thus, contents spread opportunistically from node to node in an epidemic manner with susceptibility given by the popularity of each feed.

In this section we present *Fram*, a middleware for opportunistic content distribution. We first outline the general architecture, and then describe in detail the main components of our design: the API module, the synchronization and discovery module, and the transport module. Finally, we discuss issues related to security, caching and energy consumption.

Architecture overview

In our design, the mechanisms for the opportunistic content distribution system are implemented in a middleware in the mobile nodes. The middleware allows different types of applications to be implemented on top of the basic opportunistic content distribution service and its purpose is therefore to abstract away the complexities of the underlying system from the applications. In particular, the middleware is responsible for discovering neighbors, while coping with sporadic contacts with limited contact durations. Moreover, it implements the matching, downloading and storing of content entries. In other words, the middleware implements a session layer that defines the content structure and hides networking details from applications.

Fig. 3 illustrates our middleware design and the main system components. Applications access the services of the middleware through the Application Programming Interface (API) that it exports. The API implicitly defines the content structure for applications, and it allows them to publish/subscribe to content feeds. A set of modules implement the API, the service discovery and the solicitation protocol. The middleware assumes an underlying transport layer that preserves message boundaries, provides flow control and reliable process-to-process communication. The system design does thus not assume a traditional network layer with point-to-point unicast routing. Contents disseminate in the network by means of node mobility, sharing of local contents and a receiver-driven solicitation protocol. Messages are delivered on a best-effort basis without guarantee that entries on a particular feed will be delivered in an ordered manner to all receivers.

The architecture also contains a convergence sub-layer for cross-layer interaction, particularly with the underlying radio link.

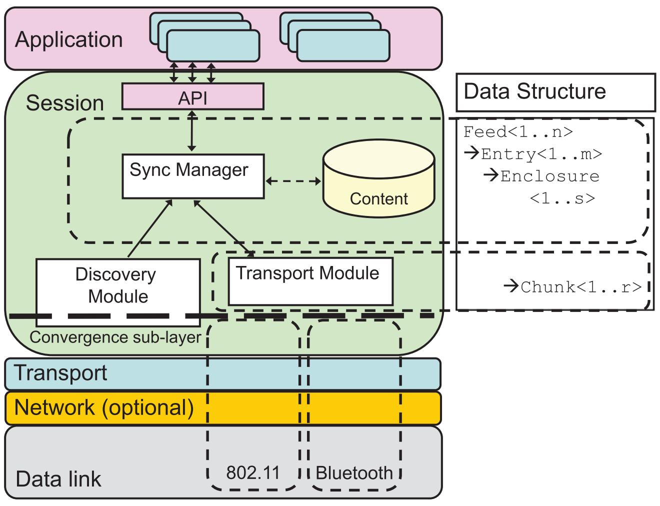


Fig. 3. The *Fram* middleware design. Applications access the middleware through an API, and the middleware implements service discovery, content discovery, solicitation and storage in corresponding modules.

Thus, the session layer architecture abstracts most of the details of the underlying radio and hides the heterogeneity of the networks away from the applications. This allows the middleware to operate on top of different radio technologies, from Wi-Fi and Bluetooth Low Energy currently available in most modern mobile devices, to modifications of 802.11 such as WLAN-Opp [6] and upcoming technologies such as Wi-Fi Aware and LTE-Direct intended to enable proximity-based services.

Lastly, the architecture assumes that each node shares content available to the applications on its device, and it specifies a protocol and mechanisms for efficient downloading of this content between nodes in the ad-hoc domain. The system does not mandate or specify a particular content caching or forwarding mechanism in addition to the interest-driven solicitations given by the application subscriptions. The content structure and solicitation protocol specified in our design are however general such that applications can implement their own caching or replication strategies, for example by discovering and subscribing to feeds that are not necessarily of interest to the local user. This could be implemented entirely using our current publish/subscribe interface. We address this issue later in Section 4.2.

Application programming interface

The API module implements the programming interface that applications use to access the services of the middleware. The API of our system is inspired by the Java Message Service (JMS) publish/subscribe API [7]. JMS was however designed for wired networks where dedicated brokers implement message delivery. The discovery of feeds also relies on a centralized directory service. In the ad-hoc domain, central servers for performing these functions are not available. Instead, both resource discovery and message distribution are performed distributively with servers being replaced by *nodes*. Thus, in addition to standard publish/subscribe/notify functions, the API needs to provide functions that allow applications to discover and create new feeds. Note that the API does not implement mechanisms to deal with potential creation of duplicate feeds by different applications. If two nodes were to create a feed with the same ID, this could lead to conflicts if the feed creators used different meta-data values. In the rest of the paper however we assume that each feed is associated with a unique creator.

Synchronization and discovery

The *synchronization manager* module processes contents from applications and solicits contents on behalf of applications. If the local content database contains data that matches a subscription, the content is delivered immediately to the application.

The *discovery* module is responsible for both neighbor and service discovery; i.e., it discovers neighboring nodes that are running the service and decides which of these are feasible to associate with. Each node advertises its existence to its neighbors with *beacon notifications* that contain the following information:

- Node identifier: A URI that uniquely identifies the node.
- Content revision number: A revision counter for the local content database. The revision number of a node is incremented whenever new content is added to the database. This helps peers to determine if re-synchronization might be beneficial in case nodes remain in range for longer durations or if they meet again after some time. In particular, two nodes only need to re-associate if at least one of them has obtained new contents since they last associated.
- Feeds Bloom filter: The list of local feeds that contain one or more entries in the form of a Bloom filter. A remote node can compare the feeds of its subscriptions to the Bloom filter to deduce if the local node has any contents of interest and therefore decide whether it wants to associate or not.

Bloom filters [8] are space-efficient data structures that provide a set-like representation of elements, requiring only a fraction of the space needed for a corresponding set with the actual elements. They trade space for accuracy since false positives can occur with some probability; a membership test returns a value of true but the element is not a member of the corresponding set. False negatives are however not possible. After receiving the Bloom filter, the node tests the IDs of its subscribed feeds against the filter. Occasionally, a false positive will result in a request for a non-existing content item. This is however not a serious issue compared to the benefits of using Bloom filters: They allow us to include a large amount of information about available contents in a possibly single, small node-advertisement message.

How beacon notifications are implemented and transmitted to other nodes may depend on the underlying radio. Therefore the discovery module is split across the main session layer and the convergence sub-layer. Typically, one would send out periodic beacons including the notification. However, if new content is added to the database of a node, a beacon notification should be sent out immediately, regardless of the beacon period. This would facilitate faster spreading of contents.

Transport module

The transport module performs session management and implements a request-reply protocol to download and discover available contents at a peer. Protocol messages are in XML format with the `message` element being the kernel of a protocol message. A protocol message has a single `node-id` element containing the ID of the message source and each message has a unique element that determines its type, given by one of the following message types: `request`, `reply` and `reject`. All other elements of a protocol message are child entries for the header fields associated with the message type.

When a peer is discovered by the discovery module, the transport module is notified and it sends a `request` message to initiate a unilateral session for downloading. The request contains either a query for a particular feed entry, or meta-data to check content availability. The peer sends a `reply` message, that answers the query establishing the session. Each download session

```
<message version="0.1">
  <node-id>olafur.helgason@ee.kth.se:1</node-id>
  <request>
    <entry>
      <id>tag:steinn.steinarr@adhocpoets.org,
          2015-06-01:Time_and_the_Water.mp3</id>
      <link rel="enclosure" type="audio/mpeg"
            ns:chunks="2-5"/>
    </entry>
  </request>
</message>
```

Listing 1. An example `request` message.

thus consists of a client node sending `request` messages and a server node sending `reply` messages (or `reject` if the server cannot accommodate the request). The server is stateless with each reply message being independent of any previous requests. Processing a request only consists of verifying that the requested contents or meta-data exist, followed by delivery.

Content solicitation in our system is entirely pull-based. At the client, a typical session alternates between *discovery* and *download* states. In the discovery state, the client node queries the server for meta-data whereas it downloads contents that match the subscriptions of applications in the download state. With this approach, each node has full control of the contents it downloads and decisions are based only on the client state with the server being stateless.

In general, a node can have multiple active sessions simultaneously, either as a client (when it is downloading) or server (when it is uploading). Note that the system does not explicitly enforce any mechanism to share download time between sessions; we simply rely on the mechanisms of the MAC layer to share the radio channel fairly. Ungraceful session termination (e.g. when nodes move out of range) is handled by a soft-state timer; if there is no activity from the peer for a certain time, the session is closed and any allocated resources are freed up.

A `request` message is used both for downloading and discovery of contents. Discovering previously known feeds or entries that may be available at a peer node is done efficiently using Bloom filters. Nodes maintain a Bloom filter with the IDs of available feeds, in addition to a Bloom filter for each feed that includes the IDs of available entries. When a node receives a `request` with an empty XML `Bloom` element, it delivers the corresponding Bloom filter in a `reply` message. After receiving the filter, the client node tests the IDs of its desired feeds or entries (or partially downloaded entries) against the filter. Subsequently, the client node sends individual requests for each entry that it wishes to retrieve. Listing 1 illustrates an example `request` message that requests four chunks from an entry that has an audio/mpeg file as enclosure.

A Bloom filter does not allow for iterating through the elements it contains and thus it cannot be used to discover previously unknown contents at a peer. The protocol therefore implements additional mechanisms for discovering previously unknown feeds and new entries on already known feeds, see Fig. 4. A request message can either contain an empty `feedlist` element or an empty `entrylist` element to indicate that it wants to receive the list of available feeds or entry ID's at the peer. The `selector` element of a request message can also be used to solicit meta-data for contents that match a particular selection criteria given by a *content selector*. A *content selector* is a string whose syntax is based on a subset of the SQL conditional expression syntax [7]. A node that receives a `request` message with a `selector` as top-level element of a `request`, evaluates the selector on the attributes of each of its available feeds. The feed elements for which the selector evaluates as true are delivered in a `reply` message. Similarly, a selector specified inside a `feed` element is evaluated against all entries of the specified feed and only those entry items that are evaluated as true are delivered. An empty selector matches all feed/entry elements and the unspecified attributes are evaluated as true by default. Since nodes can have large content libraries, specifying a selector when discovering feeds can significantly reduce the amount of meta-data delivered in a `reply` message.

Security

Our design intentionally does not include any special security mechanisms. Applications may have different security requirements and must implement the security features needed. Our design and the publish/subscribe abstraction enforce that, if security is needed, content should be secured rather than the communication channel. Entries can for example be encrypted and/or signed using standard public or private key cryptography (assuming that keys can be distributed in a secure manner).

As most other peer-to-peer systems, our system is vulnerable to security related issues such as spam, misuse and free-riding. Different solutions have been proposed to mitigate such vulnerabilities [9–11].

Caching

Nodes only store and carry contents that are of direct interest to them. Caching contents that are not relevant to a device is not included into the primary design of the *Fram* middleware. Due to node mobility, contact durations are often short. We believe that they should be utilized primarily to deliver contents of interest to nodes participating in the opportunistic content distribution system. In Section 4.2 we evaluate different caching strategies and we discuss further the benefits and drawbacks of including caching as part of the design.

Energy consumption

Although energy conservation is not a direct component of the protocol design, it should be taken into consideration since mobile nodes are battery-powered devices. We have previously shown [12] that simply turning on the Wi-Fi interface of a mobile device in ad-hoc mode significantly decreases the battery lifetime of the device, even when no data is exchanged via the wireless interface. Thus, for our middleware to be adopted widely, the energy consumption of mobile devices operating in ad-hoc mode should be reduced. In Section 4.2 we compare the performance of three different energy-saving mechanisms that can operate below our protocol design.

2.4. Internet domain

As illustrated in Fig. 1, the opportunistic content distribution system should not be considered in isolation of existing infrastructure. Instead, the system should be viewed as an extension of current network deployments. Integrating the wireless ad-hoc domain and the Internet domain could be done in existing access points. For instance, current access points could be upgraded with a dual-stack capability in order to translate incoming requests from the ad-hoc domain into the Internet domain and vice versa.

3. Implementation

We have implemented a prototype of our middleware design for the ad-hoc domain described in the previous section along with a set of applications that use the services provided by the *Fram* middleware. Our implementation runs on the Google Android Platform. The system is implemented in Java and our code consists of

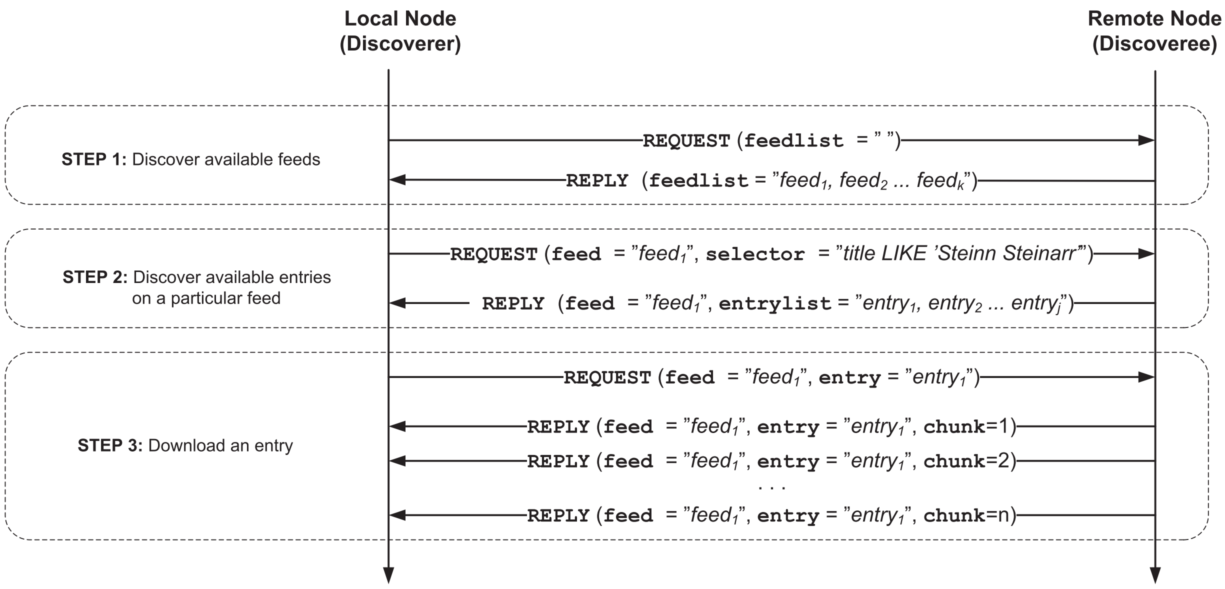

Local Node (Discoverer)
Remote Node (Discoveree)
STEP 1: Discover available feeds
REQUEST (feedlist = " ")
REPLY (feedlist = "feed1, feed2 ... feedk")
STEP 2: Discover available entries on a particular feed
REQUEST (feed = "feed1", selector = "title LIKE 'Steinn Steinarr'")
REPLY (feed = "feed1", entrylist = "entry1, entry2 ... entryj")
STEP 3: Download an entry
REQUEST (feed = "feed1", entry = "entry1")
REPLY (feed = "feed1", entry = "entry1", chunk=1)
REPLY (feed = "feed1", entry = "entry1", chunk=2)
REPLY (feed = "feed1", entry = "entry1", chunk=n)

Fig. 4. Discovering previously unknown contents on a remote node. An empty `feedlist` followed by an optional `selector` allow the local node to first discover all available feeds on a remote node, and then - all available entries under a particular feed (that also match the optional selector). Finally, the local node downloads a single entry.

55 classes and roughly 3500 lines of code. The implementation is currently based on 802.11 in ad-hoc mode and Android 4.0. The Android Java libraries (version 4.0) do however not support the ad-hoc mode of 802.11 although it is supported by both the driver and the hardware interface on the device. Therefore, our implementation requires the device to be run in privileged user mode (i.e., rooted mode) so that the interface can be reconfigured to run in ad-hoc mode. We note that even in the latest version of the Android Java libraries (version 6.0) there is still no support for the ad-hoc mode of 802.11. Furthermore, since the implementation only serves as a proof-of-concept, we use statically configured IPv4 addresses for the mobile devices in our test-bed. We also set a wake-lock in order to prevent the system from suspending its operation during our experiments (see Section 4.1).

3.1. Software modules

Our implementation consists of software modules that implement the functionality of the corresponding modules in our design in Fig. 3. Next we describe some implementation details of these modules.

Application programming interface

The middleware is implemented as an Android *service* which runs in the background and uploads and downloads data from peers that it discovers. Client applications can *bind* to the service and communicate with it by means of remote procedure calls (RPCs) through the publish/subscribe interface that it exposes. A client application wishing to receive a notification when an entry matching one of its subscriptions is downloaded, needs to implement and register a callback function that the service uses for notification. The interfaces for the service API and the application callback functions are shown in Listing 2. The remote methods exported by the service through the `IServiceAPI` interface are executed synchronously, thus blocking the local thread at the caller. In the service process, a method call is executed in a dedicated thread chosen from a pool of threads that is maintained by the Android system. The callback method in the `IClientCallback` interface is however executed asynchronously (specified by the

```
interface IServiceAPI {
    void publish(in String feedID, in Entry entry);
    void subscribe(in String feedID, in String selector);
    void unsubscribe(in String feedID);
    void discover(in String selector);
    void undiscover();
    void registerCallback(IClientCallback cb);
    void unregisterCallback(IClientCallback cb);
}

oneway interface IClientCallback {
    void notify(in String feedID, in Entry entry);
    void discoveryNotify(in String availableContents);
}
```

Listing 2. Interfaces for the service API and the application callback function.

`oneway` keyword) and therefore the service does not block when it notifies a client application.

All functions in the service API (instantiated in Listing 2) are synchronous and return immediately. As long as there is at least one feed subscription in the middleware, the local node will run the discovery process and the solicitation protocol if matching content is found. This is done continuously in the background, and the process is independent of the application. If the solicitation protocol succeeds to download new contents, the callback handler for the subscribed application is invoked with the new feed entry. Therefore, the content synchronization process is asynchronous to the application; it only receives a notification when new contents are found that matches its subscription. Note that all filtering of duplicate contents is done in the application. The application will stop receiving notifications once the `undiscover` method is invoked; the `undiscover` method unregisters the discovery callback.

Discovery module

The *discovery module* is implemented as two threads. One thread periodically broadcasts `hello` messages on a well-known UDP port and the other, a listener thread, waits for incoming `hello` messages from other nodes. The discovery module maintains a contact history cache along with the `revision` number for each peer in the cache. When a new peer is discovered, the *discovery module* notifies the *transport module* which initiates a download session with the peer. If a peer, already in the contact history cache, is seen, the transport module is notified if the peer has obtained new contents since the last association or if there are new subscriptions locally.

Transport module

The *transport module* implements both the client and server sides of a download session. The solicitation protocol is currently implemented on top of a simple transport protocol that implements message boundaries on top of TCP. The server side implementation listens on a socket and spawns a new session thread for each client. Similarly, if multiple nodes are in communication range, the transport module can create a separate client thread for each session. Currently we set the maximum number of concurrent client and server sessions to 6 in total (3 for each). If a new node tries to associate when the maximum number of sessions is reached, the server sends a `reject` message. Note that our choice for the maximum number of concurrent client and server sessions is based on the size of our test-bed which consists of 4 mobile devices. In reality there may be an arbitrary number of concurrent client and server sessions, however the higher the number of sessions, the less the bandwidth per session and the higher the complexity of the system. Since we are targeting opportunistic networks where contact durations are rather short, maintaining a large number of concurrent sessions may thus deteriorate the system performance.

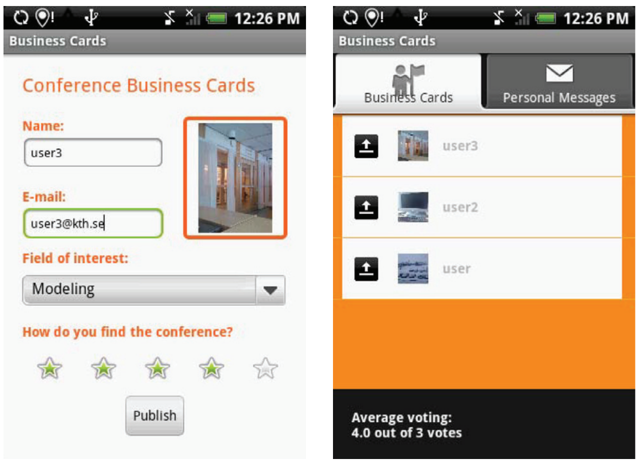
Content database

Meta-data for all available feeds and entries are stored in a SQLite database and this information is accessible to all applications on the device through the `ContentProvider` and `ContentResolver` Android Java classes. The enclosures themselves (i.e., data files) are however not stored in this database but in the corresponding Android content providers. Images, audio and video contents are for example stored in the Android `MediaStore` content provider. Thus, all media contents published or downloaded by our system are available to all applications in a standard Android manner.

3.2. Applications

To illustrate the versatility of the *Fram* middleware, we have implemented the following applications on top of it:

- *Opportunistic media blog* [13] – This application allows users to take photos or videos with their phones, caption them and publish them on a feed.
- *Personal profile sharing* [14] – This application allows users to create personal profiles (i.e., electronic business cards) and share them with other participants at a conference venue. The application also allows participants to vote on events related to the conference organization. Fig. 5 illustrates the application GUI.
- *Collaborative music sharing* [15] – Users can share music files stored on their personal mobile devices, and play them through a shared jukebox in communication range. The jukebox streams each audio file directly from the mobile device.
- *Participative light show* [16] – By using spectators' movements and ambient sounds, this application creates artistic light and sound effects in order to reflect and enhance the mood of an audience during a concert.


Business Cards
Conference Business Cards
Name: user3
E-mail: user3@kth.se
Field of interest: Modeling
How do you find the conference?

Average voting:
4.0 out of 3 votes

(a) (b)

Fig. 5. The GUI of the Personal Profile Sharing application. (a) Profile configuration and voting view. (b) Received profiles view.

4. Evaluation

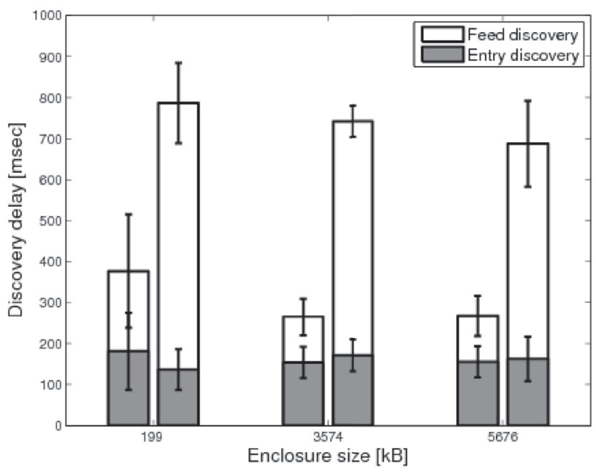
In this section we evaluate our system and dimension the system parameters. The evaluation uses two methods: experimentation and large-scale simulation.

4.1. Single node perspective

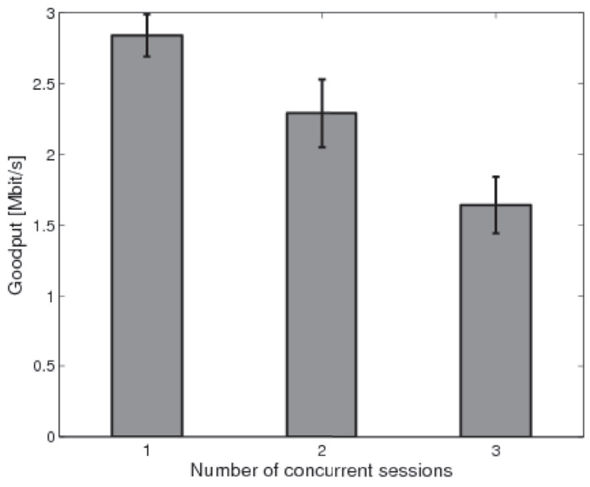
The experimental evaluation is performed on our Android implementation. We measure some key implementation metrics, such as application-level throughput (i.e., goodput), in a small and simple static scenario. The experiments are performed on identical HTC Hero A6262 mobile devices. These devices have a 528 MHz Qualcomm MSM7200A processor, a ROM of 512 MB and RAM of 288 MB and a Lithium-ion battery with capacity 1350 mAh. During our experiments, communicating nodes were stationary in an indoor office environment and placed within one meter from each other.

We have profiled our implementation of the solicitation protocol to verify correct behavior and to assess performance. For our measurements we have instrumented the code with hooks where we stamp the system clock (which provides millisecond precision). During a measurement run we turn off logging and collect the measured timestamps into a list which is printed to a file after the code section being measured has completed running. This minimizes the effect of any I/O operations due to logging or measurements on our results.

As shown in Fig. 4, a typical download session consists of three steps: (1) feed discovery, (2) entry discovery and (3) entry download. In Fig. 6(a) we show the mean delay for steps 1 and 2. We have conducted measurements for three different enclosure sizes and for each enclosure size we conduct one set where the content database only contains the actual feed and entry of interest (left-side bars) versus the case when the database has 100 other feeds available (right-side bars). For each measurement we conduct 10 runs and in the figure we show the mean value and the standard deviation. The results confirm that the total discovery delay (i.e., the sum of the feed discovery and entry discovery delays) does not depend on the size of the downloaded enclosure. When the number of feeds in the content database increases, the feed discovery delay increases due to an increase in the number of bytes transmitted in the `feedlist` in the reply message and processing delay at the server. (Observe that we do not rely on a Bloom filter during the feed discovery process, but instead request all available



(a)



(b)

Fig. 6. (a) Profiling results for the mean feed and entry discovery delays. Each group of two bars contain results with one feed in the content database (left) and 100 feeds in the content database (right). (b) The mean goodput of a download session when the number of concurrent clients is varied between one and three.

feeds at the remote node, one or 100 respectively.) We see also that the entry discovery delay remains the same since the number of entries on the feed of interest is the same in all experiments.

Our implementation supports multiple concurrent download sessions and in Fig. 6(b) we show the average goodput of a session when the number of devices concurrently downloading is between 1 and 3. Our measurement setup is as follows. Between one and three nodes (referred to as clients) are within range of a single node (referred to as server) which publishes a single entry on a feed that the client nodes are subscribed to. When the client nodes receive the first `hello` message sent by the server after the entry publication, the clients see that the server has new contents and therefore simultaneously associate with it. The client nodes discover the entry and download it. We measure the goodput G of each session as $G = B/T$ where B is the total number of bytes transmitted and T is the duration of the download session, i.e., the elapsed time from when the client discovers the node until it receives the full entry and the enclosure. The figure shows that although the implementation supports concurrent download sessions, the throughput is reduced because access to the wireless interface is serialized by the operating system at the server. The performance drop is however clearly less than 50% which is what would be seen if each node only were to serve a single client at once.

4.2. Multi node perspective

In addition to the Android implementation of our system, we have also implemented the core modules of our design for the OMNeT++ simulator, using the framework in [17]. The simulator implementation allows us to study system performance on a large scale in an urban area. A mobile node in the simulator implements the service discovery, synchronization and transport modules of our architecture described in Section 2.

Mobility scenarios

For our evaluation, we use the Walkers traces [18] captured in Legion Studio [19], a commercial simulator for designing and dimensioning large-scale spaces via simulation of pedestrian behaviors. Each trace file contains a snapshot of the positions of all nodes in the system every 0.6 s.

Fig. 7 (a) and 7(b) present the scenarios considered in our evaluation: an outdoor urban scenario, modeling the Östermalm area of central Stockholm, and an indoor scenario, recreating a two-level subway station. Both scenarios are representative of typical day-time pedestrian mobility. We here summarize the main characteristics of the scenarios; a detailed description can be found in [20].

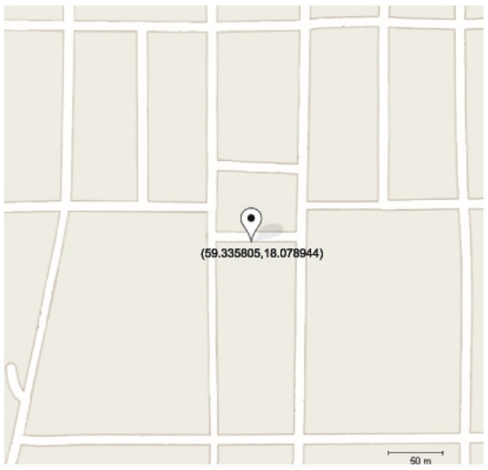
The Östermalm scenario consists of a grid of interconnected streets. Fourteen passages connect the observed area to the outside world. The active area is 5872 m^2; the active area defines the space in which nodes can move, i.e., the total surface of the streets. The nodes are constantly moving, hence the scenario can be characterized as a high mobility scenario.

The Subway station has train platforms connected via escalators to the entry-level. Nodes arrive on foot from any of five entries, or when a train arrives at the platform. The train arrivals create burstiness in the node arrivals and departures. Nodes congregate while waiting for a train at one of the platforms, or while taking a break in the store or the coffee shop at the entry level. The active area is 1921 m^2.

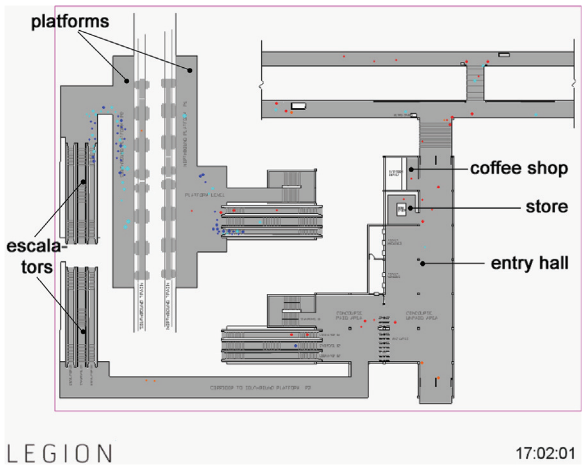
In the results to follow we focus on evaluating the system performance in high density scenarios. For fair comparison, the input parameters of the Östermalm and the Subway scenario result in approximately the same mean node density of 0.1 nodes per m^2. We note that the study is not aimed at exploring how scenarios of varied density affect the content dissemination but rather to demonstrate that the proposed middleware performs well in different environments by capturing the mobility space and the node dynamics in a realistic manner.

Performance metrics

We focus on two performance metrics: *goodput* (i.e., application throughput) and *energy consumption* from a system perspective. Since we study an open system, it is important that the metrics are normalized with respect to the nodes' sojourn time in the simulation. The system goodput is simply the sum of the number of bytes downloaded B_i by each node divided by the sum of the lifetimes of nodes in the simulation t_i, or $G = \sum B_i / \sum t_i$. We only count bytes of fully downloaded content items (entries), so the goodput is a measure of the system usefulness for the users, i.e., how much contents it can provide. Similarly, the energy



(a)



(b)

Fig. 7. Urban scenarios: (a) a grid of streets representing a part of downtown Stockholm, Östermalm, and (b) a two-level subway station.

consumption of the system is defined as $E = \sum E_i / \sum t_i$ where E_i is the energy consumed by the radio interface.

Caching

In this section we study the effects of introducing caching on top of the proposed architecture. We assume that nodes have enough battery to be altruistic and share contents with others in their vicinity, thus in this section we only evaluate the system performance in terms of goodput. For each node we introduce two types of caches: a private cache and a public cache. The *private cache* stores content items that the node is subscribed to; the *public cache* accommodates items that the node is willing to carry and forward on behalf of others. The private cache is populated when nodes obtain contents of interest from peers in direct communication range. To populate the public cache, a node downloads contents on behalf of another peer. However, nodes are initially not aware of content items that are not part of their private subscriptions. To discover and potentially download public contents, we allow nodes to relay incoming requests from peers in their vicinity as if they were the request initiators. Each request is only relayed once; in this way requests for public contents are kept close to the area in which the contents is needed. Fig. 8 illustrates how relay requests are implemented in the solicitation protocol. A `selector` is used to describe explicitly the requested contents, and new meta-data fields, `hop_cnt` and `max_hop`, are defined in order to prevent the request from propagating infinitely in the network. The exact `max_hop` value is a configurable parameter. For brevity, the relay request in Fig. 8 is shown in isolation; in practice requests for public contents and requests for private subscriptions are merged into a single `REQUEST` message. We introduce the following three strategies for relaying requests for content items that a node is unable to serve from its own caches:

- *Altruistic relay request* — A node relays every incoming request it cannot serve to peers in its vicinity. Due to node mobility [20], the relay outreach is limited to two or three hops away from the initial requesting node.
- *Greedy relay request* — A node prioritizes downloading contents for its private cache instead of its public cache. Thus, if a node discovers a neighbor which can contribute to its private subscriptions while downloading public contents on behalf of another peer, it will interrupt the current download session and initiate a new one with the neighbor offering to contribute to the private interest of the node.
- *Weighted relay request* — A node performs ordinal ranking of all requested content items during a single beaconing period based on the number of incoming requests for each item. It then allocates a certain predefined number of positions *m* for public requests and divides them evenly among *the most* and *the least* requested content IDs. Relaying a most requested item allows peers to contribute simultaneously to a number of requesting nodes; relaying a least requested item prevents penalizing nodes with less popular interests. If there are more than *m/2* most requested content items of equal request rate, the node chooses uniformly at random which of them to request. The same applies to the least requested content items.

In our evaluation scenarios we assume that all nodes carry devices that support 2 Mbps download rate from neighboring nodes, and that there are 1000 available content items (entries) in the area. The popularity distribution of those 1000 content items follows a Zipf distribution with parameter α=0.368. Every device is subscribed to 10 content items (entries) upon entering the observed area, and its private cache is initially populated with 5 randomly chosen content items out of these subscriptions. The public cache is empty upon arrival. Thus, throughout its lifetime in the simulation, each node strives to obtain the rest of the content items that belong to its subscription. Entries have a mean size of 3 KB, and a standard deviation of 1 KB. The beaconing period is set to 1 s, and the soft-state timeout for terminating a session is set to 30 s.

We define *private goodput* as the amount of bytes downloaded in the private cache of a node, divided by the lifetime of that node in the system. Similarly, *public goodput* denotes the amount of bytes downloaded in the public cache of a node, divided by the lifetime of that node in the system. We denote the mean private and mean public goodput with $\overline{G_{pri}}$ and $\overline{G_{pub}}$ respectively. All values of $\overline{G_{pri}}$ are normalized with respect to the value of a *No Caching* strategy; nodes that follow the *No Caching* strategy only store contents of private interest on their devices.

Fig. 9 shows the normalized private goodput and the overhead ratio $\overline{G_{pub}}/\overline{G_{pri}}$ for the Östermalm scenario. (Table 1 summarizes the notations used to represent the relaying strategies in the figures.) The introduction of both altruistic (strategies 2a and 2b) and greedy (3a and 3b) hop-limited relay request strategies significantly increases the private goodput, however this increase comes at a price of high overhead. Since battery capacity is considered to be an unlimited resource here, the increase in

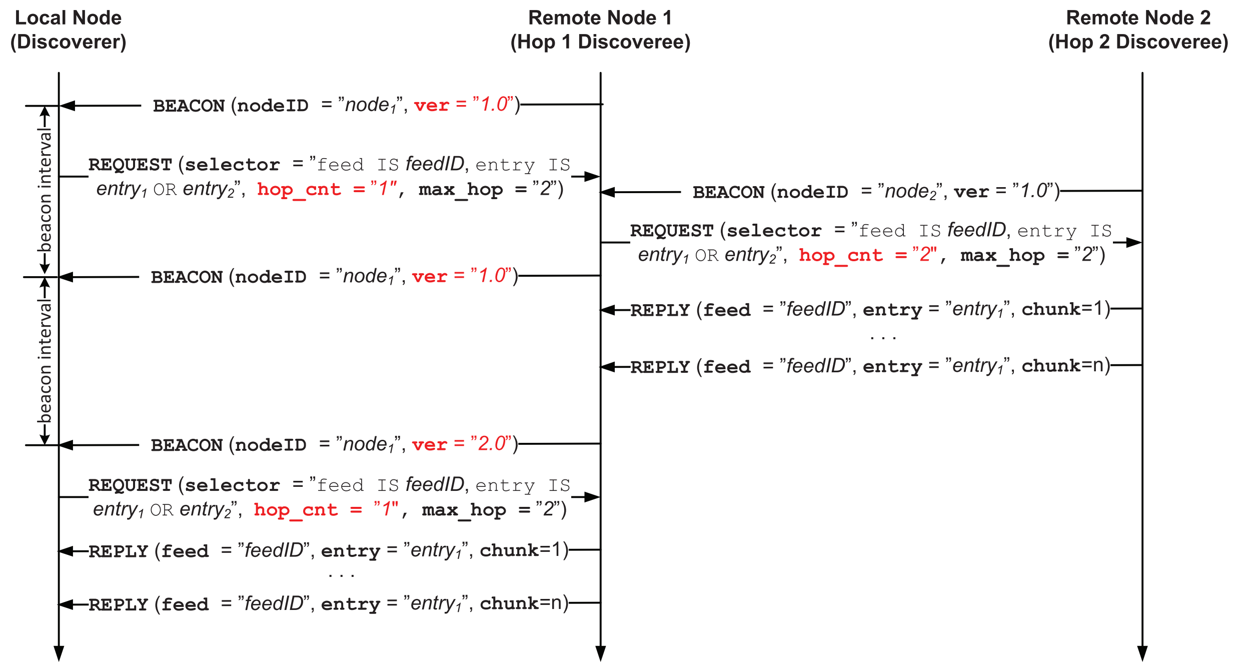

Local Node (Discoverer)
Remote Node 1 (Hop 1 Discoveree)
Remote Node 2 (Hop 2 Discoveree)
beacon interval
beacon interval
BEACON (nodeID = "node1", ver = "1.0")
REQUEST (selector = "feed IS feedID, entry IS entry1 OR entry2", hop_cnt = "1", max_hop = "2")
BEACON (nodeID = "node2", ver = "1.0")
REQUEST (selector = "feed IS feedID, entry IS entry1 OR entry2", hop_cnt = "2", max_hop = "2")
BEACON (nodeID = "node1", ver = "1.0")
REPLY (feed = "feedID", entry = "entry1", chunk=1)
REPLY (feed = "feedID", entry = "entry1", chunk=n)
BEACON (nodeID = "node1", ver = "2.0")
REQUEST (selector = "feed IS feedID, entry IS entry1 OR entry2", hop_cnt = "1", max_hop = "2")
REPLY (feed = "feedID", entry = "entry1", chunk=1)
REPLY (feed = "feedID", entry = "entry1", chunk=n)

Fig. 8. Relay request with two hops limit. The local node sends out a request with a `selector` defining the feed and optionally the entries that it is willing to download from the remote node 1. Remote node 1 does not have the contents in its cache, and relays the request to peers it its vicinity, i.e., remote node 2. Observe that remote node 1 updates the `hop_cnt` counter to reflect that the request does not belong to its private subscriptions but is instead relayed on behalf of another node. Remote node 2 delivers to remote node 1 part of the requested contents ($entry_1$), however it does not further propagate the request for the missing content items, since the relay limit `max_hop` has been reached.

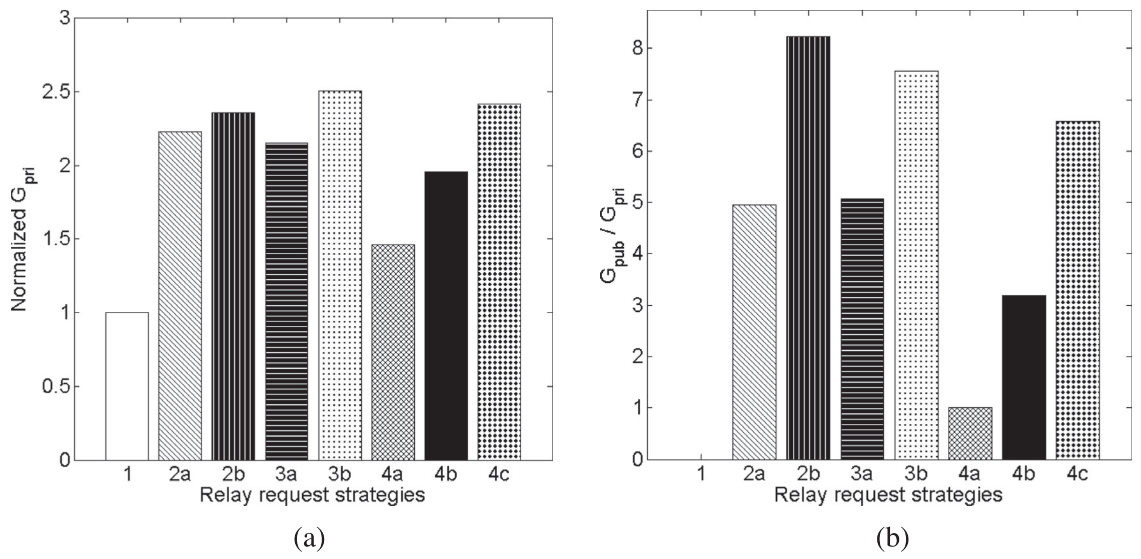


Fig. 9. Comparison of the performance of different relay request strategies in the Östermalm scenario: (a) normalized private goodput $\overline{G_{pri}}$ and (b) overhead ratio $\overline{G_{pub}}/\overline{G_{pri}}$.

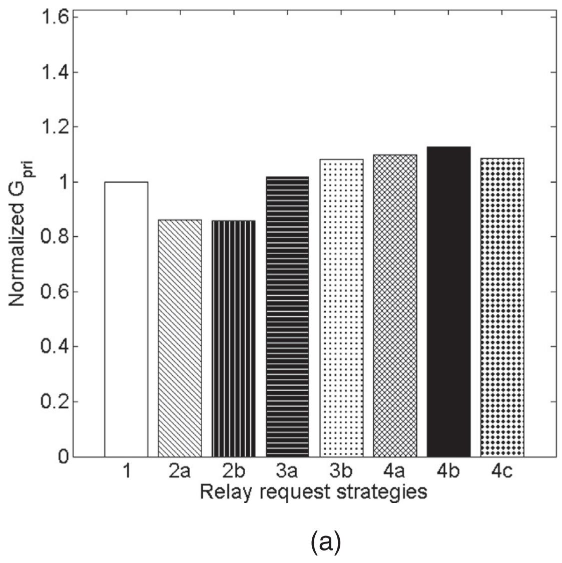
overhead may at first appear acceptable. However, in the context of battery-powered handheld mobile devices, a feasible caching strategy should provide high private goodput with low overhead. Restricting the amount of data requested on behalf of others to only $m = 2$ content items (strategy 4a) produces low overhead, $\overline{G_{pub}}/\overline{G_{pri}} < 1$ while increasing private goodput. With the increase of m however a small additional gain in private goodput comes at a price of a high overhead.

The results for the Subway scenario are presented in Fig. 10. Due to the longer contact durations among nodes (while waiting at a platform for a train to arrive, or queuing at the escalators) the increase in overhead is higher than for the constantly mobile nodes in the Östermalm scenario for both the altruistic and the greedy hop-limited relay request strategies. The increase in private goodput is negligible at best; in fact, in the case of altruistic relay requests, engaging in downloads on behalf of others harms the private interest of nodes. Introducing weights to the publicly downloaded contents again reduces the overhead, however without significant improvement in terms of private goodput.

Owing to the high overhead and uncertain benefit, we have not included public caching in *Fram*.

Energy consumption

We have previously shown that enabling the 802.11 interface of a mobile device reduces the battery life to only 25% of the lifetime with the interface turned off. [12]. This is despite the fact that no data has been transmitted or received via the interface. Once the



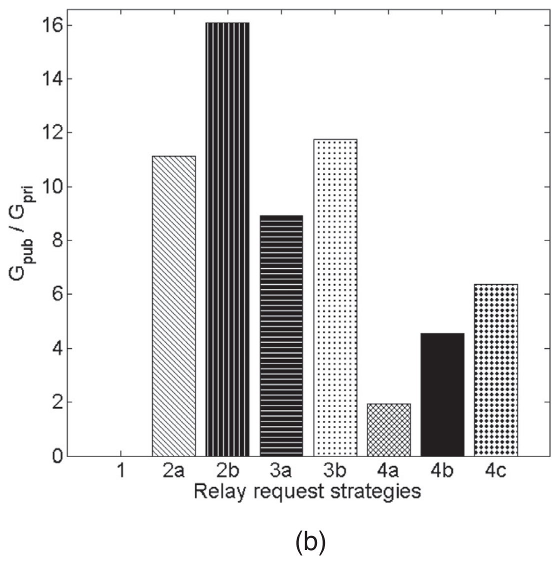


Fig. 10. Comparison of the performance of different relay request strategies in the Subway scenario: (a) normalized private goodput $\overline{G_{pri}}$ and (b) overhead ratio $\overline{G_{pub}}/\overline{G_{pri.}}$

Table 1
Notations for different relaying strategies.

Notation	Relay strategy
1	No caching
2a	Altruistic relay request with 2 hops limit
2b	Altruistic relay request with 3 hops limit
3a	Greedy relay request with 2 hops limit
3b	Greedy relay request with 3 hops limit
4a	Weighted relay request with m = 2
4b	Weighted relay request with m = 4
4c	Weighted relay request with m = 6

interface is turned on, it consumes relatively high power regardless of being in a transmit or receive state since listening in idle state consumes almost as much energy. This suggests that reducing or eliminating the idle energy cost of the 802.11 interface may be a promising strategy to reduce the overall energy consumption and prolong battery life thus enabling opportunistic communication as a viable communication mode.

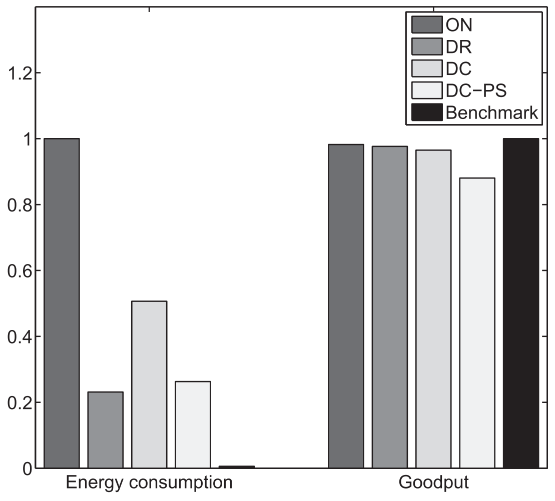
To decrease the energy consumption in mobile nodes, we evaluate the following three energy-saving mechanisms implemented on top of the *Fram* middleware.

- Dual-radio architecture (**DR**) [21]—a low-power low-energy radio interface is used for node and service discovery, and a high-power high-bitrate radio is only woken up *on demand* when a neighbor of interest is discovered. Thus, the high-power interface is only used for actual data transfer. We here assume that both radios have the same transmission range; for results with different transmission ranges we refer the reader to [21].
- Asynchronous duty cycling (**DC**) [22]—the radio interface is duty-cycled in order to decrease the energy consumption in idle state. Nodes choose uniformly at random the duration of the listening interval at the beginning of every cycle; when not in use, the radio is suspended. All nodes adhere to the same cycling interval, which is set to 10 s.
- Asynchronous duty cycling with progressive selfishness (**DC-PS**) [23]—a node duty cycles as described above while searching for content items; once a node obtains all content items belonging to its subscription it follows a *progressive selfishness* algorithm. Progressive selfishness allows nodes to further decrease their energy consumption by prolonging the time during which the radio interface is suspended; the duration of inactivity of the radio interface is based on the demand for the contents a node carries at any given moment.

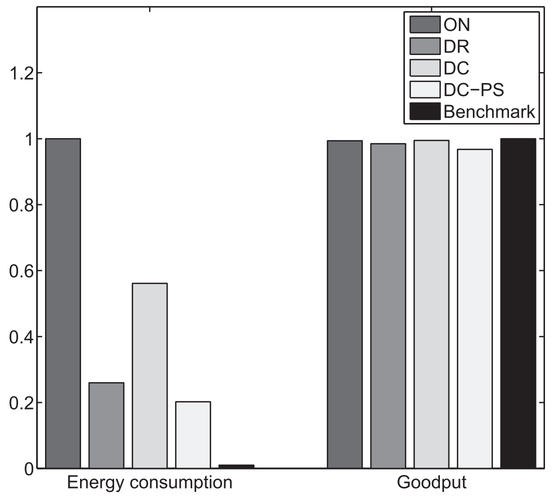
We compare the above energy-saving mechanisms to the performance of a system in which the radio interface of nodes is always turned on (**ON**) and to a benchmark (**BM**), an idealized system in which global knowledge is assumed for the location and the subscriptions of each node in the system; such a system concentrates on evaluation of the energy consumed for the actual data exchange and abstracts the discovery phase, thus it provides a lower bound for the energy consumption and an upper bound for the goodput. In the results to follow we normalize energy consumption with respect to the energy consumed by ON, and we normalize goodput with respect to the benchmark.

In our evaluation scenarios we assume that all nodes carry devices that support 2 Mbps download rate from neighboring nodes, and that there are 100 available content items in the area. Items are organized in 10 feeds, each feed containing 10 items. The popularity distribution of the feeds again follows a Zipf distribution with parameter $\alpha = 0.368$. Every device is subscribed to a single feed upon entering the area, and its cache is initially populated with 5 randomly chosen content items belonging to this feed. Thus, throughout its lifetime in the simulation, each node strives to obtain the rest of the content items that belong to its subscription. Entries have a mean size of 10 KB, and a standard deviation of 2 KB. We note that the choice of a larger entry size in this part of the study does not contradict the conclusions in the previous section. The mean entry size affects the amount of data that could be transferred over a single contact; however the entry size does not have an impact on the normalized goodput as long as the mean download time for an entry does not exceed the mean contact duration. As before, the beaconing period is set to 1 s, and the soft-state timeout for terminating a session is set to 30 s.

Fig. 11 presents the results for the Östermalm scenario when the communication range is $r = 10$ m and $r = 50$ m, respectively. At $r = 10$ m the dual-radio architecture outperforms the asynchronous duty cycling in terms of energy consumption without compromising goodput. However, we see that combining duty cycling with progressive selfishness achieves similar reduction in terms of energy consumption as DR (around 75%) at the cost of a slight decrease in goodput (less than 10%). Due to the the short contact durations, nodes tend to miss some contact opportunities

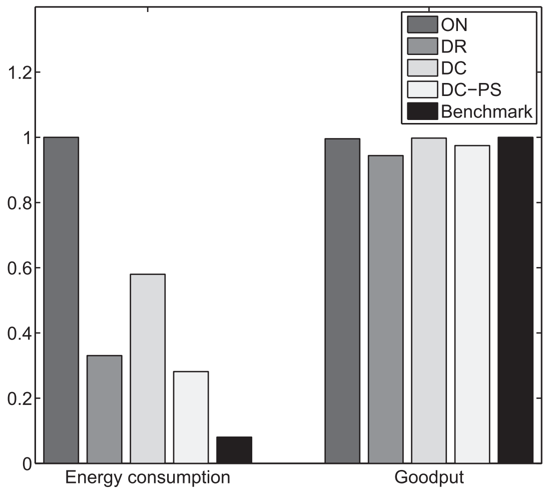


(a)

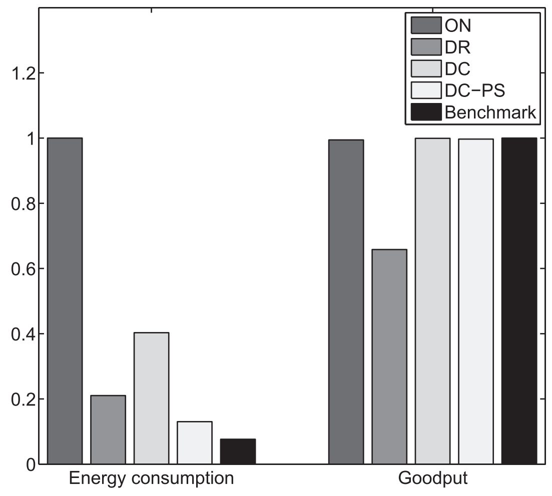


(b)

Fig. 11. Normalized energy consumption and normalized goodput for the Östermalm scenario with a cycling interval of 10 s and communication range (a) $r = 10$ m and (b) $r = 50$ m.



(a)



(b)

Fig. 12. Normalized energy consumption and normalized goodput for the Subway scenario with a cycling interval of 10 s and communication range (a) $r = 10$ m and (b) $r = 50$ m.

while their radio interfaces are turned off. With the increase of the communication range, Fig. 11(b), the contact durations become longer, and the goodput achieved by DC-PS becomes comparable to that of the benchmark.

The results for the Subway scenario are shown in Fig. 12. For both values of the communication range, the DC-PS scheme achieves the highest energy savings, outperforming both the dual-radio architecture and the asynchronous duty-cycling scheme. Furthermore, increasing the communication range to $r = 50$ m deteriorates the performance in terms of goodput for the dual-radio with up to 40%, while DC-PS maintains the goodput achieved by the benchmark. Table 2 reveals that the drop in goodput is due to the fact that many of the nodes in the Subway scenario do not obtain any content throughout their lifetime when operating with a dual-radio.

To conclude, dual-radio architectures provide an energy-saving mechanism that functions best in sparser environments. Duty cycling in combination with progressive selfishness on the other hand provides a stable solution across a larger range of configurations. Even when DC-PS experiences a slight loss in goodput (in scenarios with high mobility and short contact durations), the gain in energy savings is significant.

Table 2
Percentage of nodes that do not obtain a single content item during their lifetime.

Scenario	BM	ON	DC	DC-PS	DR
Östermalm, r = 10m	2.3%	3.6%	4.7%	8.6%	3.8%
Östermalm, r = 50m	0.3%	0.6%	0.5%	2.0%	1.2%
Subway, r = 10m	0.5%	0.6%	0.6%	1.8%	3.4%
Subway, r = 50m	0.0%	0.2%	0.1%	0.1%	22.7%

5. Related work

Our system design builds on the work in [3,24] and [25]. The work in [3,24] presented the original idea of a delay-tolerant broadcast system and evaluated its feasibility in an urban area while [25] introduced podcasting as an application for opportunistic networking. We here extend these previous works by defining a

Table 3
Comparison between our design and other systems that utilize peer-to-peer contacts between mobile devices for distributing and sharing information. Symbol × denotes that a characteristic is present in a system.

Peer-to-peer architecture	Design Characteristic			
	C1	C2	C3	C4
Fram	×	×	×	×
7DS [26,27]	×			×
Haggle [28,29]	×		×	×
DTN Architecture [30]		×		×
DoDWAN [31,32]	×	×	×	×
PeopleNet [33]	×	×		
BlueTorrent [34]	×		×	×
AllJoyn [35]	×	×		×
CAMEO [36]	×		×	×
BBS-ONE [37]	×			×
DTN-messaging [38]	×	×		×
Rio [39]				

general purpose publish/subscribe system for challenged networks and by specifying a detailed middleware and protocol design. Furthermore, we perform a large-scale evaluation and implement a set of applications as a proof-of-concept.

Other wireless peer-to-peer systems

Recently, there have been systems proposed that utilize peer-to-peer contacts of mobile hosts for distributing and sharing information in a similar way as we consider in our work. Table 3 summarizes the main differences of these systems with respect to our design (under the name *Fram* in the table) based on the following four characteristics:

- (C1) content-centric design;
- (C2) presence of a well-defined content structure;
- (C3) design based on a publish/subscribe paradigm;
- (C4) infrastructure-independence.

7DS [26] is a system for opportunistic dissemination among mobile devices of Internet data objects, identified by URL's. As such, 7DS was originally intended mainly for extending web browsing and e-mailing of mobile nodes beyond the reach of access points. There is now ongoing work in updating and extending the original 7DS architecture to provide a generic platform for communication in disruption-tolerant networks [27].

Haggle [28] is an architecture for mobile devices that facilitates the separation of application functionality from the underlying network technology. The goal is to allow applications to operate seamlessly across different networking environments and architectures. Haggle is thus not a strict protocol architecture for disruptive networks but rather proposes a node design that allows nodes and applications to adapt to the network connectivity level. In a recent redesign of Haggle [29], the focus is shifted away from point-to-point communication towards a more content-centric view, therefore making it more similar to the interest-driven dissemination of our architecture, originally proposed in [3]. There are however some notable differences in the design. In Haggle, content is unstructured but in both systems, contents have associated meta-data that facilitates searching and organizing of contents. We use a hierarchical content structure to associate content items with particular content feeds. Moreover, the actual data objects are further divided into fixed size chunks to facilitate distributed and disjoint content download. The Haggle architecture is push-based as opposed to the pull-based solicitation protocol in our architecture. It is unclear how the push-based approach avoids redundant data transmissions of already available content and how nodes can prioritize downloads according to their own preferences.

The DTN architecture [30] is a general communication architecture to enable communication in the presence of intermittent connectivity. It consists of an overlay, called the bundle layer, which operates above the transport layer. The architecture specifies the format of variable length application data units, called bundles. The goal is to deliver bundles from a sender to a receiver in the presence of intermittent and opportunistic connectivity, possibly over a wide range of different networks using different transport protocols. This is achieved by assuming that nodes store, carry and forward bundles to cope with link outages. The DTN architecture is node-centric and mainly focuses on unicast delivery of messages although some extensions for group communication have been proposed [40]. Its design philosophy is therefore significantly different from the content-centric approach we advocate in our work.

The DoDWAN middleware [31] presents a platform that supports content-based information dissemination in disconnected mobile ad hoc networks. The protocol is inspired by the publish/subscribe paradigm and is defined as a two-layer stack: the upper layer is responsible for node discovery and content exchange, and the lower layer deals with multi-hop forwarding. DoDWAN defines meta-data descriptors, however contents is not organized in a hierarchical manner. Instead selection patterns are used to map contents to the interest profile of each node. Similar to our design, nodes are discovered by periodic beaconing but in DoDWAN beacons contain all data descriptors that correspond to the announced interest profiles of nodes in range. It is however not clear how this data could be fitted into a single broadcast message. Furthermore, as opposed to our work in which a unicast request-response protocol is defined between devices in proximity, in DoDWAN content dissemination is done primarily via broadcasts. The DoDWAN middleware is evaluated via simulations as well as via a small-scale experimentation campaign in a campus setting [32].

PeopleNet [33] is a distributed geographic database where information is stored at people's mobile devices. Query requests and responses are forwarded from a mobile device via the cellular network to the geographic location which supports that particular type of request (named Bazaar). Users within the Bazaar then spread queries via peer-to-peer contacts. When a response is found for a query request the user who placed the query request is informed through the cellular infrastructure. In contrast to what we propose, PeopleNet heavily relies on a fixed infrastructure and is targeted at seeking information in contrast to broadcasting information. BlueTorrent [34] is an opportunistic file sharing application for Bluetooth enabled devices. The concept of distributing large files using small resumable atomic chunks follows our approach. However, BlueTorrent relies on Bluetooth whereas our design leverages any link-layer technologies. Furthermore, we propose to structure the data into feeds and rely on a receiver-driven content dissemination protocol.

AllJoyn [35] is an open-source platform by Qualcomm that has similar goals as our platform, namely to provide a system that relieves application developers from many of the hard problems associated with running distributed applications in a mobile environment with intermittent connectivity. The design of the AllJoyn system is however quite different from ours. AllJoyn implements a virtual distributed software bus that devices use to communicate when in range. This bus implements mechanisms such as naming, service discovery, communication sessions and a Remote Method Invocation (RMI) interface that applications use to communicate. From a developers perspective the AllJoyn system can be seen as an RMI system for opportunistic networks. In contrast, we explicitly target opportunistic publish/subscribe communication.

In [36] the authors present CAMEO, a context-aware middleware for opportunistic mobile social network. The notion of context consists of user-, device- and environmental information. Similar to our design, in CAMEO each user discovers direct neighbors through periodic beaconing. However as opposed to our proposal CAMEO also has the possibility to take optimized forwarding decisions by evaluating the probability for neighboring nodes to deliver a message to a particular destination. Moreover, in CAMEO a beacon carries all the information for node and service discovery packed in a hash value, while our implementation presents a hierarchical structure of contents stored in a node as well as a full request-reply content exchange protocol.

BBS-ONE [37] is a bulletin board system that allows for exchange of bulletin messages in networks with high mobility and churn rate, without relying on a centralized server. It supports both communication with fixed infrastructure, as well as pure peer-to-peer data exchange by exploiting IEEE 802.11 ad-hoc connections. However, as opposed to our design, messages are not structured in a hierarchical manner, but are instead mapped via keyword search. Moreover, BBS-ONE uses primarily pull- and push-based methods for transferring information among nodes while we rely solely on the publish/subscribe paradigm.

In [38] the authors present a DTN-like messaging system built on top of a peer-to-peer replication platform. The authors adjust the filtering capabilities of the peer-to-peer replication platform and explore four different routing algorithms in order to achieve high delivery ratio in a delay tolerant network. Thus every node obtains not only information relevant to its interest, but also carries data on behalf of other participants in the network. As opposed, our design does not assume any underlying routing protocols, and data is exchanged on a per-hop basis only when nodes that share a subscription come into contact.

In [39] the authors present Rio, a system solution that allows sharing of I/O such as camera, speaker or modem between mobile devices. The sharing however is not bound to nodes in proximity, and is thus infrastructure-dependent. Furthermore, the design tackles predominantly low-level system issues by splitting the stack at the device file boundary. As opposed, our middleware does not modify the I/O stack of the device, and data is exchanged among nodes in direct communication range without the help of infrastructure.

Caching and energy consumption

Most studies that evaluate the performance of opportunistic networks exploit pairwise contact opportunities between nodes. In the context of content caching, devised strategies exploit the users' social networks to try to identify nodes they frequently meet and solicit contents for them [41–43]. Few studies take an orthogonal approach, attempting to deliver data that is immediately available in a close neighborhood to the interested user [44,45]. However, it has been shown that such caching strategies consume a lot of resources without increasing the gain at the interested node [46]. In terms of energy savings, asynchronous duty-cycling schemes are suggested with respect to the inter-contact times of node pairs [47,48]. In urban scenarios with users on-the-go, the assumption of two nodes meeting in the future may not be appropriate since the inter-meeting times may be on the scale of days, and keeping track of all possible contacts during a node's lifetime is not feasible. We believe that a common case for users in urban areas is that they are regularly connected to infrastructure networks where they have access to a vast amount of contents. One of the main benefits of an opportunistic content distribution system is then that it allows users to access some contents while on the move between these occasions of Internet connectivity, such as when traveling to and from work. We therefore believe that the goal of caching and energy-saving mechanisms should be to bring what is immediately available in a close neighborhood to the interested user at a low energy cost.

6. Conclusion

In this work we presented the architecture and design of a mobile peer-to-peer content distribution system based on a publish/subscribe paradigm. Content spreads via sharing and direct interest-based dissemination and our design includes a set of basic mechanisms to discover and download contents efficiently in opportunistic networks. The system uses a decentralized content solicitation scheme that allows the distribution of contents between mobile devices without requiring Internet connectivity and infrastructure support. This scheme is efficient in the presence of intermittent contacts and short contact durations. The system design addresses key issues, in particular the structuring of contents to facilitate efficient lookup and matching of contents. We believe that our design is general and facilitates the implementation of advanced content-centric applications.

As a proof of concept, we have implemented our middleware on the Android platform along with a set of applications. We have demonstrated our system publicly and verified its correctness and experimentally evaluated performance on a small scale.

We have also implemented our system in a simulator environment and performed a large-scale evaluation in terms of both application throughput (i.e., goodput) and energy consumption. The simulator implementation consists of a detailed node and protocol implementation and uses a realistic mobility model of pedestrians in a city. Our results confirm that the system performance scales well with the number of nodes since performance improves when more nodes participate. We have further evaluated two features on top of our design: caching and energy-saving mechanisms. We presented three relay request strategies, and showed that caching contents on behalf of other nodes can significantly increase resource consumption for downloading and storage but often does not lead to any considerable increase in system performance in terms of goodput. We also introduced three energy-saving mechanisms, and showed that energy consumption in nodes could be decreased by 80% without greatly harming the system performance in terms of goodput. Thus, we claim that energy-saving mechanisms should be considered as part of the system design, whereas caching should be regarded as an add-on application-specific feature; if implemented, caching mechanisms should be such that they increase system performance while still being light on resource consumption.

Acknowledgments

This work has been partially funded by the Swedish Research Council and the ACCESS Linnaeus Center at KTH Royal Institute of Technology.

References

[1] Cisco visual networking index: Global mobile data traffic forecast update 2014 - 2019 white paper, 2015.
[2] V. Vukadinović, Ó.R. Helgason, G. Karlsson, An analytical model for pedestrian content distribution in a grid of streets, Math. Comput. Model. 57 (11-12) (2013) 2933–2944. Information System Security and Performance Modeling and Simulation for Future Mobile Networks.
[3] G. Karlsson, V. Lenders, M. May, Delay-tolerant broadcasting, in: Proc. of ACM SIGCOMM Workshops (CHANTS), 2006.
[4] 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.
[5] M. Nottingham, R. Sayre, The Atom Syndication Format, 2005, (RFC 4287).
[6] S. Trifunovic, M. Kurant, K.A. Hummel, F. Legendre, Wlan-opp: Ad-hoc-less opportunistic networking on smartphones, Ad Hoc Netw. 25 (Part B) (2015) 346–358.

[7] Java Message Service (JMS), Java message service (jms), (http://java.sun.com/products/jms/).
[8] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[9] S. Trifunovic, F. Legendre, C. Anastasiades, Social trust in opportunistic networks, in: INFOCOM IEEE Conference on Computer Communications Workshops , 2010, 2010, pp. 1–6.
[10] S. Trifunovic, M. Kurant, K.A. Hummel, F. Legendre, Preventing spam in opportunistic networks, Comput. Commun. 41 (2014) 31–42.
[11] S. Trifunovic, A. Hossmann-Picu, Stalk and lie - the cost of sybil attacks in opportunistic networks, Comput. Commun. 73 (Part A) (2016) 66–79.
[12] Ó.R. Helgason, E.A. Yavuz, S.T. Kouyoumdjieva, L. Pajevic, G. Karlsson, A mobile peer-to-peer system for opportunistic content-centric networking, in: Proc. of ACM SIGCOMM 2010, MobiHeld workshop, 2010.
[13] Ó. Helgason, E.A. Yavuz, S. Kouyoumdjieva, L. Pajevic, G. Karlsson, Demonstrating a mobile peer-to-peer system for opportunistic content-centric networking, in: Demonstration at ACM Mobiheld Workshop, 2010.
[14] S. Kouyoumdjieva, E.A. Yavuz, Ó. Helgason, L. Pajevic, G. Karlsson, Opportunistic content-centric networking: the conference case demo, Demonstration at IEEE Infocom, 2011.
[15] Z. Chen, E. Yavuz, G. Karlsson, What a juke! a collaborative music sharing system, in: Proc. of IEEE WoWMoM, 2012, pp. 1–6.
[16] I. Marfisi-Schottman, G. Karlsson, J.C. Guss, Demo: Opphos - a participative light and sound show using mobile phones in crowds, in: Proc. of ExtremeCom, 2013.
[17] Ó.R. Helgason, K.V. Jónsson, Opportunistic networking in omnet++, in: Proc. of ICST SIMUTools, OMNeT++ workshop, 2008.
[18] S.T. Kouyoumdjieva, Ó. R. Helgason, G. Karlsson, CRAWDAD data set kth/walkers (v. 2014-05-05), 2014, (Downloaded from http://crawdad.org/kth/walkers/).
[19] Legion Studio, Legion studio, (http://www.legion.com/).
[20] Ó. Helgason, S.T. Kouyoumdjieva, G. Karlsson, Opportunistic communication and human mobility, Mobile Comput. IEEE Trans. 13 (7) (2014) 1597–1610.
[21] S.T. Kouyoumdjieva, Ó. Helgason, E.A. Yavuz, G. Karlsson, Evaluating an energy-efficient radio architecture for opportunistic communication, in: Proc. of ICC E2Nets workshop, 2012.
[22] S.T. Kouyoumdjieva, G. Karlsson, Energy savings in opportunistic networks, in: Proc. IEEE/IFIP WONS, 2014.
[23] S. Kouyoumdjieva, G. Karlsson, Energy-aware opportunistic mobile data offloading for users in urban environments, in: Proc. IFIP/TC6 Networking, 2015.
[24] G. Karlsson, V. Lenders, M. May, Delay-tolerant broadcasting, IEEE Trans. Broadcast. 53 (1) (2007) 369–381.
[25] V. Lenders, M. May, G. Karlsson, Wireless Ad Hoc podcasting, in: Proc. of IEEE SECON, 2007.
[26] M. Papadopouli, H. Schulzrinne, Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices, in: Proc. of ACM MobiHoc, 2001.
[27] A. Moghadam, S. Srinivasan, H. Schulzrinne, 7DS - A modular platform to develop mobile disruption-tolerant applications, in: Proc. of NGMAST 2008, 2008.
[28] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, E. Upton, Haggle: seamless networking for mobile applications, Proc. of ACM UbiComp (2007) 391–408.
[29] E. Nordström, P. Gunningberg, C. Rohner, A search-based network architecture for mobile devices, Technical Report, Uppsala University, 2009. 2009-003.
[30] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, H. Weiss, Delay-tolerant networking architecture, 2007, (RFC 4838).
[31] J. Haillot, F. Guidec, A protocol for content-based communication in disconnected mobile ad hoc networks, Mobile Inf. Syst. 6 (2) (2010) 123–154.
[32] Y. Mahéo, N. Le Sommer, P. Launay, F. Guidec, M. Dragone, Beyond opportunistic networking protocols: a disruption-tolerant application suite for disconnected MANETs, in: Proc. ExtremeCom, Zurich, Switzerland, 2012, pp. 1–6.
[33] M. Motani, V. Srinivasan, P.S. Nuggehalli, Peoplenet: engineering a wireless virtual social network, in: Proc. of ACM MobiCom, 2005.
[34] S. Jung, U. Lee, A. Chang, D.-K. Cho, M. Gerla, BlueTorrent: cooperative content sharing for bluetooth users,, in: Proc. of IEEE PerCom, White Plains,USA, 2007.
[35] AllJoyn, Alljoyn, (http://www.alljoyn.org/).
[36] V. Arnaboldi, M. Conti, F. Delmastro, Cameo: a novel context-aware middleware for opportunistic mobile social networks, Pervasive Mobile Comput. 11 (2014) 148–167.
[37] K. Sung, S. Srinivasan, H. Schulzrinne, Bbs-one: bulletin board and forum system for mobile opportunistic networks, in: Proc. of IEEE WCNIS, 2010, pp. 370–375.
[38] P. Gilbert, V. Ramasubramanian, P. Stuedi, D. Terry, Peer-to-peer data replication meets delay tolerant networking, in: Proc. of ICDCS, 2011, pp. 109–120.
[39] A. Amiri Sani, K. Boos, M.H. Yun, L. Zhong, Rio: a system solution for sharing i/o between mobile systems, in: Proc. ACM MobiSys, New York, NY, USA, 2014, pp. 259–272.
[40] M. Demmer, K. Fall, The design and implementation of a session layer for delay-tolerant networks, Comput. Commun. 32 (2009) 1724–1730.
[41] P. Costa, C. Mascolo, M. Musolesi, G.P. Picco, Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks, IEEE J. Selected Areas Commun. 26 (5) (2008) 748–760.
[42] E. Yoneki, P. Hui, S. Chan, J. Crowcroft, A socio-aware overlay for publish/subscribe communication in delay tolerant networks, in: Proc. of ACM MSWiM, 2007.
[43] C. Boldrini, M. Conti, A. Passarella, ContentPlace: social-aware data dissemination in opportunistic networks, in: Proc. of ACM MSWiM, 2008.
[44] L. Hu, J.-Y. Le Boudec, M. Vojnovic, Optimal channel choice for collaborative ad-hoc dissemination, in: Proc. of IEEE Infocom, 2010.
[45] J. Reich, A. Chaintreau, The age of impatience: optimal replication schemes for opportunistic networks, in: Proc. of ACM CoNEXT, 2009.
[46] S. Kouyoumdjieva, S. Chupisanyarote, O. Helgason, G. Karlsson, Caching strategies in opportunistic networks, in: Proc. IEEE WoWMoM, 2012, pp. 1–6.
[47] W. Gao, Q. Li, Wakeup scheduling for energy-efficient communication in opportunistic mobile networks, in: Proc. INFOCOM, 2013.
[48] E. Biondi, C. Boldrini, A. Passarella, M. Conti, Optimal duty cycling in mobile opportunistic networks with end-to-end delay guarantees, in: Proc. European Wireless, 2014.

Ólafur Helgason is a post-doc researcher at the Laboratory for Communication Networks at KTH, Stockholm. Ólafur has previously worked as a researcher at the Systems and Services laboratory at Reykjavik University and as a professional programmer. In 2001, Ólafur was an intern at AT&T Shannon Laboratories, New Jersey, USA. He has a Ph.D. in electrical engineering from KTH (2011), Stockholm, and M.Sc. in computer science from the University of Iceland (2002). His research focuses on mobility modelling for wireless communication, opportunistic networking and mobile peer-to-peer systems.

Sylvia Kouyoumdjieva is a post-doctoral researcher at the Laboratory for Communication Networks at KTH Royal Institute of Technology, Stockholm, Sweden. She has a Ph.D. in Electrical Engineering from KTH (2015), a M.Sc. in Telecommunications from KTH (2009), and a B.Sc. in Telecommunications from the Technical University of Sofia, Bulgaria (2006). Her research focuses on opportunistic networking and mobile peer-to-peer systems.

Ljubica Pajević received her M.Sc. degree in system engineering and radio communications in 2009, and the B.Sc in telecommunications in 2007, both from the School of Electrical Engineering, University of Belgrade, Serbia. She is currently working towards a Ph.D. degree in electrical engineering at the Laboratory for Communication Networks, Royal Institute of Technology, Stockholm, Sweden. Her research interests are in the area of mobility modeling and opportunistic content distribution.

Emre A. Yavuz received his B.Sc. and M.Sc. degrees in Electrical & Electronics Engineering at METU in Ankara, Turkey in 1995 and 1998, respectively. He was a Software Engineer with Alcatel in Toronto developing safety critical real-time microprocessor firmware for embedded command, control, and communication applications in automated train systems from 1999 to 2001. He received his Ph.D. degree in Electrical & Computer Engineering at The University of British Columbia in Vancouver in 2007. He worked as a technical consultant from 2007 to 2009 prior to joining the School of Electrical Engineering at KTH in Stockholm as a post-doctoral fellow. He is now with Ericsson AB for research and development in LTE Systems.

Gunnar Karlsson is professor of KTH Royal Institute of Technology since 1998; he is director of the Laboratory for Communication Networks. He has previously worked for IBM Zurich Research Laboratory and the Swedish Institute of Computer Science (SICS). His Ph.D. is from Columbia University, New York, and his M.Sc. from Chalmers University of Technology in Gothenburg, Sweden. He has been visiting professor at EPFL, Switzerland, and the Helsinki University of Technology in Finland (now part of Aalto University), and ETH Zurich in Switzerland. His current research relates to opportunistic networking, mobility modeling and online education; he has previously worked on quality of service, wireless LAN developments and router architecture. He is a member of ACM and senior member of IEEE; he is a senior editor for IEEE Journal on Selected Areas in Communication and serves regularly on technical program committees for conferences such as IEEE Infocom. He is the 2015 recipient of the KTH Pedagogic Prize.