A Mobile Peer-to-Peer System for Opportunistic Content-Centric Networking

Ólafur R. Helgason, Emre A. Yavuz, Sylvia T. Kouyoumdjieva,
Ljubica Pajevic, and Gunnar Karlsson

Laboratory for Communication Networks
KTH, Royal Inst. of Tech.
100 44 Stockholm, Sweden
{olafurr, emreya, stkou, ljubica, gk}@kth.se

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. Contents are exchanged opportunistically when nodes are within 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 or matching and soliciting of contents. Our architecture consists of three key components. A *content structure* that facilitates dividing contents into logical topics and allows for 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 describe the design and implementation of our architecture. We also discuss potential applications and present evaluation results from profiling of our system.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

Design

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. Today, there are two ways of distributing content for mobile devices: downloading when docked to a computer with Internet access or by means of fixed infrastructure such as cellular networks/access points. The former mode is enabled by the massive storage available, but the distribution is limited to contents present at the time of docking. Access via public 802.11 networks may provide more frequent downloading opportunities when devices are on the move but their coverage is limited. The cellular network provides good coverage and continuous access to contents, however, cell capacity is a technological concern due to saturation if downloading of multimedia contents becomes very popular. Not to mention the high pricing along with the concern of so-called wireless net neutrality.

In this paper we propose a middleware architecture that allows applications on mobile devices to share contents: The devices can utilize connections with access points when in range, or distribute contents opportunistically among mobile nodes otherwise. This extends the availability of contents beyond the reach of the infrastructure while it enables their distribution. Contents are structured to facilitate efficient lookup matching and downloading under disruptive node connectivity. This requires rethinking of networking basics: While existing network architectures focus on addressing *nodes* and forward 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 while abstaining from 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.

With its content based routing and addressing, the system can be seen as a pub/sub system that decouples the communicating entities from the contents and thus it inherently allows for 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 a modest delay such as conducting local quizzes/surveys, audio/video broadcasting and on-site networking/dating profile exchange. The proposed architecture also promotes openness: anybody who wishes to publish/retrieve contents is free to do so. We therefore believe that the system has the necessary features to stimulate organic user growth, which has led to the success of many systems and services on the net. The architecture is inspired by podcasting and BitTorrent. Our operating scenario is however radically different than what is experienced on the wired net since our architecture has to cope with sporadic

MobiHeld 2010, August 30, 2010, New Delhi, India.

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 [11] [5], there are still many challenges that need to be addressed and solved by an actual system design and it is the primary focus of this work.

The rest of the paper is organized as follows. Section 2 covers related work and in section 3 we give an overview of the architecture. Sections 4, 5, 6 and 7 present the detailed design of the system and our implementation is described in section 8. Potential applications are discussed in section 9. We evaluate our implementation in section 10 and conclude in section 11.

2. RELATED WORK

The Delay-Tolerant Network Architecture [1] and Haggle [10] are two interesting communication architectures proposed in this field. DTN consists of an overlay, called the bundle layer, which operates above the transport layer. The goal is to deliver data units called *bundles* from a sender to a receiver in the presence of opportunistic connectivity using different transport protocols assuming that nodes store, carry and forward bundles to cope with link outages. Haggle is an architecture for mobile devices that facilitates the separation of application functionality from the underlying network technology to provide seamless operation despite disruptions in network connectivity. It achieves this through late binding of network interfaces, protocols and names. It is thus different from DTN since it is not a strict protocol architecture but rather proposes a node design that allows nodes and applications to adapt to the network connectivity level. 7DS [7] is another system for opportunistic dissemination of data among mobile devices and allows for extending legacy applications, such as web and e-mail, to opportunistic environments. Our system differs from systems in many fundamental aspects. It is data-centric with content-based addressing and it provides applications group-based communication via pub/sub interface. In contrast, DTN focuses on unicast source to destination communication and provides limited support for pub/sub communication. Haggle and 7DS are general purpose platforms to allow both new and legacy applications to operate in challenged environments while ours explicitly targets applications that can adapt to a pub/sub interface.

Other similar systems have been proposed. Cimbiosys [9] is a platform that provides content synchronization and replication through opportunistic peer-to-peer communication. It uses content-based filters to specify which contents are synchronized and shared by the system. BlueTorrent [4] is an opportunistic file sharing application only for Bluetooth enabled devices.

The system design introduced here builds on the work in [5] and [6]. [5] presents the original idea of a delay-tolerant broadcast system and evaluates its feasibility in an urban area while [6] introduces podcasting as an application for DTN. Similar systems to ours have been proposed.

Pub/sub systems have been widely adopted in the context of wired networks recently. Such an approach is however not suitable for mobile wireless scenarios where fixed infrastructure cannot be assumed. In [2], Costa et al. present SocialCast; a content-based routing scheme for pub/sub communications in a DTN environment. Another content routing scheme is presented in [12]. Mobile nodes run a community detection algorithm and in each community, the nodes with the highest closeness centrality act as message brokers. In contrast, our system does not include an explicit caching or message routing mechanism but is based on direct interest sharing and dissemination of content. To the best of our knowledge these content routing schemes have not been implemented but incorporating such a scheme in the context of our platform might be of interest.

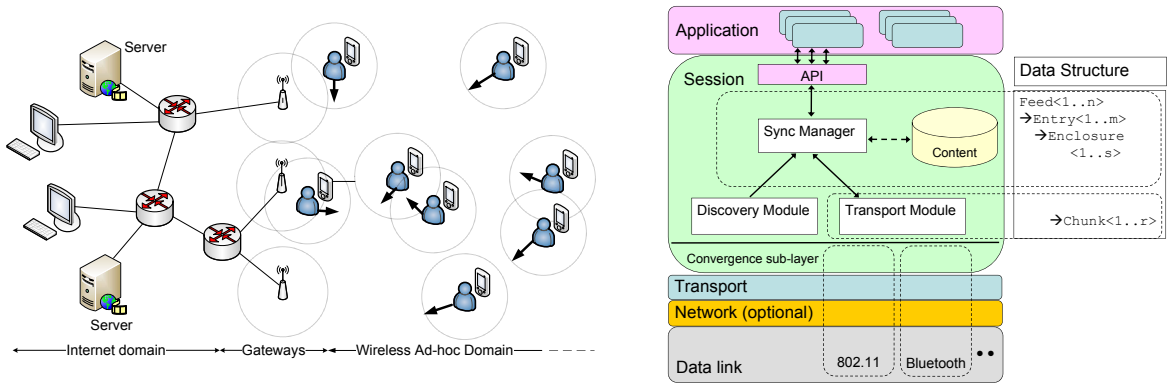
3. ARCHITECTURE OVERVIEW

A general instantiation of the system can consist of three domains as shown in Fig. 1(a). The system imposes a hierarchical structure on contents by organizing them into *feeds* where each feed consists of a number of *entries*. The sharing of contents is based on a solicitation protocol by which a node solicits entries for one or more feeds from a peer (a peer node can either be a mobile device or a gateway to the Internet, such as 802.11 access points). The content structure in the system allows for ease of searching and a higher hit rate of content queries than if they were made for individual unstructured contents. The system design does 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.

Fig. 1(b) illustrates the node design and the main components of our architecture. Applications access the services of the session layer through an API that the middleware exports. The API implicitly defines the content structure for applications and it allows them to pub/sub to content feeds. The node design is composed of a set of modules that implement the API, content solicitation on behalf of the applications, service discovery and the solicitation protocol. The architecture also contains a convergence sub-layer for cross-layer interaction, particularly with the underlying radio link such as 802.11/Bluetooth. Their architectures are quite different and thus the session layer architecture abstracts most of the details of the underlying radio and the heterogeneity of the networks away from the applications. The session layer assumes an underlying transport layer that preserves message boundaries, provides flow control and process-to-process communication above an optional network layer. Messages are delivered on a best-effort basis with no guarantee that entries on a particular feed will be delivered orderly to all receivers.

4. CONTENT STRUCTURE

Content addressing and organization adopts and extends the content structure of the Atom Syndication Format [8]. This format has primarily been used for publishing web-feeds and podcasts on the net. This content structure is quite generic and allows for more use cases than what has commonly been tried and it also maps nicely to the pub/sub semantics of our system. Contents are grouped into *feeds*. A feed is an unlimited container for *entries* that contain the actual data objects of interest. Each feed can have multiple entries published at different times by different entities. 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 update. A feed can also contain other optional meta-information such as author, subtitle and category. Similarly, each entry must also contain a globally unique ID, a title and a release timestamp and it can optionally have a range of other elements including zero or more *enclosures*. An enclosure is a single file attachment and would typically be an audio, video, or text file. To be able to efficiently transfer enclosures over the opportunistic contacts, we divide the enclosures into *chunks*, small data units of fixed size, which can be exchanged with high probability during a single contact of limited duration. Chunks are an extension to the Atom format and they allow allow an incompletely downloaded entry to be resumed with the same node or any other node that also has the entry (or parts of it). They are indexed starting from 1 and nodes can use these indices to resume interrupted



(a) (b)

Figure 1: (a) The system composed of servers, wireless gateways and mobile devices. (b) System architecture and data structures.

downloads. If a chunk is only partially received from a peer (e.g. due to lost connection) it is discarded. The recommended chunk size is 16 kB which we have found to tradeoff overhead and probability of incomplete reception well (study omitted in the paper).

5. 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 [3]. JMS however was designed for wired networks where dedicated brokers implement message delivery the discovery of feeds also relies on centralize directory service.In our operating environment, 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 the publish/subscribe functionality, we need to augment the API with a mechanism for feed discovery and for creating new feeds.

6. SYNCHRONIZATION & DISCOVERY

The *synchronization manager* processes content from applications and solicits contents on behalf of them. If the local content database contains data that matches a subscription, the content is delivered immediately to the application. The manager prioritizes content solicitations such that different applications get a fair share of the network resource.

The *discovery* module finds which neighbors are running the service and decides which ones are feasible to associate with. The module is split across the main session layer and the convergence sub-layer. The latter implements neighbor discovery specific to the underlying radio subsystem and notifies when a neighbor has been discovered. This notification includes the `node-ID` and the `revision` number of the 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 that nodes remain in range for longer durations, and thus avoid constant re-synchronization with all the neighbors only to discover if any new contents have become available. The `node-ID` is a globally unique node identifier that does not have any particular structure. The only requirement is that nodes shall choose unique addresses such as a MAC address.

Our design does not assume any existing service discovery mechanisms and includes a basic mechanism by which nodes periodically broadcast `hello` messages to their link-layer neighbors, including the `node-ID` and the `revision` constructs described above. It is expected that in many cases nodes will support more advanced and efficient service discovery than the default `hello` method such as the Service Discovery Protocol (SDP) in Bluetooth.

7. TRANSPORT MODULE

This module 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: `hello`, `request`, `reply` and `reject`. All other elements of a protocol message are child entries for the header fields associated with the message type.

Session Management

When a node discovers a new peer, it first sends a `request` message to the peer to initiate a unilateral session for downloading. The request contains either a query for a particular feed entry or for meta-data to discover content availability. The peer sends a `reply` message, establishing the session and replying to the query. Each download session thus consists of a client node sending request messages and a server node sending reply messages (or reject if the server is unavailable). 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 and then to deliver them.

Content solicitation in our system is entirely pull-based. At the client, a typical session alternates between *discovery* and *download* states. In the former state, the client node queries the server for content-meta information whereas it downloads contents that match the subscriptions of applications during the latter 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. If the client node wants to filter the contents it solicits from a particular feed (such as only soliciting content published after a certain time) it first needs to solicit feed meta-information before it can directly request the entries available at the serving node that match the request criteria.

In general, a node can have multiple active sessions simultaneously with the node being either a client (when it is downloading)

or server (when it is uploading) in each session. 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.

Content Solicitation

A `request` message contains the `bloom`, `selector`, `feed`, `entry`, and `chunks` elements. These messages are also used to query for meta-information to discover available content at a neighbor and discover new content, previously not known to the querying node. Discovering which previously known feeds or entries are available at a peer node is done efficiently by having each node maintaining a *Bloom filter* populated with the ID's of available feeds and entries at the node. A Bloom filter is a space-efficient data structure that provides a set-like representation of elements, requiring only a fraction of the space needed for a corresponding set with the actual elements. When a node receives a `request` with an empty XML `bloom` element, it delivers its Bloom filter in a `reply` message. After receiving the filter, the client node tests the ID's of its subscribed feeds or partially downloaded entries against the filter. Since false negatives are not possible, an ID not found in the Bloom filter does certainly not exist at the peer. Although false positives will occasionally result in requests being sent for ID's that are not available, the number of bytes transmitted to discover available contents is drastically reduced speeding up the content synchronization process. A Bloom filter does not allow for iterating through the element 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.

The `selector` element of a request message can be used to solicit meta-information for contents that match a particular selection criteria given by a *content selector* that has the same semantics as the message selectors previously described in section 5. A *content selector* is a string whose syntax is based on a subset of the SQL92 conditional expression syntax [3]. 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 to true are delivered in a reply message. Similarly, a selector specified inside a `feed` element will be evaluated against all entries of the specified feed and only those entry items that evaluate to true are delivered. An empty selector will match all feed/entry elements and those attributes not specified in the selector evaluate to 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.

8. IMPLEMENTATION

We have implemented our system in Java for the Google Android OS platform. Our implementation is based on 802.11 in ad-hoc mode but we also intend to support Bluetooth in the future. The Android Java libraries (version 2.2) do not currently support the ad-hoc mode of 802.11 although this is supported by both the driver and the hardware interface on the HTC Hero 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.

The middleware is implemented as an Android *service* which runs in the background and uploads and downloads data from peers

```
interface IServiceAPI {
    void publish(in String feedID, in Entry entry);
    void subscribe(in String feedID);
    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 1: Interfaces for the service API and the application callback function.

that it discovers. Client applications can *bind* to the service and communicate with it by means of remote procedure calls (RPCs) through the pub/sub 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 function are shown in listing 1. 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 `oneway` keyword) and therefore the service does not block when it notifies a client application.

The *discovery module* is implemented as two threads. One thread periodically broadcasts `hello` messages on a well-known UDP port and 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.

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.

The *content database* of the system is implemented as an *Android Content Provider*. Meta-information for all available feeds and entries is 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 content 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 content published or downloaded by our system is available to all applications in a standard Android manner.

9. APPLICATIONS

The opportunistic pub/sub service presented by the system architecture is quite generic and provides developers with variety of possibilities for application development. In this section we give examples of application categories that can be built on top of our system. Those categories encompass applications that differ in their spatial scope, as well as in the involvement of users to the data generation and the data exchange.

Local quiz: With this application, users can opportunistically initiate a local quiz or a poll. When a user initiates a new quiz instance it creates a feed and publishes the quiz as the first entry. Participants subscribe to the feed and publish their answers as new entries on the feed. Information on available quizzes could also be distributed on a dedicated discovery feed. When participants come into communication range they exchange published entries and locally update their results. In the simplest scenario where no result aggregation is needed, each user can receive the answers from other participants and then, based on higher level logic, create its own representation of the quiz results.

Social networking: Many of the current social applications that are popular on the Internet (such as Facebook or Twitter) lend themselves well to the pub/sub abstraction and can be extended into the opportunistic domain. Each user has a feed that followers subscribe to. Status updates, blogs or media files can be published as entries by the user. The actual data to be shared in each entry will be specified in the enclosure field, and users could for example define different feeds for separating content, e.g. an audio feed, a video feed etc. Applications falling into the social networking category are not expected to have any spatial limitations, thus the content can be spread opportunistically as long as there is interest in it.

Relaying sensor data: This category relates to applications that require transporting sensor data from devices in the field to a sink node or infrastructure network. Nodes that participate in the relaying of data subscribe to feeds that the sensors publish data on.

10. SYSTEM EVALUATION

The evaluation in this section is 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.

Energy consumption

We have measured the effect of our system on the battery life of the device. The Android system sends out an event notification (Intent) whenever the remaining life of the battery changes (in units of 1%). We have created a simple application that registers for these events and logs the time whenever the battery status changes. This way we can track how fast the battery is drained when various system services and applications are turned on or off. In Fig. 2 (a), we compare the battery profile for 5 scenarios: a) with the 802.11 interface turned off and our system not running, b) with the 802.11 interface turned on in ad-hoc mode but our system not running, and with our system running with the interval between `hello` messages set to c) 0.1 sec, d) 1 sec and e) 10 sec. All measurements were performed on the same device with no other active devices in range at the same time. During all measurements the display backlight was turned on. This drains the battery faster than in normal mode but prevents the device from going into idle battery saving mode which reduces the comparability of our measurements.

From Fig. 2 (a), we clearly see that the 802.11 interface significantly increases energy consumption. Running our system (in idle mode, only sending `hello` beacons) in addition to the 802.11 interface does not add considerably to the energy consumption beyond what is required by 802.11. When beaconing every 0.1 seconds[1], the battery lasts approximately 40 minutes shorter then when the `hello` messages are sent every 10 sec. We intend to add Bluetooth support to our system as well since it is significantly less power hungry than 802.11.

Solicitation protocol profiling

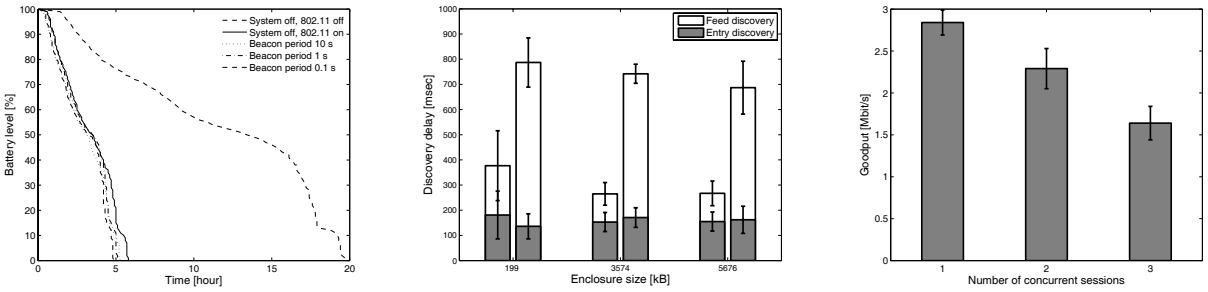
We have profiled our implementation of the solicitation protocol to verify correct behavior and assess performance. For our measurements we have instrumented the code with hooks where we stamp the system clock (which provides millisecond precision). During a measurement run we turn off logging and collect 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 described in section 7, a typical download session consists of three steps: 1) the client discovers available feeds at a server, 2) then it discovers available entries for a given feed and 3) finally downloads the entry of interest. In Fig. 2 (b) we show the mean feed discovery and entry discovery delay (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 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 reply message (which contains the list of available feeds) and processing delay at the server. 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. 2 (c) 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 subscribing 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 content and therefore simultaneously associate with it. The client nodes discover the entry and then download it and 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 enclosure.

Since the client nodes are being served concurrently, it is the responsibility of the MAC layer to share the radio channel between the download sessions. If the server would only support one session at a time the clients would be served sequentially and contention at the MAC layer is reduced. For a server that does not support concurrent sessions, the mean goodput for N sequentially served client

[1]This is the beacon period commonly used by 802.11 access points.



(a) (b) (c)

Figure 2: (a) Comparison of battery profiles when 802.11 is turned on/off and our system is turned on/off. (b) 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). (c) The mean goodput of a download session when the number of concurrent clients is varied between one and three.

nodes is given by $G_N = \frac{1}{N}(B/T + B/2T + \cdots + B/NT)$, assuming that the client nodes are not further sharing the entry among themselves. For $N = 2$ and $N = 3$ we we get $G_2 = 0.75\,G_1$ and $G_3 = 0.61\,G_1$. In our measurements we obtain the mean value $G_1 = 2.86$ Mb/sec. Using this value in the expressions for G_2 and G_3 gives $G_2 = 2.13$ and $G_3 = 1.73$ Mb/sec which are lower and higher respectively than measured values in Fig. 2 (c). This indicates that serving nodes concurrently may not be beneficial when more then two nodes are interested in the same content. In our future work we intend to conduct measurements on an implementation where nodes are served sequentially to verify if this holds in practice.

11. CONCLUSION AND DISCUSSION

We have presented a middleware architecture for mobile peer-to-peer content distribution. Content spreads via sharing and direct interest-based dissemination and our design includes a set of basic mechanisms for efficiently discovering and downloading content in opportunistic networks.

We have described the design and implementation of our system for the Google Android platform. Our experience from the implementation is that Android is a very powerful platform and quite mature despite its young age. The Java based environment provides a familiar environment with good support for most common OS primitives such as threads and concurrency, database and content storage and inter process communication through the Android service binding mechanism. Some features are however still missing, in particular support for the 802.11 ad-hoc mode (which needs to be implemented in native code).

We believe that our design is general and facilitates the implementation of advanced content-centric applications. There are however some issues that are not, or only partially addressed by our design. We do currently not address particularly the issues of privacy, security and power management. As we showed, the 802.11 interface draws significant power and it is probable that an implementation based on Bluetooth would be less battery demanding. Bluetooth however has other limitations, such as a long and inefficient discovery process, which make it ill-suited for mobile scenarios. Also, content dissemination in our system is purely interest-driven and nodes do not cache or forward any contents beyond what they are privately interested in. Content caching and forwarding is one of our primary directions for future work. We intend to release an implementation of our system as a research prototype.

12. REFERENCES

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838, April 2007.

[2] P. Costa, C. Mascolo, M. Musolesi, and G.P. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *IEEE JSAC*, 26(5):748–760, June 2008.

[3] Java message service (jms). `http://java.sun.com/products/jms/`.

[4] Sewook Jung, Uichin Lee, Alexander Chang, Dae-Ki Cho, and Mario Gerla. BlueTorrent: Cooperative Content Sharing for Bluetooth Users,. In *Proc. PerCom*, White Plains,USA, March 2007.

[5] Gunnar Karlsson, Vincent Lenders, and Martin May. Delay-tolerant broadcasting. In *Proc. ACM SIGCOMM, CHANTS Workshop*, Pisa, Italy, September 2006.

[6] Vincent Lenders, Martin May, and Gunnar Karlsson. Wireless ad hoc podcasting. In *Proc. IEEE SECON*, San Diego, CA, June 2007.

[7] Arezu Moghadam, Suman Srinivasan, and Henning Schulzrinne. 7ds - a modular platform to develop mobile disruption-tolerant applications. In *Proc. IEEE NGMAST*, Washington, DC, USA, 2008. IEEE Computer Society.

[8] M. Nottingham and R. Sayre. The Atom Syndication Format. RFC 4287, December 2005.

[9] Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C. Marshall, and Amin Vahdat. Cimbiosys: a platform for content-based partial replication. In *Proc. USENIX NSDI*, Boston, Massachusetts, 2009.

[10] Jing Su, James Scott, Pan Hui, Jon Crowcroft, Eyal de Lara, Christophe Diot, Ashvin Goel, Meng Lim, and Eben Upton. Haggle: Seamless networking for mobile applications. *Proc. UbiComp*, pages 391–408, 2007.

[11] Vladimir Vukadinovic, Ólafur Helgason, and Gunnar Karlsson. A mobility model for pedestrian content distribution. In *Proc. Simutools, SCENES Workshop*, Rome, Italy, March 2009.

[12] Eiko Yoneki, Pan Hui, ShuYan Chan, and Jon Crowcroft. A socio-aware overlay for publish/subscribe communication in delay tolerant networks. In *Proc. MSWiM*, Crete Island, Greece, 2007.