Dynamic Framework for Building Highly-Localized Mobile Web DTN Applications

Kartik Sankaran[†], Akkihebbal L. Ananda[†], Mun Choon Chan[†], and Li-Shiuan Peh[‡]
[†]School of Computing, National University of Singapore
[‡]Dept. of EECS, Massachusetts Institute of Technology
[†]{kartiks,ananda,chanmc}@comp.nus.edu.sg, [‡]peh@csail.mit.edu

ABSTRACT

Proximity-based mobile applications are increasing in popularity. Such apps engage users while in proximity of places of interest (malls, bus stops, restaurants, theatres), but remain closed or unused after the user goes away. Since the number of 'places of interest' is constantly growing and can be large, it is impractical to install a large number of corresponding native applications on the phone when each app engages the user for only a small period of time.

In this paper, we propose a dynamic framework for deploying highly-localized mobile web applications. Such web applications are deployed locally to users in proximity, and can be opened in the browser. Communication in the web app is performed over the Delay-Tolerant Network of mobile users, removing the need of an Internet connection. DTN protocols can be dynamically added or removed at run-time, allowing each application to use a protocol best suited to its needs. After usage, the web application is closed either manually by the user, or automatically when the user goes away from the place of interest.

We have implemented the framework on Android. Our analysis of the framework show that the memory and performance overhead incurred is small. Using this framework, we have written a simple DTN web application for bus stops to help the physically challenged.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Store and forward networks*

Keywords

Delay-Tolerant Networks; Smartphone; Web applications; Dynamic framework

1. INTRODUCTION

Proximity-based mobile applications have recently gained increasing popularity. In these applications, users interact with other users around them. Table 1 lists some of the popular proximity applications, along with the number of users who downloaded the application, and their average rating out of five. Many of these have more than 10 million downloads, and have high user ratings.

The rise of proximity applications has sparked an interest in scalable and energy-efficient device-to-device technologies such as LTE-direct and Bluetooth LE beacons. These technologies are expanding proximity applications to include not just interaction with people, but with physical places as well, such as stores, theatres, and restaurants. For example, users can check for daily specials in nearby restaurants, and movie combo offers in nearby theatres.

Each place of interest is typically associated with its own dedicated application on the phone. Users need to install these apps in order to use them. However, as the number of places of interest grows, installing large number of apps quickly becomes wasteful and annoying to the user. To solve this problem, what is needed is lightweight and convenient installation of proximity applications *when the user is near the place of interest*, in addition to good device-to-device communication. When users go away, the apps should no longer be active nor installed on the phone. This allows users to have highly-localized interactions, with apps engaging users only when necessary, perhaps even only for a brief period of a few minutes.

One possible solution is to deploy native mobile apps 'on-the-fly' to users who are in proximity to places of interest over device-to-device communication link. This eliminates the need to install apps beforehand and need for Internet connection to the server. Delay-Tolerant Networks (DTN) [1] are best suited to deploy apps since they exploit device-to-device technologies, working in the face of high user mobility. Unlike client-server solutions, DTN does not require an Internet connection to a central server, nor does it need access to a user's location, being inherently locality-specific.

Installation of native apps 'on-the-fly' is however still not lightweight. More importantly, users are wary of giving permission to unknown apps to access their phone's storage and private details. Use of *web applications*, as opposed to native applications, solves this problem as web applications run in the browser's security sandbox. Installation is lightweight since it only involves opening a web page. The browser informs the user when a web app attempts to access private details like location, which can be denied. Users are willing to allow such interactions since it is more akin to browsing a website.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CHANTS'14, September 7, 2014, Maui, Hawaii, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3071-8/14/09 ...$15.00.
http://dx.doi.org/10.1145/2645672.2645675.

Table 1: Examples of Social-Proximity Applications on Android

Application	Description	Downloads	Rating out of 5
Foursquare[1]	Find interesting places nearby, check-in for discounts	10,000,000+	4.2
Badoo[2]	Chatting, dating, making friends with people nearby	10,000,000+	4.5
Groupon[3]	Finding local deals and discounts	10,000,000+	4.6
Skout[4]	Discovering and meeting new people around	10,000,000+	4.1
Circles[5]	Finding people nearby with mutual interests	1,000,000+	4.5
Sonar[6]	Connect with friends and like-minded people nearby	1,000,000+	4.1
GrabTaxi[7]	Finding and booking nearby cabs	100,000+	4.1

While web apps do not have the full freedom of native applications, they are still quite powerful, having access to location, camera, and even the phone's sensors. However, they are currently limited to communication over sockets. To enable their full potential, web apps *need access to communication over the DTN*, thus making use of upcoming device-to-device technologies.

In this workshop paper, we propose and implement a dynamic framework for developing and deploying highly-localized mobile DTN web applications. This framework deploys web apps to users near the places of interest. The phone notifies the user of received web apps, and if found interesting, can be opened in the mobile browser. After use, the web app can be closed either manually, or automatically when the user leaves the place of interest. We have implemented the framework on Android, and ported it to desktop. It supports both web and native DTN applications.

Our analysis of the framework shows that the memory and performance overhead incurred is small. As a demonstration, we have written a simple DTN web application for bus stops to help the physically challenged. The app informs users when buses are arriving at the pick-up point, and is customized to physically challenged users to help them inform bus drivers that they would like to board.

By supporting both Android and web applications, the framework exposes DTN to the large community of developers, making it more likely for DTN applications to be developed for general use. Since protocols are plugged in dynamically, it is easy to modify to adapt to current advances in DTN protocols and device-to-device communication without re-compilation of the framework.

The rest of the paper is organized as follows: Section 2 discusses related work and provides a motivation for our framework. Section 3 describes the design of the framework, while Section 4 describes our sample application. Section 5 evaluates the framework. Section 6 discusses future work, while Section 7 concludes the paper.

[1]https://play.google.com/store/apps/details?id=com.joelapenna.foursquared
[2]https://play.google.com/store/apps/details?id=com.badoo.mobile
[3]https://play.google.com/store/apps/details?id=com.groupon
[4]https://play.google.com/store/apps/details?id=com.skout.android
[5]https://play.google.com/store/apps/details?id=com.discovercircle10
[6]https://play.google.com/store/apps/details?id=me.sonar.android
[7]https://play.google.com/store/apps/details?id=com.grabtaxi.passenger

2. RELATED WORK AND MOTIVATION

In this section, we discuss related work under different categories. By describing their limitations, we also provide motivation for development of a dynamic framework.

HTTP-over-DTN browsing: Efforts have been made to use DTN for web browsing [9, 10, 11, 12]. These papers concentrate on techniques for serving browsing requests over DTN, such as bundling of HTTP requests, pre-fetching, and caching. The underlying DTN is hidden from webpages.

Our framework focuses on deploying DTN web *apps*, as opposed to web pages. Web apps are similar to mobile apps: they are self-contained, i.e. they contain all the scripts and web pages required for the app to work. Also, DTN web apps are fully aware of the underlying DTN, using the DTN API exposed by our framework.

Web-based DTN Apps: Web apps such as Facebook and blogging have been written to use DTN [13, 14]. While these apps are 'DTN-aware', the work concentrates on how the apps work using DTN, and does not support localized deployment of web apps and protocols on-the-fly.

PhoneGap: PhoneGap is a framework for creating cross-platform mobile apps using web technologies. Each app runs in the PhoneGap container, which is essentially a 'super-browser': apps can access phone details (such as user contacts) via PhoneGap, normally not accessible to regular web apps. PhoneGap apps, while written in Javascript, are installed like native apps. The advantage is that several code versions are not required for different mobile platforms.

However, since the app must be installed like a native application, installation of PhoneGap apps do not meet the lightweight and convenience requirements of localized proximity applications. In addition, they lack access to DTN APIs.

QR Codes: QR codes are useful to direct mobile users to web pages online by scanning codes using their camera. While these codes are convenient to post near places of interest, they require users to look for and manually scan the codes. Discovering web apps is not 'automatic' like in our framework.

DTN middleware for mobile: Several middleware have been written on mobile for development of DTN applications. Table 2 provides a list of existing middleware, along with the type of API exposed, and a brief description of each. To the best of our knowledge, these middleware do not expose their API to web applications (with an exception of Bytewalla, discussed below), limiting their use to native mobile applications only.

In addition, unlike our framework, these middleware are static, i.e. the underlying protocols are fixed at compile-time and shared by multiple applications. It is not possible

Table 2: Existing DTN Frameworks

Framework	API exposed to developers	Brief Description
Haggle [2]	Publish-Subscribe API (attribute-based)	Uses a search-based data-centric protocol
Mist [3]	Publish-Subscribe API (topic-based)	Uses a reliable broadcast with fragmentation
MaDMAN [4]	Sockets API	Switches between TCP/IP and DTN protocol stack
ubiSOAP [5]	Service-Oriented API	Floods WSDL files and SOAP messages
MobiClique [6]	Social-Networking API	Built on top of Haggle
DoDWAN [7]	Publish-Subscribe API (attribute-based)	Floods WSDL files and SOAP messages (with attributes)
Bytewalla [8]	Bundle Protocol API	First implementation of the Bundle Protocol on Android

to load and unload protocols on-the-fly, a feature required by 'use-and-discard' proximity web applications.

Service-Adaptation Middleware: The work in [15] proposes a middleware that acts as a bridge between DTN apps written in different languages and DTN bundle service daemons running on different platforms. Bytewalla is the daemon running on Android, while PCs run the DTN2 service daemon. This middleware enables web applications to access DTN. However, like the web-based apps discussed earlier, it does not support localized deployment of web apps and protocols on-the-fly.

Dynamix: Dynamic frameworks are quite popular in the context-aware computing domain. In particular, a framework called Dynamix [16] provides context-awareness to web applications, by means of context components loaded at runtime. Architecturally, this framework is closest to our framework.

Although architecturally similar, Dynamix focuses on context awareness: its APIs are oriented around receiving 'context events'. In contrast, our framework's (DTN) APIs are communication-oriented. Dynamix's context-aware components are self-contained, while our protocol components are linked in the form of protocol stacks for each application.

To summarize this section, existing work have limitations with respect to the requirements of lightweight and convenient localized DTN web applications. Our framework has been designed to address these limitations and make such dynamic DTN applications possible.

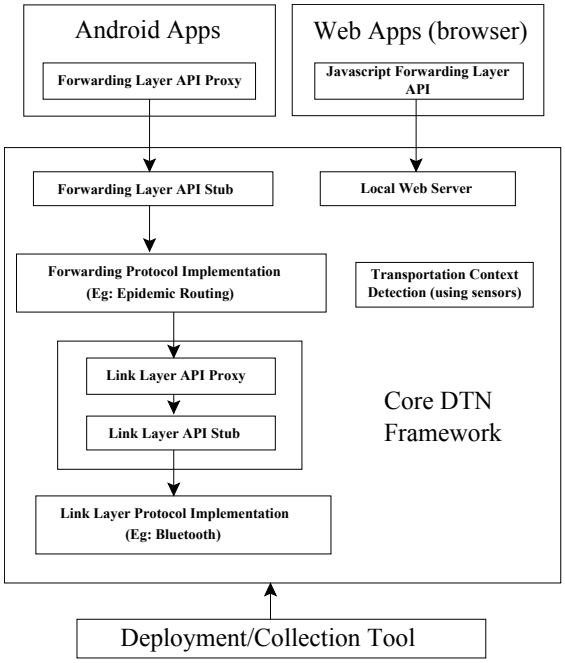


Figure 1: Design of the framework

3. DESIGN AND IMPLEMENTATION

In this section, we give a high-level overview of the design and implementation of our framework. As shown in Figure 1, it consists of three parts: the framework itself, the deployment application, and the Android/Web applications.

The framework consists of APIs, and protocol components implementing these APIs, all loaded at run-time. To support dynamic loading of code, it uses Apache Felix. It runs as a background (bound) service in Android.

We have written a simple Forwarding Layer API for applications to access routing protocols. This API supports multi-hop message transfers over the DTN. We also have a Link Layer API for one-hop communication, which supports neighbour discovery and connection-oriented communication, implemented by link layer components (Bluetooth, WiFi-direct), and used by forwarding layer components. Dynamically loaded APIs are advantageous since OSGi allows multiple incompatible versions of the API to co-exist without breaking applications.

Although Figure 1 shows only two protocols and a single protocol stack, the framework supports multiple protocol stacks, with protocols dependencies arranged in a directed acyclic graph. Every application can potentially load and use its own protocols, or even share protocol stacks. Protocol components are given a user-readable name in their config files. Applications can request for protocols with the specified name. Changing protocols involves loading a different protocol and giving it the same config name.

API components are broken into proxy and stub parts, in accordance with Android's inter-process communication (AIDL). The proxy and stub parts contain logic that shields upper layers from change in underlying protocols at run-time by saving state information, and hides underlying AIDL.

The deployment app is a 'special' DTN application that is used to deploy web apps, protocols components (`jar` files), and even native applications (it also supports collection of logs over DTN for debugging purposes). The user is notified of received web apps, which are opened in the browser, while protocol stacks are loaded into the framework.

3.1 Web app support

Web apps are provided with two Javascript libraries `DtnMessage.js` and `FwdLayerAPI.js`. The first contains convenience methods for creating DTN messages, while the second exposes the Forwarding layer API.

The framework runs a local embedded web server which receives DTN API calls from web apps via AJAX, and translates them into corresponding Java calls. To overcome the same-origin policy restriction, the server supports Cross-origin resource sharing[8]. To enable web apps to receive DTN messages, the Javascript code uses AJAX long polling.

We have implemented a low-power transportation context detection service to detect when the user is IDLE, WALKING, or in VEHICLE. In the future, we will be integrating this into the framework for automatic closing of web apps. For now, the web app is closed in the browser manually, and is not difficult for the user.

The framework runs both on Android and PCs. A subset of the Android libraries were implemented on PC so that it can compile and run largely without modification. The framework currently has full support for native Android applications, while web app support is in the prototype stage.

4. SAMPLE DTN WEB APPLICATION

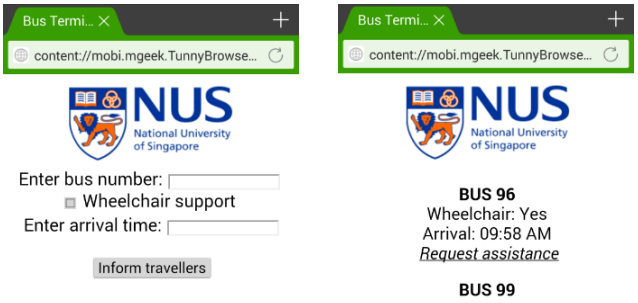
To demonstrate the usefulness of localized web apps, and illustrate how these apps work from the user's perspective, we wrote a simple app for bus stops and terminals to help the physically disabled as well as regular commuters board buses. This app is a web version of a DTN Android application written by students of the National University of Singapore (the framework has been used for two semesters by student groups in the Wireless and Sensor Networks course to build DTN apps for project work).

In bus terminals, commuters would like to know when the bus driver has been instructed to go to the pick-up point. Rather than install the LTA (Land Transport Authority) application from the play store beforehand, they can use the web app for a more localized and brief interaction. Physically disabled, such as wheelchair commuters, require assistance to board buses at bus stops and terminals. They need to inform drivers in advance so that they can board first, using a customized version of the app to do this. Our web app uses DTN to enable bus drivers to announce their allotted pick up time, regular commuters to receive this information, and wheelchair commuters to request drivers to assist them while boarding.

Since the framework has been ported to desktop, and supports multiple applications and users on a single device, the web app was developed locally before deploying it to mobile devices. This is especially important since it is easier to debug code using tools available in desktop browsers.

A device (laptop/mobile) located at the bus stop (alternatively can be placed on buses) deploys the web app wirelessly over the DTN to commuters nearby. Users carrying mobile devices running the framework receive the deployed app on-the-fly. In our prototype, received web apps are displayed in the notification bar. If interested, users can open the app in their browser. The user can choose a customized interface: for example, wheelchair people can choose the web app specialized to help them.

The app for regular users only displays arrival information sent by drivers (Bus driver's interface is shown in Figure 2a). Wheelchair people have the additional capability to inform drivers in advance that they would like to board, as shown in Figure 2b. Customized DTN protocols can be optionally bundled with the web app, and plugged into the framework at run-time. After usage (i.e. the commuter has boarded), the app can be simply closed in the browser. The framework automatically releases resources used by the app.



(a) For bus drivers (b) For Wheelchair people

Figure 2: Bus Stop Web App

Our sample application demonstrates the advantages of localized web apps: localized interactions, lightweight installation, and secure execution in the browser. Most importantly, these apps exploit device-to-device communication. In the future, our app will be extended to use swipe gestures and audio for the blind.

Description of our students' apps, documentation, APIs, and tutorials are available at the framework's website[9].

5. EVALUATION

In this section, we first compare the use of centralized server versus device-to-device communication with respect to power usage and latency experienced. We then evaluate the performance and memory overhead of our framework.

5.1 Server versus Device-to-device

Existing proximity applications have to use a central server to calculate whether a user is close to a place of interest. The phone uploads its location to the server, which informs it when it is nearby interesting places. Uploading over the cellular network is costly in terms of power. Use of device-to-device technologies can reduce power consumed, but requires periodic 'device discovery'. In this section, using power measurements on the Monsoon power meter, we quantify and compare the power usage of server-based versus device-to-device technologies, and show that there is indeed a power saving in spite of the device discovery process.

The power consumption depends on the frequency of location updates (for server-based solution) and on frequency of device discovery (for device-to-device technologies). The lower the frequency, the lower the power consumed, but at the expense of latency. We assume the device at the 'place of interest' (eg: bus stop) is powered externally, and only focus on the user's phone's power usage here.

We consider the case where location updates to the server occur over the LTE network, while the device-to-device technology used is WiFi-direct. Using the Monsoon power meter, we measured the power profile for sending a small (ping) packet to a server on a Galaxy S3 phone, as well as the power profile for device scanning. Table 3 lists the power values

[8] http://en.wikipedia.org/wiki/Cross-origin_resource_sharing

[9] http://www.comp.nus.edu.sg/~kartiks/nusdtn/

Table 3: Monsoon power meter measurements

Operation	Power (mW)
CPU (asleep)	25
CPU (awake)	85
LTE (active)	2000
LTE (tail)	490
WiFi (scan)	300

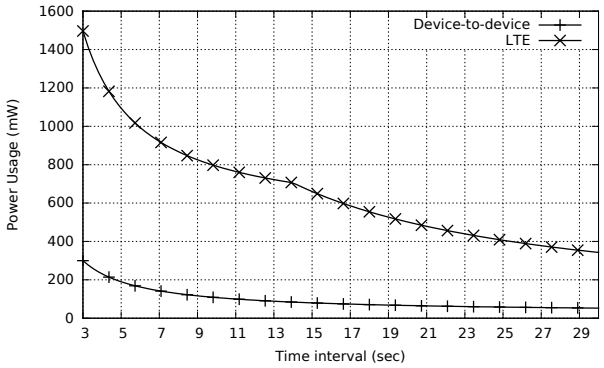


Figure 3: Power of server (LTE) v/s device-to-device (WiFi)

measured. Unlike WiFi-direct, LTE suffers from a long tail (more than 12 seconds) after the packet has been sent.

Based on these measurements, we calculated the power consumption at different frequency of location updates and scans, shown in Figure 3. For the same latency, server-based approaches would consume higher power. For example, at a (reasonable) 20 second worst-case latency, the power saving of using device-to-device technologies is 86%. Thus, use of device-to-device communication can benefit future proximity applications by being more power-efficient. Although the power to transfer web apps is not included in Figure 3, we expect that the higher bandwidth between devices would make such transfers faster and lower power than LTE as well.

5.2 Deployment Latency

The latency between a device arriving at a place of interest and receiving the deployed web app is important to users. As explained earlier, this is a function of the discovery interval used (set to 10 seconds in our deployment tool). We measured the deployment latency and found it to be 6.4 seconds on average, which is reasonable. This can be modified to tradeoff savings in power (Figure 3).

5.3 Performance Overhead

In DTN, devices exchange information when they come into range of one another. It is critical that data is transferred as quickly as possible during the limited contact duration time. Here we measure the performance overhead introduced by the framework during the data transfer.

Two aspects of the framework cause overhead during communication: the Inter-process communication (IPC) where data is copied from the DTN application to the framework, and the API Proxy/Stub (see Figure 1). We expect the Proxy/Stub overhead to be independent of data size, since it does not involve any data copying. We expect IPC overhead to vary linearly with data size. Note that IPC occurs only when data is initially passed from the DTN app to the routing protocol. After the initial copy, it is buffered in the framework for forwarding to other devices opportunistically.

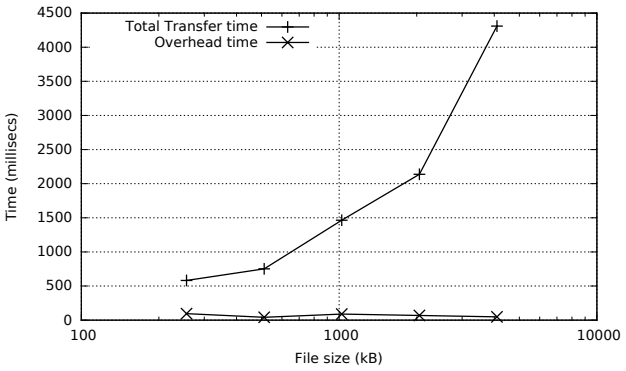


Figure 4: Overhead during file transfer

Table 5: Memory Overhead

Part of the Framework	Memory
API Proxy (application-side)	1.1 MB
Framework Service (nothing plugged in)	8.9 MB
Framework Service (2 APIs + 2 Protocols, no messages)	9.1 MB

To reduce IPC overhead for large data (audio, pictures), the framework allows data to be transferred from the app via files in the phone's storage. This removes the need for data copy, and is more convenient for the app. Only (optional) 'metadata' needs to be copied via IPC. For example, a mall application advertising a special sale would transfer product photos via files, while smaller textual data like name and price would be transferred via IPC.

Figure 4 shows the overhead involved for file transfer between two phones running the framework, using TCP over a 802.11b interface. Each data point is an average of 30 trials. The overhead is small compared to the transfer time, especially for moderate to large file sizes. Table 4 shows a breakdown of the overhead. As expected, Proxy/Stub overhead is independent of data size. IPC overhead is due to the large metadata size (32 kB) used in the experiment, but is independent of the file size. Overhead is 6% and lower for moderate to large file sizes. If IPC is not involved (i.e. data is already buffered), then the overhead is even lower.

5.4 Memory Overhead

Here we measure the extra memory used by the framework. In our implementation, the API Proxy class occupies memory in the application memory space. The framework itself runs as a service, and occupies memory separately from the application. Table 5 shows the memory overhead, evaluated using the Eclipse Memory Analyser.

Android imposes a limit on heap, which varies with OS version. Assuming a 32 MB limit, this leaves 23 MB for buffering. If each message is 1 MB, we can buffer 20 messages, which is too few. However, bulk of data is in the form of pictures/audio stored as files on the `sdcard`, and not in heap. The heap contains only the message's metadata. If metadata is 32 kB, the phone can buffer about 700 messages. As newer phones have larger RAM, we do not expect the 9 MB overhead to be significant.

Table 4: Breakdown of Framework Overhead during File Transfer (time is in millisecs)

File size	Metadata size	Proxy	IPC	Stub	Transfer time	Overhead%	Overhead% (no IPC)
256 kB	32 kB	5.66	81.03	8.40	581.12	16.36	2.42
512 kB	32 kB	8.93	25.03	8.20	752.50	5.60	2.28
1 MB	32 kB	6.53	76.09	6.75	1464.98	6.10	0.91
2 MB	32 kB	7.32	25.04	36.18	2137.29	3.21	2.04
4 MB	32 kB	13.47	25.67	8.66	4309.97	1.11	0.51

6. DISCUSSION AND FUTURE WORK

Use of WebSockets: We will be re-writing our code to use WebSockets, now increasingly supported in mobile browsers, suitable for push-based notifications of messages received, to replace AJAX long polling.

Security: While web apps run in the browser sandbox, protocols loaded in the framework have dangerous access to Android libraries. In the future, we will use OSGi's fine-grained access control to restrict a protocol's access.

7. CONCLUSION

In this paper, we proposed a dynamic framework for deployment of localized DTN web apps. The apps free users of the burden of installing multiple native apps on the phone. They are easy to open/close in a browser, and operate only during proximity interactions. To demonstrate their usefulness, we wrote an app for bus stops to help the physically disabled. Our analysis shows that the framework has low overhead. In the future, we will enhance the web app support, and analyse the performance in a real setting.

8. ACKNOWLEDGMENTS

This research was supported in part by the National Research Foundation Singapore through the Singapore-MIT Alliance for Research and Technology (SMART) program.

9. REFERENCES

[1] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[2] Erik Nordström, Per Gunningberg, and Christian Rohner. Haggle: a data-centric network architecture for mobile devices. In *Proceedings of the 2009 MobiHoc S3 workshop on MobiHoc S3*, MobiHoc S3 '09, pages 37–40, New York, NY, USA, 2009. ACM.

[3] M. Skjegstad, F.T. Johnsen, T.H. Bloebaum, and T. Maseng. Mist: A reliable and delay-tolerant publish/subscribe solution for dynamic networks. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–8, 2012.

[4] A. Petz and C. Julien. The madman middleware for delay-tolerant networks. In *Poster at HotMobile 2010 (Proceedings of the 11th workshop on Mobile computing systems and applications)*, 2010.

[5] Mauro Caporuscio, Pierre-Guillaume Raverdy, Hassine Moungla, and Valerie Issarny. ubisoap: A service oriented middleware for seamless networking. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 195–209, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, and Christophe Diot. Mobiclique: middleware for mobile social networking. In *Proceedings of the 2nd ACM workshop on Online social networks*, WOSN '09, pages 49–54, New York, NY, USA, 2009. ACM.

[7] Frédéric Guidec and Yves Mahéo. Opportunistic content-based dissemination in disconnected mobile ad hoc networks. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, UBICOMM 2007, pages 49–54, 2007.

[8] H. Ntareme and S. Domancich. Security and performance aspects of bytewalla: A delay tolerant network on smartphones. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pages 449–454, 2011.

[9] Jörg Ott and Dirk Kutscher. Bundling the web: Http over dtn. *Proceedings of WNEPT*, 2006.

[10] Aruna Balasubramanian, Yun Zhou, W Bruce Croft, Brian Neil Levine, and Aruna Venkataramani. Web search from a bus. In *Proceedings of the second ACM workshop on Challenged networks*, pages 59–66. ACM, 2007.

[11] Jay Chen, Lakshminarayanan Subramanian, and Jinyang Li. Ruralcafe: Web search in the rural developing world. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 411–420, New York, NY, USA, 2009. ACM.

[12] Mikko Pitkanen, Teemu Karkkainen, and Jörg Ott. Opportunistic web access via wlan hotspots. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 20–30. IEEE, 2010.

[13] Anders Lindgren. Social networking in a disconnected network: fbdtn: facebook over dtn. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 69–70. ACM, 2011.

[14] L Peltola. Dtn-based blogging. *Helsinki University of Technology, Department of Communications and Networking*, 2007.

[15] Hao Zhuang, Hervé Ntareme, Zhonghong Ou, and Björn Pehrson. A service adaptation middleware for delay tolerant networks based on http simple queue service. In *Proc. of the 6th Workshop on Networked Systems for Developing Regions (NSDR'12)*, 2012.

[16] Darren Carlson, Bashar Altakrouri, and Andreas Schrader. Ambientweb: Bridging the web's cyber-physical gap. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 1–8. IEEE, 2012.