Delay Tolerant Network on Android Phones: Implementation Issues and Performance Measurements

Rerngvit Yanggratoke
School of Electrical Engineering, The Royal Institute of Technology (KTH), Stockholm, Sweden
E-mail: rerngvit@kth.se

Abdullah Azfar
Dept. of Computer Science and Information Technology, Islamic University of Technology (IUT), Gazipur, Bangladesh
E-mail: azfar@iut-dhaka.edu

María José Peroza Marval, Sharjeel Ahmed
School of Information and Communication Technology, The Royal Institute of Technology (KTH), Stockholm, Sweden
E-mail: {mjpm, sharjeel}@kth.se

***Abstract*— Many regions of the world do not have access to the Internet due to lack of proper communication infrastructure. Delay Tolerant Network (DTN) provides communication in a challenging network condition such as high communication delay and intermittent connectivity. DTN is a promising solution to solve lack of connectivity problems in developing regions such as rural areas. DTN works in a store-and-forward approach, where the data is stored in a server and forwarded to a suitable carrier. The android phones can be made DTN capable to become a carrier for DTN bundles. Android phone is one of the front runners as the DTN carrier because of its portability and increasing popularity. In this paper, we have described the implementation of DTN on Android phone and the performance measurements including DTN bandwidth and battery consumption.**

***Index Terms*— Delay Tolerant Network (DTN), Wi-Fi, Android phone, Bundle, Access Point, Server.**

I. Introduction

The advancement in development of technologies has made it possible to carry powerful mobile devices. These devices are capable of Bluetooth and Wi-Fi connectivity and storing large amount of data. These devices can be used to store even gigabits of data because of the improved capacity of portable storage such as SD Card. As a result, in some conditions, physically carrying the network data in the devices can be cheaper than setting up a network infrastructure [1]. For example, it might be more cost effective to make a person carry gigabits of data everyday in his/her smart phone across twenty villages while travelling by his car or bike, rather than building an infrastructure in the village. This can be accomplished by the implementation of Delay Tolerant Networking (DTN) in the mobile devices.

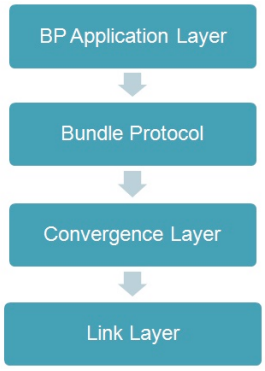

Figure 1. DTN layer architecture

DTN provides communication in highly stressed environments such as variable delays, discontinuous connectivity and high bit error rate. The current DTN implementation is based on Bundle Protocol (BP) [2]. The BP relies on the services of a convergence layer. The convergence layer converts BP messages to link layer and vice versa in order to practically make the network communication. The link layer can be any link layer protocol. For our implementation we used TCP/IP as link layer protocol. This architecture is shown in Fig. 1.

The protocol data units (PDU) of DTN are known as "bundles". The bundles may be split up into fragments and the source and the destination of the bundles are identified by the end point identifiers [3]. When the bundles are created, they are specified with the bundle expiration time. Bundles may move from one DTN node to another in a broken path [4]. Bundles may arrive to a node and find the link between the current node and the next node broken. The bundles reside in the node until the expiration time is over.

Manuscript received August 5, 2010; revised June 7, 2011; accepted July 15, 2011.

doi:10.4304/jcm.6.6.477-484

Android is an operating system designed for mobile devices [5]. The Android Software Development Kit (SDK) provides all the tools and APIs for developing. The application development is made with a new Java library named Java Android library [6]. Our target mobile device platform is Android because of its openness and profound industry support. The source code for the platform is publicly available and currently 50 leading telecom, hardware, software companies are registered as its members [7]. Thus, our challenge is to implement the protocol on the platform for facilitating the exchange of data between the mobile device and the network.

Our primary contributions are as follows. Firstly, we have developed an Open source DTN implementation on Android platform that is fully compatible with RFC 5050 [2]. Secondly, we are the first to report power consumption on DTN implementation. Thirdly, we have found that the power consumption for upload is lower than download DTN data. This insight supports people deploying DTN in practice. In particular, the deployment should favor upload over download in order to minimize power consumption.

The rest of the paper is organized as follows: related works are discussed in section 2. The implementation issues of DTN on Android phones are discussed in section 3. The performance measurements of DTN on android phone are discussed in section 4. A general discussion is presented in section 5. Finally, some conclusions are drawn in section 6.

II. Related Works

Our aim is to develop an Open source DTN implementation on Android platform that is compatible with RFC 5050 [2]. We target for Opensource software for others to be benefited from it and further improvement by the DTN community. And, it must be compatible with the standard for interoperability with other existing implementations. There are several DTN implementations available including DTN2 [8], IBR-DTN [9], Spindle [10], and Java BP-RI [11]. Nonetheless, none of them is compatible with the Android platform, Open source license, and the standard. As a result, our solution is unique.

DTN2 is an Open source implementation developed by the DTN research group. It is mainly implemented as a testing of basic DTN functionalities and it is not very efficient [9]. It follows the standard and aims to "embody the components of the DTN architecture, while also providing a robust and flexible software framework for experimentation, extension, and real-world deployment" [8]. It heavily relies on standard C++ library and Oasys system library. But neither of them is available on the Android platform.

IBR-DTN is another Open source implementation specially designed for embedded system such as Wi-Fi router [9], to allow access points (APs) to be used as a compact "plug-and-play" DTN-Modules for vehicular or stationary applications. Even though it is a great idea and it has been tested to work efficiently - consumes less main memory and has more throughput than DTN2 and our implementation – our solution is more portable, less expensive and the infrastructure involved is less complex since it is based on mobile devices. The IBR-DTN platform is written in C++ but the development language for Android platform is Java. As a result, this implementation is also not compatible with our solution.

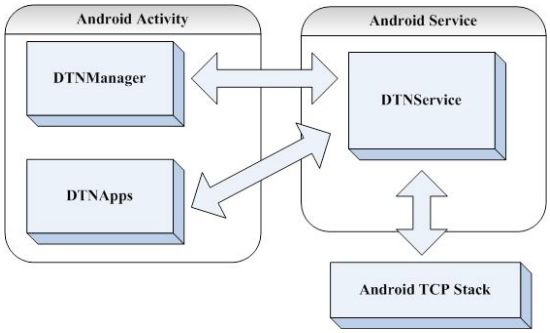

Figure 2. Major software components

Spindle is a DTN implementation built on top of DTN2 for using in the military context [10]. It follows the standard and, unlike our work, has advanced features such as efficient routing algorithm, complex name management, and knowledge-based DTN. The source code is not publicly available. Furthermore, similar to DTN2 and IBR-DTN, it is developed in C++ and as a result incompatible with our platform.

Java BP-RI is a DTN implementation written in Java. It does not comply with the standard because it follows the draft version 4 of the Bundle Protocol. Even though it is written in the same language for Android platform, it uses the Java-RMI feature, which is not available on the platform [11][12].

III. Implementation Issues

Three implementation issues on Android platform are discussed in this section. Section 3.A elaborates on technical challenges. Section 3.B explains overview of the implemented software components. Section 3.C shows the internal design of DTNService, which is the main component responsible for DTN logic.

A. Technical Challenges

During the beginning of our work, we identified three viable alternatives to implement DTN on Android. The alternatives are porting from DTN2 [8], porting from Java BP-RI [11], and implementing from scratch in Android. We finally decided to take the first option, porting from DTN2, because it is the best option in term of flexibility to adapt to Android platform, software license, efforts required for existing DTN developers to understand and use the result. The details of the analysis are discussed in [13].

While porting DTN implementations from DTN2, we face three major technical challenges including size of the software, differences between programming languages, and missing of system programming library.

The size of DTN2 software is very large. It composes of approximately 4MB of source code and 394 source files. Porting huge software to another platform is not trivial. In particular, if we made even a small design

mistake, it will cost us a lot of time to correct. As a result, one has to be very careful in designing and implementing software as this size.

While DTN2 is implemented in C++ [8], the major language for Android is Java. As a result, we have to find workarounds when there are some features available in C++ but not in Java. For example, in C++, one can inherit multiple classes but this is not possible in Java. We redesign and implement new programming classes in order to support this.

DTN2 relies heavily on OASYS, system-programming library written in C++. For example, many main data structures in DTN2 are inherited from classes in OASYS. However, the aforementioned library is not available in Android SDK. Thus, porting DTN2 to Android is not straightforward. We have to cope with this either by finding equivalent classes in the SDK or implementing from scratch if they are not available.

B. Software Components Overview

There are three software components in the implementation. They are: DTNService, DTNManager, and DTNApps. These components interact with each other and with the Android TCP/IP stack. The interactions are illustrated in Fig. 2.

DTNService is a backend application serving DTN communication. As a result, even though the user is using other applications such as making a phone call or reading text messages, he/she is still able to send/receive DTN transmission. To be able to run in backend in the Android platform, this component is implemented as Android Service [14]. And, this component uses TCP/IP stack of the Android platform to achieve network communication.

Because DTNService is running in backend, there is a need for a user interface for interacting with the service. This is where DTNManager shows up. It is a frontend application for the user to configure, monitor, and manage the DTNService. This frontend application is designed as Android Activity [15]. The user interface for DTN manager is shown in Fig. 3. The DTNManager supports several management actions including starting, stopping, restarting, configuring, and monitoring status of DTNService.

DTNApps are the applications running on top of the DTNService. As a result, they are in BP application layer as shown in Fig. 1. Two sample DTN applications developed are DTNSend and DTNReceive. DTNSend is a DTN application allowing users to send text messages over DTN. On the other hand, DTNReceive is a DTN application allowing users to receive text messages over DTN. Because both of them are frontend applications similar to DTNManager, they are mapped to Android Activity [15].

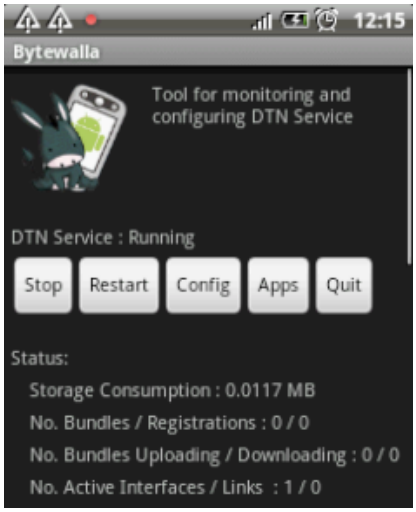

Figure 3. DTN Manager, GUI for managing DTN service

C. DTNService Internal Design Summary

The design of DTNService follows very closely the design of DTN2, the DTN reference implementation written in C++ by the DTNRG [8]. Hence, the design has similar characteristics of the reference implementation by which it is a modular design. In other words, the system composes of several plug-in components working together. When there is a new function to be added to a particular component, other parts need not to be changed. This makes the system very easy to extend. For example, if anyone would like to add new routing algorithm to the DTNService, he/she can simply extend the BundleRouter class without making any modification to the other parts.

The DTNService composes of nine running modules including Bundle Daemon, Contact Manager, TCP Convergence Layer, Discovery, Persistent Storage, Registration, Bundle Router, Fragmentation manager and APILib.

DTNService is an event driven system. There are several types of events, for example, bundle receiving event, bundle transmitted event, or contact initiation event. The system works according to the event handling functions defined in event handling components including Bundle Daemon, Bundle Router, and Contact Manager.

The communication among the running modules with each other in order to respond to an event is illustrated in Fig. 4. The arrow represents the communication between each module. The brief summaries of all modules are discussed below.

Bundle daemon is the main event handler of the system. It is the central processing unit and responsible for communicating with other module for processing the bundle event. Every bundle event will be checked first by the daemon. If the Bundle daemon determines that other two event processing components including Bundle Router and Contact Manager should process the event as

well, it will forward event to the components. Otherwise, it will just remove the event.

Contact Manager is the service in charge of detecting new opportunities of connections. Each opportunity of connection ("opportunistic link") with a neighbor ("contact") is under the control of Contact Manager which also does the scheduling of the links.

TCP Convergence Layer is the transport mechanism over TCP that the DTN application uses for transmitting bundles to a next hop.

Discovery is the method by which other nodes can be aware of the "existence, availability, and addresses of other DTN participants" [16]. Discovery is based on IP protocol where each node sends and listens to IP UDP announcements to discover remote neighbors.

Persistent storage is the storage mechanism that stores data objects permanently on the disk. Persistent storage is a generic implementation. It can store any kind of objects on the disk. It also stores unique identifier of every available object in the database to identify the object.

Registration is defined as "A registration is the state machine characterizing a given node's membership in a given endpoint" [2]. This module handles the specified registration created from the Bundle Daemon. Every bundle received by the daemon will be checked with this module and if it is matched, it will be delivered to the registration for further processing.

Bundle router is the main decision maker regarding forwarding the bundles. Depending on the implementation, it may contain the routing table as a database for making the routing decision. The router implemented here is the static Bundle Router which mainly relies on the routing table database configured before the system starts. The router is used by the Bundle Daemon for checking and finding an appropriate next hop to send the DTN bundle. Then, if it finds the next hop, it will forward the DTN bundle to the Contact Manager for initiating a communication with the hop.

Fragment manager makes fragments of the large bundles. It also keeps the state of all the fragmentary bundles and partially received bundles. Finally it reconstructs the whole bundle from the received fragments.

APILIb provides Application Programming Interface (API) for making a DTN application on Android platform. The API will communicate with Bundle Daemon to achieve the API calls. As a result, this is the channel for other components such as DTNApps to programmatically use DTNService. This component is implemented by Android Binder [17]. A DTN application developer can bind with this Binder. Then, the developer can call an API from the Binder interface. For example, if the developer would like to write an application for sending a DTN bundle, he/she can call "dtnsend" Binder interface provided in this component.

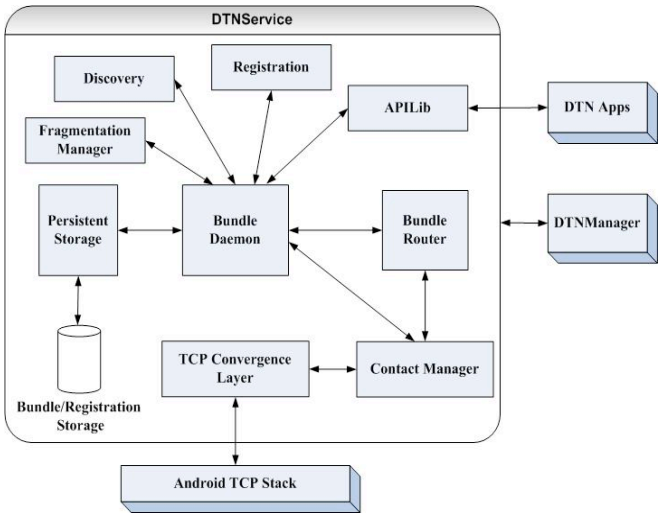

Figure 4. DTNService internal design

IV. Performance Measurements

Our test environment consists of two servers running DTN2 and an Android phone as shown in Fig. 5. The specification of the servers and android phone are shown in table 1 and 2.

We made two custom codes in order to make the tests. The first code is a script to generate DTN bundles with different sizes. The script inputs parameters including the number of DTN bundles, initial bundle size, the incremental value, number of repetition of bundles with the same size, source Endpoint ID, and destination Endpoint ID of the DTN bundles.

The second script is for recording the performance of the Android phone. This script was added to the Android DTN system. For getting DTN Bandwidth, the recording code supports recording of the upload/download time of each individual DTN bundle. For getting the battery consumption, the code hooks on the battery status change API of the Android phone. Every time the battery level is changed, the upload/download size is recorded.

In order to evaluate the performance, 105 DTN bundles are generated to be routed from one server to the other. The parameter for bundle repetition was set to 3. As a result bundles with 35 different sizes with each size having 3 instances were generated. This was done to get the average bandwidth value for bundle of each size. The initial bundle size was 100KB. Each of the remaining bundle size was incremented by 100KB. So the size of the last 3 bundles was 3.5MB.

TABLE I.
SPECIFICATION OF THE DTN2 SERVERS

CPU	Intel Celeron 1.8 GHz
System Memory	512 MB ECC DDR
Total Hard Disk Size	40 GB
Operating System	Linux Ubuntu 8.04
DTN Software	DTN2 version 2.6

TABLE II.
SPECIFICATION OF THE ANDROID PHONE

Model	HTC Tattoo
Processor	Qualcomm® MSM7225™, 528 MHz
Operating System	Android™ 1.6
DTN Software	Android DTN
Phone Memory	256 MB
Network Interface	Wi-Fi®: IEEE 802.11 b/g
Battery	Rechargeable Lithium-ion battery Capacity: 1100 mAh

A. Bandwidth Analysis

The time for downloading the 105 bundles is shown in Fig. 6. There was 189013.1 KB of data in total. In this case the bundles were downloaded from the server to the Android phone. The graph shows a linear increase in download time with the increase of bundle size. This follows a linear equation of the general form y=mx+b. This is expected because while downloading the bundles, there were no other download activities running which can consume the bandwidth.

The average download bandwidth is calculated below:

Average Download Bandwidth
= Total Bundle Size / Total Download Time
= 189013.1 KB / 3680.36 s
= 51.36 KB/s

The bandwidth analysis for upload time is shown in Fig. 7. The bundles were sent from the Android phone to the other server. As expected, the upload time increased linearly with the increase of bundle size. An average value for the upload bandwidth is as follows:

Average Upload Bandwidth
= Total Bundle Size / Total Upload Time
= 189013.1 KB / 3134.999 s
= 60.29 KB/ s

A comparative study between the download and upload bandwidth is shown in Fig. 8. For small bundles of size 100 KB, the download and upload times are identical. Up to the bundle size of 1.5MB, there is a very small difference in upload and download time. After that, the difference increases as the download time increases. A significant observation made from this figure is that the download time is always higher than the upload time.

B. Battery Consumption Analysis

The battery consumption was recorded with a fully charged battery of HTC tattoo phone. The 105 bundles had a total of 189013.1 KB = 184.68 MB of data.

Fig. 9 shows the cumulative battery consumption for downloading the bundles. No other applications were run in the phone during the period the data were recorded. This ensured the battery power was consumed only by the DTN software. The battery consumption rate was high for first 22 MB of data. But after that there was a drop in battery consumption till 75 MB of data. Then it went high a bit and continued in a linear way. At the end 44% of the total battery charge was consumed for downloading 184.68 MB of data. The average battery consumption rate is calculated as follows:

The HTC Tattoo phone has 1100 mAh for 100% battery power. This means 1 % battery power= 110 mAh. As a result, the Average DTN download battery consumption rate

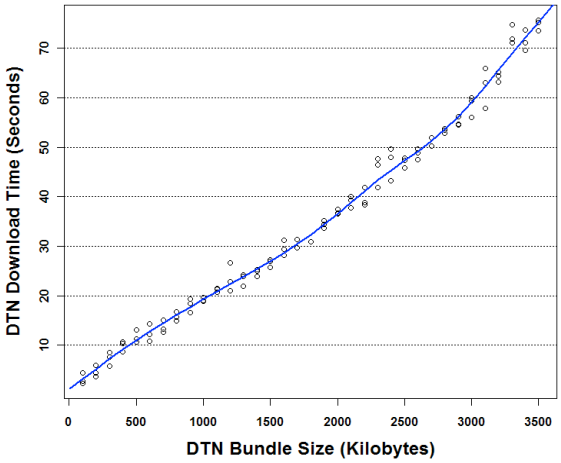

Figure 6. Download bandwidth

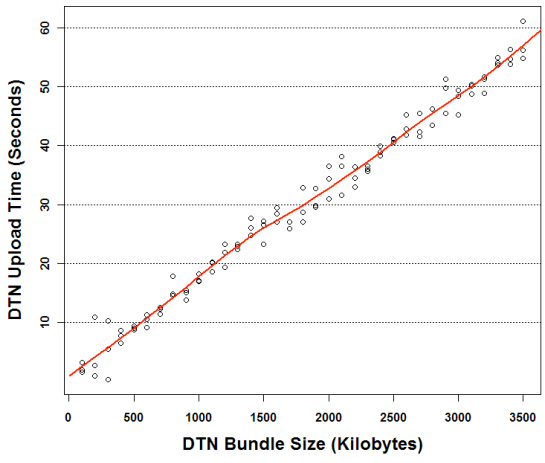

Figure 7. Upload bandwidth

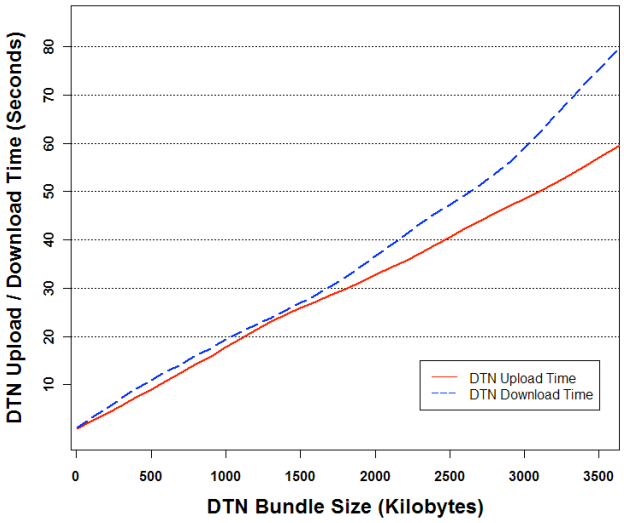

Figure 8. Comparisons between download and upload bandwidth

= 44*110 mAH / 184.68 MB
= 26.20 mAH/MB

Fig. 10 shows the cumulative battery consumption for uploading bundles. Unlike download battery consumption, the upload battery consumption maintains a very linear increase. 36% of the total battery power was consumed for uploading 184.68 MB of data. The average battery consumption rate is calculated as follows:

Average DTN upload battery consumption rate
= 36*110 mAH / 184.68 MB
= 21.44 mAH/MB

A comparative study is shown between the cumulative download and upload battery consumption in Fig. 11. The battery consumption for uploading data is always lower than the battery consumption for downloading data.

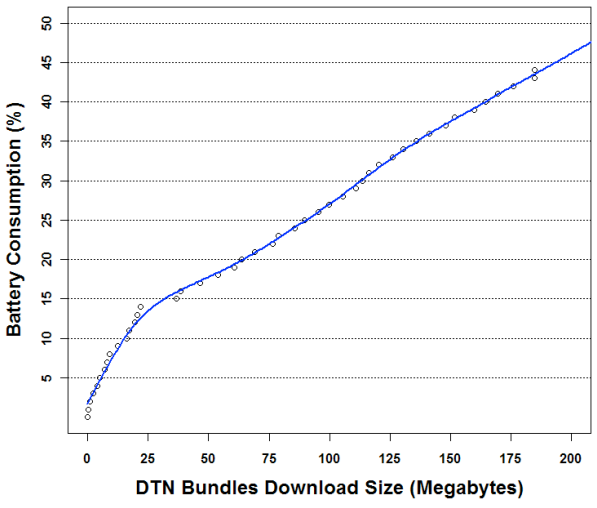

Figure 9. Cumulative battery consumption for download

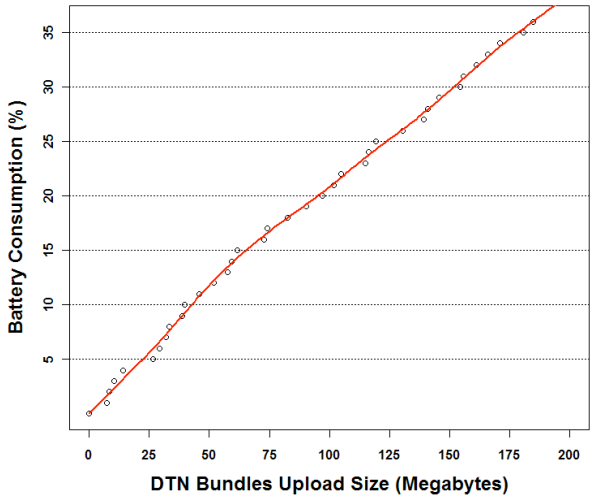

Figure 10. Cumulative battery consumption for upload

V. Discussion

Our implementation has some limitations. The routing algorithm is based mainly on the routing entries in the routing table. The entries are specified in the configuration file before the DTNService is started. As a result, all the DTN routes have to be known before the system begins operation. This is suitable for a simple network topology, for example, when there are few DTN nodes in the system such as servers in the villages, city, and the data carrier. In real world situation where there are possibly much more DTN nodes and complex network configurations, this routing table will be very difficult to manage and may make the whole network inefficient.

If a disruption in the connection happens, the bundle is discarded and must be transmitted again once the connection is back. This approach is not suitable in environments with often disruptions in connectivity. The

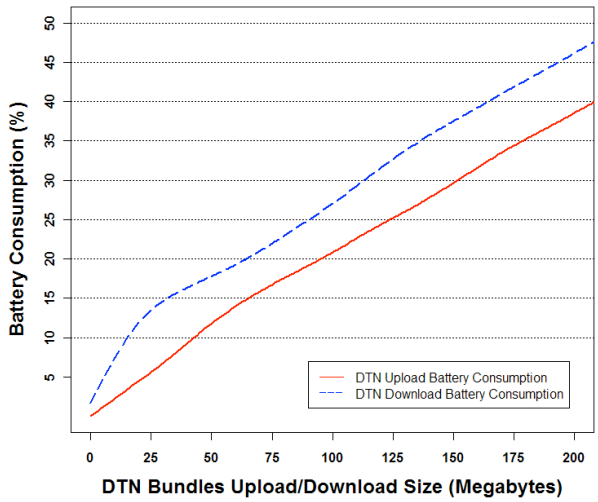

Figure 11. Comparison between battery consumption for download and upload

sender node might have to send the same bundle several times, and might not be able to transmit it completely before the bundle expires.

The DTNAPI implemented here is a shared process application programming interface (API). As a result, all the DTN applications implemented in this project runs in the same process as DTNService. This is fine if there are no DTN applications consuming too many resources running simultaneously with the DTNService. But this is not the best practice for making an API for interacting with a backend service on Android platform. This is because an Android application running in the same process shares the Dalvik Java Virtual machine (Dalvik Java VM) [5]. And, a Dalvik Java VM will be shutdown if it consumes too many resources. As a result, if there is one DTN application exhausts system resources such as main memory, the DTNService and other DTN applications will be affected as well.

Considering the performance issues, this is the first time where the power consumption on DTN implementation has been measured. From our measurements we can say that download time is always higher than the upload time. On the other hand, battery consumption for uploading data is always lower than the battery consumption for downloading data. So, upload data consumes less power than download data. Moreover, upload power consumption linearly increases with bundle size but download power consumption does not show this property. In particular, downloading consumes high power for first 0 - 40 MB and consumes less power and increases linearly for 40 - 200 MB.

VI. Conclusions

Implementation of DTN on Android platform is a relatively new concept. The goal of this paper was to develop DTN software for Android platform. The software supports static routing and shared process application programming interface environment. A lot of work can be done based on this. Effective routing algorithms / protocols are required for real world deployment. Both Delay Tolerant Routing for Developing Regions [18] and Probabilistic Routing Protocol (PRoPHET) [19] are the promising routing algorithm / protocols to be implemented. Reactive fragmentation can be implemented for supporting environments with disruptions in connectivity. The sender node can start transmitting an entire bundle, and in case of a failure in the connection, both the remaining and the received portion of the bundles can be converted into valid fragments of DTN bundles. An API can be created, which will be called from different processes other than the DTNService to solve the problem of resource exhaustion. This can be accomplished by a technique called Inter-process communication (IPC). IPC on Anroid platform is done by using a combination of AIDL [20] and Parcelable [21]. The security concerns were not taken into account in this implementation. The main focus was to make the network working. Security mechanisms can be developed by implementing the Bundle Security Protocol [21] on the Android phone.

Acknowledgment

This work was a part of a Communication Systems Design (CSD) project in the Royal Institute of Technology (KTH), Sweden. We would like to thank Professor Björn Pehrson, Marco Zennaro, Avri Doria, Elwyn Davies and Hervé Ntareme for their continuous support throughout the project.

References

[1] Networking for Communications Challenged Communities, http://www.n4c.eu/N4Cproject.htm, last visited: December 2009.

[2] K. Scott, S. Burleigh, "Bundle Protocol Specification", *IETF RFC 5050*, IETF Network Working Group, November 2007.

[3] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture", *IETF RFC 4838*, IETF Network Working Group, April 2007.

[4] Keith Scott, "Disruption Tolerant Networking Proxies for On-the-Move Tactical Networks", *Military Communications Conference, 2005. MILCOM 2005. IEEE*, Vol. 5, pp. 3226–3231, October 2005, ISBN: 0-7803-9393-7.

[5] Android Developers, "What is Android", http://developer.android.com/guide/basics/what-is-android.html, last visited: December 2009.

[6] Java RMI, http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp, last visited: December 2009.

[7] Open Handset Alliance, http://www.openhandsetalliance.com/oha_members.html, last visited: December 2009.

[8] DTN2, *DTN Reference implementation by the DTNRG*, http://www.dtnrg.org/wiki/Code, last visited: December 2009.

[9] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, "IBR-DTN: an efficient implementation for embedded systems", *Proceedings of the Third ACM Workshop on Challenged Networks*, San Francisco, California, USA, 2008, pp. 117-120, ISBN:978-1-60558-186-6.

[10] R. Krishnan et al, "The SPINDLE Disruption-Tolerant Networking System," *Proceedings of Military Communications Conference, 2007. MILCOM 2007 IEEE*, Orlando, FL, USA, October 29-31, 2007, pp. 1-7, ISBN: 978-1-4244-1513-7.

[11] J. Deshpande, *Java BP-RI*, http://irg.cs.ohiou.edu/ocp/downloads/BP-RI-Impl-Doc.pdf, last visited: December 2009.

[12] J. Agüero, M. Rebollo, C. Carrascosa, V. Julián, "Towards on embedded agent model for Android mobiles", *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Dublin, Ireland, Article No. 37, 2008, ISBN: 978-963-9799-27-1.

[13] A. Azfar et al, "Bytewalla: Delay Tolerant Networks on Android phones Final Report", http://www.tslab.ssvl.kth.se/csd/projects/092106/sites/default/files/Bytewalla_Final_Report_v1.0.pdf, last visited: December 2009.

[14] Android Developers, "Android Service", http://developer.android.com/reference/android/app/Service.html, last visited: December 2009.

[15] Android Developers, "Android Activity", http://developer.android.com/reference/android/app/Activity.html, last visited: December 2009.

[16] R. Beverly and D. Ellard, "*DTN IP Neighbour Discovery (IPND) draft-irtf-dtnrg-ipnd-00*", http://tools.ietf.org/html/draft-irtf-dtnrg-ipnd-00, last visited: January 2010.

[17] Android Developers, "Android Binder", http://developer.android.com/reference/android/os/Binder.html, last visited: December 2009.

[18] M. Demmer and K. Fall, "*DTLSR: Delay Tolerant Routing for Developing*

Regions", http://www.cs.berkeley.edu/~kfall/papers/dtlsr-nsdr07.pdf , last visited: December 2009.
[19] A. Lindgren and A. Doria, "*Prophet routing protocol*", http://www.dtnrg.org/docs/specs/draft-lindgren-dtnrg-prophet-02.txt, last visited: December 2009.
[20] Android Developers, "Designing a Remote Interface Using AIDL", http://developer.android.com/guide/developing/tools/aidl.html, last visited: December 2009.
[21] Android Developers, "Android Parcelable", http://developer.android.com/reference/android/os/Parcelable.html , last visited: December 2009.

Rerngvit Yanggratoke is doing his PhD with topic "Autonomous Resource Allocation in Cloud Computing" in School of Electrical Engineering with The Royal Institute of Technology (KTH) in Stockholm, Sweden. He got his MS in Erasmus Mundus NordSecMob program specialized in Security and Mobile Computing from Aalto University School of Science and Technology (TKK), Finland and Royal Institute of Technology (KTH), Sweden. He received his BSc degree in Computer Engineering from Chulalongkorn University, Bangkok, Thailand in 2006. He worked as a software engineer in the company named Openface Internet during the period April 2006 – April 2008. He also served as a system administrator in Dhammasociety Fund, Thailand during the period April 2008 – August 2008. He was provided full scholarship for two years studying in his MS from the European Union. His major research interests are delay tolerant networking, information system security, and high performance distributed system.

Abdullah Azfar is working as an Assistant Professor in the Computer Science and Information Technology department of Islamic University of Technology (IUT), Gazipur, Bangladesh. He got his MS in Erasmus Mundus NordSecMob program specialized in Security and Mobile Computing from Norwegian University of Science and Technology (NTNU), Norway and The Royal Institute of Technology (KTH), Sweden in 2010. He received his BSc degree in Computer Science and Information Technology from Islamic University of Technology (IUT), Gazipur, Bangladesh in 2005. He served as a lecturer in Islamic University of Technology during the period March 2006 – July 2008 and July 2010 – December 2010. He also served as a lecturer in Prime University, Dhaka, Bangladesh during the period October 2005 – February 2006. He received the Erasmus Mundus scholarship from the European Union for his MS studies. His research interests are mainly focused on Information Security, VoIP Communication, Cryptography and Mobile Computing.

María José Peroza Marval got her MS in Internetworking program specialized in Networking and Radio Communication Systems in Royal Institute of Technology (KTH), Sweden. She received her BSc degree in Electronic Engineering from Simon Bolivar University, Caracas, Venezuela in 2005 (graduated with an Honourable Mention for her bachelor thesis). She worked as a Network Engineer in the company named MOVILNET, Venezuela, during the period May 2006 – October 2007. She also worked as an Operations Analyst in Dayco Telecom, Venezuela, during the period September 2005 – May 2006. Her research interest is mainly focused on conventional Networking and Optical Networking.

Sharjeel Ahmed is doing his MS in Information and Communication Technology Entrepreneurship in Royal Institute of Technology (KTH), Sweden. He received his BSc degree in Computer Science from COMSATS Institute of Information Technology, Lahore, Pakistan in 2007. He worked as a software engineer during the period March 2007 – August 2008. He has innovated in a number of fields of information technology and software development. With his research-based ideas and a vision apart, he has evolved from being a software engineer to an internet entrepreneur and now working on his ideas to start new startups.