A Comprehensive Data Management Framework for Opportunistic Communication on Mobile Phones

Sathyam Doraswamy
IIT Delhi

Arvind Subramaniam R
IIT Delhi

Aaditeshwar Seth
IIT Delhi

1. INTRODUCTION

Several of our non-profit partners working in rural areas complain about poor data connectivity from their mobile phones. To instrument this, we deployed a simple application on Android mobile phones of two field staff located in the state of Jharkhand in India to continuously probe 2G GPRS EDGE connectivity across several days. We found that the connectivity was quite flaky and underwent frequent disruptions as the staff moved around for their work. This motivated us to develop a comprehensive data management framework that can run on mobile devices and help application developers cope with several issues including communication on flaky connections, data synchronization, support for transactions, and consistency management. Most previous work in the area of supporting communication in poorly connected regions has focused on connection management and session persistence across disconnections, while we focus more on data management challenges that arise in these scenarios. We have built and deployed an application for media transfer using this framework, and are now using this experience to improve the framework.

Our connectivity-testing Android application logged signal strengths and HTTP ping latencies to www.google.com to check for connection availability, and uploaded the traces every few hours to our server for analysis. The application was deployed on Samsung Galaxy Fit phones provided by us to two staff working with our field partner. Figure 1 shows the HTTP ping latencies (~ 2RTTs) and availability plotted on the map of Ranchi, the main city in which the field staff are located. The points in red indicate no connectivity, green points indicate moderate latency and the blue ones indicate high latency values. As can be seen, the mobile devices often run into areas of poor availability, and we found the mean time between disconnections and the maximum disconnection period to be 83mins and 30mins respectively.

2. SYSTEM OVERVIEW

Based on the state of 2G connectivity we observed, and feature requests obtained from conversations with several social sector organizations working in rural areas, we propose the following communication framework for flaky Internet environments.

DEV '13, January 11-12, 2013 Bangalore India

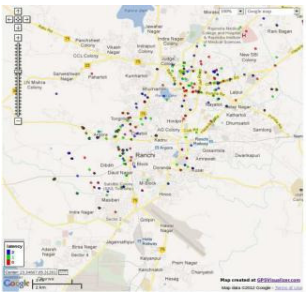

Figure 1: Latency and Availability

A client-side library on the mobile device is provided to application developers, to maintain a local datastore that serves all read/write requests made by the application. This datastore is synchronized automatically by the library whenever connectivity is available to a global datastore. A server-side library running off the global datastore in the Internet is then used by the application developers to write the corresponding server-side application logic. Thus, the application running on the user's mobile device renders itself using the local data store, and hides disconnections altogether from the user, while transparently managing data synchronization. Figure 2 shows a block diagram of the framework.

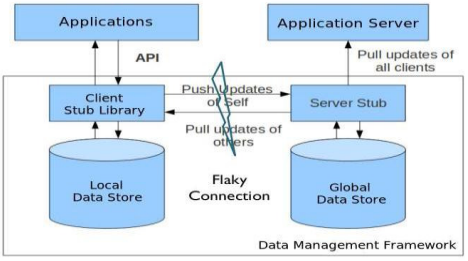

Figure 2: Data management framework for flaky Internet connections

This approach is different from previous work on using third-party proxies to intermediate between mobile devices and the application servers [2, 5]. We are instead providing a library interface and therefore application providers are not dependent on third-parties to run a proxy. Moreover, as we explain next, our framework goes beyond just providing session layer persistence across disconnections, but provides an entire suite of APIs for data management for applications that need to operate in flaky connectivity environments.

2.1 Key-value get/put API access

Our framework uses a key/value storage mechanism. A key-value pair get/put API into a flat table provides a simple primitive that can be generalized across several applications. The API can be

used to store application data, transaction data to be communicated to the correspondent side, and even content objects. The data-types can further automatically define how to internally store the data. The most important functionality this generic method allows is that the actual act of synchronizing data between the client and server data stores can then be handled exclusively by the framework, without requiring any help from the application. The application only has to read and write data from/to its local datastore, and irrespective of how the key/value pairs are internally used by different applications, the framework can independently synchronize the local and global stores.

2.2 Namespaces

Partitioning the keys into namespaces allows an easy way to create subsets in the datastore that need to synchronized with each other while skipping the rest of the data. We allow applications to subscribe and unsubscribe users to namespaces by maintaining a server side access list which is consulted whenever a connection is established between a particular client and server stub library. The actual policy of which namespaces can different users be allowed to access is an application level policy; the framework only provides an API for applications to initialize the namespace access lists. In case the API is invoked by the application at the client-side library, the request is conveyed to the server-side library in the next connection session.

2.3 Opportunistic Communication

Our requirement is to synchronize key-values pairs in the subscribed namespaces between the client-side and server-side data stores during opportunistic connection intervals. It is clearly imperative to ensure that data transmission in each interval resumes from the point where the last transmission left off, and several previous studies have proposed different ways to ensure session level continuation [2]. In addition, we want to be able to club together updates that belong to a transaction or a single group update that needs to be committed at the remote endpoint in an atomic manner. To meet these objectives, we choose the simple and well known method of transmitting data in small bundles with a bundle acknowledgement protocol, to ensure that all bundles are reliably transmitted. A transaction identifier is additionally incorporated to mark bundles that belong to the same transaction group, and are processed only when all bundles comprising the transaction have been transmitted. Currently the bundle transmission happens over a TCP connection. We are also implementing a UDP based reliable transport layer with a different flow control mechanism that initial experiments have shown to perform better than TCP's flow/congestion control mechanism. We will also build functionality in the future to detect the type of connection that is available, whether EDGE (2G) or HSDPA (3G) or WiFi, and accordingly choose a suitable transport layer to invoke. Similarly, although we are using a static bundle size of 50KB currently, the bundle size in the future will be specified to match the type of connection.

3. IMPLEMENTATION

We have implemented a preliminary version of the data management framework described above for Android devices. Both the client-side and server-side libraries are packaged as .jar files that can be included for application development. The client stub library uses the Android Services interface to run two services in the background, to send and receive data from the server. The server stub library is in Java and similarly runs two threads, to send and receive data.

We also implemented a simple application that is used to post audio and video recordings, and photographs, to a server. The users simply have to click an appropriate record button to capture the photo or video or audio, and the captured content is automatically inserted into the framework for transmission. We deployed this application on Galaxy Fit phones provided by us to three users working with our field partner in Jharkhand, and the application is being used actively to collect photographs and videos of local events happening there.

We are currently building some additional features in our framework:

Namespace locking: Application may want to prevent write-write conflicts on the same namespace. We are building an out-of-band namespace locking procedure using SMS: a client-side API will trigger an SMS protocol to the server to lock the given namespace and prevent any other user from requesting a lock. The lock is to be given up only after the updates made locally by the user are synchronized completely with the server, or the lock times out in an eventuality that the user never gets a connection opportunity for synchronization.

Handling conflicts: Conflicts may still arise in locked namespaces if locks were to time out. Another conflict scenario arises in non-locked namespaces if data needs to be tagged with the login-id of the user making the update, but this login-id is shared by several users. This happens often in real life. While possible conflicts may be detectable using application specific logic to selectively accept only one update, we feel the need to build a merge procedure much like in a version management system where each update is uniquely tagged.

A preliminary version of the framework is ready and we hope to release a full version in the next few months.

4. REFERENCES

[1] S. Keshav, “Design Principles for Robust Opportunistic Communication”, *NSDR, 2010*

[2] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharya, “A Policy-Oriented Architecture for Opportunistic Communication on Multiple Wireless Networks”, *Technical report, University of Waterloo, 2005*

[3] K. Fall, “A Delay Tolerant Network Architecture for Challenged Internets”, *SIGCOMM, 2003*

[4] D.B. Terry, M.M. Theimer, and K. Petersen, “Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System”, *SOSP, 1995*.

[5] D. L. Johnson, V. Pejovic, E. M. Belding and G. V. Stam, “VillageShare: Facilitating content generation and sharing in rural networks”, *ACM DEV, 2012*

[6] S. Guo, M. Derakhshani, M.H. Falaki, U. Ismail, R. Luk, E.A. Oliver, S. Ur Rahman, A. Seth, M.A. Zaharia, and S. Keshav, “Design and Implementation of the KioskNet System”, *Computer Networks, 2011*

[7] A.Pentland, R.Fletcher and A. Hasson, “DakNet:Rethinking Connectivity in Developing Nations”, *IEEE Computer, 2004*

[8] A.Mahla, D. Martin, I. Ahuja, Q. Niyaz and A. Seth, “Motivation and Design of a Content Distribution Architecture for Rural Areas”, *ACM DEV, 2012*

[9] M. Demmer, B. Du and S. Surana, “TierStore: A Distributed Storage System for Developing Regions”, *FAST, 2008*