A Comprehensive Data Management Framework for Opportunistic Communication on Mobile Phones

ABSTRACT

Several of our non-profit partners working in rural areas complain about poor data connectivity from their mobile phones. In this paper we begin with studying the state of 2G GPRS EDGE connectivity in a rural location. We then use insights from this study 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 used this framework to develop two applications and will soon be making our solution available for public use.

1. INTRODUCTION

We are working with several civil society and non-profit partners in rural areas of India who use participatory media as a means to bring about community empowerment and awareness, and rely heavily on digital content capture in the field for their work. This includes organizations like Video Volunteers [26] and Drishti [27] that train members from rural communities in video capture using handycams, and help them create short films on social issues such as explaining the rights and entitlements to people for availing different government schemes, motivate voices against corruption, criticize domestic violence against women, etc. These films are then screened in public gatherings in the nearby villages. We also work closely with the startup, Gram Vaani [28], that provides technology solutions and empowerment programming on community radio and mobile phones, and has similar requirements to capture audio content and photographs of various cultural and social development oriented events in rural areas. All these organizations face a problem of poor Internet connectivity to upload to the Internet various content their field staff capture in remote areas.

Although there is significant talk since the last several years of growing Internet penetration, broadband connectivity right down to the 600,000 villages of India [22], and the increasing penetration of ICTs in remote areas, yet the data connectivity challenges faced by our partners *mysteriously* still remain significant. We have similar personal experiences from our field trips to several rural areas mostly in the poorer states of India.

In this paper, we demonstrate a few studies on the state of GPRS EDGE 2G connectivity measured at a rural location. While this may not be representative of 2G coverage in rural India, it gives us several insights into the kind of connectivity issues that are likely to arise in these settings. We document network characteristics including connection availability, spurious timeouts in TCP, and rate and latency stability on GPRS EDGE networks. We then use these insights, and significant previous research that exists in the space of rural Internet connectivity [2, 12, 14, 15, 4], to build a communication and data management framework for mobile devices. This framework is made available to application developers as a library, and helps applications to transparently deal with disconnections, improve the utilization of opportunistic connection intervals to transfer data, manage data and provide synchronization of offline operations whenever connectivity becomes available, help resolve merge conflicts, and even prevent data consistency issues common in poorly connected scenarios. Our contribution here is a comprehensive data management solution for application developers, which goes beyond the challenges of opportunistic connection management that most previous work has dealt with.

We have built two applications on this infrastructure in close consultation with our field partners. Both applications aim to enable audio/video documented stories from rural hinterlands to reach the outside world and help our partners present a better case for improving public services, and also aid greater communication within and across communities for sharing contextually relevant information.

The rest of the paper is organized as follows: In Section 2 we describe several network measurements to characterize the state of EDGE connectivity in a rural area in India. Section 3 describes in detail the communication and data management framework we have built. Section 4 outlines the applications we have created using this framework. This is followed by a review of related work, and finally the

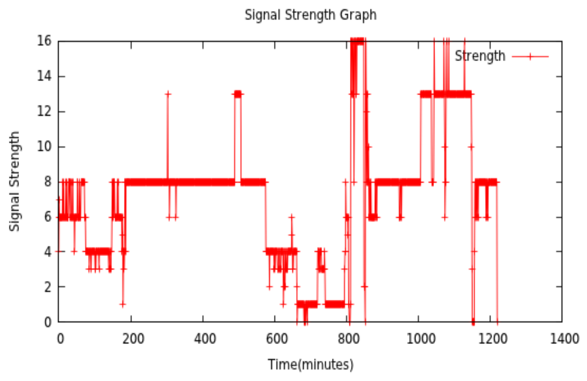

Figure 1: Signal strength variations across one day

conclusions from our work so far.

2. STATE OF 2G CONNECTIVITY

In this section, we describe several observations made from pilot tests we ran in rural areas in the state of Jharkhand in India, to document the state of EDGE connectivity. These observations motivate the design for the framework we propose in the subsequent section.

2.1 Connection availability

We wrote a simple Android application to log the signal strengths and HTTP pings to $http://www.google.com$ to check for connection availability. Figure 1 shows a sample trace of signal strength across 24 hours. The mean time between connectivity losses in traces collected off the phones given to our field staff for their day-to-day operations was 38 minutes with a 95%ile of 133 minutes. The staff travel to different areas during the day, and we can see that signal strengths often drop, dead spots causing loss of connectivity are frequent, and during our field trips we personally experienced poor browsing experience on our phones as well. In another work we observed that GPRS base stations were actually powered off at night on a daily basis [21]. This clearly shows that despite grand visions of ubiquitous connectivity advocated by the governments and telecom providers, the networks on the ground are not well provisioned, and solutions are required to handle mobile phone based media content delivery and sharing in these environments.

2.2 TCP performance

We next study TCP traces we captured from moderately sized file uploads. Our experience with downloads was similar, but we mostly show uplink characteristics in this paper since we are immediately more concerned with getting content from the field than transmit in the reverse direction. We rooted the phone for this purpose and compiled the tcpdump[24] library for Android to record the traces. The BotSync[25] application was used to initiate data transfers. Figure 2 shows a sample trace; clearly there are several timeouts followed by slow-start recoveries that affect the TCP performance. A zoomed-in timeout is shown in Figure 3. Here we can see the spurious timeout potentially caused because of a sudden spike in latency – several retransmissions are made but we can see from the ack-data gaps that the jump in cumulative acks at the 11th second could not have been

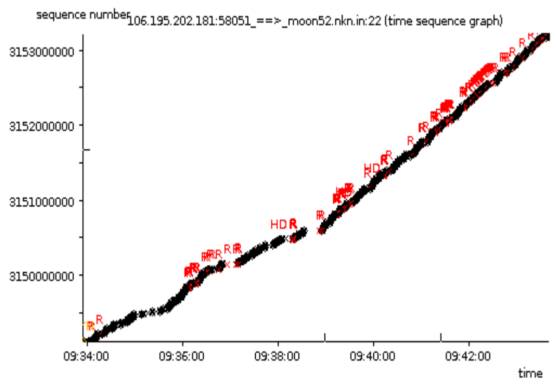

Figure 2: TCP sequence-time graph

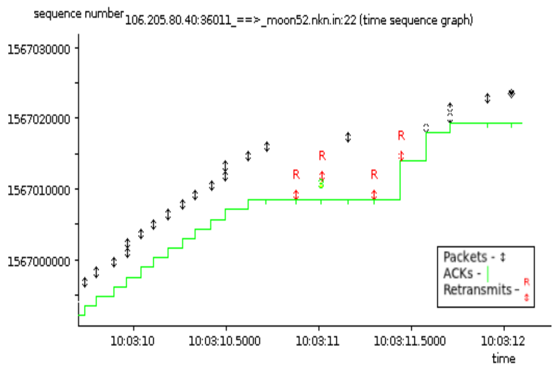

Figure 3: TCP sequence-time graph with timeout

triggered because of receipt of the retransmitted packets, and was rather caused because of regular acks sent in receipt of the original data but only extraordinarily delayed because of a latency spike in that RTT time interval.

On average, we were able to achieve TCP throughputs of only 101.18Kbps while the advertised EDGE throughputs are much higher. In the following section, we try to understand the reasons behind this gap and make recommendations for an alternate transport layer for GPRS connections.

2.3 Network characteristics

We begin with a brief overview of GPRS networks, and then proceed with several pilot tests to understand the behavior. While several studies on GPRS have been done in the past [6, 10, 8], we were not able to find any recent studies on the latest GPRS EDGE standards run off today's mobile phones. The phones used in earlier studies were computationally under-provisioned and self-limited TCP throughputs by advertising a small TCP receive window; we do not expect this to be true in most phones available today (we did this study on a Galaxy Fit phone with a a Single core, 600 MHz, ARM11 processor). Further, as we describe next, GPRS channels are shared with regular GSM voice channels with voice getting a priority over data, and hence we are keen to observe GPRS performance in the specific context of rural areas.

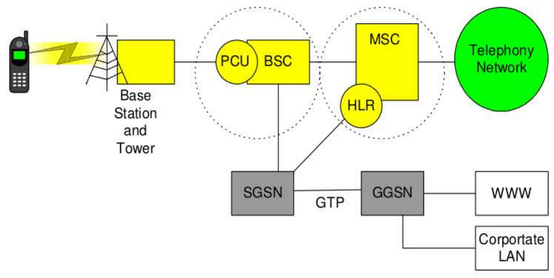

Figure 4: GPRS network architecture [29]

GPRS works on top of the already present GSM network with the addition of a few nodes:

PCU (Packet Control Unit): These nodes differentiate circuit switched voice from packet switched data
SGSN (Serving GPRS Support Node): These nodes handle packet data routing, handover and IP address assignment
GGSN (Gateway GPRS Support Node): These nodes serve as a gateway into the IP Internet, and also run firewalls

The GGSN and SGSN are critical for data performance since they carry buffers to hold data both on the uplink and downlink, while the PCU and other nodes in the radio access network control the scheduling of slots to different mobile terminals.

GPRS uses TDMA and allocates slots to competing users on demand, ie. in case data is waiting to be transmitted from the SGSN/GGSN to the mobile terminal, or vice versa. Each TDMA frame consists of 8 time slots that could be allocated to one or more mobile terminals, for voice or data, with voice getting a preference over data. Each TDMA frame may further be encoded at different rates between 72.4Kbps and 171.2Kbps based on the signal strengths seen at the mobile terminals and base station. For these reasons, we expect GPRS data rate and latency characteristics to be heavily influenced by cross-traffic and other network conditions, that would go on to influence TCP performance. We try to understand this next.

2.3.1 *Experimental setup*

We set up a UDP server in the Internet cloud that could be used to ECHO packets of varying sizes. Client applications running on the phone could send large UDP packets and receive a small reply, or send small packets and request for a large reply, to separately study the uplink and downlink capacities. The packets could also be marked as belonging to a packet train, in which case the server would reply only after the train test had completed, and report the timestamps of having received each individual packet in the train. We ran a number of tests using this simple tool that is available for download from our website.

2.3.2 *RTT variations*

We studied RTT variations at large and small granularities by using UDP ECHO packets of 1400 bytes. Figure 5 reports the mean and standard deviation of RTT values seen during

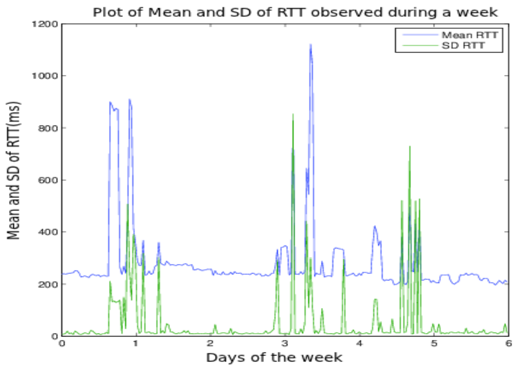

Figure 5: Mean and the standard deviation of RTT observed during a week

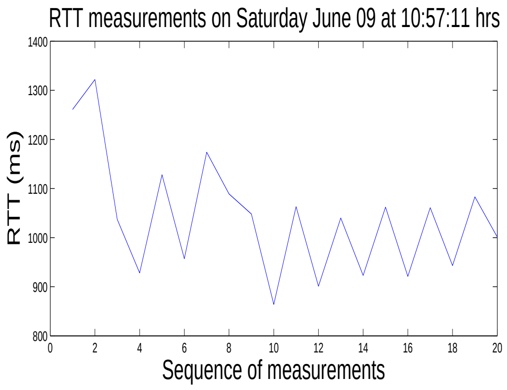

Figure 6: RTT values observed on Saturday June 09 2012 at 10:57:11 hrs

the course of a week, and Figure 6 shows the RTT variation in one trace across a shorter timescale of 1 minute. While the RTT values vary widely from 200ms to 1100ms at large timescales, the variations are lower at shorter timescales but can still be significant enough to cause spurious TCP timeouts as noticed in Figure 3. The latency variations probably happen because of scheduling delays in the GSM network caused by competing voice and data traffic from other users.

2.3.3 *Available bandwidth*

To measure the available bandwidth on a GPRS connection, we used the standard technique of sending a packet train to saturate the link and measure the time taken for the train transmission at the remote endpoint. This is shown in Figure 7 for a train of length 5. The total data transmitted between TA_1 and TA_2 measured at the remote endpoint is $(n-1)*S$ bytes for a train of length n with packets of size S, giving an aggregate as follows:

$$Datarate \propto \frac{Amount\ of\ data\ transferred}{Total\ time\ taken} = \frac{(n-1)S}{TA_2 - TA_1} \tag{1}$$

To measure data rates at varying timescales, and also to evaluate the effect of packet sizes, we recorded the delay

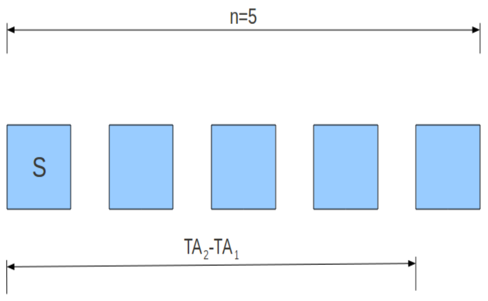

Figure 7: Packet Train Test to measure Data rate

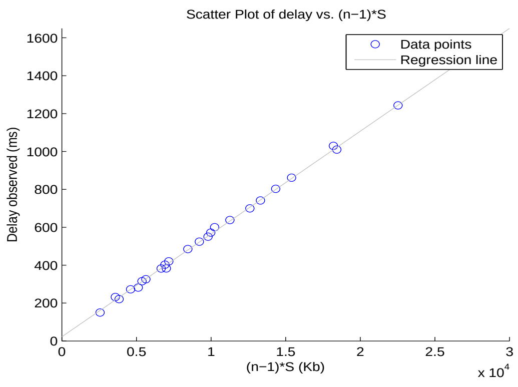

Figure 8: Dependence between (n-1)*S and the delay $TA_2 - TA_1$ (Uplink)

($TA_2 - TA_1$) for packets of different sizes and for trains of different lengths. The tests were performed for packets of sizes (512, 768, 1024, 1400, 2048) and trains of length (6, 8, 10, 12, 14) packets. The value (n-1)*S was computed for each packet size and train length combination. Figure 8 shows a scatterplot of the values observed while probing the uplink. Clearly, $TA_2 - TA_1$ is directly proportional to (n-1)*S, showing that the data rates are stable across timescales of within a single RTT and have no dependency on the packet sizes either. Tests for the downlink were similar, but with a higher data rate.

Figure 9 shows the mean data rates and the standard deviation for uplink during the course of a week. Here too, we can see that the data rate is mostly stable at approximately 150Kbps, showing that the available throughput for any mobile terminal stays fairly the same, most of the time. This indicates that GSM scheduling probably ensures a throughput based fairness across different mobile terminals, despite short terms delay variations. Thus, it is interesting to notice that although GPRS latencies are variable but the data rates are quite stable and the bandwidth delay product is likely to vary between 4KB (150Kbps times 200ms) to 20KB (150Kbps times 1100ms). TCP connections will suffer if the buffer sizes are inadequate to absorb the latency bursts. We therefore next evaluate the buffer sizes in the network.

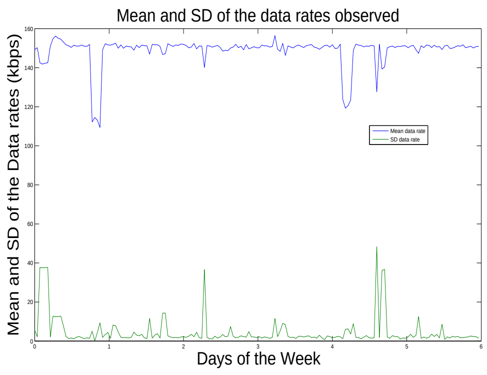

Figure 9: Mean and standard deviation of data rates observed during a week

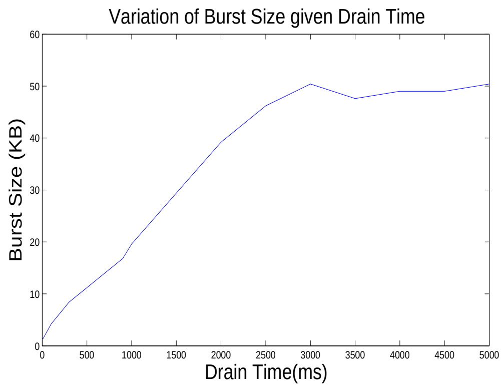

Figure 10: Burst size vs. drain times observed during a week

2.3.4 Buffer sizes

Assuming that the bottleneck buffers exist at the GGSN or SGSN interfaces [8], we send monotonically increasing lengths of packets trains and wait for a minimum drain time duration so that the trains do not experience any packet loss. Figure 10 shows a sample of the values recorded indicating that the buffer size saturates at approximately 50KB taking 3s to drain at a rate of 133Kbps. We also rigorously evaluated the buffersizes at different times of day and observed the values to be consistent, indicating that these are per-user buffers isolated from each other rather than shared buffers for which active queue management would become important at times of congestion.

This shows that the buffer sizes are almost three times the maximum bandwidth delay product that can be expected on EDGE links, and therefore while these are large enough to absorb bursts but the latency under load can be significantly higher than the RTTs we measured in a quiescent state of the network [13]. For bulk data transfer though, this does give some useful ideas on designing a better flow control scheme for transport layers on EDGE networks where the data rates are stable but the latencies may vary within bounds that can be easily accommodated in the buffers already provisioned in the existing infrastructure.

2.4 Recommendations

Based on the observations made above, we make several recommendations that guide the design of the communication framework we propose in this paper.

2.4.1 Session layer delay tolerance

Given that connection availability is quite flaky in rural environments, we concur with the need for session level delay tolerance as also proposed by several other studies [2, 14, 15]. We go much beyond the previous work by not just handling connection state persistence across disconnections, but also propose a generic application development API based on key-value pairs for data management across disconnections. We feel that such a comprehensive data management framework will prove instrumental in guiding the development of applications within a consistent framework, rather than different applications having to evolve different policies for managing complex aspects such as transactions, consistency, reactive Vs proactive synchronization, etc.

2.4.2 Transport protocol

We observed that GPRS links can be characterized as having a uniform data rate but variable latencies, with more than large buffer sizes to absorb any latency bursts. The default flow and congestion control mechanisms of TCP are however inefficient in such a scenario because:

* Spurious timeouts are caused due to sudden latency spikes
* Slow start is wasteful; an initial congestion window consistent with lower bounds on the bandwidth delay product can be easily used instead

Since the buffer sizes are large enough and per-user based, a simple and more efficient flow control scheme could operate as follows: Instead of dispatching data in TCP's self-clocked manner, pump data into the network at the given data rate uniformly over time in a CBR manner, but to avoid packet loss and consequent retransmissions be careful that the data in transit never exceeds the buffer size. This should not be hard to ensure since the buffer sizes are almost three times the observed maximum bandwidth delay product, and therefore any exceptional events such as a sudden drop in signal strength or an increased latency should be detectable within three RTTs.

We further do some simple experiments to compare the data transfer rates of a 2.3MB file over TCP and a constant bit rate UDP. For upload, while TCP transfers took on average 212s with a standard deviation of 14.6s, UDP transfers only took 152s with a standard deviation of 1.9s and loss rates of 0.8%. This shows that indeed a more efficient transport layer flow control scheme makes sense for GPRS links. We are implementing a transport layer protocol on UDP based on these ideas as part of future work, and instrumenting different 3G connections as well to see if connection based profiles for flow and congestion control can be developed for cellular data connections.

3. SYSTEM DESIGN

Based on the state of 2G connectivity described in the previous section, and conversations with several social sector organizations working in rural areas, we enumerate some goals that should be provided by a comprehensive communication and data management framework in flaky Internet environments.

- *Opportunistic communication*: Since the connectivity can fluctuate between on and off, the framework should automatically detect link layer events and use any available connection opportunities to send and receive data to and from mobile phones. Basic principles of delay tolerant communication including session layer persistance and bundling of data to allow transfers to span across multiple disconnections [4, 2, 12] should be followed.
- *Alternate transport layers*: The framework should detect the type of underlying connection and accordingly choose a suitable transport layer implementation to transfer data during opportunistic connectivity intervals. For example, the standard TCP implementation can be used if WiFi connectivity is available but a modified UDP-based transport layer proposed by us in the previous section could be used on EDGE connections.
- *Generic application development interface*: The framework should provide a generic API to application developers so that an entire class of applications that work in offline mode are able to use the framework. Applications should be able to issue non-blocking read/write requests that are cached until connectivity is available, and then opportunistically synchronized.
- *Transactions support*: Applications should be able to club several write requests into transactions that support the ACID (Atomicity, Consistency, Isolation, Durability) properties. In case only a fraction of the write operations within a traction were successfully synchronized during an opportunistic connection interval, an intermediated layer should hold them off from a commit until the rest of the requests have not arrived.
- *Policy based summarization and prioritization of updates*: Several applications working with media objects often require only thumbnails or low resolution images to be updated in priority over the actual more bulky media object. The framework should allow applications to specify broad policies on summarization and prioritization of certain types of objects, and automatically take care of the underlying machinery to do this.
- *Ease of consistency maintenance*: While managing and resolving write-write conflicts in cases where multiple users make an update on the same shared resource is typically an application layer policy, the framework should provide functionality to tag different groups of updates so that consistency restoration is easier. Most modern version control systems such as Git adopt a similar approach to group the set of updates made by a user in a single commit, so that an entire update can be dropped or rolled back.

- *User and device identity conflicts*: An interesting requirement conveyed to us by the NGO Digital Green [23] was that the same login/password was often used by several of their staff to access their MIS system, and thus it became hard for them to isolate updates made by different users. The framework should take such exceptional cases into account and isolate updates not just based on user and device identities, but potentially also build an out-of-band locking or update tagging subsystem.

3.1 Overview

Based on the design goals enumerated in the previous section, we propose the following communication framework for flaky Internet environments. A client-side library on the mobile device is provided to application developers, to maintain a local data store cache of all read/write requests made by the application. The data store is synchronized with a global data store whenever connectivity is available. A server-side library running off the global data store in the Internet is then used by the application developers to write the corresponding server-side application logic that may result in pushing updates back to the user's or other users' data stores. The application running on the user's mobile device then renders itself using the local data store, in an effort to hide disconnections altogether from the user and transparently manage data synchronization.

This approach is different from previous work on using third-party proxies to intermediate between mobile devices and the application servers [2, 12]. We are instead providing a library interface and therefore application providers are not dependent on third-parties to run a proxy. Moreover, our framework goes beyond just providing session layer persistance across disconnections, but provides an entire suite of APIs for data management for applications that need to operate in flaky connectivity environments.

The different modules in our framework are the following, shown in Figure 11.

1. Client stub library
2. Server stub library
3. Client and server-side application components

3.1.1 *Client stub library*

The client stub library is called by the application developer through a simple key-val get/put API described in more detail later. The library takes care of various underlying activities to be performed:

1. Execute the application get/put requests on the local data store and maintain a log of all updates.
2. Push the updates to the server stub during connectivity intervals. Take care of using the appropriate transport layer implementation based on the connection profile on which Internet connectivity is available.

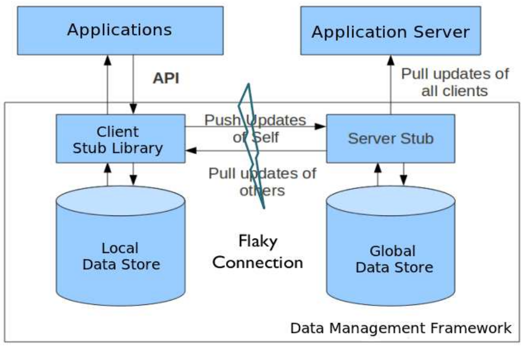

Figure 11: Data management framework for flaky Internet connections

3. Pull relevant updates from the server stub and add them to the local data store once an entire transaction update is reflected locally.

The data store attached to the library maintains all application data including media objects in a persistent key-value database. Namespaces are used to decide which subsets of the global data store are to be maintained locally. For example, to build a video sharing application, a public namespace can be used to maintain all metadata for videos marked as being publicly shared, group level namespaces can be used to share videos within smaller groups, and per-user namespaces can be used for each individual user. Users can then be subscribed to these namespaces to have their contents be automatically synchronized at the local data stores, and the application on the mobile device renders itself using whatever data is available in the store. Then, if the user adds a new video to their individual and group namespaces, the updates are propagated to the global data store and then pushed out to other subscribers on the group namespace.

3.1.2 *Server stub library*

Similar to the client stub, a server-side stub library is provided to the application developers to manage access to namespaces, data sharing across namespaces, etc on the server side. The server stub library also has an attached key-value data store. Listeners can be initialized on different namespaces in the data store to get triggered if a client update on a particular namespace arrives at the server. The server can then make its own updates that are written into the appropriate namespaces on the server-side data store. Whenever the client library establishes a connection to the server, all pending updates on the namespaces to which the client is subscribed are written out to the client stub and consequently to its local data store.

In the future, we will enhance the server stub library for scalability by distributing namespaces across multiple servers, and with access control and security to control access across different namespaces.

3.1.3 *Client and server-side application components*

Application developers are expected to use the library API to write a client-side and a server-side application, much like what they would do anyway. Our goal has been to provide a generic API that will fulfil most common data management requirements of these applications designed to work in disconnected environments. This includes the handling of disconnections, a simple primitive of namespaces to structure data synchronization activities, and advanced features to help handle consistency and transactions. We next describe the API and other functionality in more detail.

3.2 Key-value get/put API access

A key-value pair get/put API into a flat table provides a simple primitive that can be generalized to be useful to most applications. The API can be used to store application data, any transaction data, and even content objects. The datatype 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 data store, and irrespective of how the key/value pairs are internally used by different applications, the framework can independently synchronize the local and global stores.

As an example, the following set of key/value pairs can be used to store metadata and content object for a photo added into the data store.

```
KEY: "AppName.UserId.Photo.Id"
VALUE: 652

KEY: "AppName.UserId.Photo.Name"
VALUE: "Rally in Nagri Jharkhand"

KEY: "AppName.UserId.Photo.Date"
VALUE: Tue Jul 31 15:30:00 IST 2012

KEY: "AppName.UserId.Photo.Image"
VALUE: 98jhys86k2j90kalq08ka1k9...
```

The application can define its own logic to interpret the namespace nomenclature, field names, expected data-types, etc, while the framework only needs to see this as a flat table in which certain namespaces need to be synchronized across a set of users. We also allow applications to specify transaction ids for groups of key/value pairs that need to be synchronized and committed atomically. This is described later.

3.3 Namespaces

Partitioning the keys into namespaces allows an easy way to create subsets in the data store that need to synchronized with each other while skipping the rest of the data. We allow applications to subscribe and unsubscribe users to namespaces. An access list is maintained in the framework and is consulted whenever a connection is established between a particular client and server stub library; the namespaces to which the user on the client side is subscribed are then synchronized between the client and server side data stores.

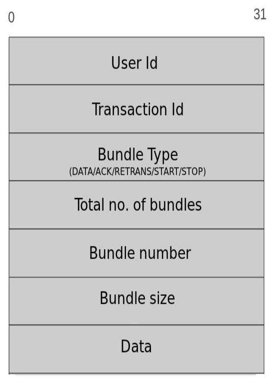

Figure 12: Format of a fragment

Currently we do not support authentication, group based access, and different read/write permissions, but that can be easily built in. As an example, a namespace access list for two users to read other's photo streams can be specified as follows:

```
User: "User1", Namespace: "AppName.User1.*", Perm: RW

User: "User1", Namespace: "AppName.User2.Photo.*", Perm: R

User: "User2", Namespace: "AppName.User2.*", Perm: RW

User: "User2", Namespace: "AppName.User1.Photo.*", Perm: R
```

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.

3.4 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, 16]. In addition, we have a requirement 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. 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 and to identify bundles that belong to the same transaction group. Figure 12 describes the bundle structure.

Each bundle contains the following fields:

1. **User Id**: To identify the user who originated this update or to whom the update is destined

2. **Transaction Id**: The unique id of the transaction of which this bundle is a part

3. **Bundle type**: This is used to specify the type of the bundle. A bundle can be of the following types:

   - **"DATA"**: If the bundle contains data of key-value pairs in the data field
   - **"ACK"**: If the bundle is an acknowledgement to a successful bundle receipt. Note that although we assume an underlying reliable transport layer to transfer the bundles, even TCP does not provide any method to applications to query whether the data written to TCP in a socket send() call was actually acknowledged by the receiver [2]. Therefore, during unclean connection breaks such as when a mobile terminal loses EDGE connectivity, applications could mistakenly believe that up to a window size amount of data was actually received at the remote endpoint because the local send() call never returned an error. Hence a bundle-level acknowledgement protocol becomes necessary.
   - **"RETRANS"**: To request for retransmission of a specific transaction or a particular bundle, depending on which fields are initialized in the bundle header. The data field is expected to be blank.
   - **"START"**: A poll to request the remote endpoint to initiate any new data transfer for synchronization. An API is also provided to the application to forcefully do a refresh. In the future, we will enhance this to also be able to specify a particular namespace that needs to be synchronized in priority.
   - **"STOP"**: To instruct the remote endpoint to stop any data transmission. This becomes useful in a scenario where the user wants to stop the transmission of a bulk update and use the phone for other browsing activities. An API is provided to the application to forcefully stop any data transfer in the library.

4. **Total number of bundles**: The total number of bundles into which this transaction update was divided

5. **Bundle number**: The sequence number of this bundle

6. **Bundle size**: The size of the data field of this bundle

7. **Data**: The actual data

Currently the bundle transmission happens over a TCP connection. As mentioned in the previous section, we are also implementing a UDP based reliable transport layer that will give a better performance than TCP on EDGE networks. 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.

3.5 Additional features

We are currently developing the following additional features:

Data summarization: Applications will be able to mark key-value pairs for media objects as summarizable or not. Summarized content will then be transmitted in priority over the full content objects. An additional field for default Vs on-demand synchronization will be added, to control whether the full object should be synchronized by default or only transmitted on-demand.

Conflict resolution: We currently commit all sets up updates made by different users once the data for an entire transaction has been received. But if multiple users use the same login userid then conflicts can arise and a merge procedure would be needed. We therefore want to provide especially at the server-side a functionality to browse all recent updates, use application logic to infer any potential conflicts, and selectively accept only one update or merge multiple updates after conflict resolution. We will do this by tagging each update, much like in a version management system.

Namespace locking: To completely avoid any conflicts of the type mentioned above where several users may use the same login userid, and also to help applications avoid any logical write-write conflicts, we want to build an out-of-band namespace locking procedure using SMS. Our idea is that the application can use a client-side API to lock a particular namespace in case a user wants to update a critical section resource. This will trigger a real-time SMS protocol to the server to lock the given namespace and prevent any other user from requesting a lock or performing a write. 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.

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

4. IMPLEMENTATION AND FIELD EVALUATION

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 have implemented two applications using this framework, one of which was recently deployed in the field but is facing some issues and needs to be updated. Figure 14 shows a screenshot of a content sharing application for staff at community radio stations to post images [1] to a Facebook

[1]One of our partner field organization ran public information campaigns at 5 community radio stations on the topics of NREGA (Indian rural employment guarantee scheme), PDS (the public distribution system for food subsidy), and

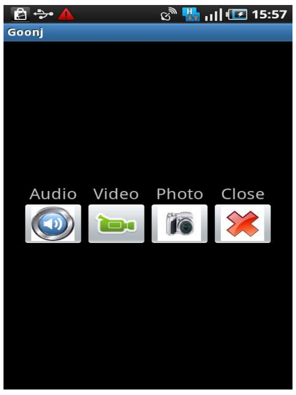

Figure 13: Simple application to upload data from the field

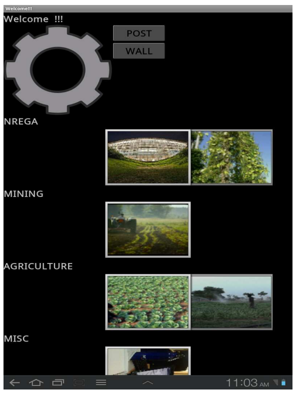

Figure 14: Content sharing linked to a Facebook page

page. Figure 15 shows the application architecture; here the server-side application uses the library to receive updates from different users and posts them to a Facebook page. Updates made on the Facebook page are polled by the application and inserted into the namespaces of different users, to get pushed back to their respective phones. We deliberately made the application simple for the phone users, so they only have click a photo or video and post it. We have yet to demonstrate this application to the intended users and have them use it.

A second application we developed is only meant to post audio and video recordings, and photographs, to a server. A screenshot is shown in Figure 13. The users only have to click the appropriate record button, capture the photo or video or audio, and save it; 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. Two of the three users had prior exposure to computers and none of them had used smartphones with a touchscreen before. However, all three of them quickly took to using the application without much training and used it to post photographs and videos of local events happening in the state of Jharkhand in India. Unfortunately the application developed a bug and we have been unable to guide the users over the phone to upgrade the application; we are waiting for another field visit and also plan to come up with an easy to use automatic upgrade mechanism to avoid similar issues in the future.

5. RELATED WORK

This paper builds upon several pieces of work. Delay tolerant networking was first introduced in the context of planetary networks and sensor networks [4], and several studies extended the idea to rural area networks to use vehicles and USB keys as message ferries to move data between remote rural locations and Internet access points [14, 15, 17, 18]. This paper operates in an easier environment over a single hop of disconnection, ie. online/offline connectivity to the Internet. [2] operates in a similar scenario but only focuses on utilizing opportunistic connection opportunities; the actual data management API available to applications is much simpler, restricted to simply dropping and reading files from a shared directory. We have placed more emphasis on the data management interface and evaluations of 2G connectivity in rural environments. Our paper thus complements the set of design principles enumerated in [1] for opportunistic communication.

Our proposed solution to helping application developers handle consistency problems is simple and suited to the specific use-case of online-offline connectivity of mobile devices to a server. Other studies such as [19, 20] focus on harder scenarios with disconnection possibilities between all pairs of nodes, and propose solutions that are an overkill in our case.

We also did extensive evaluations of the GPRS EDGE network connections. Our techniques are similar to those used in several other studies [7, 11], but executed specifically on EDGE networks probed via mobile phones. [6] also evaluated a GPRS network in a different setting several years back, and proposed to use a transparent proxy located close to the GGSN that would alter the receive window advertisements to maintain a steady amount of data in transit. They refrained from suggesting any changes in TCP for the practical reason of avoiding any protocol modifications at the endpoints. In our case however, since we are already deploying a communication framework at the endpoints, we feel comfortable in actually altering the transport protocol implementation to achieve greater performance benefits. In the future, we will evaluate our proposed transport layer modifications against TCP Eifel [9] that uses the TCP timestamp option to detect spurious timeouts.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we described the need of several partner organizations to send multimedia content from the field, and evaluated the problems they face in using typical 2G networks in rural areas. Based on our evaluations, we described a communication and data management framework for application developers to write applications on mobile phones to work in flaky Internet environments. Our framework is comprehensive and meets several common needs of appli-

RTE (right to education act), and expressed the need to also receive photos and videos of interviews and site visits to gather more data for the campaigns.

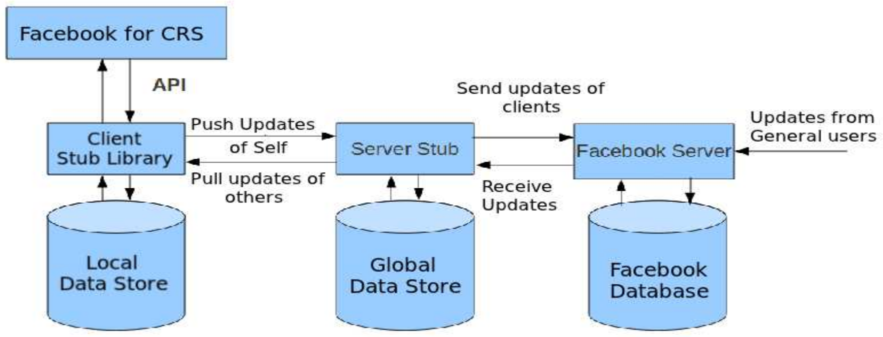

Figure 15: Application for posting data to a Facebook page

cations including their ability to function across disconnections, synchronize data between the phone and an Internet server, manage transactions, manage conflict resolutions to restore consistency, and additional features such as the ability to summarize large data objects. Much future work in terms of field evaluations and release of an open-source framework remain to be done.

7. ACKNOWLEDGEMENT

This project was funded as part of a Ford Foundation grant to design and deploy an Internet measurement framework for cellular data connectivity in India.

8. 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] J. Scott, P. Hui, J. Crowcroft, and C. Diot, “Haggle: a Networking Architecture Designed Around Mobile Users”,*IFIP WONS workshop, 2006.*

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

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

[6] R. Chakravorty, J. Cartwright, and I. Pratt, “Practical experience with TCP over GPRS”, *GLOBECOM, 2002*

[7] A. Shriram and J. Kaur, “Empirical evaluation of techniques for measuring available bandwidth”, *INFOCOM 2007*

[8] P. Benko, G. Malicsko and A. Veres, “A Large-scale, Passive Analysis of End-to-End TCP Performance over GPRS”, *INFOCOM, 2004*

[9] R. Ludwig and R. H. Katz, “The Eifel algorithm: Making TCP robust against spurious retransmissions”, *ACM CCR, 2000*

[10] A. Wennstrom, A. Brunstrom, J. Rendon, and J. Gustafsson, “A GPRS Testbed for TCP Measurements”, *International Workshop on Mobile and wireless Communications Network, 2002*

[11] J. F. Kurose , K. W. Ross, “Computer Networking: A Top-Down Approach”, *Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2010*

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

[13] V. Cerf, V. Jacobson, N. Weaver, J. Gettys, “BufferBloat: What’s Wrong with the Internet?”, *Vol 9 Issue 12, ACM Queue, 2012*

[14] 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*

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

[16] A. C. Snoeren and H. Balakrishnan, “An End-to-End Approach to Host Mobility”, *ACM Mobicom, 2000*

[17] 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*

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

[19] K. Petersen, M. Spreitzer, D. terry and M. Theimer, “Bayou: Replicated Database Services for World-wise Applications”, *SIGOPS, 1996*

[20] J.J. Kistler and M. Satyanarayanan, “Disconnected Operation in the Coda File System”, *Computer Systems, 1992*

[21] Anonymized for blind review

[22] Broadband Penetration - “http://bit.ly/GK956W”

[23] Digital Green, “http://digitalgreen.org/”, India.

[24] Tcpdump, “http://web.eecs.umich.edu/ timuralp/tcpdump-arm”.

[25] BotSync, “https://www.botsync.com”.

[26] Video Volunteers, “http://www.videovolunteers.org/”, India.

[27] Drishti Media, “http://www.drishtimedia.org”, India.

[28] Gram Vaani Community Media, “http://www.gramvaani.org”, India.

[29] MorganDoyle Limited, GPRS Tutorial, “http://bit.ly/TveAMB”