ELMR: Efficient Lightweight Mobile Records

Arvind Kumar, Jay Chen, Michael Paik, Lakshminarayanan Subramanian
New York University
arvind.kumar@nyu.edu, jchen@cs.nyu.edu, mpaik@cs.nyu.edu, lakshmi@cs.nyu.edu

ABSTRACT

Mobile devices are increasingly being used as clients for a wide suite of distributed database-centric healthcare applications. This is particularly true in developing regions where the bulk of healthcare delivery is handled by community health workers due to lack of doctors. The wide availability of GSM cellular services and the ubiquity of cheap mobile phones make these latter a promising platform for enabling new healthcare applications in these settings. In this paper we describe Efficient Lightweight Mobile Records (ELMR), a system that provides a practical protocol for accessing and updating database records remotely from low-end mobile devices using the 140-byte SMS channel. Achieving this objective is a challenging task due to stringent bandwidth and message cost constraints. The design of ELMR employs a reduced query set, semantic compression, and a user-centric intermittent database consistency model to reduce bandwidth and per-packet costs. Additionally, ELMR includes an SMS reliability layer to cope with spotty wireless service, and a lightweight privacy model to prevent identity spoofing and theft of sensitive data.

1. INTRODUCTION

In rural regions around the world, especially underdeveloped areas, it is often difficult to gain access to basic healthcare. In these areas much of the burden of healthcare delivery falls on community health workers (CHWs) with limited skills and expertise. The massive penetration of cellular services in such regions around the world position mobile devices and applications to revolutionize the way healthcare is delivered by providing a channel for these CHWs to gain on-demand access to relevant data to improve quality of treatment. Most countries in Africa have over 50% cellular coverage [2] and a significant fraction of their rural population owns or has access to a mobile phone.

Despite this promising trend, the usage of mobile devices by rural populations has been extremely constrained due to low purchasing power and extremely high usage costs. In a recent survey of AIDS patients in rural Ghana citesmarttrack, we found that while 30-40% of the rural population owned a mobile phone, nearly 30% of the users made less than one call a week and more than 80% of the users made less than one call a day [5] From the service provider's perspective, existing usage rates need to be maintained to break even in these markets especially given low user densities in rural areas. In addition, we found that most users use low-end mobile devices and are unfamiliar with smartphones.

Several recent research and developmental efforts around the world such as OpenRosa [4], OpenMRS [3], and Voxiva [6] have explored the use of mobile phones as a low-cost computing platform for distributed healthcare applications. However, these existing systems are not scalable and sustainable in developing contexts for a variety of reasons. First, the software implementation itself is typically too heavyweight for low-end mobile phones, calling for the additional expense of smartphones. In addition these systems rely on SQL database implementations on standard TCP/IP networking stacks. These in turn depend on GPRS network connectivity which is feasible only in urban settings; in most rural settings only voice and Short Messaging Service (SMS) capabilities (140 byte packets) are available.

On the opposite end of the spectrum are lightweight applications designed with high access (and therefore only SMS) in mind such as FrontlineSMS [1]. In these systems only SMS messages are used, so very little infrastructure is required and low-end mobile phones are adequate. The main drawback of these systems is that forms and structured data are not designed as a part of the messaging system. The result is feature-poor systems using ad-hoc messages and responses that do not utilize the limited bandwidth efficiently.

To address these limitations, we designed and implemented Efficient Lightweight Mobile Records (ELMR), a generic record system that is both feature-rich and cost effective. ELMR runs on low-end mobile phones and across the extremely bandwidth-constrained SMS channel enabling health workers and patients in rural areas to fetch and update their patient records and doctors to remotely track their patients' health. ELMR em-

ploys a client-server architecture, where a mobile client provides an interface to a stationary physical database. An append-only database model suited to data collection is applied, and the rare cases of data inconsistency are handled by the server administrator and users. ELMR also provides a lightweight reliability protocol to handle SMS message losses and a simple authentication mechanism for privacy. Finally, ELMR employs a suite of optimizations including semantic compression, cached updates, and message aggregation to make maximum use of limited bandwidth.

2. MOTIVATION

In this section, we describe the typical usage scenario of mobile devices for healthcare in rural developing regions and the corresponding shortcomings of existing mobile solutions in these contexts.

2.1 Typical Usage Scenarios

For the past three years, we have been directly working with several large hospitals and AIDS care centers in India, Ghana and South Africa, each of which serves over a million patients per year. Our application scenario is motivated by the healthcare delivery setting in these regions. There are three types of agent in these contexts: CHWs, patients and doctors. Patients reside primarily in rural areas, and 30-75% of rural patients possess cell phones [5]. CHWs and doctors may either be stationed in specific locations or conduct mobile health camps across different rural areas on a periodic basis.

The typical usage scenario we envision consists of a doctor or CHW diagnosing a patient in a mobile camp or primary health care center (PHCC), using a mobile phone to fetch and update the patient's medical records.In most rural areas, GPRS connectivity is scarce and most health workers use low-end cell phones which have basic Java capacity. Therefore, communication on mobile phones should either be voice-based or SMS-based, of which we choose the latter as the former requires speech recognition tools or call centers, which represent increased hardware and software requirements or human capital cost, respectively. We describe three usage scenarios that occur on the field:

Scenario 1: A CHW conducts a mobile camp in a village where the health worker measures the CD4 count[1] of each patient and updates his health record. The CHW or a remote doctor may be able to fetch the CD4 history of a patient to track the progress of the HIV infection.

Scenario 2: Patients can use mobile phones to report their medication consumption along with any side-effects or other indications of adverse reaction.

Scenario 3: A CHW diagnoses a patient with a collection of symptoms but is unsure of the diagnosis, and may use the system to search for health records of other patients who have displayed the same set of symptoms.

In all of these example scenarios and most others in this context, the total amount of information that is being fetched and updated by the mobile device is relatively small compared to the size of the database. Hence, it is reasonable to expect the information to be compressable into one or few SMS messages. ELMR uses semantic compression techniques to minimize the number of bits for every operation, enabling ELMR to aggregate several operations into a single SMS message.

2.2 Pitfalls of Existing Approaches

To understand the necessity for ELMR and its design rationale we describe three simple ways of implementing a mobile record system:

Non-interactive SQL: In this case a full-fledged database system (e.g. Microsoft Access) runs on the client mobile device. During the day the user interacts with the local database and at the end of the day user manually synchronizes with the central database. This approach has two benefits. First, this allows the system to operate in disconnected environments. Second, this approach is not bandwidth constrained since all the operations can be synchronized with the centralized database in bulk over a high bandwidth connection or physical connection to the central database. This non-interactive approach is currently adopted by many existing systems [3, 4, 6].

There are fundamental problems with this approach. The period of non-interactivity with the central database can be very high in such a system, particularly if the user is physically remote from the central database location; in many scenarios, the health workers or patients may not regularly travel to areas with higher-bandwidth GPRS connectivity to synchronize the database. We found that a significant fraction of rural patients visit the neighboring town once in 6 months [5] During such extended periods, the devices could be damaged or stolen, irretrievably destroying collected data. Also, such solutions require powerful (and commensurately expensive) mobile devices with significant storage capable of running computationally intensive database applications. Furthermore, the advanced features provided by a complete database system are not needed. Finally, without some definite (and simple) model for intermittent consistency, user updates and queries to the database may yield stale or incorrect results.

Interactive SQL: In this case a user-friendly frontend interfaced with a SQL query engine running on each mobile device communicates with a centralized database running on a remote server. The application intqeracts

[1]CD4 count represents the level of T-helper cells in the bloodstream and is used along with viral load tests to indicate the prognosis of HIV in an infected patient.

with the query engine, which in turn sends SQL queries to the remote server and receives responses. Many of the existing interactive systems [3, 4] depend on data connectivity solutions such as GPRS/EDGE and rely on the TCP/IP socket interface, not present on low-end phones common to the contexts we are considering.

SQL over SMS: Existing SMS based systems [1] are inefficient and offer only limited functionality. Sending a single complex SQL query can easily result in sending large numbers of SMS messages, and a single poorly written query (e.g. `select * from foo`) can return huge datasets using large numbers of costly messages; in the regions we are targeting each SMS costs 5-25 cents and thus the total cost for such a query could run into the hundreds of dollars for a large table.

Optimizing the semantics of the usage scenario to the low bandwidth of SMS is the main problem that ELMR seeks to address. In the next section, we describe the design of ELMR in greater detail.

3. ELMR DESIGN

At a high level, the architecture of ELMR is simple: a client running on the user's mobile phone accesses the remote database by sending and receiving SMS messages to and from the server. The novelty lies in the manner in which these messages are encoded.

Each type of data interaction (i.e. form or information request) is defined with a schema, the sum of which are contained in a Schema Description File. This file is an XML document defining each schema's fields and their associated datatypes, interaction modes (e.g. read/write, read-only), etc. Identical copies of this file are held on remote clients and the server.

The client maintains a small queue of recent operations. When the queue is full, or the client needs immediately access to data present on the server, it builds the required SMS messageges for the queued operations using the appropriate schema(s) and sends the messages to the server. Upon receipt, the server decodes the messages according to the same schema definitions and responds accordingly.

3.1 Restricted Set of Operations

ELMR's motivating problem is medical record systems, and it targets developing regions where the only data connectivity is via SMS. While there are innumerable possible SQL queries that may be issued in order to access the database in the traditional SQL model, if we look at domain-specific operations required for medical record systems (e.g. create, fetch, update, and search), only a few specific queries are useful, and we design ELMR around these. This allows the client application to be simple and lightweight.

We support only the following operations: `create`, `append`, `update`, `destroy`, `search`, `fetch`, and `aggregate fetch`. In a given command message, the first 3 bits contain the operation ID and the subsequent 8 bits contain the schema ID. Following these are various bits reserved for control information and the payload containing the variable-length data sent to perform the operation specified by the operation ID.

3.2 Semantic Compression

ELMR employs semantic compression which reduces the number of messages exchanged between the client and server to complete an operation by reducing payload space required for each command's requisite data. Each schema, each field of each schema, and each operation is assigned an identifier using a small number of bits, concatenated appropriately to form a small fixed-width record.

To better utilize the space in each SMS message we limit the types of fields in a schema. These simple data types encompass the fields typical in medical forms:

1. *Date*
2. *Integer*
3. *String*
4. *Boolean*
5. *Multiple Choice*

We optimize by automatically assigning the minimum number of bits to each datatype (e.g. 1 bit to booleans) during schema encoding and decoding. In addition, each integer and date field carries a precision modifier and each multiple choice contains a finite number of options, which aids in correctly assigning the minimum number of bits to make the most of the 140-byte SMS payload. Variable-length strings have a 1-byte length value prepended to them. An example of a definition might be

```
<field name="weight"
       type="integer"
       min="0"
       max="511"/>
```

which would cause the semantic compression engine to appropriately allocate 9 bits to the value of this field. Future implementations of this engine will take data distribution into account to provide better compression over large data sets with high probability, e.g. allocating 8 bits as most weights fall under 256 and using extra bits as necessary for outliers.

3.3 Lightweight SMS Reliability

SMS message delivery is not 100 percent reliable, nor are the messages that do arrive guaranteed to be in order. In rural regions SMS loss rates have been observed to be as high as 10%. Since ELMR uses SMS service for communicating with the server, reliability is important

to our system. Implementing a reliability model similar to TCP/IP would be excessively complex and expensive over SMS. For example, a sliding window protocol is designed for efficient bulk data transfer, but in ELMR bulk data transfer is not a primary design goal. Also, protocols that require multiple rounds of acknowledgements are out of the question since each message incurs a relatively high cost. To keep the cost as low as possible for the user, we propose a thin layer of reliability that attempts to, but does not guarantee 100% reliability.

In ELMR every operation is client-driven and not server-driven, i.e. pull and not push. Interactions between the client and server are divided into sessions, each of which is comprised of one or more aggregated operations. The basic protocol for reliability is a stop-and-wait protocol across a session as opposed to across each message. We briefly outline our protocol below.

Session Definition:

1. Every session has a unique 8 bit identity.
2. Each message in the session contains a 4-bit sequence indicator.
3. A session contains at most 16 messages.
4. A session has 3 phases.

Phase 1: The client initiates a session[2] and then sends at most 16 messages to the server. The first of these messages has a single bit set indicating that it is the first message in the session and uses its sequence number field to indicate the total number of messages in the session. The server waits for SMS messages and does not send any acknowledgements.

Phase 2: After either a certain amount of time passes and the session times out or all messages are successfully received by the server, the server sends a message with a 16-bit ACK vector. The vector contains 1's for the sequence numbers which have been received, and 0's for those wihch have not. The client retransmits these missing messages, if any, when the ACK vector is received. If the server is waiting for missing messages but does not receive them, it attempts one more round of sending an ACK vector and waiting for a response, then fails the session. Once the server has received all messages, either in the first round or in a subsequent retransmission round, the ACK may either be sent immediately or piggybacked on a response in phase 3. If messages are still not received the server resends a NACK one last time. If messages are received the server sends an ACK.

[2]it may optionally send an initialization vector to initialize its encryption, and would then an ACK back from the server containing the same IV.

Phase 3: The server sends up to 16 messages in response to the client in a session. In this case, the client does not ACK the server messages unless the client receives some of the messages and others are lost. This is the mirror of phase 2 from the client side. If all messages are received or all messages are lost, the client does not send a NACK.

In this protocol, a session may not complete under three scenarios:

1. All messages from client to server are lost.
2. All messages from server to client are lost.
3. Transmission of the initialization vector is lost in either direction.

In any of these cases, the client times out and asks the user whether he wishes to retry the session.

3.4 User Driven Consistency

Consistency requirements vary depending on the environment and application of the system. In scenarios where explicit consistency guarantees are required, timestamps and conflict resolution mechanisms can easily be incorporated into our design to resolve conflicting updates in favor of the latest update. In addition, inconsistencies can either be handled by the system administrators or the users themselves. However, in other scenarios where perfect consistency is not required ELMR allows reduced consistency at a lower cost. This is useful for a data collection system where only a single CHW has ownership over certain records and immediate updates are not necessary. To facilitate these common types of intermittency requirements we have designed ELMR to operate with both immediate and lazy acknowledgements.

Immediate Acknowledgements - In an immediate acknowledgement model the server sends an acknowledgement immediately after each session initiated by the client. The client also sends an acknowledgement back to the server in order to notify the server that the client has received the response from the server.

Lazy Acknowledgements - In the lazy acknowledgement model the client does not expect an immediate acknowledgement from the server. Instead, the client caches transaction for which no acknowledgements are received and marks them as not committed, or 'dirty'. The server is then responsible to acknowledge these transactions in aggregate during future interactions with the client. The client deletes these cached transactions only when it receives an acknowledgment from the server.

3.5 Privacy

Table 1: Example user permission information maintained by the server

Phone Number	ID	Secret Key	Permissions
918932	213	132781029	*Schema1,op1,op2,flag* *Schema2,op3,op4,noflag*

Privacy and secrecy are of paramount importance in any system dealing with medical records. ELMR allows any client to send a request, but the server decides whether a request can be executed or not. In order to restrict access to the database, in ELMR, we implement security based on symmetric key stream encryption. The server maintains a list of phone numbers, their access types, and their secret keys. Each client is given a secret key at the time when client application is deployed on client's cell phone. Periodically, the client generates an initialization vector meant to create an updated version of its symmetric key. The periodicity of this behavior is tunable based on cost/privacy constraints as the two additional messages do incur some economic cost. Additionally, a 16-bit CRC is added to each message. Together, these prevent replay attacks and substitution attacks.

ELMR implements simple privacy and authentication models. The privacy model of ELMR addresses two basic questions: first, who can access the parts of the database and second, to what extent can an authorized user access each record? The authentication model ensures that the sender is actually who he says he is. In ELMR these requirements are satisfied by the server maintaining in a table the authorized users and their permissions (Table 1).

The first, second, and third columns are self-explanatory. The last column specifies the list of possible operations on different schemas with an extra flag field that is used to provide record level access control. If the flag field is present for a particular user that means only those record would be available to the user which have got their primary field equivalent to the user ID specified in the privacy table.

Upon receiving the properly decryptable message, the server verifies that the client under consideration has sufficient privileges to perform the operation which is being requested. If the client does not possess these privileges, the server aborts the request and returns an access violation message back to the client.

3.6 Further Optimizations

ELMR is designed to be a cost-effective solution. Therefore, minimizing the number of SMSs exchanged is essential. There are a number of simple optimizations that are incorporated into ELMR that are helpful and implemented to various degrees. We have not yet evaluated these in detail as they are minor compared to the other aspects of our system design.

Batched Updates - In ELMR the client caches updates and append transactions in its local memory and sends them to the server only when client has sufficient data to construct a nearly complete SMS. This greatly reduces the number of SMSs transferred.

Lazy Updates - We optionally queue updates as above, and send them opportunistically appended to realtime fetch requests, in order to save payload space.

Further Optimizations - Depending on the consistency level required for the application, a further optimization could be to send the delta between the cached version and the updated version of the record. In some cases this would be smaller than simply sending the entire set of operations.

4. EVALUATION

We performed a preliminary evaluation of ELMR using real medical forms used by hospitals in Africa. Due to privacy concerns we were unable to obtain live data to drive our evaluation, so we were forced to use simulated field entries for our results. Our evaluation is composed of two parts: we evaluate the effectiveness of ELMR at reducing the size of messages and analyze the message overhead of our reliability protocol.

4.1 Message Size

We began this portion of the evaluation by translating a complete new patient form and a checkup form into the ELMR schema format along with the possible range for each field in the form. A HIV simple patient intake form (Intake), and a longer followup form for HIV/TB were translated for use with ELMR. These two forms were chosen since they are common medical database operations in the regions we are focused on. We updated the schema by automatically assigning the optimized type for each field using ELMR's semantic compression scheme. We then populated the data in the forms using random numbers generated according to the length of the numeric fields, and single words taken from a small dictionary randomly. To arrive at a lower bound on the benefits of using ELMR we have not optimized the Intake or HIV/TB forms in any way. These forms are both quite large (109 and 195 fields respectively), and would likely be streamlined for field use. Also, strings can often be converted to multiple choice with a large dictionary of drugs and typical fill-in-the-blank answers. Figure 1 illustrates the resulting average size per data type. As the effectiveness of semantic compression is a function of the data types, we broke down the results according to the data types of the original forms. Gzip is the average compression rate for single messages. Gzip(100) refers to the average operation across 100 aggregate operations. ELMR is the

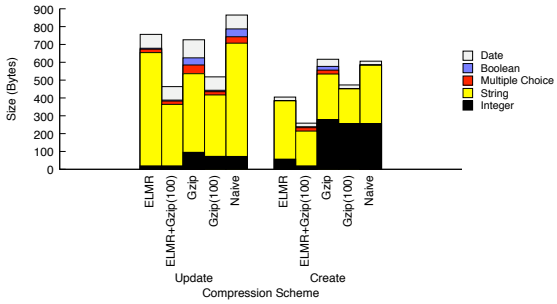

Figure 1: Field sizes after various compression schemes

Figure 2: Lightweight SMS reliability vs SMS loss rate

benchmark where only semantic compression is used. We observe that semantic compression improves upon the uncompressed version for both Intake and HIV/TB. We can see that the uncompressed performs the most poorly in both cases. The Gzip compression performs only slightly better than uncompressed for HIV/TB and worse for Intake, and the Gzip(100) further compacts strings across messages. ELMR is not currently optimizing string fields using semantic compression, but if we simply compress the strings in ELMR using Gzip we achieve the best results as seen in ELMR+Gzip(100) reducing the unmodified payload size by nearly 50% in the aggregate case.

4.2 Reliability and Consistency Overhead

Because there is no direct comparison to be made with existing systems that rely on TCP/IP, we implemented a simple SMS message simulator to measure the overhead of ELMR's SMS reliability and consistency protocol across different loss rates. Using the same messages as before, we simulated our protocol over a range of loss rates from 0 to 20%. We setup this experiment with each operation being sent as a session with no message aggregation and immediate acknowledgements. The message payloads are semantically compressed. Finally, if an entire session is dropped, we simulate a user resending the message after the timeout. Our results are shown in Figure 2. We observe from the results that the overhead is low even for loss rates of up to 20%. Our protocol manages to provide reliable communication while incurring minimal additional overhead well beyond the expected maximum loss rate of 10%.

5. CONCLUSION

In this paper we have presented ELMR, a system designed to meet the urgent need for better communications in healthcare in the context of the developing world. It has been designed around the context-specific goals of robustness and cost-effectiveness, and the more general goals of privacy and efficacy.

We believe that ELMR fills a significant gap in service provision in the context in question and hope to deploy pilot projects shortly in Africa and India with our partners.

6. REFERENCES

[1] FrontlineSMS. http://www.frontlinesms.com/.
[2] International Telecommunication Union. http://www.itu.int/.
[3] OpenMRS. http://openmrs.org/wiki/OpenMRS.
[4] OpenRosa. http://www.openrosa.org/.
[5] A. Sharma, M. Paik, A. Meacham, G. Quarta, P. Smith, J. Trahanas, B. Levine, M. A. Hopkins, B. Rapchak, and L. Subramanian. The Case for SmartTrack. *IEEE/ACM Conference on Information and Communication Technologies and Development (ICTD)*, 2009.
[6] Voxiva. http://www.voxiva.com/platform.php.