Design and Implementation of a Short Message Service Data Channel for Mobile Systems*

Earl Oliver
David R. Cheriton School of Computer Science
University of Waterloo
eaoliver@uwaterloo.ca

ABSTRACT

The Short Message Service (SMS) is one of the most ubiquitous wireless technologies on Earth. Each year hundreds of billions of messages are sent, demand continues to grow, and competition between cellular providers is driving prices down. These trends create practical opportunities for the use of SMS in mobile systems. In this paper we present the design and implementation of an SMS-based data channel, or *SMS-NIC*, that is lightweight and runs on multiple mobile platforms. Through integration with an existing mobile platform, we show that the SMS-NIC has little operational overhead and provides efficient, reliable transport for large messages send over the cellular network, where message losses frequently occur.

We motivate the design of the SMS-NIC through a characterization of SMS using workloads consisting of bursts of messages between sender and receiver. This analysis differs from previous SMS studies by focusing on transmission patterns that differ from normal cell phone use. Through this characterization we show that message delay and message loss are affected by the transmission order, the time of day has minimal effect on transmission rate, delay, and loss, and that losses and high delays significantly outweigh the rate or message reordering.

1. INTRODUCTION

Since its conception in 1991 [8], GSM has evolved to one of the most ubiquitous technologies on the planet. One of the services provided by GSM is the Short Message Service (SMS). SMS allows cell phones to exchange short messages with each other or various services such as Internet search, calendar notification, e-voting, etc. In 2005, over a trillion Short Message Service (SMS) messages were sent and received by cell phones all over the planet [10]. This number is predicted to increase to a staggering 3.7 trillion messages by 2012 [11]. In many countries, competition between GSM service providers, coupled with the growing demand for Multimedia Messaging Service (MMS), has driven the cost of sending SMS messages (a.k.a *wireless text messages*), to fractions of a penny or free. Today in the United States, unlimited SMS packages cost as low as $5/month [4]. As use of the cell phones continues to climb, we expect the prices for basic SMS service to continue to fall.

The low-cost of SMS and the ubiquity of today's cellular networks presents interesting opportunities for its use in mobile systems. Today there are many mobile systems that could benefit from using SMS as a data channel. In particular, DakNet [9], KioskNet [13], Haggle [12], and DieselNet [1], could exploit SMS to improve and coordinate routing, provide end-to-end message delivery notification, track vehicles, establish cryptographic session keys, etc.

As a data channel, SMS is greatly inferior to EVDO and GPRS/EDGE cellular data services. SMS has significantly lower data rates, high latency, a small fixed message size of 140 bytes, and messages are *frequently* lost during transport. However, data services are very expensive, sparsely deployed in developing regions, and while they can send megabytes of data effortlessly, exchanging kilobytes of data is sufficient for many applications. In 1999, the Enhanced Message Service (EMS) [2] was defined as an application level extension to SMS. Using EMS, devices may send messages as large as 918 bytes. This is an improvement; however, for SMS to be used as a general purpose data channel, we require a means to reliably transfer much larger messages.

In this paper we present the design and implementation of an *SMS-NIC*, which provides a reliable data channel built on top of SMS. We have designed the SMS-NIC to run efficiently and reliably on a variety of resource constrained mobile devices. The SMS-NIC is implemented in Java Micro Edition and complies with the CLDC profile. Therefore the core of the SMS-NIC can run on both personal computer environments and CLDC enabled cell phones and smart phones. Environment specific functionality such as logging, sending and receiving messages, and UI feedback are achieved through abstract interfaces that are passed into the SMS-NIC at startup. The current release of the SMS-NIC has full support for both BlackBerry[1] and Linux; however, support for other devices can be added with minimal effort.

To motivate the design of an SMS-NIC, we present a preliminary characterization of SMS that builds upon previous work by Zerfos et al. [16, 7]. In these papers Zerfos examines SMS traces collected by a cellular carrier in India over a three week period. The traces consist of over 59 million SMS messages exchanged by more than 10 million users (approximately 10% of India's total mobile subscribers). These traces are used to classify the current uses of SMS and measure how conversation threads progress across a series of messages. One of the key contributions of this paper was as a preliminary classification of the behaviour of SMS messages

*Technical Report CS-2007-42

[1]http://www.blackberry.com

are they traverse the cellular network. In particular, the authors observe that nearly 5.1% of messages are lost during transit due to expiration or denial of delivery. They measure message delay and found that 73.2% of messages reach their recipient within a 10 second delay, 17% require more than one minute, and the remainder take over an hour and a half. A data set this size is probably an accurate macro representation of SMS; however, we believe that an aggregate characterization does not provide the details needed to design a data service built on top of the SMS.

This paper builds on the work by Zerfos by examining the behaviour of SMS from a micro perspective. We examine a system of two cell phones connected to commodity PCs and measure the characteristics of SMS while sending messages between the two. As our study is biased towards a design of an SMS data channel, we focuses on traffic patterns that differ significantly from *normal* [7] human generated SMS traffic. To maximize throughput, the SMS-NIC must send messages as fast as possible - much faster than a human cell phone user could manually send messages. While previous work observes the presence of mass message senders, it does not examine them as an isolated group.

This paper makes the following contributions:

- We examine the characteristics of SMS for traffic patterns not considered in previous studies.
- We present the design and implementation of an SMS data channel for mobile system.
- We detail the integration of our SMS-NIC with an existing mobile system.

This paper is organized as follows. Section 2 characterizes the behaviour of SMS when sending many messages between cellular clients. Section 3 describes the design, architecture, and implementation of the SMS-NIC. In Section 4 we detail the integration of the SMS-NIC with an existing mobile system. We conclude in Section 5.

2. SMS CHARACTERIZATION

When designing a network protocol, it is important to understand the characteristics of the underlying network. Previous studies [16, 7] characterizing SMS have examined the service from an aggregate macro perspective. These studies do not consider the characteristics of SMS for single users sending bursts of messages and do not provide enough insight to design an SMS data channel. We seek to better understand the following properties of SMS:

- **Transmission rate**: The time required to transmit an SMS message from the phone is affected by phone's signal strength, medium contention, and communication latency with the phone circuitry.
- **Delay**: Once an SMS message has been accepted for delivery it is subject to several sources of delay: propagation delay as the message traverses the cellular network, queuing delays throughout the network, and transmission delay as the message waits for the recipient to be available. The network may also impose delays on a per client basis to prevent from flooding the network with messages.

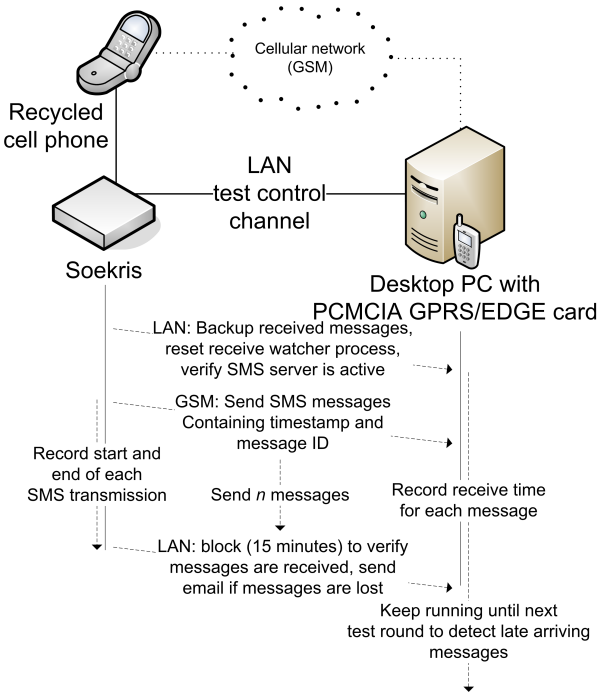

Figure 1: High level test overview.

- **Loss rate**: SMS messages can be lost due to transmission failure, congestion in the cellular network(s), or be rejected in transit by data corruption. If a receiver is not available, a message may also expire in the network while waiting for delivery.
- **Message reordering**: Depending on the design of the cellular network, messages may arrive in a different order than they were sent.

These properties may be affected by the time of day, the day of the week, and the number of messages sent at a particular time. In the following sections we examine these criteria and present our results.

2.1 Test Bed

Our test bed consists of a low-power, low-cost computer from Soekris Engineering (net4801) and a Pentium 4 desktop PC. The Soekris computer has the same configuration currently used in [13] to provide low-cost Internet to rural regions. The Soekris has a single USB port, which is used to attach a recycled Nokia 3390 phone. The Pentium PC was equipped with a PCMCIA PCI card that housed a Sony Ericsson GC82 EDGE card. The cell phone and EDGE card were on the same cellular network (Rogers) and located in the same room. Sending and receiving SMS messages was performed using an application called Gammu[2]. There are several open source packages for interacting with cell phones. Gammu was chosen because of its active support for the Nokia FBUS protocol and the Hayes AT instruction set. Both machines were accessible over the University LAN, which was used as a test data channel. Both computers were synchronized to the University's NTP server. Accurate time synchronization is essential for tracking an

[2]http://www.gammu.org

SMS message as it is transferred across the test bed.

The cellular signal strength on the Nokia was approximately 60% throughout the tests. Unfortunately, the phone did not support querying the signal strength directly; the signal strength was observed manually by periodically checking the indicator on the phone's display. The EDGE card did support signal strength monitoring, and it had a 90% to 100% signal throughout the experiments.

Linux support for USB attached cell phones and PCMCIA cellular cards is marginal. In the event of a transmission failure (message loss), it was necessary to manually verify through the Gammu logs a loss was not due to a communication failure between the computer and cellular device. Both computers were equipped with sendmail and configured to send warning emails to the experimenter when a failure was suspected.

Figure 1 illustrates the test bed setup and provides a high level overview of the test procedure that we will now describe.

2.2 Evaluation

We evaluate the characteristics of SMS using two forms of tests. Our primary evaluation of the characteristics of SMS consists of sending 10 messages per hour from the Soekris (*client*) and the desktop PC (*server*). We chose to send 10 messages because we believe it represents a typical message size (approx. 1400 bytes) for our application (as discussed in Section 4). In each test, the client first connects to the server over a LAN connection and backups up previously received messages. This was done purely as a precaution; in the event of a failure, we can reconstruct our results from the received files. While connected to the server, a process is invoked in the background that periodically checks for newly received SMS messages and records their arrival time. The scanning period of this process is 50 ms, which introduces on average a 25 ms margin of error on our evaluation of delay.

Once the environment is correctly configured, the client begins sending SMS messages to the server. The client sends an SMS using a synchronous call to the Gammu application. Gammu sends SMS messages over the serial connection with the Nokia cell phone using Nokia's proprietary FBUS protocol [15]. We measure transmission time by sampling the timestamp with millisecond resolution before and after the call to Gammu. Therefore transmission time consists of both the wireless transmission time and the time to communicate with the cell phone. Each SMS message sent to the server consists of the following data: an integer representing the current hourly test, an index for the current message, and additional random characters to pad the message to a full 140 bytes (160 7-bit characters [3]). In preliminary experiments we found that a message being full or partially full had no effect on its transmission time or delay; however, we chose to pad messages to 140 bytes because in practice, it would make sense to maximize the data payload.

On the server side, SMS messages are received using the SMS daemon, *smsd*, service provided by Gammu. smsd communicates with the EDGE card using the Hayes AT instruction set. Unfortunately, this requires that smsd poll the card for new messages. We configured smsd to poll the card in 1 second intervals. By polling we introduce, on average, an addition 0.5 seconds of delay. This was the most aggressive polling interval possible. Messages retrieved from the phone are each written to file on the server and their arrival time is recorded by the previously mentioned scanning process. The time to write the file has negligible effect on delay.

When the client has finished sending, it connects to the server to wait for all messages to be received. A failure to delivery all of the messages triggered an email to the authors. Periodic anomalies (such as excessively long delay) were then verified manually.

2.3 Analysis

Over a period of seven days, we successfully transmitted a total 1644 messages from the client to the server. Our transmission success rate was 97.85% with 15 messages rejected by the network, and 21 messages that failed to send due to software failure on the client side. Contrary to our expectation, we found that the day and time had no effect on the transmission rate and loss rate of SMS messages, and had only a small effect on delay. However, the number of messages sent at a single time had a significant effect. The following sections detail our characterization of the transmission and loss rate, and delay of SMS. Aggregate results over the seven day period are summarized in Table 1.

2.3.1 Transmission Rate

The transmission rate was the most consistent variable in our study. This value was calculated for each message successfully sent by the client. Our average transmission data rate was approximately 25.052 bytes/second (0.18 SMS messages/second). Figure 2 illustrates the transmission rates over the course of the experiment. The gap on Wednesday night corresponds to a failure at the client side. No messages were sent for two hours. This period of software failure was not included in the results of this study.

2.3.2 Delay

SMS delay was a highly variable quantity in our study. Average delay over all successfully received messages was 289.31 seconds. The minimum and maximum delay was 3.19 and 14534.32 seconds respectively and the standard deviation was a staggering 1247.83 seconds. These values were inflated by a series of highly delayed tests during Thursday as illustrated in Figure 3. Although this period of high delay is based on true SMS

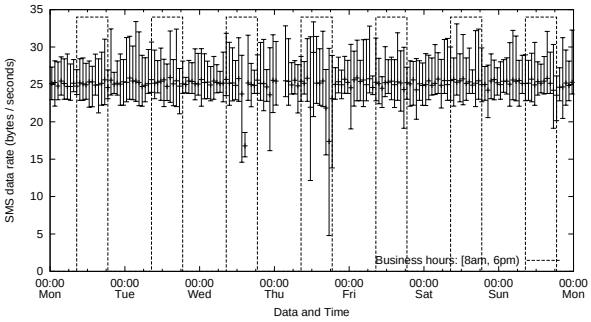

Figure 2: SMS transmission rate over seven day period.

	Mean	Minimum	Maximum	Median	Standard deviation
Time (sec)	5.59	4.19	29.23	5.63	0.76
Rate (bytes/sec)	28.63	5.47	38.19	28.42	209.75
Delay (sec)	289.31	3.19	14534.32	14.00	1247.83

Table 1: Aggregate SMS transmission time, data rate, and delay over seven day period.

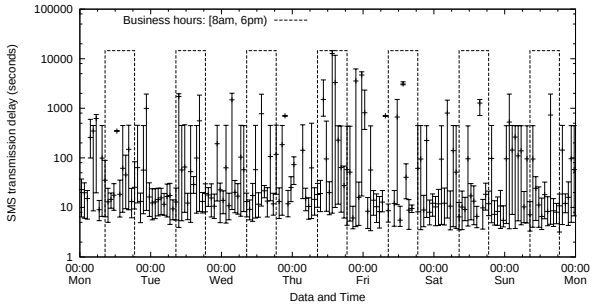

Figure 3: SMS transmission delay over seven day period.

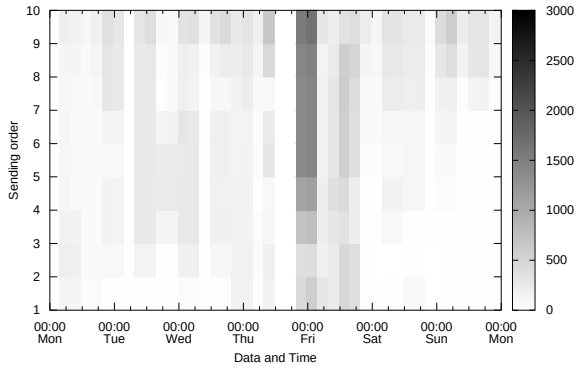

Figure 4: SMS delay (seconds) over seven day period with respect to the transmission ordering.

traffic, it is unlikely that Thursday regularly exhibits high SMS delay. We re-ran our experiment on a subsequent Thursday and Friday to reassess SMS traffic on those days. We found that delay on the subsequent Thursday was consistent with the other days of the week, and ruled out a correlation between delay and the day of the week. The most useful aggregate result of our experiment was the median delay. This value was found to be 14 seconds and can be observed in the delay CDF in Figure 5.

We found that the order that a message was sent had an effect on the delay. Mean delay for the first and second message sent was 218.9 and 170.4 seconds respectively and increased to 425.8 seconds for the tenth message sent. Median delay was also lower for both the first and second messages; however, median delay was roughly bounded to 14-15 seconds for the subsequent messages. The mean and median delays with respect to transmission order are illustrated in Figures 6 and 7. This trend is also visible in Figure 4, which depicts delay with respect to order for each day of the week.

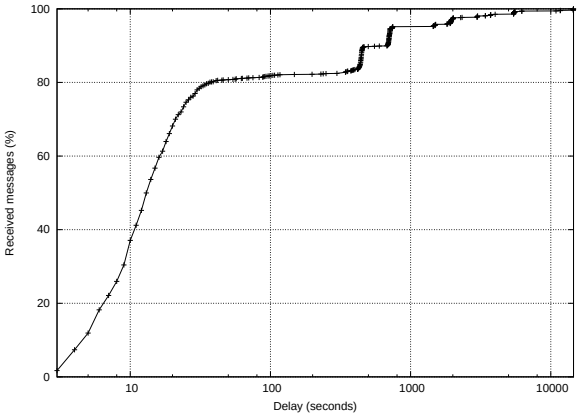

Figure 5: Cumulative distribution function for SMS delay using one second sample intervals.

Note that the high delay times illustrated on Thursday night (up to 14534.32 seconds) have been omitted from this diagram because these results dwarf all other results. This diagram does not illustrate delays over 4000 seconds.

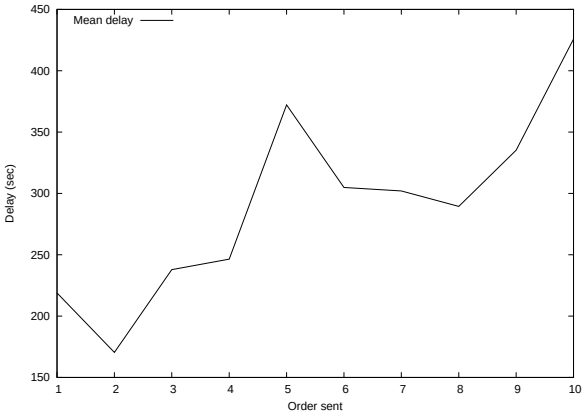

Figure 6: Mean delay with respect to the order sent.

Because our mean delay is heavily included by a few high delay cases, in the rest of this paper will consider the median delay as a good approximation for delay in the SMS network.

2.3.3 Loss

The loss rate was also a consistent variable in our study. Over the week of our test, the SMS loss rate was 3.89%. We observed peaks in losses during business hours on both Thursday and Friday. Re-running the experiment on a subsequent Thursday and Friday found no evidence that SMS traffic was consistently in-

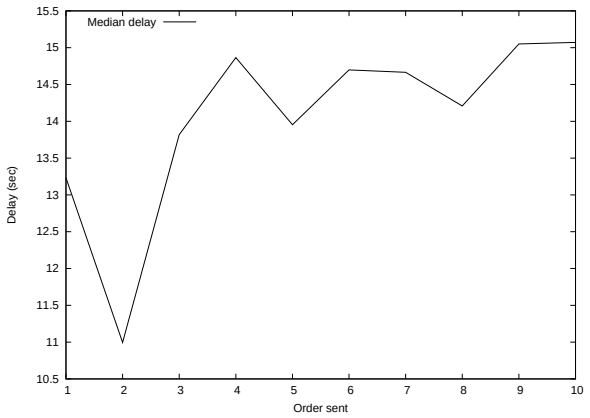

Figure 7: Median delay with respect to the order sent.

creased on these days. We found that the spike of losses on Monday morning was also an anomaly. These results are illustrated in Figure 9. We examined loss rate with respect to the order that messages were transmitted. We found no correlation between the loss rate and transmission order. These results are illustrated in Figure 8.

We also observed that SMS messages were duplicated at a rate of 3.1%. Duplicate messages are often a side effect of poor communication between the phone and the service provider. By manually examining the Gammu smsd server logs, it was verified that the messages were not received twice from the cellular network. The duplications were caused by a bug/feature of Gammu related to resetting its connection with the phone. While the duplication rate is not a measured characteristic of SMS, we include this observation because because it is relevant to the design of the SMS-NIC.

3. DESIGN AND IMPLEMENTATION

We begin the design of the SMS-NIC with a summary of the key points from our characterization:

- **Transmission failures occur**: Transmission errors due to both software errors and a busy network occur approximately 2.2% of the time. The sending component of the SMS-NIC should adapt to local failures.
- **Transmission rate is consistent**: The SMS-NIC does not need to adapt to transmission variability. Delay is the only variable duration.
- **Transmission order affects delay**: When sending a burst of messages, the first messages sent have lower delay than subsequent quantity.
- **Loss rate is low**: The SMS standard has a built-in delivery receipt mechanism where the delivery of each message is acknowledged. A delivery receipt effectively doubles the number of messages exchanged. With a loss rate as low as 3.89% the SMS-NIC should use a less aggressive acknowledgement (ack) strategy.
- **Day and time have little effect**: Contrary to our expectations, the day of the week and time of day have little effect on the rate, delay and loss of SMS.
- **Messages are reordered**: Messages arrive out of order 2.53% of the time; however, when a message does not arrive as expected, is far more likely to be delayed or lost than reordered.
- **Messages remain intact**: Although it is not part of our characterization, it is worth mentioning that SMS guarantees message integrity [2].

Throughout this section we will refer to data passed by applications into the SMS-NIC as *data*. Data is fragmented and transmitted over the cellular network as SMS *messages*.

3.1 Protocol

The communication protocol between two SMS-NICs is designed to handled a wide variety of applications and user defined settings while maximizing the payload of a single message. Each SMS message transmitted contains a small fixed size header and message payload. Our current design consists of *short messages*, *standard messages*, and *control messages*. Short messages consist of data that is small enough to fit into a single SMS message. We predict that short messages will be the bulk of the traffic over the SMS-NIC, so we have included this special case to reduce the header size by two bytes. Standard messages consist of two or more SMS messages

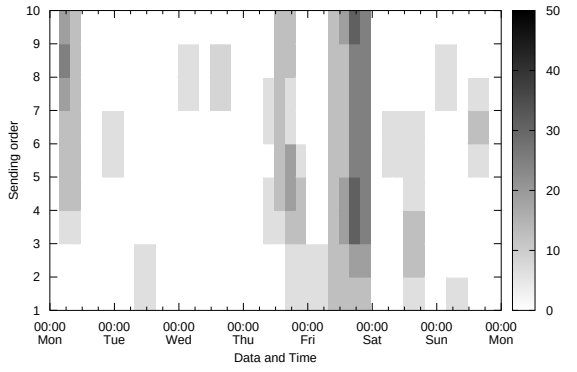

Figure 8: SMS loss rate over seven day period with respect to the transmission ordering.

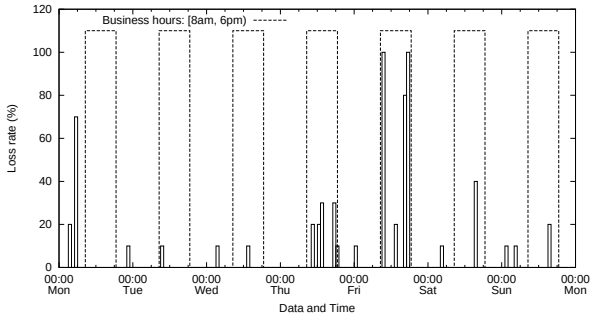

Figure 9: SMS loss rate over a seven day period.

up to 32 KB in size (243 messages in the best case). We believe that 32 KB is a practical and reasonable upper bound for an SMS based data channel. Systems that require larger messages should switch to a cellular data service such as GPRS/EDGE. We will discuss control messages later in this section.

Figure 10 illustrates the composition of short, standard, and control messages. The first byte consists of a two bit protocol version, a three bit message type, and three flag bits. The first flag bit indicate that window size value has been specified, which we will discuss later in this section. The second bit indicates that the length of the data is present. The final bit signals that the payload is compressed. The window size and length flags were introduced as an optimization. The sender initially sends all messages containing the window size and length of the overall data. Once the receiver has sent its first ack, subsequent messages sent from the sender do not contain the length. This saves an additional two bytes in the message payload. Similarly, the receiver also acks the RTT as measured by the sender and the sender's chosen window size.

There are three types of messages not included in Figure 10. As of GSM 3.40 [2], SMS messages can be transmitted on one of 65536 ports. Unfortunately, many devices do not support transmitting messages on a specific port. To support these devices, we have included additional types that include an additional two byte port field. These messages are received on the default port (zero), and processed as if they were received on different ports.

Two retransmission strategies were considered when designing the SMS-NIC: Simple Method to Aid ReTransmission (SMART) [6] and selective retransmission (SR) [5]. SMART has several favourable characteristics. First, the protocol is aggressive about retransmission. When an expected message does not arrive, the message is assumed to be lost and retransmits it. Through our characterization we see that message losses are more frequent than reorderings; SMART's assumptions hold for SMS. Second, the protocol avoids the use of timers to determine which messages have been correctly received and to recover from message losses. Timers are only used to recover from a loss of all messages and all acks. The transmission rate of SMS is low enough that frequently setting and resetting timers is not a costly operation; however, on many mobile platforms, timers are implemented as independent threads. These threads

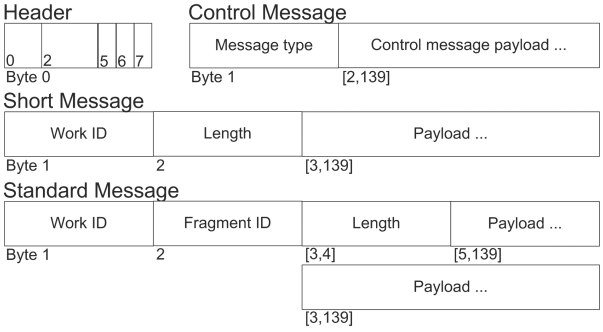

Figure 10: The format of SMS messages exchanged between SMS-NICs.

increase the memory and CPU footprint of the application and should be avoided.

In SR schemes, each ack contains a bit-mask for the messages that have been received. This way, the sender knows exactly which messages must be retransmitted, again, without the use of timers. Normally in SR, if the window size is large, the bit-mask can become a significant overhead. However, because we have chosen to limit the application data size to 32 KB, all 243 messages can be comfortably acked using only 31 bytes of an SMS message. We have therefore chosen to use an SR scheme to handle retransmissions.

Through our characterization, it is clear that we require a windowing mechanism to limit the number of messages sent. In this protocol, we provide flow control implicitly by limiting the window size to the number of messages that can be transmitted in two RTT for each transaction. The SMS-NIC calculates the window size for each transaction at the sender. Once the sender has determined the window size, the value is sent to the receiver and remains fixed for the remainder of the transaction. The window size is calculated by first determining the RTT for the current transaction. The RTT is calculated by recording the timestamp when the first message is sent to the receiver. When receiving the initial message, the receiver always sends an ack message back to the sender. The sender then subtracts the initial transmission time from the time the first ack was received to produce the RTT.

The transmission time is used to determine how many messages can be sent in an RTT window. Message transmission time was highly consistent in our characterization. The SMS-NIC therefore calculate this value by taking the exponential moving average of the transmission times of each message. Averaging the transmission rate globally simplified the design of the SMS-NIC, and allowed us to decouple the application level data from the process of queueing fragments (messages) for transmission.

Once the RTT and transmission time are known, the sender calculates windows size as follows:

$$window_{transaction} = \frac{2(RTT_{transaction})}{transmission\_time_{global}}$$

Because an initial high delay can result in an unreasonably large window size, the SMS-NIC places an upper bound on window size to two times the window size obtained using the mean values from our characterization. The maximum window size is 28 messages. Once the window size is known, the value is transmitted at the head of all messages until the receiver acks that it has received the new window size. The window size ack is included as a field in a standard data message ack.

Although we have not evaluated this technique for optimality, our current design is conservative in its use of space and we have found that it works reliably in practice. We now present the software architecture of our implementation this protocol.

3.2 Architecture

Nearly every mobile environment, including cell phones, smart phones, and embedded systems, exposes a different set of APIs to applications. In addition to the characterization of SMS, the design of the SMS-NIC was influenced by a need for portability. The design

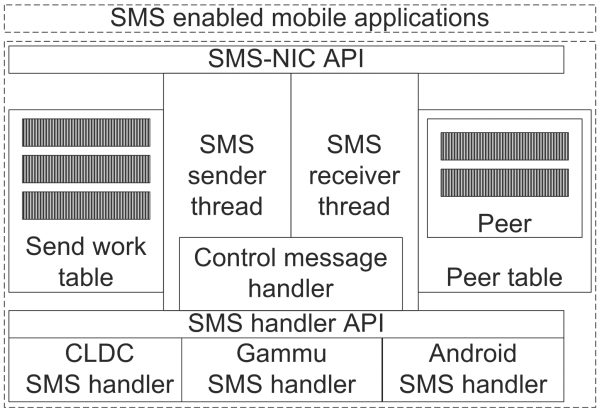

Figure 11: Architecture of the SMS-NIC including example SMS Handlers.

should be modular and The SMS-NIC should also have a small memory footprint and minimize the allocation and copying of memory. It should also minimize the use of threads and timers, which are both limited resources on a mobile device. Finally, for the NIC to provide the most value to existing mobile systems, it should be easy to integrate with minimal change to the mobile application.

The architecture of the SMS-NIC, as illustrated in Figure 11, consists of the following high level components:

- **SMS-NIC API**: The SMS-NIC API is the single interface into the SMS-NIC. This API mirrors traditional *send* and *recv* interfaces and includes flags to indicate that the data should be compressed, and a 16 bit port number to differentiate applications. The SMS-NIC API has five sub-interfaces: the SMS Data, SMS Address, SMS Logger, SMS View, and SMS Compression interfaces. Receive buffers within the SMS-NIC implement the SMS Data interface, which allows fully received data to be passed directly to applications without allocating new memory. The SMS Address interface is designed to ease integration of the SMS-NIC into existing applications. This interface is implemented by objects in the host application that are responsible for maintaining addresses and removes the need for redundant copying of addresses (typically strings) into the SMS-NIC. The SMS Logger is a simple interface that allows the host application to specify how/where the SMS-NIC should log its activities. The SMS View API provides a call back mechanism for the host application to be notified of changes to the SMS Model. This interface is primarily intended to support a GUI in the host application. Finally, each mobile platform provides a custom set of functions to compress and decompress data. We abstract this functionality from the SMS-NIC using the SMS Compression interface.

- **SMS Handler API**: Each mobile platform sends and receives SMS messages differently, and has a different representation of an "SMS connection" to a remote host. The SMS Handler API is the second major interface in this architecture, which is designed to abstract the SMS APIs of the host environment from the NIC. This interface provides only two functions: *sendSMS* and recvSMS. Through these two abstract functions, the SMS-NIC is able to send and receive SMS messages on a variety of mobile platforms.

- **SMS Model**: The SMS Model stores the current state and statistics of the SMS-NIC. This class allows objects in the host environment that implement the SMS View API to register their interest in changes to the model. As in the model-view-controller design pattern, when a change is made to the model, all views are notified of the change.

- **SMS Send Work**: The SMS Send Work component is a wrapper for data sent from the application layer. This component maintains the state of a transaction, which includes the number of transmission failures, the measured RTT, and the fixed window size. This component also contains a pointer to the original application data and is responsible for providing SMS sized *chunks* to the SMS Sender for transmission as SMS messages. Chunks are retrieved from the work object by a linear sequence number starting from the beginning of the given data. Each chunk returned from this component consists of a small header and a data payload. The reciprocal of this component is the *SMS Receive Work*, which reassembles messages into application data on the receiver's side.

- **SMS Sender**: The SMS sender is one of the two threads in the SMS-NIC. The primary function of this component is to dequeue data passed into the SMS-NIC for transmission, compress it if necessary, wrap the data in an SMS Send Work object, and transmit pieces of the data as SMS messages. The sender stores SMS work objects in a table, which is keyed by a one byte integer *work identifier* that is chosen in increasing order by the sender to uniquely represent a transaction. With an identifier space so small, it is possible for collisions to occur if more than 256 pieces of data are enqueued for transmission. We believe this scenario is rare, and is currently handled by an error returned to the sender signalling it to retry later.

  The SMS Sender maintains two sending queues: a high priority queue for control messages and a second queue for sending messages containing application data. Messages are placed in both queues by the sender thread and by the Control Message Handler when processing received acks. The use of a high priority queue is particularly important when responding to an initial message from a sender. If ack messages experience unusually high queuing delays, the sender's calculation of RTT and window size will be inflated. The sender transmits messages from the queues according to the current calculated window size. The sender is also responsible for recovering from transmission failures. In many cases, this involves simply pushing the SMS message back onto the sending queue; however, if transmission errors to a particular sender persist, the sender may drop the message and flag the corresponding work object for deletion.

To minimize the use of threads and timers in the SMS-NIC, the sending thread is not always blocking waiting for messages to send. Each blocking operations times out to allow the sending thread to discard received data that is incomplete (i.e. the sender stopped sending), expire data that was unsuccessfully sent, and retransmit data when a receiving acknowledgement has been lost.

- **SMS Receive Work**: On the receiving side of the SMS-NIC, data is reassembled within an SMS Receive Work object. This class maintains a receive buffer for the full data size and a bitmap of all received messages. The bitmap is stored with a pre-formatted control message. When sending an ack, this special message is retrieved from the work object and forwarded to the sender. As mentioned above, this component implements the SMS Data interface, which allows completed data to be passed to the application layer unmodified. Although reducing a Receive Work object to an SMS Data objects leaks SMS-NIC control information, it is more efficient than reallocating and copying a fully received array of data.

- **SMS Peer**: The SMS Peer is a simple component that serves as a mapping from SMS source address (phone number) to a set of receive work objects. This component is motivated by the fact that the space of work identifiers, while large enough for our application, is small enough to cause collisions at the receiver. When an SMS peer contains no work objects it is considered empty and is deleted.

- **SMS Receiver**: The SMS Receiver is the second thread in the SMS-NIC. The primary roll of this component is to block on the arrival of an SMS message. Once a message is received it takes two paths: control messages are passed to the Control Message Handler and data messages are passed to an SMS Peer corresponding to the message's source address. If a received messages completes an work object, then the work object is removed form its peer container and placed in an application receive queue.

  A secondary roll of the Receiver is to enforce upper bounds on both the data size and the number of concurrent work objects that can be handled by a receiver. Messages for data that are either too larger than a user specified limit or exceed the 32 KB are discarded. The Receiver drops messages for new receive work to prevent SMS based denial of service attacks.

- **Control Message Handler**: The Control Message Handler implements the communication protocol between two or more SMS-NICs. Architecturally, this component has access to both the sending and receiving data structures and is responsible for handling and dispatching all inter-NIC control messages.

The architecture of the SMS-NIC can be implemented on a variety of platforms and languages. In the next section we present the decisions made when implementing this architecture. We also detail the integration with an existing mobile system.

4. MOBILE SYSTEM INTEGRATION

When implementing the SMS-NIC, we considered both C++ and Java Micro Edition (JME). C++ is an efficient language for implementing low-level functionality and is supported on a range of mobile and embedded platforms; however, we found that OS support for sending and receiving SMS messages using C++ libraries was poor. Alternatively, Java runs on most mobile devices as an always-on JVM (ex. BlackBerry, Symbian, and Android), or as a third party application such as IBM's J9 JVM that runs on top of Windows Mobile. JME also supports sending and receiving SMS messages as part of the Connective Limited Device Configuration (CLDC)[3], which makes it an ideal platform for the SMS-NIC.

Ease of integration with an existing mobile system was a key design decision. We considered two open source systems implemented in Java that we believed could immediately benefit from an SMS based data channel: the Haggle Project [12] and the Opportunistic Connection Management Protocol (OCMP) [14]. Haggle is a network architecture designed to enable communication in the presence of intermittent network connections. Haggle assumes a lack of end-to-end infrastructure and relies on opportunistic connections between mobile devices to disseminate data. OCMP is a disconnection-tolerant, policy-driven session layer that allows applications to fragment data for transmission across multiple NICs opportunistically. In contrast to Haggle, OCMP is heavily dependent on infrastructure. Data is transmitted over different network interfaces to an Internet resident *proxy*. The proxy then resembles data and forwards it to another OCMP user (again opportunistically), or over the Internet to a legacy server. Internally, OCMP abstracts NICs through a common *Connection* API. We chose to integrate the SMS-NIC into OCMP because of this clean separation of NICs from the rest of the system.

The integration with OCMP had a minimal effect on the OCMP code base. Integration took less than three hours to perform. When finished, OCMP was able to send and receive messages on both Linux and CDLC enabled devices.

Our integration with OCMP consisted of the following steps:

1. To communicate across multiple NICs, OCMP *users* have multiple addresses. For example, they may have an IP address on both a WLAN and WWAN, a Bluetooth address, or a DTN end-point ID [13]. These addresses are stored in a common *Address* class. We modified this class to implement the SMS Address API and added a string telephone number as a new address type.

2. We extended OCMP's logging mechanism to allow the SMS-NIC to log its operations into the standard OCMP log.

3. The utility gained by compressing data is variable based on the content. OCMP leaves it to the application to compress data. We followed this trend and did not implement the SMS Compression interface.

[3]http://java.sun.com/products/cldc/

	GPS position (1 msg)	**2 KB RSA public key** (13 msgs)	**4 KB BLOB** (27 msgs)
Calculated average (sec)	39.18	103.64	193.47 (1 loss)
SMS-NIC (sec)	37.32	97.23	212.11 (3 losses)

Table 2: Performance of the SMS-NIC compared to an average measurements.

4. We modified OCMP's configuration sub-system with several new parameters. These parameters included the maximum number of concurrent work items, the maximum data size supported, and the number of concurrent peers to support. At startup, OCMP initializes an instance of the SMS Model with these configuration parameters.

5. To support sending and receiving SMS messages on Linux, we developed a simple wrapper for Gammu that implements the SMS Handler interface. This component consists of a single thread that invokes Gammu's smsd server, and polls for new messages. When a new message is received, it is placed in a queue, until the SMS-NIC performs a recvSMS call on the handler. To send messages we used the same technique used in the experiments. Messages are sent from the handler synchronously by invoking Gammu in client mode and sending the message.

   To support sending messages on CLDC enabled devices, we developed a second SMS Handler that utilized CLDC's Connector class. Unlike the Linux handler, using CLDC allowed the receiving thread to block when calling recvSMS - a far more elegant solution. Sending messages was also simplified by making a single function call instead of invoking an external application.

6. The core of OCMP currently has no GUI component so we extended OCMP's logging mechanism to view changes to the SMS Model. When a change to the model is made (i.e. a message is transmitted or received), the OCMP logger records an entry with the current statistics from the SMS Model.

7. Finally, we introduced a new class, the *SMS Connection* class, which implements OCMP's Connection API. Like the SMS-NIC API, the Connection API provides a simple set of operations for sending and receiving data on the NIC that is contains. The SMS Connection class was therefore a simple wrapper of the SMS-NIC that consisted of less than 30 lines of code. Like the TCP/IP and DTN Connection objects included with OCMP, the SMS Connection object is instantiated upon OCMP startup on both the client and proxy components of the system. During startup we initialize the SMS-NIC by passing it the OCMP logger, a Linux/Gammu SMS Handler, and the SMS Model containing its configuration parameters.

We performed a simple trial to evaluate the performance of the SMS-NIC while integrated with OCMP. We placed three files on an OCMP *client*, and transferred them through the SMS-NIC, over the cellular network to the SMS-NIC on the OCMP proxy. This trial was performed on the same Linux test bed as our characterization experiments. Table 2 illustrates the performance of the SMS-NIC against a calculated scenario with an average transmission rate, median delay, and an average loss rate. We found that the overhead of the OCMP and SMS-NIC, and the presence of real losses introduced only a small difference over the average case.

We note one limitation to our SMS integration with OCMP: it does not scale. By design, the proxy is a centralized component. With a single cell phone for sending and receiving messages, the SMS data channel will rapidly fail. The solution to this problem is to increase the proxy's SMS capacity. In future work we plan to add support for sending and receiving messages from an SMS gateway on the Internet. This functionality can be easily added using a new class that implements the SMS Handler interface.

5. CONCLUSIONS

In this paper we have characterized the behaviour of SMS when exchanging many messages in series between two clients. We have observed that the SMS transmission rate is a consistent value and transmission errors occur at a rate as low as 2.2%. We have found that transmission order affects delay and that the first messages transmitted has a lower delay than subsequent messages. We found that the loss rate of SMS is approx. 3.89% and that messages arrive out of order approx. 2.53% of the time. Finally, contrary to conventional wisdom, the day of the week and time of the day have little effect on the transmission rate, loss rate, or delay of SMS messages.

We have used our characterization to shape the design and implementation of an SMS data channel for mobile systems. We have integrated this data channel, or SMS-NIC, into an existing mobile systems platform (OCMP) and have developed a simple application for sending and receiving files over SMS. The compact design and lightweight implementation of the SMS-NIC made integration with OCMP effortless. We strongly believe that many mobile systems could benefit from the use of an SMS data channel, and anticipate more sophisticated applications using this work to emerge in time.

6. REFERENCES

[1] Umass dieselnet, 2007. Available from: `http://prisms.cs.umass.edu/dome/index.php?page=umassdieselnet`.

[2] 3rd Generation Partnership Project. 3GPP Specification, 2007. Available from: `http://www.3gpp.org/ftp/Specs/html-info/23040.htm`.

[3] 3rd Generation Partnership Project. Technical Realization of the Short Message Service (SMS); Release 6. v6.5.0, September 2007.

[4] AT&T Inc. Thumbs Up! AT&T Announces Unlimited Messaging Plans, 2007. Available from:

http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=23733.

[5] BT Doshi, PK Johri, AN Netravali, and KK Sabnani. Error and flow control performance of a high speed protocol. *Communications, IEEE Transactions on*, 41(5):707–720, 1993.

[6] S. Keshav and S. P. Morgan. Smart retransmission: Performance with overload and random losses. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, page 1131, Washington, DC, USA, 1997. IEEE Computer Society.

[7] X. Meng, P. Zerfos, V. Samanta, SHY Wong, and S. Lu. Analysis of the Reliability of a Nationwide Short Message Service. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1811–1819, 2007.

[8] C. Peersman, S. Cvetkovic, P. Griffiths, and H. Spear. The global system for mobile communications short message service. *IEEE Personal Communications*, 7(3):15–23, 2000.

[9] A. Pentland, R. Fletcher, and A. Hasson. Daknet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, 2004. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1260729.

[10] Portio Research. The Staggering Success of SMS: 1 trillion SMS messages worldwide in 2005, 2006. Available from: http://www.portioresearch.com/opinion1_sms.html.

[11] Portio Research. Worldwide SMS revenues to hit $67bn by 2012, 2006. Available from: http://www.portioresearch.com/press6.html.

[12] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A networking architecture designed around mobile users. *To flapper in Proceedings of the Third Annual Conference on Wireless On demand Network Systems and Services (WONS 2006)*, 2006.

[13] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 334–345, New York, NY, USA, 2006. ACM Press.

[14] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya. A policy-oriented architecture for opportunistic communication on multiple wireless networks. 2006. Available from: http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/06/ocmp.pdf.

[15] Wayne Peacock. Nokia FBUS Protocol Made Simple, 2005. Available from: http://www.embedtronics.com/nokia/fbus.html.

[16] Petros Zerfos, Xiaoqiao Meng, Starsky H.Y Wong, Vidyut Samanta, and Songwu Lu. A study of the short message service of a nationwide cellular network. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 263–268, New York, NY, USA, 2006. ACM.