Exploiting the Short Message Service as a Control Channel in Challenged Network Environments

Earl Oliver
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
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 SMS in today's mobile systems. In this paper we present the design and implementation of a robust SMS-based data channel, or *SMS-NIC*, that runs on a variety of mobile platforms. Through integration with an existing mobile system, we show that the SMS-NIC has little operational overhead and provides efficient, reliable transport for large messages sent over the cellular network.

We motivate the design of the SMS-NIC through a characterization of SMS using workloads consisting of bursts of messages between cell phones tethered to Linux PCs and between smartphones. This analysis differs from previous SMS studies by focusing on transmission patterns that differ from normal SMS use. Through this characterization we show that bidirectional traffic and the choice of hardware have a significant effect on transmission rate, delay, and message reordering. We also show that burst size has no effect on SMS, losses are rare, and messages may be duplicated during transport.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]

General Terms

Experimentation, Design, Reliability.

Keywords

Short message service, sms-nic, data channel, mobile systems, challenged network

CHANTS'08, September 15, 2008, San Francisco, California, USA.

1. INTRODUCTION

Since its introduction in 1991 [7], GSM has evolved into 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 services such as Internet search, calendar notification, e-voting, etc. In 2005, over a trillion SMS messages were sent and received by cell phones all over the planet[1]. This number is predicted to increase to a staggering 3.7 trillion messages by 2012[2]. 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 to fractions of a penny or free. Today in the United States, unlimited SMS packages cost as low as $5 per month[3]. As the 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 present interesting opportunities for its use in mobile systems operating in challenged network environments. Today there are many mobile systems that could benefit from using SMS as a control channel. In particular, DakNet [8], KioskNet [4], Haggle [9], 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, GPRS/EDGE, and other cellular data services. SMS has significantly lower data rates, high latency, a small fixed message size of 140 bytes, and messages can be 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 [2] (EMS) 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 evaluation of a general purpose data channel built on top of SMS. We have designed this *SMS-NIC* to run efficiently and reliably on a variety of resource constrained mobile devices. The SMS-NIC is implemented in Java and complies with the Java Con-

[1]http://www.portioresearch.com/opinion1_sms.html
[2]http://www.portioresearch.com/press6.html
[3]http://tinyurl.com/6az5gh

nective Limited Device Configuration (CLDC) profile. By abstracting OS specific functionality such as logging, sending and receiving SMS messages, and UI feedback, the SMS-NIC can run on both personal computer environments and CLDC enabled cell phones and smartphones. The current release of the SMS-NIC has been evaluated on both BlackBerry, (a CLDC compliant device), and Linux. Other platforms can be supported with minimal effort.

To motivate the design of the SMS-NIC, we present a characterization of SMS that builds upon previous work by Zerfos et al. [11, 5]. 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 messages exchanged by more than ten 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. Their work provides a preliminary classification of the behaviour of messages as 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 found that 73.2% of messages reach their recipient within a ten 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, by exchanging messages between pairs of devices. Our testbed consists of combination of BlackBerrys, Nokia cell phones, and a PCMCIA EDGE card. Because our study is biased towards a design of an SMS data channel, we focuses on traffic patterns that differ significantly from *normal* [5] human generated SMS traffic. Our tests send SMS 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 is organized as follows. In the next section we present the characterization of SMS using a variety of commodity hardware. This characterization shapes the protocol design and architecture of the SMS-NIC in Section 3. Section 4 provides implementation details and performance benchmarks. We conclude in Section 5.

2. CHARACTERIZING SMS

When designing a network protocol, it is important to understand the characteristics of the underlying network. Previous studies [11, 5] 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 hence do not provide enough insight to design an efficient SMS-based data channel. We seek to better understand the following properties of SMS:

- **Transmission time**: The time required to transmit an SMS message from the device is affected by the cellular signal strength, medium contention, and in some cases communication latency with the device 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 is delivered to the recipient. The network may throttle or artificially delay messages from some users to prevent them from flooding the network.
- **Loss rate**: SMS messages can be lost due to transmission failure, congestion in the cellular network(s), or be rejected in transit due to 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.

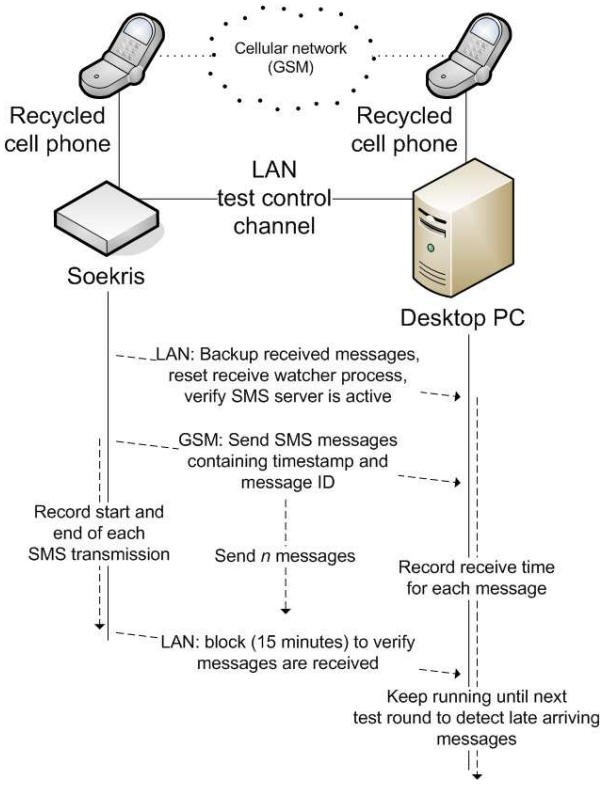

Figure 1: Tethered cell phone test configuration.

These properties may be affected by the time of day, the day of the week, burst size, and the device used to transmit messages. In the following sections we examine these criteria and present our results.

2.1 Experiment setup

We evaluate the behaviour of SMS using two configurations that reflect how the SMS-NIC would be used in practice: cell phones connected to commodity PCs and between SMS capable smartphones.

2.1.1 PC with tethered cell phone

This configuration mimics the hardware configuration found in existing challenged network deployments [8, 4, 1]. The testbed consists of a low-power, low-cost headless computer

	Configuration	Mean	Minimum	Maximum	Median	Std. dev.
Transmission time (sec)	Tethered cell phone	5.59	4.19	29.23	5.63	0.76
	Smartphone	4.03	2.98	39.36	3.34	3.81
	Smartphone (bidirectional)	9.59	2.24	67.90	3.41	14.10
Delay (sec)	Tethered cell phone	289.31	3.19	14534.32	14.00	1247.83
	Smartphone	52.22	0.616	388.87	9.66	72.25
	Smartphone (bidirectional)	203.02	1.98	645.56	173.44	174.19
Inter-message arrival time (sec)	Tethered cell phone	67.32	0.0	14511.02	1.68	591.84
	Smartphone	8.51	0.28	92.90	3.51	16.91
	Smartphone (bidirectional)	28.14	2.90	274.08	5.59	39.97

Table 1: Aggregate SMS transmission time, delay, and inter-message arrival time for tethered cell phone and smartphone test configurations.

from Soekris Engineering (net4801) and a Pentium 4 desktop PC. Both computers ran Ubuntu Linux and were each equipped with a recycled Nokia 3390 cell phone attached over USB. Sending and receiving SMS messages was performed using an application called Gammu[4]. There are several open source packages for interacting with cell phones. Gammu was chosen because of its active support for Nokia's proprietary FBUS[5] protocol and the Hayes AT instruction set. Both computers were accessible over a LAN, which was used as a test control channel. The LAN connection also allowed the computers to synchronize their clocks to the university's NTP server. Clock synchronization provided millisecond accuracy when tracking sent and received messages. Finally, both cell phones were on the same cellular network (Rogers), contained voice-only SIM cards, and were located within several meters of each other. Figure 1 illustrates this testbed setup and provides a high-level overview of the test procedure.

Our primary evaluation of SMS consists of sending ten messages per hour for one week from the Soekris (*sender*) to the desktop PC (*receiver*). We expect that ten messages (1400 bytes) represent a typical workload for the SMS-NIC. In each test, the sender first connects to the receiver to invoke a background process that periodically checks for newly received SMS messages every 50 ms and records their arrival time. Once the environment is correctly configured, the sender begins sending SMS messages to the receiver by making a synchronous call to the Gammu application. Gammu sends SMS messages over the serial connection with the Nokia cell phone using the FBUS protocol. Messages are placed in the cell phone's SMS "outbox" and transmitted by the phone as if they had been created by a user.

We measure transmission time by sampling the timestamp before and after the call to Gammu. Our measurement of transmission time therefore consists of both the wireless transmission time and the time Gammu spends communicating with the cell phone. Each SMS message sent consists of an integer representing the current hourly test and an index for the current message.

On the receiver side, SMS messages are received using the SMS daemon service, *smsd*, packaged with Gammu. smsd communicates with the cell phone through Gammu. Unfortunately, this requires that smsd polls the cell phone's SMS "inbox" for new messages. We configured smsd to poll the card in one second intervals. By polling we introduce, on average, an additional 0.5 seconds of delay. This was the most aggressive polling interval possible. Messages retrieved from the phone are each written to a file on the receiver and their arrival time is recorded by the background scanning process.

When the sender has finished sending, it connects to the receiver and waits for all messages to be received. A failure to deliver all of the messages is logged. Periodic anomalies, such as excessively long delay, were then verified manually.

Unfortunately, the cell phones did not support programmatic querying of the signal strength or the cell tower that they were connected to; both variables could have an effect on our study. The signal strength was observed manually by periodically checking the indicator bars on the phones' displays. The cellular signal strength on the cell phones was approximately 60% throughout the tests.

In several tests the PC's recycled cell phone was substituted for a Sony Ericsson GC82 EDGE PCMCIA card. Communication with the card was erratic due to poor Linux driver implementation. We omit these results from our study; however, the use of unreliable hardware illustrates an important variable in our study, NIC dependency, which we revisit in Section 2.2.2.

2.1.2 Smartphone to smartphone

A second series of experiments was designed to assess the effect of signal strength, cell tower changes, and provide a tighter relationship between the test driver and sending and receiving messages. Recall that when using Gammu, a message is sent by first placing the message in the SMS outbox on the phone. The phone then transmits the message while Gammu polls the outbox to check if the message has been sent. In contrast, on smartphones, a messages is sent using a blocking call into the OS that passes the message directly to the GSM stack for transmission. Similarly, receiving messages consists of a blocking call into the OS. When a new message arrives, it is passed to the application immediately; thus eliminating the polling delay.

The smartphone test suite consists of two Java applications, a sender and receiver, each running on a BlackBerry 8820 smartphone. The sender consists of two threads: one that sends messages in a closed loop and a second that receives any messages echoed back to it. At startup, the sender application prompts the user for the number of messages to send. In this round of tests, we send bursts of 10, 20, 40, and 80 messages. The receiver consists of one thread that operates in closed loop and blocks on the receipt of a message. The receiver may be invoked in an "echo" mode, which will

[4]http://www.gammu.org
[5]http://www.embedtronics.com/nokia/fbus.html

	Configuration	Mean	Minimum	Maximum	Median	Std. dev.
Cell tower duration (sec)	Smartphone	21.72	0.52	178.96	9.60	26.61
	Smartphone (bidirectional)	33.91	0.56	776.31	12.31	71.55

Table 2: Time associated with a GSM cell tower.

cause the application to echo messages back to the sender before blocking to receive another message. Placing the receiver in echo mode allows us to evaluate the behaviour of bidirectional SMS traffic. Both applications record the current timestamp (in milliseconds) when sending and receiving messages to local files on the device. Both devices used the same voice-only SIM cards used in the previous experiment. Their clocks were synchronized to the GSM network time. On both devices we record the signal level of the cellular interface (RSSI) and the GSM cell tower ID that the device is currently connected to.

2.2 Analysis

Over a period of seven days, we successfully transmitted a total 1644 messages across the tethered cell phone testbed, and 745 messages between the two smartphones. Our transmission success rate was 98.5% with 15 messages rejected by the network, and 21 messages that failed to send due to software failure on the sender side. Contrary to our expectation, we found that the day and time had no effect on service. This would imply that our cellular network is provisioned to handle bursty workloads. The following sections detail our characterization of the transmission and loss rate, delay of SMS. Aggregate results for both the cell phone and smartphone tests are summarized in Table 1.

2.2.1 Transmission rate

The transmission rate was the most consistent variable in our study. Unidirectional transfer yields similar results across both the cell phone and smartphone tests. The difference is attributed to the Gammu delays in storing the message to the cell phone's outbox and polling the phone to check if the message has been sent. Under bidirectional conditions, we see a significant increase in transmission time. We attribute this increase purely to medium contention; both the device and network compete to transmit their message, which causes the ALOHA protocol underlying GSM's control channel to introduce random delays.

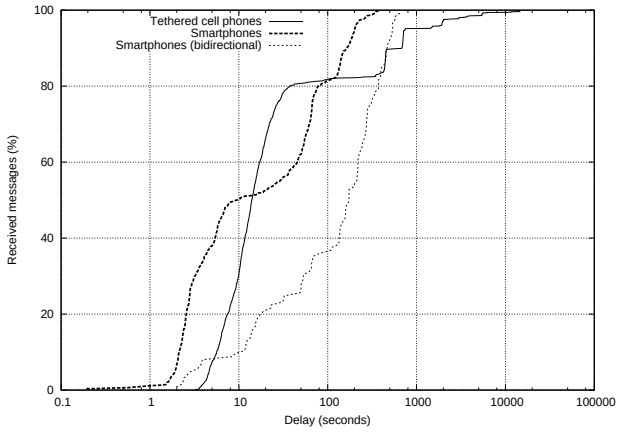

Figure 2: Cumulative distribution function for SMS delay.

2.2.2 Delay

SMS delay was a highly variable quantity in our study; a variable that is directly dependent on the hardware used. We measured delay as the difference between the receiver receiving the message and the sender returning from sending call. Average delay over all successfully received messages on the cell phone test was 289.31 seconds. This value is inflated by a series of messages that took over four hours to deliver. Failure to deliver message was primarily due to communication failures with the phone. Concurrent polling of the phone's SMS inbox for newly receiving message and deleting copies of retrieved messages, while the phone was receiving messages from the network, frequently caused the underlying FBUS protocol to enter an inconsistent state. In many cases Gammu was unable to retrieve the messages even though they were already present on the phone. Moreover, substituting a cell phone for the EDGE card increased average delay by nearly a factor of two; reducing the polling period to five seconds had very little effect. Although these increases in delay are not due to the cellular network, it is important to note the effect that hardware can have. From the perspective of an application using SMS, unreliable hardware is indistinguishable from poor network services.

In the absence of software failures, cell phones performed nearly as well as the unidirectional smartphone. Minimum and median delay is higher for cell phones due to polling delays on both the sender (to verify that the message has left the phone's outbox) and receiver. Burst size also has no effect on delay, which is contrary to our hypothesis that the SMS network is leaky bucket regulated.

Bidirectional traffic between the smartphones had a significant effect on delay. Average delay increased by a factor of four from 52.22 to 203.02 seconds. These results are illustrated in Figure 2. This increase is attributed to medium contention that leads to random ALOHA delays; signal strength in both the unidirectional and bidirectional tests was relatively stable. Occasional drops in signal strength frequently resulted in a cell tower change. This is expected behaviour as defined in the GSM standard [7]. In some cases changing cell towers translated into both increased transmission time and delay; in others, the tower change had no effect. Interestingly, bidirectional communication did increase the duration that each phone spent on a single cell tower (see Table 2). This increase is likely the result of ALOHA delays on the control channel, which prevent the tower from signaling a handover to another tower; however, without low-level access to the smartphone's GSM stack, it is impossible to say for sure.

2.2.3 Loss

The average SMS loss rate across all tests was 3.89%. During the cell phone test, we observed peaks in losses during business hours on two days of the study. However, re-

running the experiment on a subsequent week found no evidence that SMS traffic was consistently increased on these days. We found no correlation between the loss rate and transmission order or burst size. The loss rate is also independent of the device used.

We observed that SMS messages are duplicated at a rate of 3.1% and 0.8% on the cell phone and smartphone testbeds respectively. Duplicate messages are often a side effect of poor communication between the phone and the service provider. In the smartphone case, duplicate messages were directly correlated with a change in cell tower. It is likely that the message was received by the device, but communication was severed before the tower could confirm delivery; thus triggering a resend. On the cell phone test, manual examination of the smsd logs found that approximately 2% of duplicate messages were caused by a bug/feature of Gammu that reset the connection with the phone when the communication protocol entered an inconsistent state.

2.2.4 *Message reordering*

Surprisingly, the rate of message reordering is highly dependent on the device used. Cell phones have a reordering rate of 2.53%. The smartphone tests reorder 31.72% and 41.95% of messages for unidirectional and bidirectional traffic! This bizarre result was verified manually by observing communication with the cellular network, (visible as arrow icons on the BlackBerry's screen), and displaying the message contents to the screen as they arrive.

3. DESIGN

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

- **NIC dependency**: The choice of hardware has an impact on the behaviour of SMS.
- **Bidirectional traffic**: Concurrent transmission by both the mobile and network under bursty conditions increases transmission time, delay, and message reordering.
- **Message reordering**: Bursts of messages may experience significant reordering.
- **Variable inter-message arrival times**: Variable delay coupled with message reordering causes messages transmitted at a constant rate to arrive intermittently.
- **Losses**: Messages are rarely lost and occasionally duplicated. Messages that do not arrive in the expected order are more likely to be reordered than lost.
- **Burst size**: Burst size does not have an effect on the behaviour of SMS.
- **Messages remain intact**: Although it is not part of our characterization, it is worth mentioning that SMS guarantees message integrity.

Our design is further motivated by the requirement to minimize monetary cost, maximize throughput, and recover from lost or re-ordered SMS messages. Although we predict that SMS messages will continue to decrease in cost, the service is not yet free; we must minimize the number of messages exchanged. In contrast, throughput is maximized (34.77 bytes/second) by sending messages continuously back to back. Our flow and error control strategies are subject to these constraints. Moreover, they must operate over the shared SMS channel with the previously described behaviour.

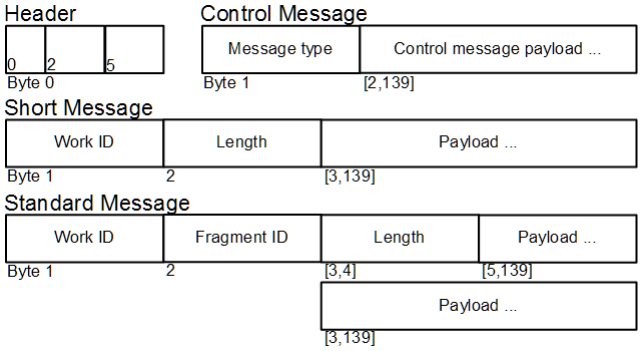

Figure 3: The format of SMS messages exchanged between SMS-NICs.

3.1 Protocol

The communication protocol between two SMS-NICs is designed to support a wide variety of applications and user defined settings while maximizing the payload of a single message. We outline the message structure and flow and error control methods.

3.1.1 *Message format*

Each transmitted SMS message contains a small fixed size header and message payload. Messages exchanged between two SMS-NICs consist of *short messages*, *standard messages*, and *control messages*. The format of these messages is illustrated in Figure 3. Short messages consist of data that is small enough to fit into a single SMS message. We expect that short messages will be common, so we have included this special case to reduce the header size by two bytes. Standard messages are designed to provide fragmentation and reassembly of larger data. These messages consist of a 5 byte header, and a 135 byte payload. We have chosen to limit the maximum data size supported by the SMS-NIC to 32 KB. We believe that 32 KB (approximately 243 SMS messages) is a practical and reasonable upper bound. 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.

The first byte consists of a two bit protocol version, a three bit message type, and three flag bits. The first bit signals that the payload is compressed. The last two bits are currently unused; however, we foresee using one bit to signal that the payload is encrypted.

3.1.2 *Flow control and error control*

The SMS-NIC provides flow control and error control through a simplified version of the NETBLT protocol [3]. In NETBLT, the sender communicates with the receiver by exchanging a series of large data aggregates called buffers. When transferring a buffer, the sender fragments it into a set of packets. The packets are then sent across the network to the receiver where they are reassembled into the original

buffer. When the last packet in the buffer arrives, the receiver checks to see if all the packets in the buffer have been correctly received. The receiver then sends an acknowledgement (ack) to the sender containing a bitmap that indicates which packets have been received or lost.

NETBLT has several distinct advantages that make it suitable for use in an SMS network. Bidirectional traffic is minimized through the use of a cumulative ack when all of the packets have been received. The cumulative bitmap ack also tolerates message reordering, random losses, and variable inter-arrival times. Because burst size has no effect on SMS, NETBLT may transmit all packets in one continuous burst. This property significantly reduces the complexity of the underlying implementation. Finally, the low loss rate of SMS ensures that the quantity of ack messages will be low.

The SMS-NIC's protocol differs from NETBLT by using only one buffer that is bound to 32 KB in size; thus removing the need to coordinate block transfer and to reassemble blocks of data at the receiver. Like NETBLT, buffers are fragmented into SMS message sized *chunks* for transmission to the receiver. The sender initiates communication by sending a single message to the receiver that contains the size of the overall data, an integer representing the current transaction, and the first chunk of data to exchange. Upon receiving the initial message, the receiver may accept or reject the session by sending an ack back to the sender. If either message is lost, the sender will retransmit the original message several times before failing.

After receiving the initial ack accepting the transaction, the sender begins transferring each remaining chunk of data. Each chunk is transmitted as a full SMS message to the receiver's phone number. When sending each message, the sender adapts to transmission failures by resending messages; however, it is not aware of losses in the network. The sender continues to transmit chunks until none remain.

Each message sent by the sender causes an ack timer to be set. When the timer expires, the sender begins retransmitting non-acked chunks to the receiver. The sender's ack timer is set to a static value of 200 seconds (approximately four times the mean delay of the smartphone experiment). In practice, we found that a relaxed timer at the sender prevented many redundant retransmissions of messages; retransmissions that almost always arrive after the original message.

On the receiving side, messages are received and their associated data chunks are placed into a pre-allocated buffer. While receiving messages, the receiver exponentially averages the inter-arrival time of the messages. Each received message also causes a timer to be set to three times the average inter-arrival time. This technique allows the receiver to adapt to fluctuations in delay that cause message inter-arrival times to grow. If the timer expires, the receiver transmits an ack indicating the chunks that have been received. The timer is then reset to twice its previous value. If no messages arrive after three ack retransmissions, the receiver discards the current buffer.

Like NETBLT, when the last chunk has been placed in the buffer, the receiver sends a complete ack to sender. Because this final ack may be lost, the receiver maintains record of the completed data and retransmits completed acks as needed.

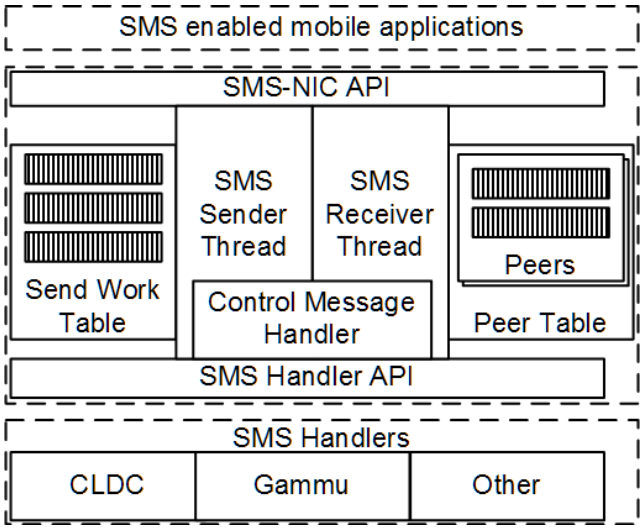

Figure 4: Architecture of the SMS-NIC including example SMS Handlers.

3.2 Architecture

As an open source project, the architecture of the SMS-NIC was influenced by the need for platform portability and code modularity. To run on a variety of resource constrained devices, the architecture minimizes memory allocation and copying by reusing objects, timers, and threads as often as possible. To provide clean integration with existing mobile systems and applications, the SMS-NIC provides a series of APIs that abstract the surrounding environment from the internal operation of the NIC.

The architecture is illustrated in Figure 4. It consists of the following components:

- **SMS-NIC API**: The SMS-NIC API is the application interface into the SMS-NIC. This API allows *host applications* to send and receive data, register a view, configure, and start and stop the SMS-NIC. This interface provides 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. This interface 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 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 internal state of the SMS-NIC. 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 (optional) SMS Compression interface.

- **SMS Handler API**: Each mobile platform has a unique method for sending and receiving SMS messages. The SMS Handler API is designed to abstract these differences from the internal operation of the SMS-NIC.

- **SMS Send Work**: This component contains the data sent from host application data and is responsible for providing SMS message sized chunks of data to the SMS Sender for transmission. Chunks are retrieved from the work object by a linear sequence number starting from the beginning of the given data. This component also maintains the state of a transaction, which includes the messages that have been successfully received, the last time ack was received or message was sent. The reciprocal of this component is *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 configured to do so, wrap the data in an SMS Send Work object, and transmit chunks 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.

  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 sender is also responsible for recovering from transmission failures. In many cases, this involves simply pushing the SMS message back onto the sending queue.

  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, expire data that was unsuccessfully sent, and retransmit data or acks when an ack or receive timeout occurs.

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

- **SMS Peer**: The SMS Peer is a simple component that stores a set of partial Receive Work objects from a specific source address (phone number). This component also temporarily stores the work identifier of completed work. This prevents new Receive Work objects from being allocated when highly delayed messages arrive after the work has completed. When an SMS Peer contains no work objects or completed work identifiers, 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. When a message is received it takes one of 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 message completes a work object, then the work object is removed form its peer container and placed in the 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 larger than a user specified limit or exceed 32 KB are discarded.

- **Control Message Handler**: The Control Message Handler implements the communication protocol between two or more SMS-NICs. 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 briefly overview the decisions made when implementing the SMS-NIC and its integration with an existing mobile system.

4. IMPLEMENTATION AND EVALUATION

We considered both C++ and Java Micro Edition (J2ME) when implementing the SMS-NIC. C++ is an efficient language for implementing low-level functionality and is supported on a range of mobile and embedded platforms; however, we chose J2ME because it is supported on both Linux and many mobile platforms. Our implementation only uses libraries that are present in both Java's CLDC and Java Standard Edition. The SMS-NIC therefore runs on most systems, including most smartphones and most cell phones. Platform specific functionality such as logging, compressing data, visual feedback, and sending and receiving SMS messages are abstracted from the core of the SMS-NIC though a series of abstract classes. This abstraction allows the SMS-NIC to be ported to new Java platforms by simply writing 'plugins' specific to the new platform.

The current release of the SMS-NIC provides plugins for both BlackBerry (CLDC) and Linux. This work is open source under the Apache License and can be downloaded from: *http://blizzard.cs.uwaterloo.ca/eaoliver/sms.html* .

We evaluate our implementation of the SMS-NIC by integrating it with an existing mobile system: the Opportunistic Connection Management Protocol (OCMP) [10]. OCMP is a disconnection-tolerant, policy-driven session layer that is used extensively in the KioskNet Project [4] to provide DTN-like service to rural kiosks in developing regions. The OCMP

	GPS position (1 msg)	**2 KB RSA key** (16 msgs)	**4 KB BLOB** (31 msgs)
SMS-NIC (sec)	37.32	97.23	212.11
Calculated average (sec)	39.18	103.64	193.47

Table 3: Performance of the SMS-NIC compared to expected values derived from our characterization.

client fragments data on behalf of disconnection-tolerant applications and transmits the fragments over multiple NICs opportunistically. Fragmented data is then reassembled at a central server, or *proxy*, on the Internet. The proxy then forwarded the reassembled data to legacy servers on the Internet. Our integration augments OCMP's existing high-delay, high-capacity "DTN-NIC" by providing a low-delay, low-capacity control channel. Details of our integration with OCMP can be found in [6].

We performed a series of trials to evaluate the performance of the SMS-NIC while integrated with OCMP. We configured OCMP to transmit all data over the SMS-NIC, and scheduled files for upload on the *client*. OCMP read the files and transferred them through the SMS-NIC, over the cellular network to a receiving SMS-NIC on the proxy. This trial was performed on the same Linux testbed as described in 2.1.1. Table 3 illustrates the average performance of several sample workloads. This table compares the SMS-NIC with a calculated scenario with an average transmission rate, median delay, and an average loss rate. The overhead of OCMP was negligible. The SMS-NIC performed nearly as well as the expected average derived from our characterization.

5. CONCLUSION

In this paper we have characterized the behaviour of SMS when exchanging bursts of messages between tethered cell phones and smartphones. We have used this characterization to shape the design and implementation of an SMS-based data channel for mobile systems operating in challenged network environments. Through integration with an existing mobile system, we have shown that the SMS-NIC has little operational overhead and is robust to the conditions of the cellular network. The compact design and lightweight implementation of the SMS-NIC made integration with OCMP effortless. As the cost of basic GSM service continues to fall throughout the world, we strongly believe that many mobile systems could benefit from the use of an SMS control channel.

6. ACKNOWLEDGEMENTS

My thanks to Hossein Falaki, Nilam Kaushik, and my supervisor Prof. S. Keshav for their feedback.

This research was supported by grants from the National Science and Engineering Council of Canada, the Canada Research Chair Program, Cisco Research, and Nokia Research.

7. REFERENCES

[1] Umass dieselnet, 2007.

[2] 3rd Generation Partnership Project. 3GPP Specification, 2007.

[3] D. D. Clark, M. L. Lambert, and L. Zhang. Netblt: a high throughput transport protocol. *SIGCOMM Computer Communication Review*, 17(5):353–359, 1987.

[4] Shimin Guo, Mohammad Hossein Falaki, Earl A. Oliver, Sumair Ur Rahman, Aaditeshwar Seth, Matei A. Zaharia, Usman Ismail, and Srinivasan Keshav. Design and implementation of the kiosknet system. In *Proceedings of the International Conference on Information and Communication Technologies and Development (ICTD 2007)*, pages 300–309, 2007.

[5] 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.

[6] Earl Oliver. Design and implementation of an sms control channel for mobile systems. Technical Report CS-2007-42, University of Waterloo, February 2008.

[7] 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.

[8] A. Pentland, R. Fletcher, and A. Hasson. Daknet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, 2004.

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

[10] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya. Manuscript: 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`.

[11] 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.