Buzzenger: Asynchronous (Time-Sliced) Missed Call Duration Messaging

Brian G. Omwenga
Nokia Research - Africa
Pauline W. Githinji
Nokia Research - Africa

Abstract

We describe the Buzzenger project, a mobile phone based missed-call-duration interpretation project that makes use of a time-sliced messaging communication protocol useful for transmitting small data packets that can form complete messages at no cost. This project takes the basic beeping habits (missed calling/flashing) and extends them into a formalized messaging protocol. This can be particularly useful for sending small structured messages at no cost where useful applications can be developed for emergency situations, feedback, etc. where the user doesn't have sufficient funds (phone credit) to place a call or send an SMS.

1. INTRODUCTION

The cost of mobile communication in developing countries is at times much higher than other forms of digital communication. This cost is relatively high, especially when considering the transmission of small data packets. Most mobile communications—notably, data and multimedia—are charged at a constant rate regardless of the payload's size. If therefore you were to consider sending an SMS, which might contain just a single word or sentence, the user will have to pay a relatively higher cost per character if they did not utilize the entire message space provided. In most countries, like here in Kenya, this fact has given rise to the popular concept of flashing or beeping (that is, messaging through missed calls), because service providers don't charge callers for placing a missed call.

The findings of a study conducted by the Nokia Research Center – Africa, entitled Young Africa [1] outlined challenges of African mobile users such as coverage, consistency, speed and cost which formed the premise in the design of this project. Based on this study, our team set out to develop a practical, free, mobile phone based implementation of the asynchronous missed-call messaging duration interpretation.

Our project focused on developing a simple software application that could relay messages accurately from one node to another by purely sending missed calls of varying durations. More than a simple proof of concept, the project also offered a good foundation to consideration of duration encoded messaging for other small data messaging systems and structured messaging that may be useful in further ICT4D settings such as m-health related applications, disaster recovery, simply surveys, etc. The protocol can be used in numerous applications that require simple structured feedback and further at no cost to the user.

2. RELATED WORK

Encoding of signal pulses is a common basis for many messaging mechanisms. The Morse code communication was the first electric telegraphy system in the world and was invented by Samuel Finley Breese Morse in the 1836 [2]. This system sent pulses of electric current along wires which controlled an electromagnet that was located at the receiving end of the telegraph system. This has evolved to the present day bits and bytes message transmission mechanism.

Missed call based projects such as *PengYo* application [3] by Mark Bilandzic et al have utilized beeping primarily for notification purposes and instant feedback. Beeping (missed calling) is accepted as a popular practice and has grown in popularity especially in the developing world. Many of the applications of the beeping concept have primarily been for notification purposes, as well as call-back requests. Nevertheless, few projects have touched on interpreting beep durations as a comprehensive messaging platform.

3. PROTOCOL OVERVIEW

The Asynchronous (time-sliced) Missed Call Duration Messaging protocol had the primary objective of encoding messages into distinct missed call durations that would then be relayed and interpreted by a recipient node.

We conceptually divided the development of the protocol into three modules; the presentation module that handled the user interface; the parsing module that handled the encoding and decoding of messages; and the transport module that managed the sessions and the missed calling. A summary collaboration diagram of how these modules interact is represented in figure 1.

3.1. Presentation Module

Based on any given messaging use case, the presentation module handled the desired user interface implementation. Such messages could take the form of images, templates, words, audio, video, graphics, etc. that would be stored within the applications database.

As the module handling interaction with a user the presentation module managed all user triggered activities which include:

- Sender functions: message composition, sending a message
- Receiver functions: message alert, viewing a message

Message composition involves selecting the recipient contact, which can be selected from the phone's contacts or simply keyed in as a phone number. The design of the message composition interface is dependent on the particular use case as well as the designer's preference. For example, in the case of our sample implementation project, Buzzenger, we explored pictorial messages, message templates and the classical short messaging compose interface. Once a message is composed, a simple send button fulfilled the function of initiating transmission of the message.

Depending on the architecture of choice (as discussed in section 4 below), once a complete message has been relayed, the application alerts the recipient node using several notification mechanisms. These could be typical message tones such as those used for incoming SMS messages, or triggering the execution of any other given function.

3.2. Parser Module

The parser module was designed to operate in two modes; when sending a message and when receiving a message. When sending a message, the parser module encodes a message it receives from the presentation module by fetching the assigned duration codes for specific messages. The composed message will then be represented as a stream of missed call durations that are then relayed onto the transport module for transmission.

When receiving a message, the parser module obtains a stream of durations from the transport module and proceeds to decode them by mapping them onto the predefined provided message database. This are then relayed to the presentation module.

The parser module together with the application database represents an area of flexibility on the design and desired use of the protocol. The contents of the messages database for use by the application can include varied messages such as pictorial, template, text, audio, etc. that a developer may wish to use.

3.3. Transport Module

The transport module was designed to handle all missed call pulses. The module would place a series of missed call durations from the stream received from the parser module. Other functions executed by the transport module included:

- Application wake up and sleep (control bits)
- Session management (handling multiple message sessions from multiple senders)

The applications on both devices define a series of control bits for the purpose of handshaking. These bits are used to define wakeup, wakeup acknowledgement, sleep and sleep acknowledgement.

A sending application does not begin relaying a message unless it receives a wakeup acknowledgement from the receiving node device. This solves the problem of attempting to send a series of missed calls to a device that does not run the application. If the sending application sends a wakeup signal to a specified receiving device and does not receive a wake up acknowledgement after a reasonable time, it reports this through an error message to the user notifying them that the user may not be using such application. It may also occur that the transport module receives a busy signal from the receiver, after which it shall attempt to make the call again after a given timeout.

Once the transport module has been woken up and has started receiving missed calls, from a given number, it would intercept any more calls coming from that number for purposes of the missed call interpretation application. This may involve avoiding the logging of the calls on the phones call log, or initiating alerts for in

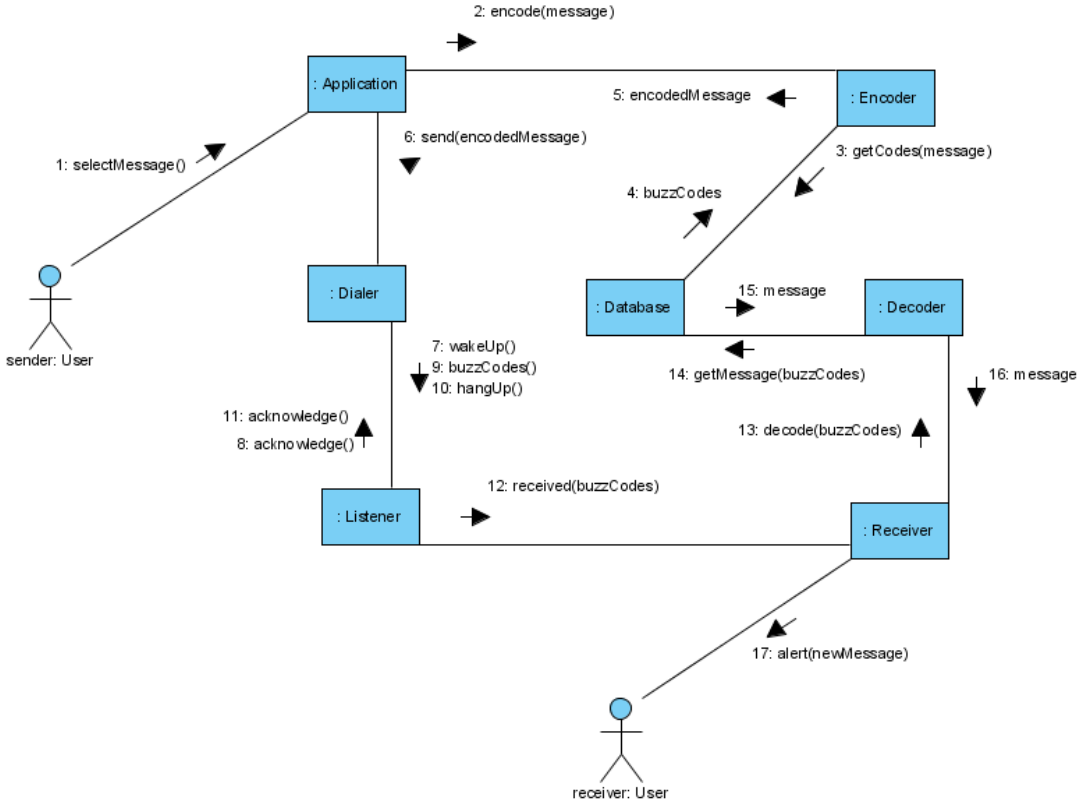

Figure 1: Collaboration diagram

coming calls from that number. The recipient's phone therefore does not ring uncontrollably over the period the message is being relayed. Session management is handled simply by associating every missed call with a given sending phone number. The application can therefore receive multiple messages concurrently.

4. DESIGN CONSIDERATIONS

Several design decisions regarding the efficiency and effectiveness of using the protocol also need to be considered with respect to any given use case. Some of these design considerations include:

4.1. Deployment Architectures

Developers can choose between many varied deployment designs on how to deploy the protocol to best serve their given use case. The options vary between a centralized, distributed or hybrid architecture.

In centralized deployments, a server can be used to receive all missed calls from a given phone running the application. The server then decodes the messages and performs a given function defined by the developer once such a message is received. This architecture is useful for emergency, disaster reporting and data collection applications.

Distributed architectures involve deploying the application to both sending and receiving devices (mobile phones). This deployment architecture can be used for simple messaging applications such as the one our team developed. Hybrid architectures would be useful for such functions as store and forward or group messaging applications.

Further design considerations touched on economic and operational feasibility. Since the application would perform free messaging for the users, and made use of existing resources (mobile network infrastructure), there seemed to be few to no challenges affecting the economic feasibility of the project. The operational considerations are covered below.

4.2. Operational Considerations

Operational decisions that affected design considerations of the protocol and further applications were determined by the following factors:

- The level of accuracy to which we could measure a missed call; and

- The maximum duration in which a missed call would last.

While the maximum length defined the bounds of the universal set of missed calls we could operate within, the level of accuracy defined the level of granularity and therefore the number of distinct unique missed calls we could allocate to different messages. The GSM standard [4] defined the different states a phone call is in.

In analyzing the maximum duration with which a missed call would last, we simply placed a call within the available different service providers locally, and realized that the service providers will leave a call at the *alerting* state for an average of 40 (forty) seconds after which they would drop the call. This was tested across three different networks (Safaricom, Airtel and Telkom Kenya) and was found to be the same. This discovery set a benchmark of the maximum number of seconds a missed call could have and set the boundaries for the universal set of missed calls.

We then proceeded to analyze the level of accuracy with which we could measure a missed call between two devices and discovered that this was affected to a great extent by latency. In a networked communication system, there exists a challenge in obtaining signals in a timely fashion due to the latencies caused by the communication medium as well the networked device.

This latency occurs due to a combination of lags experienced when establishing a connection (start signal), relaying the message (transmit signal) and when closing the connection (end signal). After a signal has been transmitted by a sending node, the lag occurs in the shared communication medium as well as on the receiving node, due to properties and processes unique to each; such as, resource availability, processing capabilities and extra processing functions (e.g, displaying of caller screens, writing to call logs, etc).

To extend the accuracy of measuring these durations, it was imperative to establish these lags as a lack of accuracy would only result in miscalculated durations and inevitably distorted messages as well as a heavily reduced list of available durations that could be used for encoding. After several tests it was established that this lag is consistent between similar devices. Nevertheless, it was difficult to isolate which part of this lag was attributable to device properties and which was from the mobile network. We therefore devised a calibration algorithm to establish this consistent lag as outlined below:

- *Step 1:* Set-up of a universal set of pre-defined control pulses to be communicated amongst all devices [x_1, x_2, x_3]. This set of universal control pulses are known to all devices or acquired during initialization of a new app installation onto a device

- *Step 2:* When sending a message, the sending device sends out the pre-defined control pulses to the recipient device

- *Step 3:* The recipient device "listens" and records time slices for the message received

- *Step 4:* The recipient device compares its recorded time slices against the pre-defined control pulses and computes its lag

- *Step 5:* The pre-defined time slices are adjusted to factor in the updated consistent lag. This involves either decreasing or increasing the margin for error allocated between each time slice

We were able to observe a 0% error rate (in delivery) by widening the margin of error to 1.5s which meant that distinct buzz codes would be 3s apart (i.e. by allowing an error margin of 1.5s to the previous and next codes for each code). The higher the margin of error the longer the messages would take to send. This margin could probably be reduced, even to 0.3s, though we did not have adequate resources and time to test it sufficiently.

5. IMPEMENTATION

Our internal design and prototyping of Buzzenger involved several iterations. The primary design question was how to encode and decode the messages. Our initial message designs only made use of a single unique code to represent a single message. With accuracies of up to half of a second (+/- 1.5) and a sample space of 40 seconds, we could only represent a limited number of messages. Although this would be sufficient for emergency messaging, we explored more flexibility options that we could use within the encoding process without necessarily affecting the usability of the service. We settled on a (2X2) matrix representation of message encoding that exponentially increased the number of possible messages we could send, affecting the length of sending the message by a factor. Each unique message was therefore encoded as an (x,y) duration coordinate that was used to map onto the message database as shown in figure 2 below:

		Call 1 duration			
		1	2	3	...x
Call 2 duration	1	message 1,1	message 2,1	message 3,1	message x,1
	2	message 1,2	message 2,2	message 3,2	message x,2
	3	message 1,3	message 2,3	message 3,3	message x,3
	...y	message 1,y	message 2,y	message 3,y	...message x,y

Figure 2: Interpretation matrix

It was discovered that applying an x-dimension interpretation matrix that was larger than a 2X2 matrix would affect the usability and user experience of the application as user would have to wait for minutes to have the entire message relayed from one node to the other.

We then considered the universal control bits (wake-up, start, stop, etc) that the application would use. It was important to establish distinct duration control signals that would be unique from any other missed call that a users' device could get simply because the application used the common messaging channel and mode that devices experience on a fairly regular basis – phone calls.

We established that a 0.5s missed call pulse could be recorded by a device. The likelihood of a human user sending a 0.5s missed call was fairly slim, and therefore a device initiated 0.5s pulse was established as the wake-up control bit for the application. For ease of programming as well as for conserving message space, the 0.5 second duration was also selected as the start and stop bit for sending messages depending on the current application state, whether sending or receiving. In the off chance that a non-application initiated missed call lasts the duration of the chosen control bits, upon waking up the application would acknowledge this by sending a missed call to that callers number and if no message stream in the form of further missed calls come from such number, the application would simply time out and go to sleep.

With all this established, we set about developing the different features of the Buzzenger project. As earlier discussed, our presentation module covered a messaging interface which incorporated pictures, templates and text messaging. The module also handled an inbox and outbox functionality together with the necessary user alerts.

Figure 3 shows the different kinds of interfaces we implemented in our presentation module; pictorial, template or textual after selecting a recipient for the message. These messages are held within the applications' database that is common amongst the different devices. Pictorial messages are simple icons such as smileys or emoticons. For the purposes of the trial among university students, we included a number of other seemingly common pictorial messages that we felt were familiar with the focus group such as the "love hut" for "I love you" and the "traffic lights" for "stuck in traffic". The second type of messages we considered were the template messages. Templates messages are actually a common functionality among most mobile devices, nevertheless, these are charged as actual text messages under normal circumstances. The user could select any of the included messages like "I'm in class", "In the hospital feeling sick", etc. to send to a recipient. Finally, we included the classical text messaging possibility, with only limited vocabulary that could be held by our message database. Users could compose simple messages in the very same way that they composed text messages, with the primary implementation difference that each word was assigned a unique missed call duration code.

Figure 3: Buzzenger user interface (pictorial, template and text composition)

Once a user selects or composes a message to send and hits the send button, messages are then encoded by the application into a series of missed calls of pre-defined duration and sent over the mobile network to the receiving device.

The Buzzenger application was developed on the Qt platform. Qt is a multiplatform C++ GUI toolkit provided by Nokia. It provides application developers with all the functionality needed to build applications with

state-of-the-art graphical user interfaces. Qt is fully object-oriented, easily extensible, and allows true component programming. The target device was the Nokia Symbian series 60 devices with feature pack 1.

The choice of platform was driven by the design considerations that required access to the telephony stack as well as the ability of the mobile device to support multi-tasking.

- Access to the telephony stack was necessary since monitoring the call states was necessary for the management of event triggers to enable the application execute functionalities for accurately measuring durations.

- Multi-tasking was necessary since the application runs in an active wait state on the phone as a background application, listening to all missed calls to pick out any wake-up control bits.

After internal conceptualization, the application was jointly developed in collaboration with University of Nairobi computer science students and faculty through iterative prototyping. Further analysis is currently underway to determine possibility of porting the concept to other platforms and devices.

6. EVALUATION

In order to learn more about the usability and practicality of missed call messaging, Buzzenger was deployed to a controlled focus group of approximately 17 university students and their activities and perceptions logged for analysis. Some findings include:

Two out of every three students preferred the use of the pictorial (emoticons) form of messaging. We were able to observe a 0% error rate (in delivery) by widening the margin for error to 1.5s. Some students took issue with the length of time it took to send messages, as the application would have to send multiple missed calls. The time taken to send a message only depends on the duration code assigned to it, be it an icon message, template or the composed message (i.e. an icon assigned a 1s code would take the same time as a template message with a 1s buzz code). Though in the case of composed messages, the time taken would depend on the number of words that the user has composed (a total of their individual codes). All the respondents found it easy to use the application, probably because the presentation layer modeled typical messaging interfaces that they were familiar with. The primary challenge on the project was its operational feasibility. It was observed that such a solution would not be met with enthusiasm by mobile phone operators since it translates to more network traffic with no revenue inflow.

7. FUTURE WORK

We are currently looking into open sourcing the Missed Call Duration Interpretation protocol, specifically APIs for the transport and parsing layer to other developers who may wish to utilize it for sending messages.

We are also working on a more elaborate emergency use case as well as analyzing how to best deploy this service among network providers who may take issue with excessive missed calls. We believe that this protocol can have far reaching effects within ICT4D in the many different use cases it can be intelligently applied by various developers for the transmission of small structured data packets.

8. CONCLUSION

The Buzzenger project and concept probably has more far reaching applications within ICT4D especially regarding the transmission of very small structured data packets. Its application can extend into areas such as surveys, structured questionnaires, disaster reporting systems, etc. which require simple structured messages that can be represented as missed call time slices and transmitted at no cost.

9. REFERENCES

[1] Nokia Research Center Africa (2010), Young Africa – Development and Empowerment of Young People in Africa

[2] Morse code: *International Morse code recommendation* – ITU

[3] Mark Bilandzic et al, *A Mobile Application to Support Phatic Communication in the Hybrid Space,* ITNG 2009

[4] SMG, *Digital cellular telecommunications system (Phase 2+); Physical layer on the radio path; General description (GSM 05.01 version 5.4.0),* GSM Technical Specification