Emergency Messages in the Commercial Mobile Alert System

Paul Ngo, Duminda Wijesekera

Chapter 13

EMERGENCY MESSAGES IN THE COMMERCIAL MOBILE ALERT SYSTEM

Paul Ngo and Duminda Wijesekera

Abstract The U.S. Department of Homeland Security initiated the Commercial Mobile Alert System (CMAS) to ensure that emergency situations are effectively communicated to the general public. CMAS uses the existing commercial telecommunications infrastructure to broadcast emergency alert text messages to all mobile users in an area affected by an emergency. One of the limitations of CMAS is that the maximum message size is 90 characters of plaintext. This paper proposes an enhancement to CMAS that provides more detailed information within the 90-character text using an encoding technique. The viability of the enhancement is demonstrated using a prototype that generates and broadcasts CMAS emergency alerts to Android phones, on which an emergency response application intercepts, decodes and displays the alerts to users.

Keywords: Commercial Mobile Alert System, emergency response, alert messages

1. Introduction

Protecting assets against man-made and natural emergencies is a priority. However, due to the unexpected nature of emergencies, preparing for, responding to and recovering from an emergency are always challenging. Indeed, the emergency problem space is often overlooked until an emergency arises, often unexpectedly.

In the aftermath of the September 11, 2001 terrorist attacks, communications were identified as a major bottleneck for emergency and rescue operations. Telecommunications service providers experienced an extremely high overload of calls in and out of the stricken areas, which caused congestion at access and core networks, resulting in many calls being blocked or rejected [8]. However, mobile users were still able to send and receive text messages.

In 2006, the U.S. Federal Government established the Worker Adjustment and Retraining Notification (WARN) Act that supported research and development efforts related to the Common Mobile Alert System (CMAS) [3]. CMAS utilizes the existing commercial telecommunications infrastructure to broadcast emergency alerts and warnings to a specified geographic area. The current messaging protocol standard, however, is limited to 90 plaintext characters, which is not enough to communicate detailed information. This paper describes an enhanced encoding technique that enables the broadcasting of more detailed information while satisfying the 90-character constraint.

2. CMAS Limitations

In 2006, the U.S. Government initiated the Commercial Mobile Alert Service (CMAS) to broadcast emergency alert text messages to the public [5]. Unlike the short message service (SMS) point-to-point communications protocol, CMAS uses a dedicated broadcast control channel to send text message alerts, which can reach millions of wireless subscribers within minutes. Note that CMAS does not require subscriber registration; the service is available as long as a user is within range of a cellular access point. While CMAS is designed to communicate information to the general public during emergency situations, it inherits the following weaknesses from the cellular broadcast service:

- CMAS alert messages cannot broadcast to an area smaller than a cell site, which is defined in the Federal Information Processing Standard (FIPS) code [4]. The area of a cell site varies depending on the population density and can be too large for targeted broadcast alerts in a small-scale emergency (e.g., a burning building or an apartment gas leak).

- CMAS disseminates three types of alerts: (i) Presidential alerts; (ii) imminent threat alerts; and (iii) AMBER alerts [11]. CMAS is not designed to broadcast alerts for local emergencies.

- The CMAS specification [2] states that the Common Alerting Protocol (CAP) version 1.2 is to be used to communicate emergency alerts. However, CAP 1.2 was designed for department and agency communications across different levels of government (e.g., federal, state and local). Also, most of the information in a CAP 1.2 message is not relevant to emergency mobile broadcasting and the message structure does not meet the requirements associated with local emergencies.

- CMAS broadcast messages are limited to 90 characters of plaintext [2]. This size limitation restricts the ability to disseminate detailed and informative emergency messages.

The first three limitations are addressed by our ERApp emergency application for the Android mobile platform [7]. ERApp filters CMAS messages based on the GPS location and displays alerts only if a user is within the affected area. We also introduced the Emergency Alert System (ERAlert) to generate CMAS alerts specifically for local emergencies. Additionally, we suggested enhancements to the CAP 1.2 message structure by adding XML tags to enable relevant communications. This paper addresses the fourth limitation by employing an encoding scheme that enables detailed information to be sent in a 90-character message.

3. CMAS Enhancement

This section describes the CMAS enhancement for encoding and delivering detailed emergency information messages. The solution affords the flexibility to tailor messages specific to emergency situations.

3.1 ERApp Considerations

In 2003, OASIS sponsored the CAP initiative to provide messaging protocols that facilitate inter-agency emergency communications. CAP, however, is intended to facilitate communications between emergency systems and operators; it was not intended for one-way broadcast alerts to a population. For example, the sender ID field is not relevant to users who receive broadcast alert messages.

The GSM/UMTS cellular broadcast service technical specification does not address broadcast alert messages for an area smaller than a cell site [1, 2]. The ERApp solution, however, overcomes this challenge by enhancing the CAP 1.2 message structure [6] to include three additional tags: (i) affected area; (ii) spreadable; and (iii) location. Broadcast localization is achieved by intercepting and filtering a CMAS cellular broadcast service alert message based on the distance between the location of the emergency and the recipient.

To support the ERApp implementation, the CAP 1.2 message structure must be expanded to accommodate XML tags and values for the additional information [9, 10]. We propose an encoding scheme that enables more emergency data to be placed within the 90-character block. Upon receiving the CMAS broadcast alert message, ERApp decodes the enhanced XML message into its original format and displays the emergency information to the recipient.

3.2 CMAS Architectural Enhancement

ERApp requires a "codepage" to decode an emergency alert message. The codepage contains a list of emergency message formats and region-specific emergency tag repositories, which can be identified by the namespace of the unique XML schema. The unique uniform resource identifier (URI) or uniform resource name (URN) in the namespace is contained in the emergency XML alert message. Note that each emergency tag repository contains a location-specific list of emergency name and value pairs. The codepage can be downloaded automatically to a mobile device during handover (i.e., the process of transferring an ongoing call or data session from one cellular network to another). Enabling codepage download during handover requires a minor enhancement to the current architecture. To simplify our discussion, we consider the Global System for Mobile Communications (GSM). Other networks such as Code Division Mul-

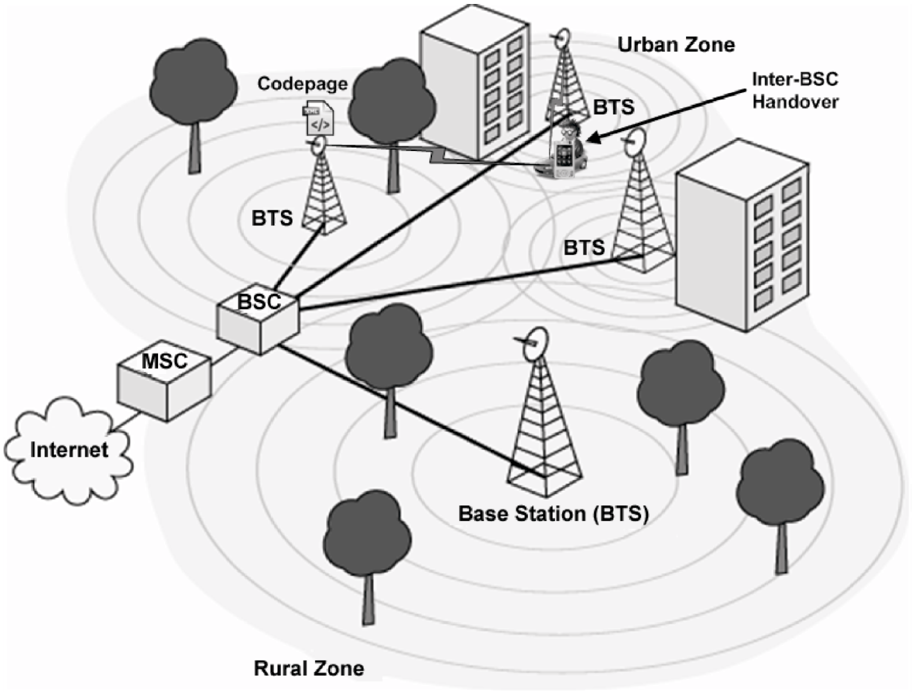


Figure 1. Handover enhancement.

tiple Access (CDMA), Universal Mobile Telecommunications System (UMTS) and Long Term Evolution (LTE) have similar handover procedures.

GSM has four forms of handover: (i) intra-BTS handover; (ii) inter-BTS intra-BSC handover; (iii) inter-BSC handover; and (iv) inter-MSC handover. Although each form has different implementation details, the synchronization procedure between the mobile station (MS) and the base transceiver station (BTS) is common for all four handovers. During synchronization, the codepage is transmitted from the BTS to the MS. Figure 1 shows a user with an Android phone driving from an urban area to a rural zone. When the inter-BSC handover occurs, the new codepage from the rural area is sent to the user's Android phone during the synchronization process.

A second approach is to request the codepage based on the GPS location at the time of ERApp installation. This enables codepage download during a non-emergency when bandwidth may be more readily available. This approach is also better suited for fixed cellular devices that are more likely to remain in one designated cell.

To enhance messaging details in the available 90-character text, we propose two encoding/decoding methods for inclusion with ERApp: (i) predefined method; and (ii) just-in-time method. The predefined encoding method requires ERAlert and ERApp to use the message format specified in the codepage,

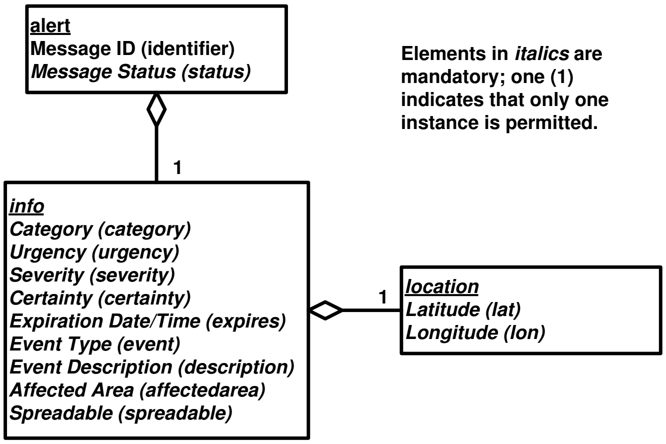


Figure 2. CAP object model for CMAS messages.

which is communicated prior to the broadcast of an alert message. A major advantage of this method is that more bytes are available for emergency data because the tag and attribute names are not encoded. The AMBER alert is a good candidate for the predefined encoding method. However, the predefined method suffers from a loss of flexibility with regard to including or excluding tags or attributes associated with a specific emergency.

The just-in-time encoding method does not require predefined message formats, provided that the alert message complies with the emergency tag repository identified in the codepage. Although the just-in-time method requires more bytes than the predefined method for the tag and attribute names, it offers the flexibility required for specific emergency alerts.

3.3 CMAS Encoding Schemes

Figure 2 presents the CAP object model for CMAS messages. Current encoding schemes limit the utility of the static data fields in CMAS messages. The enhanced structure, however, expands the XML schema to include more expressive information for emergency reporting. To realize the full benefit of the expanded structure, an encoding scheme is required that maximizes the amount of information that can be expressed in each element. Before describing the proposed enhancement, we review the WBXML and prime power encoding schemes currently used by CMAS.

3.3.1 WBXML Encoding. The WBXML encoding converts XML tags and attributes to the associated byte representation. WBXML encodes one byte for a beginning tag, one byte for a value and one byte for an ending tag. A similar method is used to encode an attribute. The details of the WBXML encoding are specified in Algorithm 1.

Algorithm 1 : WBXML Encoding Algorithm (Input: XML Stream)

```
Require: xmlStream $\neq$ null
 1: tokenSteam← new ByteArrayOutputStream()
 2: handler← new WBXMLContentHandler()
 3: reader← XMLReader.createXMLReader()
 4: reader.setContentHandler(handler)
 5: xmlSource← new InputSource(xmlStream)
 6: reader.parse(xmlSource)
 7: tokens← handler.getTokens()
 8: while tokens.hasNext() do
 9:    aToken← (Token)tokens.next()
10:    tokenStream.write(aToken.getValue())
11: end while
12: return  tokenStream
```

The WBXML algorithm requires a valid XML stream as its input. In Line 1, a new token stream is created as a new byte array output stream object to store the encoded values. A WBXML handler is then created that implements callback methods (e.g., beginning tag, closing tag and text value) and loads the codepage based on the namespace specified in an XML alert message. The algorithm creates the XML reader in Line 3 and registers the WBXML handler with the XML reader in Line 4. In Line 5, the algorithm creates a new input source object from the XML stream, which is provided as input for the reader.parse method in Line 6. The reader.parse method evaluates tags and attributes in pre-order. When the parser identifies a beginning or closing tag name, it calls the appropriate callback methods defined in the handler to identify the tag name in the tag repository. The handler.getTokens call in Line 7 returns the list of byte tokens to be written into the token stream. Lines 8 through 11 recursively evaluate the list of tokens and write out the encoded byte values to the output token stream, which is returned in Line 12. This technique is useful for encoding values of known tags and attributes.

3.3.2 Prime Power Encoding.

The prime power encoding method encodes an XML document as a single, albeit large, integer. Note that significant CPU power and computational time may be required to encode and decode a simple XML document. The details of the prime power encoding are specified in Algorithm 2.

Like the WBXML algorithm, the prime power algorithm requires a valid XML stream as input. In Line 1, the algorithm creates the XML prime power content handler. The handler implements the callback functions and loads the codepage based on the namespace specified in an XML alert message. In Line 2, the algorithm creates the reader. The handler is then registered with the reader in Line 3 so callback functions are referenced when XML tags are encountered. In Line 4, xmlSource is created from xmlStream. In Line 5, the reader.parse method performs post-order lookups from the leaf node to the root

Algorithm 2 : Prime Power Encoding Algorithm (Input: XML Stream)

```
Require: xmlStream $\neq$ null
 1: handler$\leftarrow$ new XMLPPContentHandler()
 2: reader$\leftarrow$ XMLReader.createXMLReader()
 3: reader.setContentHandler(handler)
 4: xmlSource$\leftarrow$ new InputSource(xmlStream)
 5: reader.parse(xmlSource)
 6: return handler.getPPValue()
```

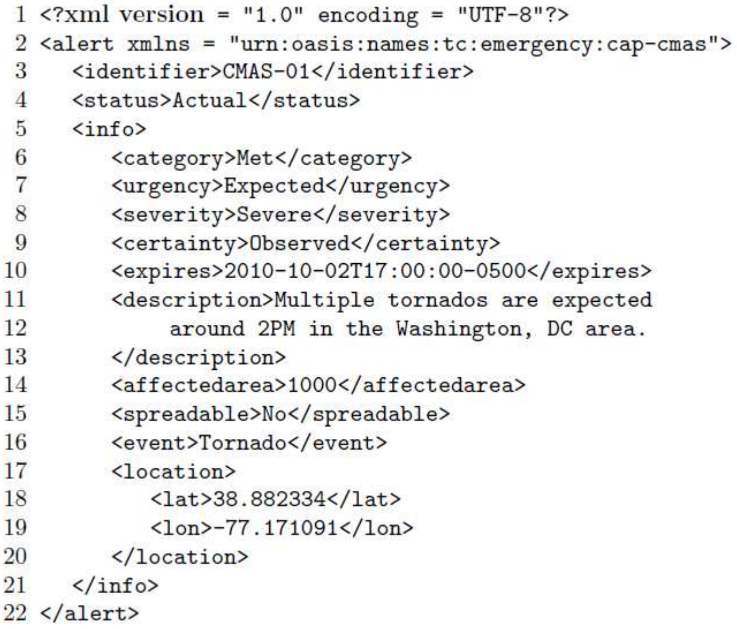
node. The first three prime numbers are reserved for the default node, internal node and leaf node. At every node and node value, the parser performs the prime encoding by taking the smallest available prime and raising it to the power of the integer value represented in the tag repository. Finally, in Line 6 the handler returns the encoded integer value corresponding to the XML document.

Consider the severity tag in the tornado example in Figure 3. The severity tag has an integer value of 2 and a severity value of 2. To encode the severity tag, the parser first encodes the severity tag value. Because the severity value is the leaf node, the parser raises the leaf prime number 3 to the power of 2, yielding a value of 9. The parser continues to encode the severity tag by raising the internal node prime of 2 with the tag integer value of 2. Therefore, the severity tag has the integer value of 4. To complete the encoding, the parser multiplies the tag integer value of 4 with 1953125, which is the result of raising the next available prime of 5 to the tag value integer of 9. The prime power encoding for the severity tag yields the large integer value of 7812500. Note that the integer values can be extremely large and become unmanageable very rapidly.

3.3.3 CMAS Encodings. In order to provide more meaningful CMAS alert messages for XML trees, we introduce the just-in-time and predefined XML encoding schemes. These encoding schemes significantly reduce the number of required encoded bytes.

Just-in-Time Encoding. The just-in-time encoding algorithm has two phases: (i) preprocessing phase; and (ii) encoding phase. The preprocessing phase builds the XML tag repository from the XML schema. For each level of depth in the XML tree, the preprocessing phase examines all possible tag names and associates each unique instance with an integer value starting at 0. The process is repeated for all tag values, attribute names and attribute values.

The preprocessing phase generates a codepage for all region-specific tag repositories. Each tag repository is identified and retrieved according to the unique namespace of the XML schema. The generated codepage is location-specific and pertains to emergencies that occur regularly in the associated region. For example, the codepage for areas in Florida can describe hurricane and



```
<?xml version = "1.0" encoding = "UTF-8"?>
<alert xmlns = "urn:oasis:names:tc:emergency:cap-cmas">
  <identifier>CMAS-01</identifier>
  <status>Actual</status>
  <info>
     <category>Met</category>
     <urgency>Expected</urgency>
     <severity>Severe</severity>
     <certainty>Observed</certainty>
     <expires>2010-10-02T17:00:00-0500</expires>
     <description>Multiple tornados are expected
          around 2PM in the Washington, DC area.
     </description>
     <affectedarea>1000</affectedarea>
     <spreadable>No</spreadable>
     <event>Tornado</event>
     <location>
        <lat>38.882334</lat>
        <lon>-77.171091</lon>
     </location>
  </info>
</alert>
```

Figure 3. CMAS mobile alert message for a tornado warning.

tornado related tags, while the codepage for areas in California can describe earthquake and wildfire tags.

Algorithm 3 : Preprocessing Algorithm (Input: XML Emergency Schema)

```
Require: xmlSchemaStream $\neq$ null
depth $\leftarrow$ getDepth(xmlSchemaStream)
maxTags $\leftarrow$ getMaxTags(xmlSchemaStream)
numTagBits $\leftarrow$ $log_2$(maxDepth) + $log_2$(maxTags) + 1
maxTagValues $\leftarrow$ getMaxTagValues(xmlSchemaStream)
maxAttrValues $\leftarrow$ getMaxAttrs(xmlSchemaStream)
maxValues $\leftarrow$ $max$(maxTagValues, maxAttrValues)
numValueBits $\leftarrow$ $log_2$(maxValues) + 1
numEncodingBits $\leftarrow$ $max$(numTagBits, numValueBits)
encodingScheme $\leftarrow$ getEncodingScheme(numEncodingBit)
return encodingScheme
```

As shown in Algorithm 3, the preprocessing phase uses the depth and the maximum number of tags, tag values, attribute names and attribute values. Furthermore, the preprocessing phase examines all static value tags and consolidates them to fit the selected encoding schema. Note that preprocessing generates a combined tag in the tag repository, which significantly reduces the number of encoded bytes.

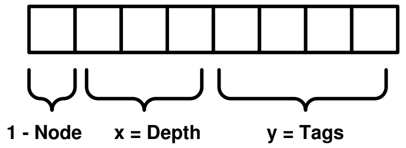


Figure 4. Node depth and tags.

The just-in-time encoding algorithm (Algorithm 3) requires a valid XML emergency schema as input. All the nodes in the schema are examined in Line 1 to compute the depth and in Line 2 to determine the maximum number of tags (maxTags). Line 3 computes the number of bits required to encode the tags. The algorithm then computes the maximum number of tag values (maxTagValues) in Line 4 and the maximum number of attribute names and attribute values (maxAttrValues) in Line 5. The maximum value (maxValues) between the two is computed in Line 6. The algorithm then computes the number of bits to encode the values. The number of bits required for encoding the XML schema is computed in Line 8. In Line 9, the encoding scheme (encodingScheme) is determined from one of the four values (e.g., 8 bits, 16 bits, 32 bits or 64 bits). The encoding is returned in Line 10.

An application of the just-in-time encoding scheme is illustrated in Figure 4. As an example, a one-byte encoding scheme is used to encode an XML document with depth $x = 8$ and maxTags $y = 16$. The number of encoding bits (numTagBits) z_1 is computed as:

$$z_1 \geq log_2(x) + log_2(y) + 1$$

where x is the depth and y is the maximum number of tags at each depth.

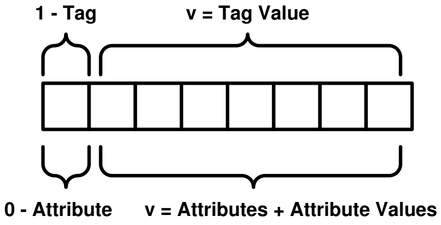


Figure 5. Attribute depth and tags.

Figure 5 illustrates the encoding of a tag value, attribute name and attribute value. Note that the depth does not have to be encoded because it is incorporated into the node. To compute the number of encoding bits for the values (numValueBits) z_2, the maximum number of tag values (maxTagValues) and the maximum number of attributes and attribute values (maxAttrValues) must be determined first. The number of encoding bits for the values is computed

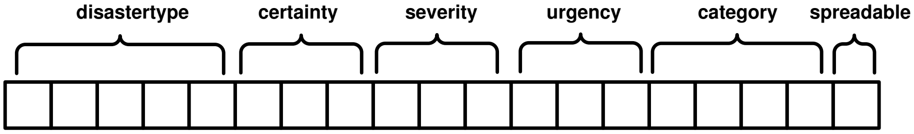


Figure 6. Combined tag.

as:

$$z_2 \geq log_2(v) + 1$$

where v is the maximum number between tag values, attribute names and attribute values for each tag (maxValues). Finally, the number of encoding bits (numEncodingBits) z required is given by:

$$z = max(z_1, z_2).$$

In Figure 5, we use an 8-bit encoding scheme to encode a known tag or a known attribute. With this encoding scheme, the maximum number for tag values, attribute names and attribute values is 128 for each tag.

Consider, for example, the tornado alert message discussed previously (Figure 3). The category, urgency, severity, certainty and spreadable tags have static values. If each tag is encoded separately, ten bytes are required to encode the five tags and five tag values. As shown in Figure 6, combining the tags realizes an encoding that uses only three bytes. Note that the category tag has twelve possible values and requires four bits, whereas the spreadable tag is a Boolean value and only requires one bit.

Algorithm 4 : MXML Encoding Algorithm (Input: XML Stream)

```
Require: xmlStream $\neq$ null
 1: xmlStream$\leftarrow$ combinedTags(xmlStream)
 2: handler$\leftarrow$ new MXMLContentHandler()
 3: reader$\leftarrow$ XMLReaderFactory.createXMLReader()
 4: reader.setContentHandler(handler)
 5: xmlSource$\leftarrow$ new InputSource(xmlStream)
 6: reader.parse(xmlSource)
 7: masterNode$\leftarrow$ handler.getMasterNode()
 8: outputStream$\leftarrow$ new ByteArrayOutputStream()
 9: masterNode.writeStream(outputStream)
10: arrayBytes$\leftarrow$ outputStream.toByteArray()
11: outputStream.close()
12: return arrayBytes
```

After the preprocessing phase, the XML alert message document is ready for the encoding phase. Algorithm 4 specifies the CMAS encoding scheme, which we refer to as the mobile XML (MXML) encoding.

The MXML algorithm requires the codepage and a valid XML stream as input. In Line 1, combined tags in the tag repository generate a new XML stream. The MXMLContentHandler function in Line 2 creates the handler, which implements the callback functions and loads the codepage based on the XML namespace specified in the XML alert message. In Line 3, the XMLReader function creates the reader and, in Line 4, the handler is registered with the reader. In Line 5, xmlSource is created from xmlStream. In the encoding phase, the MXMLContentHandler creates the internal tag repository. The reader.parse method is called in Line 6 with xmlSource as the parameter; the reader examines each XML tag value, attribute name and attribute value in pre-order. The parser then encodes the XML document, starting from the root element at depth zero and continues recursively to the other elements.

The encoding method uses a pre-order traversal to encode every tag name, tag value, attribute name and attribute value. For every value encountered by the parser, the value in the tag repository is correlated based on the depth level and its parent node. The value is encoded by inserting the byte representation into a byte array. The encoding bytes in the array from the child elements are appended to the parent node. If the value is not statically known (i.e., the value is not registered in the tag repository), then it is encoded as a text value with null bytes representing the beginning and end of the text. After the encoding phase is complete, a byte array stream containing the encoding bits is returned. The encoding byte array stream must be converted into an array of characters using Base64 encoding to provide human-readable text. The additional message length provided by the encoding is given by:

$$Base64\ Length = (Bytes + 2 - ((Bytes + 2)\ mod\ 3))/3 * 4.$$

An additional consideration is that the cellular broadcast service uses an independent broadcast control channel with dedicated bandwidth to send emergency alerts. Because there is no competition with other channels for bandwidth, emergency information can be segmented into multiple messages if the alert exceeds the standard 90 characters. ERApp uses the message header fields to reconstruct the original emergency information in its entirety.

Predefined Encoding. Similar to the just-in-time encoding algorithm, the predefined encoding algorithm has two phases: (i) preprocessing phase; and (ii) encoding phase. The implementation mirrors the just-in-time algorithm, but with a slight modification in the preprocessing phase. Specifically, in the case of the predefined encoding method, the alert message format is defined and stored in the codepage during the preprocessing phase. ERAlert and ERApp use the prescribed message format to encode and decode the alert message, respectively. The encoding phase proceeds as described for the just-in-time encoding.

4. Experimental Evaluation

In order to evaluate performance, alert information is classified as either static or dynamic. Static data fields contain predefined values that emergency

Table 1. Performance evaluation summary.

Encoding Algorithm	**Alert Time**	**Encoding Length**	**Base64 Length**
WBXML	179 ms	267 bytes	356 bytes
Prime Power	Infeasible	N/A	N/A
MXML (Predefined)	160 ms	98 bytes	132 bytes
MXML (Just-in-Time)	158 ms	118 bytes	160 bytes

operators can select based on the situation; examples include category, urgency, severity, certainty and event. Dynamic data fields contain values that cannot be predefined due to intractable uncertainty; examples include event location, expiration time and description.

We revisit the tornado example to illustrate the encoding of static and dynamic data. The data element, category, represents the static environment and is specified as <category>Met</category>. Let s be the description of the category data field and let $D(s)$ be its length such that the normal value of $D(s)$ is 24 characters.

All the tokens and their values are defined in the tag repository for CMAS alerts. Each token is represented by an integer value starting at zero. The category field and MET values are encoded as 0 and 2 (bytes, not integer values), respectively. Therefore, the CMAS encoding requires only two bytes instead of the eight bytes needed to represent two integers; thus, the corresponding $D(s)$ value is two bytes.

For the dynamic data category, data fields and values are encoded according to their data types. For example, the expiration time may be encoded as a string representing the timestamp, which requires up to 20 characters. However, the data field also can be encoded as a long value for milliseconds or an integer value for seconds. Because the number of milliseconds provides more detail than is necessary, the expiration time is encoded as an integer value that requires only four bytes.

The WBXML, prime power, MXML just-in-time and MXML predefined algorithms were executed for CMAS mobile alert messages corresponding to the tornado example. A Dell Latitude E6400 with dual core processors was used as the computing platform. Because CMAS only allows 90 characters per broadcast, the tornado alert was segmented into two broadcast messages and reconstructed by the receiving ERApp. Table 1 shows the results for the average alert time, encoding length and the Base64 encoding length. The MXML predefined encoding provides the shortest length for readable text (i.e., Base64 encoding length). The MXML just-in-time and MXML predefined encodings used less bytes to encode the message than the original CMAS encoding scheme while also minimizing the alert time. Note that the prime power encoding was declared to be infeasible because it exhausted the CPU utilization rate and was unable to encode the alert message even after several hours of processing.

5. Conclusions

The CMAS extension described in this paper enables detailed emergency information to be incorporated in alert messages while complying with the 90-character message specification. The utility of the approach is demonstrated by the ability to install the ERApp emergency response application on an Android platform and receive detailed emergency alert messages.

References

[1] Alliance for Telecommunications Industry Solutions, Implementation Guidelines and Best Practices for GSM/UMTS Cell Broadcast Service, ATIS-0700007, Washington, DC, 2009.

[2] Alliance for Telecommunications Industry Solutions, Commercial Mobile Alert Service (CMAS) via GSM/UMTS Cell Broadcast Service Specification, ATIS-0700006, Washington, DC, 2010.

[3] Federal Communications Commission, Common Mobile Alert System (CMAS), Washington, DC (`www.fcc.gov/cgb/consumerfacts/cmas.html`), 2011.

[4] National Institute of Standards and Technology, Federal Information Processing Standards Publications, Gaithersburg, Maryland (`www.itl.nist.gov/fipspubs/index.htm`).

[5] National Public Safety Telecommunications Council, Commercial Mobile Alert Service Architecture and Requirements, Version 0.6, Littleton, Colorado (`www.npstc.org/download.jsp?tableId=37&column=217&id=703&file=PMG-0035_Final_Recommendations_v0_6.pdf`), 2007.

[6] P. Ngo and D. Wijesekera, Using ontological information to enhance responder availability in emergency response, *Proceedings of the Semantic Technology for Intelligence, Defense and Security Conference*, 2010.

[7] P. Ngo and D. Wijesekera, Enhancing the usability of the Commercial Mobile Alert System, in *Critical Infrastructure Protection V*, J. Butts and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 137–149, 2011.

[8] Northern Virginia Resource Center for Deaf and Hard of Hearing Persons, Emergency Preparedness and Emergency Communication Access Lessons Learned Since 9/11 and Recommendations, Fairfax, Virginia (`tap.gallaudet.edu/emergency/nov05conference/EmergencyReports/DHHCANEmergencyReport.pdf`), 2004.

[9] Organization for the Advancement of Structured Information Standards, Emergency Data Exchange Language (EDXL) Distribution Element, v1.0, OASIS Standard EDXL-DE v1.0, Burlington, Massachusetts (docs.oasis-open.org/emergency/edxl-de/v1.0/EDXL-DE_Spec _v1.0.pdf), 2006.

[10] Organization for the Advancement of Structured Information Standards, Common Alerting Protocol Version 1.1 (Approved Errata), Burlington, Massachusetts (docs.oasis-open.org/emergency/cap/v1.1/errata/CAP-v1.1-errata.html), 2007.

[11] U.S. Department of Justice, AMBER Alert, Washington, DC (www.amberalert.gov).