Network Working Group
Internet-Draft
Expires: August 29, 2003

M. Ramalho
Cisco Systems, Inc.
February 28, 2003

RGL Codec Description Document
draft-ramalho-rgl-desc-01.txt

Status of this Memo

Copyright Notice

Abstract

This document describes the operation of the RGL codec which obtains significant lossless compression of speech/audio packet payloads encoded with ITU-T Recommendation G.711 PCM (mu-law or A-law, IETF RTP payload types PCMA or PCMU) with trivial complexity and virtually no delay. Full documentation can be found at www.vovida.org [14].

The RTP payload format proposed for this codec is described in draft-ramalho-rgl-rtpformat-01.txt [4].

Table of Contents

1. Introduction

This document describes the operation of the RGL (short for Ramalho G.711 Lossless) codec which obtains significant lossless compression of speech/audio packet payloads encoded with ITU-T Recommendation G.711 PCM (mu-law or A-law, IETF RTP payload types PCMA or PCMU) with trivial complexity and virtually no delay. Full documentation for this codec can be found at www.vovida.org [15]. The remainder of this introduction provides the motivation behind the creation of the RGL codec.

To improve bandwidth efficiency for IP transport of normal telephony audio signals, audio signals are often compressed by the interworking hardware between the PSTN/GSTN and the transport IP network. The types of audio compression used are often optimized for speech, such as G.723.1 and G.729A. These coders compress the PSTN/GSTN PCM signals (defined in ITU-T Standard G.711) from 64 kbps to bit rates of 8 kbps or less.

There is a desire, and sometimes an absolute need, in many VoIP applications to send voice data over IP networks to the end systems in the identical PCM format it was presented to the VoIP system by a PSTN/GSTN interworking device. Most examples of this need are associated with the fact that many voice coders do not transport the signal with the required fidelity for the application using the channel (e.g., DTMF/TTY/TDD pass-through and modem or FAX pass-through).

For example, generalized audio is often poorly reproduced when coded and decoded by most speech coders (e.g., music on hold). This particular problem could be resolved via the use of slightly higher bandwidth audio coders (on the order of 16 kbps for voiceband quality). Another example, arguably more important, is the case when a voiceband modem signal is being transported. V.90 modems typically require the entire 64kbps signal to be transported unmodified from end-to-end[EndNote:1]. A "codec switch" from a speech codec to G.711 upon determination that a voiceband modem is present could also resolve this problem with a slight increase of modem set-up delay. A third example is in a network where bandwidth is plentiful and there is no need or desire to further degrade the PSTN signal via a lossy transcoding. As many VoIP transport providers desire not to degrade the audio quality over the distortions already created by the G.711 companding, there is often a need to transport the signal in the identical PCM format presented to the PSTN/GSTN interworking device[EndNote:2].

Additionally, a method of lossless compression for G.711 signals may be of utility at the beginning of end-to-end media transmission.

During this time, lossless compression can be used until it is determined whether the channel is being used only for normal speech (i.e., not music or other audio signals or modems) and an informed decision to transition to a speech-specific codec (or other general audio codec) can be made. For the case where the call is a normal voice call and an informed decision has yet to be made, the lossless compression techniques described below can be used to obtain compression gain over G.711 during this period[EndNote:3].

2. Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC2119 [2].

3. Companding Codec Background Information

ITU-T Recommendation G.711 specifies the dominant companding methods used in the PSTN/GSTN, the so-called mu-law and A-law companding (with mu=255 and A=87.56). The companding codecs in this specification are actually piecewise linear approximations to the non-linear mu-law and A-law defining equations and these equations may be found in many coding and telecommunications textbooks [5],[6]. The primary aim of both "mu-255" and "A 78.56" law codecs is to quantize larger signals more coarsely and smaller signals more finely, resulting in a "flatter" SNR over a wider dynamic range while using only 8 bits. For example, the (eight bit) "mu-255" companding coder approximates the SNR attained by a 13 linear codec for low signal input levels[Endnote:4]. The tradeoff is less SNR at high signal levels than the equivalent 13 bit linear codec owing to the coarser quantization of larger input signals. This tradeoff saves 5 bits (13-8) or 40kbps!

Both mu-law and A-law companding effectively use eight linear segments on either side of analog zero (16 linear segments). The "mu-255" law consists of a 15 segment characteristic with the two innermost segments about zero having the identical slope; the "A 78.56" law has the four innermost segments having the identical slope, resulting in 13 areas of distinct slope.

In both cases, log 16 = 4 bits convey the segment information. Further, the amplitudes within each segment are quantized to 16 levels, requiring an additional 4 bits. These eight bits are organized as:

Bit 1:	Sign (p bit)
Bits 2 to 4:	Segment number (s bits)
Bits 5 to 8:	Level within a segment (l bits)

Although not necessary for description here, a natural binary code (NBC) has been used in both G.711 specifications for representing the segment number s and the level number l. The same NBC representation being used for both positive and negative values, resulting in a folded binary code. Lastly, in accordance with limitations of early transmission systems that had a "ones density" requirement, the "mu-255" code was constructed so that low amplitude signals are represented with codewords with more ones than zeros[Endnote:6]. Unfortunately this particular coding of the linear segments (bits 1-4) is such that adjacent segments S(i) and S(i+1) are such that adjacent segments usually differ in coding by more than one bit.

The important thing to note, however, is that small signals do not "excite" the linear segments that represent the larger segments. For

example, low "background noise" (a very quiet input signal) may only span quantization levels in the two linear segments about zero. In this case, Bit 2, Bit 3 and Bit 4 remain constant for every sample during this "background noise" interval ("111" for "mu-255" and "000" for "A 78.56"). Therefore, if one knew a priori that the signal contained in a given audio frame only spanned these two segments, you need not transmit Bits 2, 3, or 4 - resulting in a 37.5% bandwidth savings. The lossless compression for G.711 signals of the RGL codec exploits similar opportunities to save bandwidth at trivial additional computational cost.

Before describing the method, it is instructive to note an important property of the resultant coding produced by speech/audio signals that have been coded by G.711 coders. The speech/audio so encoded will typically be zero mean. This is because these signals typically are: 1) based on acoustic signals (which are usually zero mean over any significant observation interval), 2) have been coupled by devices that are only able to transduce acoustic signals to electrical signals in a zero mean sense (ignoring "dc biasing" of such transducers as microphones), and 3) the signals have been high pass filtered (typical PSTN/GSTN cutoffs of around 100 Hz). As a result, virtually all real world (i.e., not artificially generated) G.711 encodings are expected to be biased around the signal magnitudes about the "analog" zero. Nevertheless, the RGL coding methodology accommodates, although sometimes less efficiently, G.711 encodings where this real-world property does not hold (i.e., artificially generated signals).

4. RGL Codec Description

G.711 PCM is grouped into "speech/audio" frames for transport over packet networks. The default audio segment size currently specified in RFC 1890 [8] for G.711 VoIP packets is 20 msec, which would result in 160, eight-bit samples or 160 bytes of G.711 coded audio. Typical frame sizes for speech encoders are 10 msec (e.g., G.729), 20 msec (e.g., GSM), and 30 msec (e.g., G.723.1). One can also place more than one speech/audio frame per packet. The following RGL method described in this document operates on speech/audio frames independent on their length and each such frame is referred to as an "audio frame". In the description that follows "audio samples" (individual G.711 PCM sample encodings) are indexed with the sample index i and the audio frames are indexed with the frame index j.

For each sample we map the G.711 mu-law and A-law output alphabet from the most negative value to the most positive value according to the following table. This table maps the most negative quantization level of the G.711 alphabet (either mu-law or A-law) to q(0) and the most positive to q(255); the anchoring codepoint column will be described shortly.

Table 1: G.711 (mu-law and A-law) to 0-255 Linear Codepoint Mapping

0-255 Linear Quantization Codepoint	G.711 mu-law encoding	G.711 A-law encoding [EndNote:7]	Anchor Codepoint (value)
q(255); most +	1000 0000	1111 1111	
q(254)	1000 0001	1111 1110	
q(253)	1000 0010	1111 1101	
******	**** ****	**** ****	
q(137)	1111 0110	1000 1001	
q(136)	1111 0111	1000 1000	
q(135)	1111 1000	1000 0111	
q(134)	1111 1001	1000 0110	
q(133)	1111 1010	1000 0101	
q(132)	1111 1011	1000 0100	
q(131)	1111 1100	1000 0011	
q(130)	1111 1101	1000 0010	
q(129)	1111 1110	1000 0001	00000= 0 (B(j)={0,1 ... 7})
q(128); 0+ lvl	1111 1111	1000 0000	00001= 1 (B(j)={0,1 ... 7})
q(127); 0- lvl	0111 1111	0000 0000	00010= 2 (B(j)={0,1 ... 7})
q(126)	0111 1110	0000 0001	00011= 3 (B(j)={0,1 ... 7})
q(125)	0111 1101	0000 0010	00100= 4 (B(j)={0,1 ... 7})

q(124)	0111 1100	0000 0011	00101= 5 (B(j)={0,1 ... 7})
q(123)	0111 1011	0000 0100	00110= 6 (B(j)={0,1 ... 7})
q(122)	0111 1010	0000 0101	00111= 7 (B(j)={0,1 ... 7})
q(121)	0111 1001	0000 0110	01000= 8 (B(j)={0,1 ... 7})
q(120)	0111 1000	0000 0111	
q(119)	0111 0111	0000 1000	01001= 9 (B(j)={0,1 ... 7})
q(118)	0111 0110	0000 1001	
q(117)	0111 0101	0000 1010	01010=10 (B(j)={0,1 ... 7})
q(116)	0111 0100	0000 1011	
q(115)	0111 0011	0000 1100	01011=11 (B(j)={0,1 ... 7})
q(114)	0111 0010	0000 1101	
q(113)	0111 0001	0000 1110	00001=12 (B(j)={0,1 ... 7})
******	**** ****	**** ****	SEE NEXT TABLE FOR ALL
******	**** ****	**** ****	ANCHORING POINT LOCATIONS
q(4)	0000 0100	0111 1011	
q(3)	0000 0011	0111 1100	
q(2)	0000 0010	0111 1101	
q(1)	0000 0001	0111 1110	
q(0); most -	0000 0000	0111 1111	B(j) = 8 anchoring location

For a given audio frame j, we note the most negative and most positive quantization level spanned by the signal in this frame. Label these levels as Y[POS](j) and Y[NEG](j), respectively and let their value be equal to their q(.) codepoint. Let Y[MAX](j) be the number of contiguous codewords spanned from Y[POS](j) to Y[NEG](j), inclusively, for audio frame j (i.e., Y[MAX](j) = [Y[POS](j) - Y[NEG](j) +1]). Let B(j) be The minimum number of bits needed to assign the Y[MAX](j) number of quantization levels. That is B(j) = ceil(log2 Y[MAX](j)) where "ceil()" denotes the integer ceiling function and "log2" is the logarithm base 2. For example, if the resultant sample encoding spanned fifteen contiguous quantization levels, four bits can be used to encode (describe) these 15 levels. The RGL encoding to be described will always use B(j) number of bits per sample to encode each sample in frame j.

The RGL encoder codes the individual sample points i of frame j (which span from Y[POS](j) to Y[NEG](j)) by a binary count up from an "anchoring location". If possible, this anchoring location is specified by an "anchoring codepoint" which is encoded as "side information" for the RGL frame j. All possible anchoring codepoint locations are shown in the following table. On rare occasion, we will use an "explicit anchor" whereby the anchoring location is not an anchoring codepoint quantization level, but rather an explicit location conveyed via a second byte of "side information" in the RGL frame. We call this explicit anchoring location an "explicit anchor".

The exact algorithm that determines whether an "explicit anchor" byte is required will be described shortly.

The RGL encoding endeavors to send as "side information" (the number of bits, B(j), and the location of the "anchor codepoint") in one "overhead" byte for majority of frame encodings. As mentioned above, on rare occasions it is required that two bytes of side information be sent. Since B(j) can range from 0 bits (only one G.711 value was contained for the every sample i in frame j) to 8 bits (almost or all of the entire G.711 encoding range was used in the frame), we would normally require 4 bits to encode this information; however we are able to use three bits by using an anchoring codepoint reserved to signal the B(j) = 8 case (this will be described later). These three bits are defined as the "N bits" of the first overhead byte. Because a goal of the RGL codec is to use only one overhead byte for most encodings, 5 bits remain in the first overhead byte that are used to describe (encode) an anchoring codepoint location. These five bits are defined as the "A bits" of the first overhead byte. This, in turn, allows for up to 32 possible "anchoring codepoint" locations. We will use one of these possible codepoint locations to signal when we require an "explicit anchor" be sent in the second overhead byte. Additionally we will also use another anchoring codepoint location to signal a B(j) = 8 encoding (this will be described shortly in the N bit table discussion below). Thus there are 30 locations (32-2) available for use as anchoring point locations.

Recalling that most real-world signals are zero mean, we begin with the assumption that the signal (linear q(.)) range will be about the "analog zero" level (i.e., about the q(127) or q(128) levels in the above table). Assuming that we wish to anchor the codepoint on the most negative value of Y[NEG](j), the anchoring point need only be specified to reside in the negative portion of the codespace. If B(j) = 8 (i.e., no RGL compression possible) the anchor is defined to be the most negative quantization level (q(0)). If B(j) does not equal 8, then a near-optimal strategy for defining the anchoring codepoint locations would be to assign the 30 anchoring locations approximately logarithmically from the quantization level representing zero amplitude.

The anchoring codepoint locations chosen are defined in the following table.

Table 2: Full Listing of All Anchoring Point Locations

0-255 Linear Quantization	Anchor Codepoint {A5,A4,A3,A2,A1}
q(129)	00000 = 0 (if B(j) = {0,1 ... 7})

q(128); 0+ lvl	00001 = 1 (if B(j) = {0,1 ... 7})	
q(127); 0- lvl	00010 = 2 (if B(j) = {0,1 ... 7})	
q(126)	00011 = 3 (if B(j) = {0,1 ... 7})	
q(125)	00100 = 4 (if B(j) = {0,1 ... 7})	
q(124)	00101 = 5 (if B(j) = {0,1 ... 7})	
q(123)	00110 = 6 (if B(j) = {0,1 ... 7})	
q(122)	00111 = 7 (if B(j) = {0,1 ... 7})	
q(121)	01000 = 8 (if B(j) = {0,1 ... 7})	
q(119)	01001 = 9 (if B(j) = {0,1 ... 7})	
q(117)	01010 = 10 (if B(j) = {0,1 ... 7})	
q(115)	01011 = 11 (if B(j) = {0,1 ... 7})	
q(113)	01100 = 12 (if B(j) = {0,1 ... 7})	
q(111)	01101 = 13 (if B(j) = {0,1 ... 7})	
q(108)	01110 = 14 (if B(j) = {0,1 ... 7})	
q(105)	01111 = 15 (if B(j) = {0,1 ... 7})	
q(102)	10000 = 16 (if B(j) = {0,1 ... 7})	
q(99)	10001 = 17 (if B(j) = {0,1 ... 7})	
q(96)	10010 = 18 (if B(j) = {0,1 ... 7})	
q(92)	10011 = 19 (if B(j) = {0,1 ... 7})	
q(88)	10100 = 20 (if B(j) = {0,1 ... 7})	
q(84)	10101 = 21 (if B(j) = {0,1 ... 7})	
q(80)	10110 = 22 (if B(j) = {0,1 ... 7})	
q(75)	10111 = 23 (if B(j) = {0,1 ... 7})	
q(70)	11000 = 24 (if B(j) = {0,1 ... 7})	
q(65)	11001 = 25 (if B(j) = {0,1 ... 7})	
q(60)	11010 = 26 (if B(j) = {0,1 ... 7})	
q(54)	11011 = 27 (if B(j) = {0,1 ... 7})	
q(48)	11100 = 28 (if B(j) = {0,1 ... 7})	
q(41)	11101 = 29 (if B(j) = {0,1 ... 7})	
q(0); most neg	11110 = 30 (only if B(j) = 8)	*Note 1*
Explicit Anchor	11111 = 31 (if B(j) = {0,1 ... 7})	*Note 2*

Note 1: If B(j)=8, codepoint {11110} is used to signal an 8 bit per sample encoding (see N bit table and discussion below). All B(j)=8 bit encodings are anchored at q(0).

Note 2: The anchoring location is explicitly provided via a second overhead byte (see discussion below).

Note that a few anchoring codepoints are placed above the 0- level to compensate for potentially inaccurate analog G.711 A/D encoding bias (small positive DC offset in the analog A/D converter front end circuitry). These remaining anchor codepoints are spaced approximately logarithmically from the "analog zero" to the most negative quantization level.

The following table specifies the value of the N bits.

Table 3: Mapping for the N bits of First Overhead Byte

B(j)	N Codepoint {N3, N2, N1}
Eight Bits (if A bits = {11110})	000
Zero Bit (if A bits != {11110})	000
One Bits	001
Two Bits	010
Three Bits	011
Four Bits	100
Five Bits	101
Six Bits	110
Seven Bits	111

Figure 4

First note that a B(j)=8 encoding is signaled by the N bits all zeros AND the A bits = 30 ({11110}). All other N bit combinations with A bits = 30 are reserved for use by the RTP payload format or for other future signaling use. As we will later see in RGL encoding section, the N bits and the A bits are packed in the first RGL overhead byte. Thus there are seven reserved first overhead bytes; they are when the A bits = 30 and the N bits are not all zeros (0x3E, 0x5E, 0x7E, 0x9E, 0xBE, 0xDE and 0xFE). The RGL RTP payload format will make use some of these codes to define the RGL frames in the RTP payload. As of the writing of this document, the details of the RTP payload format for the RGL codec have not been agreed upon at the IETF. The present draft for the RGL RTP payload format is draft-ietf-ramalho-rgl-rtpformat-01.txt [16].

Note that all B(j) = 8 encodings are always anchored at q(0). A second item to note is for any encoding other than an eight bit encoding (i.e., B(j) != 8) that the anchoring location is determined via the A bits. Lastly note that A codepoint {11111} (= 31) is used to signal that an "explicit anchoring location" is required to be sent in the second overhead byte (i.e., an explicit anchor byte). We will discuss when this byte is needed in the following section.

The second item to note is for any encoding other than an eight bit encoding (i.e., B(j) != 8) that the anchoring location is determined via the A bits. Lastly note (also for B(j) != 8) that A codepoint {11111} (= 31) is used to signal that an "explicit anchoring location" is required to be sent in the second overhead byte (i.e., an "explicit anchor byte").

4.1 Basic RGL Encoding Algorithm

Before describing the detailed RGL encoding details, it is instructive to describe in rough terms what the RGL encoding endeavors to do and then to describe some atypical corner cases that accommodate all possible, including "non real-world", G.711 encodings (e.g., artificially generated G.711 encodings).

The basic RGL encoding methodology is as follows. First we map the G.711 mu-law or A-law encoding to the 0-255 linear codepoint representation in the above table for each sample i in frame j. Then we determine the signal range in a given frame j by noting Y[POS](j) and Y[NEG](j) and determine B(j) = ceil(log2 Y[MAX](j)) as discussed above. If B(j)=8, then the default anchoring codepoint is q(0), the N and A bits in the first overhead byte are defined as above and all samples i in frame j are encoded as a binary count up from q(0). For all other cases (i.e., B(j) != 8) we determine if Y[NEG](j) can be represented by an existing codepoint location. If it can, then we simply chose that codepoint, set the N and A bits in the first overhead byte as appropriate, and encode all samples i in frame j as a binary count up from this anchoring codepoint location. If Y[NEG](j) is not an anchoring codepoint location, then we choose a tentative anchoring location to be the next most negative codepoint location, denoted Y[ANCHOR](j). If the number of contiguous codewords spanning from Y[POS](j) and Y[ANCHOR](j) can still be represented by B(j) bits, then we simply chose that codepoint, set the N and A bits in the first overhead byte as appropriate, and encode all samples i in frame j as a binary count up from this anchoring codepoint location. However, it is possible that the difference between the tentative anchor Y[ANCHOR](j) from Y[NEG](j) could have resulted in a span that required more than B(j) bits to represent. In this case, we can save one or more bits per sample (e.g., 80 bits for an 80 byte G.711 frame or 160 bits for a 160 byte G.711 frame) by simply encoding Y[NEG](j) exactly (using 8 bits) at the cost of sending a second overhead byte: the so-called "explicit anchor byte". In this case, we set the N bits (according to B(j)) and the A bits (to denote use of an explicit anchor), send the explicit anchor in the second overhead byte (Y[NEG](j), coded as a binary count up from q(0)) and we encode all samples i in frame j as a binary count up from the location of the "explicit anchor" (i.e. Y[NEG](j)) location.

Now, before describing the detailed RGL algorithm, it is instructive to consider two atypical corner cases.

Atypical Case One:

The first atypical case is the case when the entire mu-law or A-law frame contains only one value, which we call the "zero bit per sample" (B(j)=0) case. To make room for signaling this case, the table above uses the same N codepoint to signal both the eight

bit and zero bit case depending on the value of the A bits. An eight bit encoding is signaled in the overhead byte by using the N = {0 0 0} bits and the A = {1 1 1 1 0} codeword. Note that when a "zero bit" encoding case occurs in a typical embodiment of a mu-law or A-law PCM system, the one value is most likely a value around the "natural zero" of the G711 encoded space (i.e., near the 0+ (q(128)) or 0- level (q(128))). Since we have anchoring codepoints at these levels (actually we have continuous anchoring codepoints from q(121) through q(129), inclusive), we expect to be able to compress this frame to exactly one byte! However, to accommodate all possible artificially generated G711 signals that could have the single level at any quantization level, we must use an "explicit anchor" if an anchoring codepoint is not available at that quantization level.

Atypical Case Two:

The second atypical case to consider is when less than eight bits can represent the signal range (Y[POS](j) to Y[NEG](j), inclusive) and Y[NEG](j), is below the most negative anchoring codepoint location. If the signal range is less than 8 bits, we expect that the signal range should be such that Y[NEG](j), is well above the most negative codepoint available, q(41), as natural signals are zero mean signals. Thus, for example, a 7 bit encoding (representing up to a 128 bit range) is expected to live approximately between q(64) (about 64 levels below 0-) and q(192) (about 64 levels above 0+); note that q(64) is well q(41). However, to represent all possible artificially generated G711 signals, we must resort to using an "explicit anchor" to represent Y[NEG](j) if Y[NEG](j) is below the lowest available anchoring codepoint (for zero through seven bit encodings, as all eight bit encodings are always anchored at q(0)).

4.2 Detailed RGL Encoding Algorithm

Knowing the constraints presented by the two atypical cases above, the algorithm for the RGL coder follows the following six steps:

Step 1: Map the G711 A-law or mu-law quantization codepoints to their 0-255 linear quantization codepoints.

Step 2: Calculate the number of bits per sample required for this coding. The number of bits required is B(j) = ceil(log2(Y[MAX](j)) where Y[MAX](j) is the number of contiguous codewords spanned from Y[POS](j) to Y[NEG](j), inclusive, for audio frame j. Set the N

bits to represent this number of bits.

Step 3: If B(j)=8 (the anchoring location for this frame is q(0) by default) set Y[ANCHOR](j)= q(0), set the A bits appropriately (i.e., to {11110}) and >> Skip To Step 5 << below. Otherwise continue.

Step 4: For B(j) !=8, find the "tentative anchoring codepoint" location Y[ANCHOR](j).

If Y[NEG](j) is more negative than the lowest anchoring codepoint, q(41), >> Skip To Step 4A <<.

If Y[NEG](j) is an anchoring codepoint location, >> Skip To Step 4B <<.

If Y[NEG](j) does not represent an anchoring codepoint location but is above the lowest available anchoring codepoint, >> Skip To Step 4C <<.

Step 4A: An explicit anchor must be used to represent Y[NEG](j) for this case (i.e., set Y[EXPLICIT_ANCHOR](j) = Y[NEG](j)). This explicit anchor will need to be encoded and will sent as a second overhead byte. When an explicit anchor is required, the A bits need to be set to reflect that an explicit anchor will be used (i.e., set to {1 1 1 1 1}), code the explicit anchor as a binary count up from q(0), and place this value in the second overhead byte (details on how to pack this explicit anchor are described below). >> Skip To Step 5 <<.

Step 4B: Set Y[ANCHOR](j) = Y[NEG](j) and set the A bits to reflect this anchoring location. >> Skip To Step 5 <<.

Step 4C: Set Y[ANCHOR](j) to the next most negative anchoring codepoint location. Find the number of bits required for coding with the tentative anchor Y[ANCHOR](j). Let Y[NEW_MAX](j) be the number of contiguous codewords spanned from Y[POS](j) to Y[ANCHOR](j), inclusive. Define the number of bits per sample needed if the tentative anchor is used, B[NEW](j), to be B[NEW](j) = ceil(log2(Y[NEW_MAX](j))). If B[NEW](j) = B(j), then Y[ANCHOR](j) is to be used as the anchor location for this frame j, set the A bits to signal the use of this anchor (there will not be a second overhead byte used for this case) and >> Skip To Step 5 <<.

If B[NEW](j) is more than B(j), the setting an explicit anchor is required (as it s use will result in a more efficient coding of the samples) and >> Skip To Step 4A <<.

Step 5: If the coding is not a "zero bit" per sample encoding (i.e., B(j) != 0), anchor the "all zeros" codepoint containing B(j) bits at Y[ANCHOR](j) (if explicit anchor not needed) or Y[EXPLICIT_ANCHOR](j) (if explicit anchor was required) as appropriate.

Step 6: If the coding is not a "zero bit" per sample encoding (i.e., B(j) != 0), use a simple binary counting up from the anchoring location (Y[EXPLICIT_ANCHOR](j)] or Y[ANCHOR](j), as appropriate) for each successive, more positive quantization level up to Y[POS](j) (this is possible using exactly B(j) bits) for every sample in the frame j. Label these bits as Z bits beginning with Z1 as the least significant bit through as many Z bits are necessary for the coding (again, exactly B(j) bits for each sample). For example, a five bit encoding would have bits Z5,Z4,Z3,Z2,Z1. Pack the Z bits for all samples in the G711 frame as defined in the following section.

4.3 RGL Frame Encoding

The RGL encoding for a given frame j is described in this section.

The first byte of an encoded frame is the side information, encoded as: {N3,N2,N1,A5,A4,A3,A2,A1}. The N bits are as defined in the N bit table above and the A bits are defined using the algorithm above and

set in accordance with Table 2.

Note that if a second overhead byte (the explicit anchor byte) is determined to be necessary, it will follow the first overhead byte and the first overhead byte will have all A bits set to one. If this explicit anchor is required, the location of the explicit anchor is coded as a binary count up from q(0) and is defined by the "E bits". The E bits are labeled as E1 through E8, with E1 being the least significant bit. The format of the explicit anchor byte is {E8,E7,E6,E5,E4,E3,E2,E1}.

For example, a three bit encoding anchored at Y[NEG](j) = q(124) would be encoded {0 1 1 0 0 1 0 1}. For example, a five bit encoding anchored at Y[NEG](j) = q(121) would be encoded {1 0 1 0 1 0 0 0}. For example, a seven bit encoding that requires an explicit anchor at level q(58) (00111010 in binary) would have the first overhead byte encoded {1 1 1 1 1 1 1 1} and a second overhead byte encoded {0 0 1 1 1 0 1 0}. Any eight-bit encoding is anchored, by default, at the most negative quantization level (i.e., q(0)) and will have the first byte encoded as {0 0 0 1 1 1 1 0}. It is interesting to note that the eight bit per sample encodings always expand the input G.711 frame of samples by one byte and that the first byte of an RGL frame will always be {0 0 0 1 1 1 1 0}. This fact will be exploited in the RTP payload format defined for use with the RGL codec [4].

If the encoding is a "zero bit" per sample coding (i.e., B(j)=0) we are finished. If the coding is not a "zero bit" per sample encoding (i.e., B(j) !=0), the remaining bytes of an encoded frame contain the Z bits. Each sample is encoded and sent by concatenating the appropriate number of Z bits (i.e., exactly B(j) bits), for every sample in frame j. We label the first sample in frame j as sample i and the last sample in a M sample frame as sample (i+M-1). For example, a three bit audio frame would be encoded:

{Z3(i) Z2(i) Z1(i) Z3(i+1) Z2(i+1) Z1(i+1) Z3(i+2) Z2(i+2)
Z1(i+2) ... Z3(i+M-1) Z2(i+M-1) Z1(i+M-1) }.

Pack these Z bits into bytes for transmission as a "RGL encoded" frame. If Z1(i+M-1) is not the least significant bit in the last byte, pad remaining bits in the last byte to zero.

Thus an encoded RGL frame consists of:

First overhead byte: {N3,N2,N1,A5,A4,A3,A2,A1}
(always present)

Explicit anchor byte: {E8,E7,E6,E5,E4,E3,E2,E1}
(only if all A bits in first
overhead byte are equal to 1)

The Z bits, packed into bytes per above: {Z[B(j)](i) ... Z1(i+M-1)}
(only if B(j) !=0)

Note that the Z bits may have zero padding after bit Z1(i+M-1), if required.

It is possible to calculate the length of the RGL encoded frame by knowing only two pieces of information: the length M (number of samples) of the G.711 input frame and the first overhead byte. Since different encoding systems can use different frame sizes, the frame size M must be passed to the RGL encoding function. This fact will be noted in the RTP payload format defined for use with the RGL codec [4]. A reference implementation of this encoding is provided as a C-language function at www.vovida.org [17].

4.4 RGL Frame Decoding

Determine from the first overhead byte the number of bits used in the encoding and if a second overhead byte containing an explicit anchor was used for this frame. If all A bits are ones (i.e., {A5,A4,A3,A2,A1}={1,1,1,1,1}), then the next byte is an explicit anchor byte. This byte represents the explicit anchor and was computed by counting in binary up from counting from the most negative quantization level (i.e., q(0)). Otherwise, the anchoring codepoint or the default anchor of q(0) for the B(j)=8 case is used as the anchor for this frame.

Determine the number of bits per sample by decoding the N bits (and the A bits, if necessary, to determine between zero or eight bit per sample encodings). If a zero bit per sample encoding was used, duplicate the 0-255 linear codepoint corresponding to the anchor for each sample in the entire frame. If a one or more bit per sample encoding was made, decode the Z bits for each sample in the frame and then add the anchoring code point quantization value to each sample to obtain the 0-255 linear quantization level for each sample in the frame. Finally, map the 0-255 linear codepoints to their A-law or mu-law counterparts, as appropriate.

Lastly, note that there are seven "reserved" first overhead byte combinations; those containing {A5,A4,A3,A2,A1} = {1 1 1 1 0} and {N3,N2,N1} != {0 0 0}. These seven "reserved" first overhead bytes

(0x3E, 0x5E, 0x7E, 0x9E, 0xBE and 0xFE) should never be generated by a RGL encoder (version 1.0.0 or later). Prudence dictates that if they are found in the decoding that the RGL decoder should indicate an error. A reference implementation of this decoding is provided as a C-language function at www.vovida.org [18] and has this and other error reporting mechanisms fully commented.

5. Analysis of RGL Codec

This section is an abbreviated version of an qualitative and quantitative analysis of the RGL codec that may be found at www.vovida.org [19].

Before presenting the following RGL codec compression estimates it is instructive to note some properties of the RGL codec. The first property is that Voice Activity Detection (VAD) is not recommended for use with the RGL codec as the RGL codec obtains high compression during periods of non-speech. As will shortly be noted, the average compression is primarily a function of the compression obtained during "non-speech" segments. Therefore the compression results are highly related to the level (power) of the so-called "background noise condition".

The following table presents average compression results using a speech corpus and methodology more fully described at www.vovida.org [20] (a small portion of the TIMIT database) and assumes various background noise conditions and Voice Activity Factors (VAFs).

The following average compression results assume the following.

- RGL codec compresses based on 10 millisecond G.711 frames.
- The voice activity factor is varied from 35% to 40% to 45% to 50%.
- "Worse Case Loudness" talkers and "Nominal Loudness" talkers.
- "Artificial Zero", "Near Zero", "Very low", "Low" and "Moderate" background noise conditions.

The 10 msec frame size is used because it is close to the optimum frame size during voiced periods of speech. Since most conversations only have one person speaking at a time, 50% is a reasonable real-world upper bound Voice Activity Factor (VAF). Most speech models assume a two state Markov (on-off) model with voice activity factor for speech bursts in the 40 to 45% range; however these models consider bursts as speech activity (energy) over a relatively long speech interval (e.g., 60 to 100 msec). As the RGL codec resolves to the frame size (in this case to 10 msec frames), a 45% VAF for most speech coders should equate to a lower "equivalent RGL" VAF. Thus a 40 to 45% "RGL VAF" is somewhat conservative compression assumption for an average compression estimate. The "worse case loudness" talker was modeled by concatenated speech with virtually no silence between utterances and the resultant speech scaled to the worst case G.711 input level (e.g., the absolute magnitude of the largest or smallest signal value was mapped to the highest or lowest G.711 quantization

level, respectively). A moderate talker is one that typically exercises one less bit per sample per frame than the "worse case loudness" talker (6dB lower, resulting in 12.5% more compression than the worse case talker). The "Artificial Zero" noise condition is representative for an IVR system in which the silence periods between voice prompts are represented by a single G.711 quantization level. As described previously, frames consisting of a single quantization level near "analog zero" are compressed to one byte (an 80 to 1 compression for 10 msec frames). Thus the "artificial zero" frames obtain 98.75% compression. This property makes this codec very attractive for IVR application where no loss of fidelity is desired. The noise conditions of "near zero", "very low", "low", "moderate" and "high" were simulated by white noise exciting approximating background noise levels of -52db, -46dB, -40dB, -33dB and -22dB relative to the maximum G.711 signal input level.

Table 4: Example RGL Average Compression Results

Talker Loudness	Background Noise	Voice Activity Factor 35%	40%	45%	50%
Loud	Artificial Zero	68.4%	64.0%	59.7%	55.4%
	Near Zero	44.4%	41.9%	39.4%	36.9%
	Very Low	36.3%	34.4%	32.5%	30.7%
	Low	28.2%	26.9%	25.7%	24.4%
	Moderate	19.4%	19.4%	18.8%	18.2%
Nominal	Artificial Zero	72.8%	69.0%	65.3%	61.6%
	Near Zero	48.8%	46.9%	45.0%	43.2%
	Very Low	40.7%	39.4%	38.2%	36.9%
	Low	32.5%	31.9%	31.3%	30.7%
	Moderate	24.4%	24.4%	24.4%	24.4%

This table demonstrates that the average compression percentages are largely influenced by the background noise condition. Forty percent average compression is possible for relatively low noise conditions, while low to mid-twenty percent is possible for moderate background noise conditions. Recall that the RGL codec is a lossless codec that reproduces non-speech (e.g., music on hold) with present G.711 fidelity and produces no fidelity loss under noise conditions that deteriorate the quality of most speech-only coders (lower MOS scores and lower speech intelligibility, see [11] or [12] for details). Note the high compression results for IVR applications (with low artificial background noise between prompts).

6. Future Extensions for RGL Codec

Other coding methods could be employed to reduce the number of bits per sample needed to represent the signals between Y[POS](j) to Y[NEG](j). An example would be difference encoding whereby the data is sifted to find the median or average difference between individual samples and this difference taken out in order to re-code the resulting samples. Often such a technique will further reduce the range of the "difference" signal allowing it to be coded in a fewer number of bits. The author has investigated a "brute-force" first-difference and has determined that this technique alone is not very effective in a mu-law or A-law encoding due to the nature of the companding (a large difference occurs in the linear q(.) domain when the signal passes through the inner segments). Typical increases in compression efficiency is in the 3% range and therefore do not justify the added complexity required (although small, it requires an additional array to perform the first difference and the corresponding arithmetic and comparison operations).

Yet another example would be variable bit length encoding that would exploit a histogram of the values between Y[POS](j) to Y[NEG](j) to obtain a more compact (in bits) representation of the signal span. Such techniques can be found in [13]. These techniques are obviously possible with the corresponding increase in complexity; however, the RGL codec has been designed for extremely low complexity. The author is working on a "first difference" scheme applied to the present methodology of the RGL codec that has a minor increase in complexity.

7. Changes from Previous Versions of RGL Codec

This draft describes the operation of the Version 1.0.0 RGL codec. The previous version (and only other version) was Version 0.1.0.

The reason the RGL codec was revised to Version 1.0.0 was to accommodate a simple RTP payload format for it. The newly proposed RTP payload format is described in companion draft draft-ietf-ramalho-rgl-rtpformat-01.txt [21]. To create a Table of Contents (TOC) required for some of the packetization formats proposed in the RTP payload document, the RGL codec was revised to Version 1.0.0 to create seven "reserved first RGL bytes". The seven reserved first overhead bytes were previously described in Section 4 (Figure 4) and are 0x3e, 0x5E, 0x7E, 0x9E, 0xBE, 0xDE and 0xFE. These reserved overhead codes were created by deleting an anchoring codepoint location at q(36) which was present in the previous version of the RGL codec.

The new, Version 1.0.0, RGL codec does not have q(36) as a potential anchoring codepoint location. This is not significant as this anchoring codepoint was not used for most (if not all) real-world signals (see discussion in the "Atypical Case Two" portion of Section 4.1 (Section 4.1) for an explanation of why this is so). If, by chance, the anchor codepoint at q(36) would have been used, the Version 1.0.0 RGL codec would instead use an explicit anchor to anchor at this location (at the cost of an additional byte in the RGL frame). Since no other anchoring codepoints were modified between RGL versions, all possible Version 1.0.0 RGL frames can be successfully decoded by an earlier version RGL decoder (as the earlier version would simply use the explicit anchor provided by the Version 1.0.0 encoder). Thus this modification was made in a manner that is backwardly compatible with earlier version RGL decoders. Additionally since the anchoring codepoint at q(36) was not used by any of the speech files tested, this modification was accomplished in such a way as to not affect the affect the earlier compression results in Section 5 (Section 5).

8. Open Source/IPR Issues

Reference implementations of the RGL encoder and the RGL decoder written as C-language functions can be found at www.vovida.org [22]. Use of this open source reference code is subject to the "Vovida Software License" terms found at http://www.vovida.org/About/license.html.

9. Security Considerations

The RTP payload format proposed for the RGL codec is described in draft-ramalho-rgl-rtpformat-01.txt [4]. The security considerations of using the RGL codec (with the RTP payload format) is described in that document. This document simply describes the operation of the RGL codec.

10. IANA considerations

As described in the companion RTP payload format document draft-ramalho-rgl-rtpformat-01.txt [4], it is RECOMMENDED that this codec be referred to as: RGLv1 (v1 for version 1).

When and if the RGL codec becomes mainstream, IANA registration may be necessary.

11. End Notes

[EndNote:1] Many PSTN/GSTN telephony systems use "robbed-bit" signaling, leaving only 56 kbps of the 64kbps channel transported reliably through the network. V.90 systems need access 64kbps channel and determine, using framing heuristics, the 56 kbps subset of the "reliable" bandwidth.

[EndNote:2] This need is sometimes driven by the VoIP transport provider's desire to state to a governmental agency (e.g., a public utility commission) that they transport voice using the existing PSTN standards for voice quality.

[EndNote:3] This is due to the facts that: 1) speech is typically present only in one direction of transmission, and 2) the speech (usually a "hello" or similar utterance) does not occupy the entire initial greeting period.

[EndNote:4] See Figure 5.10, page 241 of for a graph showing SNRs versus input signal level for both single-frequency sinusoidal and Gaussian input characteristics.

[EndNote:5] To be precise, A-law uses "seven" segments on either side of analog zero; the segment closest to zero is a "double width" segment (representing 32 quantization levels, whereas the other segments represent only 16 levels). For the purposes of this explanation, A-law can be thought of as having eight "16 quantization level" segments, with the knowledge that the "double width" segment near zero has the same slope.

[EndNote:6] A similar ones-density requirement existed for E1 systems. Thus the "A 78.56" codebook design was, with a different technique, designed to have similar ends (i.e., high ones-density for low amplitude signals).

[EndNote:7] For A-Law encoding, the "character bits" (bits to be sent) are obtained by inverting the even bits in this column (bits in this column are labeled 1,2 .. 8 from left to right). See the ITU-T G.711 standard for more information.

References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[4] Ramalho, M., "RTP Payload Format for RGL Codec", draft-ietf-ramalho-rgl-rtpformat-01.txt (work in progress), February 2003.

[5] Jayant, N., "Digital Coding of Waveforms", Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-211913-7, 1984.

[6] Haykin, S., "Digital Communications", John Wiley & Sons, New York, NY, ISBN 0-471-62947-2, 1988.

[7] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[8] Schulzrinne, H., "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996.

[9] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[10] Baugher, M., Blom, R., Carrara, E., McGrew, D., Naslund, M., Noorman, K. and D. Oran, "The Secure Real-TIme Transport Protocol", draft-ietf-avt-srtp-05.txt (work in progress), June 2002.

[11] Deller, R., Proakis, J. and J. Hansen, "Discrete-Time Processing of Speech Signals", Macmillan, New York, NY, ISBN 0-02-328301-7, 1993.

[12] Syrdal, A., Bennett, R. and S. Greenspan, "Applied Speech Technology", CRC Press, Boca Ratton, FL, ISBN 0-8493-9456-2, 1995.

[13] Hans, M. and R. Schafer, "Lossless Compression of Digital Audio", IEEE Signal Processing Magazine, IEEE Signal Processing Society, July 2001.

[14] <http://www.vovida.org>

[15] <http://www.vovida.org>

[16] <RGL_Format>

[17] <http://www.vovida.org>

[18] <http://www.vovida.org>

[19] <http://www.vovida.org>

[20] <http://www.vovida.org>

[21] <RGL_Format>

[22] <http://www.vovida.org>

Author's Address

Michael A. Ramalho
Cisco Systems, Inc.
1802 Rue de la Porte
Wall Township, NJ 07719-3784
USA

Phone: +1.941.708.4650
EMail: mramalho@cisco.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.