Network Working Group
INTERNET-DRAFT
Expires: May 2002

Richard Price, Siemens/Roke Manor
Robert Hancock, Siemens/Roke Manor
Stephen McCann, Siemens/Roke Manor
Mark A West, Siemens/Roke Manor
Abigail Surtees, Siemens/Roke Manor
Paul Ollis, Siemens/Roke Manor

21 November, 2001

Universal Decompressor Tokens for EPIC
<draft-price-rohc-signaling-epic-01.txt>

Status of this Memo

This document is a submission to the IETF ROHC WG. Comments should be directed to the mailing list of ROHC, rohc@cdt.luth.se.

Abstract

This draft describes a set of tokens for the Universal Decompression Algorithm. The new token set allows the universal decompressor to understand compressed messages generated by EPIC.

Table of contents

1. Introduction

This draft describes a set of tokens for the Universal Decompression Algorithm [UNIDEC]. These tokens configure the universal decompressor to understand compressed messages generated by the Efficient Protocol Independent Compression [EPIC] scheme.

The EPIC scheme is designed to provide efficient compression of arbitrary protocol stacks. EPIC takes as its input a BNF (Backus Naur Form) description of the protocol to be compressed, which is stored at the compressor and decompressor and used as part of the compression process. Since EPIC is pre-programmed with knowledge of how the protocols behave, the compression ratio obtained is very high and the processing and memory requirements are low.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

3. Overview of BNF

The basic idea of EPIC is to provide a BNF (Backus Naur Form) description of the relevant protocol stack at the compressor and decompressor. BNF is a "metasyntax" commonly used to describe the syntax of protocols and languages. An example BNF description taken from [SIP] is given below:

```
host           =     <IPv4address> | <hostname>

IPv4address    =     1*<digit> "." 1*<digit> "." 1*<digit> "."
                     1*<digit>

hostname       =     *(<domainlabel> "." ) <toplabel> [ "." ]

domainlabel    =     *<alphanum>

toplabel       =     *<alpha>

alphanum       =     <alpha> | <digit>

alpha          =     <lowalpha> | <upalpha>

upalpha        =     "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
                     "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
                     "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
                     "Y" | "Z"

lowalpha       =     "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                     "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                     "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
                     "y" | "z"

digit          =     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                     "8" | "9"
```

Figure 1: Example BNF description of a protocol

All versions of the BNF metasyntax are based around the concept of a "BNF rule". Each BNF rule describes the syntax of part of the protocol stack, and is defined in terms of existing BNF rules. The "top-level" BNF rule describes the entire protocol or protocol stack.

The following basic constructs are used to define new BNF rules:

BNF_rule = ...	Defines a new BNF rule in terms of existing BNF rules

<BNF_rule>	Reference to an existing BNF rule
<a> | <b> ... | <z>	Choice of different BNF rules

For example, a new BNF rule for an alphanumeric character can be defined in terms of the BNF rules for letters and digits as follows:

```
alphanum      =     <alpha> | <digit>
```

Clearly new BNF rules cannot be defined in terms of existing BNF rules ad infinitum, so one or more "fundamental" BNF rules must also be provided. For example, many variants of BNF allow new BNF rules to be specified in terms of ASCII text as illustrated below:

```
digit         =     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                    "8" | "9"
```

4. Overview of EPIC

As explained above, the EPIC compressor and decompressor each contain a BNF description of the protocol stack to be compressed. At the EPIC decompressor this BNF description is parsed to reconstruct the uncompressed message. The information required in the compressed message is simply a set of instructions on the next BNF rule to parse when more than one choice is available. For example, consider the simple protocol defined by the following BNF:

```
hello_message  =     "Hello " <name>

name           =     "Tom" | "Dick" | "Harry" | "Sally"
```

Figure 2: Simple protocol described using BNF

When reconstructing a hello message generated using the above BNF, all that the decompressor needs to know is the correct choice of name to insert after the "Hello " text. Therefore each message can be compressed down to just 2 bits.

A number of variants exist on the basic BNF metasyntax, each offering different fundamental BNF rules. For example the Augmented BNF described in [RFC-2234] and used in [SIP] offers a selection of fundamental rules including the following:

"string"	String of ASCII characters
[<ABNF_rule>]	Optional ABNF rule
x*y <ABNF rule>	List of between x and y occurrences of ABNF_rule. If x is omitted then x = 0, and if y is omitted then y = infinity

The variant of BNF described in [EPIC] contains a library of 15 fundamental BNF rules, offering the standard rules such as OPTIONAL

and LIST, as well as some additional compression-specific rules such as INFERRED.

EPIC can compress a protocol described using any variant of BNF, provided that the actions to take for each fundamental BNF rule have been defined. This draft describes how to compress protocols defined using ABNF as per [RFC-2234], as this covers a wide range of protocols including [SIP].

5. Converting ABNF into universal decompressor tokens

The universal decompressor described in [UNIDEC] can be programmed to understand the compressed messages generated by a wide range of compression algorithms. This is accomplished by sending an appropriate set of tokens to the decompressor before compression begins (for example the tokens can be appended to the front of the first compressed message).

[UNIDEC] offers a simple mnemonic language that can be used to describe new decompression algorithms. The set of tokens required for decompression of EPIC messages will be described using this mnemonic language.

The base set of tokens required to implement EPIC is given below:

```
:first_token            main
:uncompressed_start     2048
:uncompressed_end       2048
:circular_buffer        2048
:compressed_start       compressed_message_start
:compressed_end         0
:compressed_pointer     0
:stack_free             0
:stack[0]               0
:stack[1]               0
:stack[2]               0
    :                   :
:stack[**depth**]       0

:bit_offset             0
:index                  0

:main

CALL (**top_level_rule**)
COMPARE (0, 0, 0, 0, 0)

; Additional tokens inserted here

:compressed_message_start
```

```
   ; The first compressed message starts here
```

Note that any text enclosed in ** ** is not part of the mnemonic language, but instead must be taken from the ABNF description itself. For example, in the above set of tokens the text **top_level_rule** must be replaced by the name of the top level ABNF rule (in other words, the ABNF rule which describes the entire protocol or protocol stack to be compressed).

The remaining tokens required to program the universal decompressor are derived directly from the ABNF description of the protocol to be decompressed. Exactly one subroutine will be provided for every ABNF rule.

Just as ABNF rules are defined in terms of simpler rules, the subroutines corresponding to high-level ABNF rules will call subroutines for lower-level rules using the CALL token. Consequently, the amount of stack space that must be reserved in the [UNIDEC] byte buffer is equal to the maximum depth of the ABNF tree. In the above set of tokens, the **depth** parameter should be replaced by this value.

For simplicity, it is assumed that every ABNF rule used to describe the protocol takes one of the following forms:

```
and_rule       =     <rule_1> <rule_2> ... <rule_n>

or_rule        =     <rule_1> | <rule_2> | ... | <rule_n>

text_rule      =     "string"

optional_rule  =     [<rule>]

list_rule      =     x*y (<rule>)
```

If a particular ABNF rule does not fall into one of the above categories then it must be broken down into a set of simpler ABNF rules. For example, consider the following rule:

```
hostname       =     *(<domainlabel> "." ) <toplabel> [ "." ]
```

This rule can be rewritten as the following five simpler rules:

```
hostname       =     <dummy_rule_1> <toplabel> <dummy_rule_2>

dummy_rule_1   =     *<dummy_rule_3>

dummy_rule_2   =     [<dummy_rule_4>]

dummy_rule_3   =     <domainlabel> <dummy_rule_4>

dummy_rule_4   =     "."
```

Once the BNF has been rewritten so that all rules take one of the above five forms, each BNF rule is then converted into a set of universal decompressor tokens as follows:

5.1. ABNF "and" rule

ABNF description: and_rule = <rule_1> <rule_2> ... <rule_n>

Each instance of an "and" rule is replaced by the following set of tokens:

```
:**and_rule**

CALL (**rule_1**)
CALL (**rule_2**)
  :     :
CALL (**rule_n**)
RETURN
```

Note that as explained previously, any text marked with ** ** is taken from the ABNF rule being replaced. For example:

<byte> = <hex_digit> <hex_digit>

This ABNF rule is replaced by the following tokens:

```
:byte

CALL (hex_digit)
CALL (hex_digit)
RETURN
```

5.2. ABNF "or" rule

ABNF description: or_rule = <rule_1> | <rule_2> | ... | <rule_n>

To find the correct set of tokens for an ABNF rule of the above form, first let k = ceiling(log2(n)). The correct set of tokens to use is then:

```
:**or_rule**

HUFFMAN ($compressed_pointer, $bit_offset, index, 1, **k**, 0)
SWITCH ($index, **rule_1**, **rule_2**, ... , **rule_n**)
```

As an example, consider the following instance of an "or" rule:

host = <IPv4address> | <hostname>

In the universal decompressor mnemonic language this becomes:

```
:host

HUFFMAN ($compressed_pointer, $bit_offset, index, 1, 1, 0)
```

```
SWITCH ($index, IPv4address, hostname)
```

5.3. ABNF "text" rule

ABNF description: text_rule = "string"

Let k denote the number of characters in the text string. Also let string_bytes denote the set of ASCII bytes corresponding to the text string (specified as decimal integers separated by whitespace). The correct set of tokens to use is given below:

```
:**text_rule**

COPY-LITERAL (**text_rule**_text, **k**, $uncompressed_end)
RETURN
:**text_rule**_text .byte **string_bytes** .short
```

For example, consider the following ABNF rule:

```
name        =     "Tom"
```

This ABNF rule is replaced by the following tokens:

```
:name

COPY-LITERAL (name_text, 3, $uncompressed_end)
RETURN
:name_text          .byte 84 111 109 .short
```

5.4. ABNF "optional" rule

ABNF description: optional_rule = [<rule>]

The ABNF "optional" rule is replaced by the following set of tokens:

```
:**optional_rule**

HUFFMAN ($compressed_pointer, $bit_offset, index, 1, 1, 0)
SWITCH ($index, **rule**, **optional_rule**_not_present)
:**optional_rule**_not_present
RETURN
```

5.5. ABNF "list" rule

ABNF description: list_rule = x*y (<rule>)

Let k = ceiling(log2(y - x)). The above ABNF rule is then replaced by the following set of tokens:

```
:**list_rule**_start

ADD ($**list_rule**_counter, 1)
```

```
   CALL (rule)

   :**list_rule**
```

```
HUFFMAN ($compressed_pointer, $bit_offset, **list_rule**_counter, 1,
         **k**, **x**)
COMPARE ($**list_rule**_counter, **y**, **list_rule**_start,
         **list_rule**_start, **list_rule**_end)

:**list_rule**_end
RETURN
```

Once each of the ABNF rules has been replaced by the correct set of tokens, downloading these tokens to the universal decompressor will configure it to understand EPIC compressed messages.

6. Worked example for a simple BNF description

This chapter lists the set of universal decompressor tokens corresponding to the simple BNF description of Figure 2. For simplicity the tokens are given in the [UNIDEC] mnemonic language, although an automated "BNF-to-tokens" converter would typically output the tokens directly as a byte stream.

The first step is to split up the two specified BNF rules into simpler rules which fall into the five categories of Chapter 5. This is accomplished as follows:

```
hello_message  =     <rule_1> <name>

rule_1         =     "Hello "

name           =     <rule_2> | <rule_3> | <rule_4> | <rule_5>

rule_2         =     "Tom"

rule_3         =     "Dick"

rule_4         =     "Harry"

rule_5         =     "Sally"
```

The corresponding set of tokens that must be downloaded to the universal decompressor is given below:

```
:first_token            main
:uncompressed_start     2048
:uncompressed_end       2048
:circular_buffer        2048
:compressed_start       compressed_message_start
:compressed_end         0
:compressed_pointer     0
:stack_free
:stack[0]
```

```
   :stack[1]

   :bit_offset             0
   :index                  0
```

```
   :main

   CALL (hello_message)
   COMPARE (0, 0, 0, 0, 0)

   :hello_message

   CALL (rule_1)
   CALL (name)
   RETURN

   :rule_1

   COPY-LITERAL (rule_1_text, 6, $uncompressed_end)
   RETURN
   :rule_1_text         .byte 72 101 108 108 111 32 .short

   :name

   HUFFMAN ($compressed_pointer, $bit_offset, index, 1, 2, 0)
   SWITCH ($index, rule_2, rule_3, rule_4, rule_5)

   :rule_2

   COPY-LITERAL (rule_2_text, 3, $uncompressed_end)
   RETURN
   :rule_2_text         .byte 84 111 109 .short

   :rule_3

   COPY-LITERAL (rule_3_text, 4, $uncompressed_end)
   RETURN
   :rule_3_text         .byte 68 105 99 107 .short

   :rule_4

   COPY-LITERAL (rule_4_text, 5, $uncompressed_end)
   RETURN
   :rule_4_text         .byte 72 97 114 114 121 .short

   :rule_5

   COPY-LITERAL (rule_5_text, 5, $uncompressed_end)
   RETURN
   :rule_5_text         .byte 83 97 108 108 121 .short

   :compressed_message_start

   ; Insert the first compressed message here
```

7. Further enhancements

This draft has presented a simple version of the [EPIC] protocol compression scheme.

The following sections discuss a number of further enhancements that can be made to improve the efficiency of this base scheme.

7.1. Huffman encoding

For each of the BNF rules that require some information to be sent in the compressed header (this includes the "or" rule, "optional" rule and "list" rule), a HUFFMAN token is provided to decompress this information.

If n different choices are available, the compressed message currently includes log2(n) bits that are interpreted as an integer from 0 to n - 1.

To reduce the average number of bits required to transmit a choice to the decompressor, it is possible to take into account the probability that each choice will be used in practice. For example:

```
name          =     "Tom" | "Dick" | "Harry" | "Sally"
```

If all four names occur on a roughly equal basis then it is most efficient to communicate the choice of name as a 2-bit integer. However if the name "Tom" occurs 95% of the time then the following encoding will give better efficiency:

Uncompressed text:	Compressed bits:
"Tom"	0
"Dick"	10
"Harry"	110
"Sally"	111

The probability that each choice in a BNF description will be used can be discovered by applying the scheme to a selection of messages. The number of times that each choice is used is recorded, and the results are scaled to give the necessary probabilities.

Note that the selection of messages provided to EPIC in this "learning" phase should reflect as accurately as possible the mix of messages that will be compressed. More accurate probability values give a higher compression ratio when the compression scheme is applied.

The resulting set of probability values can then be used to build a "Huffman tree" as described in [HUFF]. This tree indicates how to

encode each of the choices in an optimally efficient manner, with more common choices communicated to the decompressor using fewer bits than rare choices. The [UNIDEC] draft describes how to convert a

given Huffman tree into parameters for the HUFFMAN token so that it can be downloaded to the decompressor.

7.2. Additional BNF rules

This draft provides a set of [UNIDEC] tokens to decompress a protocol defined using ABNF. The advantage of supporting ABNF is that it is the variant of BNF used to describe many text-based protocols including [SIP]. This means that the ABNF description of SIP can be converted directly into a set of tokens as explained above.

The [EPIC] draft provides a library of additional "fundamental" BNF rules that can be used to describe protocols to be compressed in greater detail.

For example, the version of BNF defined in [EPIC] provides rules for sequence numbers (which increase by 1 for each consecutive message), length fields (which contain the length of the message) and so on. This improves the compression ratio because these values can typically be inferred at the decompressor rather than being sent in the compressed message.

Consequently, any protocol described using the [EPIC] version of BNF will be compressed more aggressively than a protocol described using ABNF. Future versions of the draft will include additional BNF rules to improve the compression ratio in this manner.

7.3. Dynamic compression

The version of EPIC currently described in this draft can only perform individual message compression (i.e. messages cannot be compressed dynamically relative to previously sent messages).

A future version of the draft will describe the tokens needed to support dynamic compression.

7.4. Generic compression

For a given BNF description, the version of EPIC in this draft can compress any message generated by the BNF. Note however that it cannot compress messages that are illegal in the BNF it is currently using.

The simplest way to ensure that EPIC can compress arbitrary messages is to download an additional set of tokens for generic message compression to the universal decompressor. For example, sets of tokens for LZ77, [DEFLATE] and [LZW] decompression are provided in [UNIDEC]. These tokens can then be invoked as a "backup" to ensure a reasonable level of compression even for messages that are illegal

in the BNF provided to EPIC.

A more advanced method for combining EPIC with generic message compression is to provide a BNF rule for generic compression as per

Section 7.2. This rule can then be invoked for any part of the message which contains freeform text, and consequently would benefit from generic compression techniques.

8. Security Considerations

This draft describes a set of tokens for direct use in the Universal Decompression Algorithm. Consequently the security considerations for this draft match those of [UNIDEC].

9. Acknowledgements

Header compression schemes from [ROHC] have been important sources of ideas and knowledge. Basic Huffman encoding [HUFF] was enhanced for the specific tasks of robust, efficient message compression.

Thanks to

Carsten Bormann (cabo@tzi.org)
Christian Schmidt (christian.schmidt@icn.siemens.de)
Max Riegel (maximilian.riegel@icn.siemens.de)
David Keogh (david.keogh@roke.co.uk)
Lawrence Conroy (lwc@roke.co.uk)

for valuable input and review.

10. Intellectual Property Considerations

This proposal in is full conformity with [RFC-2026].

Siemens may have patent rights on technology described in this document which employees of Siemens contribute for use in IETF standards discussions. In relation to any IETF standard incorporating any such technology, Siemens hereby agrees to license on fair, reasonable and non-discriminatory terms, based on reciprocity, any patent claims it owns covering such technology, to the extent such technology is essential to comply with such standard.

11. References

[UNIDEC] "Universal Decompression Algorithm", R. Price et. al., <draft-ietf-rohc-sigcomp-algorithm-00.txt>, Internet Engineering Task Force, November 14, 2001

[ROHC] "RObust Header Compression (ROHC)", Carsten Bormann et al, RFC3095, Internet Engineering Task Force, July 2001

[EPIC] "Framework for EPIC-LITE", Richard Price et al, <draft-ietf-rohc-epic-lite-00.txt>, Internet Engineering Task Force, October 23, 2001

[SIP] "SIP: Session Initiation Protocol", Handley et al, RFC2543, Internet Engineering Task Force, March 1999

[FLOWS] "SIP Call Flow Examples", Alan Johnston et al, <draft-ietf-sip-call-flows-05.txt>, Internet Engineering Task Force, June 2001

[HUFF] "The Data Compression Book", Mark Nelson and Jean-Loup Gailly, M&T Books, 1995

[DEFLATE] "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, P. Deutsch, May 1996

[LZW] "LZW Data Compression", Mark Nelson, Dr. Dobb's Journal, October 1989

[RFC-2026] "The Internet Standards Process - Revision 3", Scott Bradner, Internet Engineering Task Force, October 1996

[RFC-2119] "Key words for use in RFCs to Indicate Requirement Levels", Scott Bradner, Internet Engineering Task Force, March 1997

[RFC-2234] "Augmented BNF for Syntax Specifications: ABNF", Crocker et al, RFC2234, Internet Engineering Task Force, November 1997

12. Authors' Addresses

Richard Price Tel: +44 1794 833681
Email: richard.price@roke.co.uk

Robert Hancock Tel: +44 1794 833601
Email: robert.hancock@roke.co.uk

Stephen McCann Tel: +44 1794 833341
Email: stephen.mccann@roke.co.uk

Mark A West Tel: +44 1794 833311
Email: mark.a.west@roke.co.uk

Abigail Surtees Tel: +44 1794 833131
Email: abigail.surtees@roke.co.uk

Paul Ollis Tel: +44 1794 833168
Email: paul.ollis@roke.co.uk

Roke Manor Research Ltd
Romsey, Hants, SO51 0ZN
United Kingdom