Packet header compression formal notation requirements

Julije Ozegovic
OPAL COMPUTING d.o.o.
Split, Croatia
E-mail: julije.ozegovic@vest.hr

Abstract: **Robust header compression (ROHC) [1] is recently developed to be used in wireless last hop links of global Internet. It is built around an extensible core framework that can be tailored to compress new protocol stacks by adding additional ROHC profiles. The role of formal notation is to provide a simple method for ROHC profile specification. It is to be used in the phase of profile refinement to achieve maximal compression ratio possible, as well as in the phase of software implementation to provide unambiguous profile specification. In this paper, requirements expected to be achieved with profile formal notation system for ROHC are elaborated.**

1. INTRODUCTION

Header compression exploits the strict format of header fields to enhance the transport of Internet packets over wireless links. In large proportion of flows that utilize short packets, like IP/UDP/RTP headers that can waste up to 75% of bandwidth, header compression is sufficient.

The ROHC working group effort to provide robust header compression framework resulted in state machine and profiles specified using "RFC box notation". This is actually a simple graphical representation with English explanation added where needed. The profile refinement and specification clarity were achieved by implementing changes in English text.

Alternative solution for profile specification was introduced in Efficient Protocol Independent Compression (EPIC-Lite) [2] on the basis of Bachus-Naur form (BNF) [3]. The proposed BNF formal language induced lot of discussion because of its stack oriented functionality. In proposed two stack model, particular field processing is postponed by pushing its value to the "control stack".

The authors of EPIC introduced "A Formal Notation for Header Compression" [4] where explicit stack manipulations are avoided with the "LABEL" concept. Instead of using second stack, postponed field value is stored in a variable defined by LABEL method. This way, particular field processing can be postponed to the more appropriate time.

Simultaneously, alternative "Generic Header Compression Notation for ROHC" [5] was proposed with its hierarchical header extension of EPIC concept. Uncompressed header structure definition is separated from actual compression.

Based on ROHC WG discussions, improved version of [4], the "Protocol Enabled BNF-Based Language (PEBBLE)" [6] is published to be the official ROHC starting point for formal notation development.

This work resulted in "Formal Notation for Robust Header Compression(ROHC-FN)" [7], which served the role of template text for San Francisco IETF-56 formal notation discussions. IETF-56 was the place where a necessity for "Formal notation requirements" was recognized [8].

This paper is organized as follows. In Chapter two, the scope of ROHC formal notation is given. In Chapter three, the formal notation requirements are elaborated. In Chapter four, the notation extensibility is mentioned. Conclusions and future work are given in Chapter six.

2. SCOPE OF ROHC FORMAL NOTATION

The ROHC formal notation is developed to ease completion of the following tasks and goals

1. Profile writing
2. Profile refinement
3. Profile human readability
4. Profile compactness
5. Profile machine readability
6. Profile validation
7. Profile clarity
8. Hand written code generation
9. Machine automated code generation
10. Profile interpretation

2.1 Human requirements

From the above, human ROHC user can benefit of human readability, clarity and compactness for profile writing and refinement, and for hand written code generation.

2.2 Machine requirements

Machine ROHC processing can benefit of machine readability for profile validation, automated code generation and profile interpretation.

2.3 Profile interpretation

Profile interpretation is a method of profile execution which enables the profile to be downloaded to compressor and decompressor. This way, new profiles can be introduced without need to download new compressor and decompressor software.

In profile interpretation environment, processing is organized in two phases:

1. Offline processing - profile interpretation to build necessary data structures and parameters
2. Online processing - profile usage for compression and decompression

To optimize offline processing, profile can be enhanced with preprocessed parameters.

Opposite to the profile interpretation is (hand or machine) profile implementation in code. This way, new profiles can be introduced by installation of new compressor and decompressor software.

3. FORMAL NOTATION REQUIREMENTS

The scope of formal notation can be achieved if formal notation system developed can be used for the following tasks:

1. Specification of packet classification parameters
2. Specification of uncompressed header structure
3. Specification of header format
4. Specification of compressed header structure
5. Specification of context management
6. Specification of notation extensions

3.1 Packet classification parameters

Packet classification procedure is performed on uncompressed packet header to achieve two goals:

1. Recognize protocol stack (profile) to which packet belongs
2. Recognize packet flow (context) to which packet belongs

Packet classification is currently considered to be implementation specific. This situation is acceptable, unless formal notation system is intended for profile interpretation.

In profile interpreting systems, the main achievement is the ability to download a new profile in machine readable form to the compressor and decompressor, instead of installing the new software with new profile hard coded.

In such a system, it can be foreseen that new profile can support a totally new, previously unsupported protocol stack. In this case, it is essential that formally specified profile submits enough information to recognize a new type of header structure.

The packet classification actually takes place on compressor side. Under ROHC framework, decompressor uniquely classifies compressed packets using Context Identifier (CID).

3.1.1. Profile selection

To recognize protocol stack, some information from or for link layer can be used. Different link layer protocols can use different codes. To support multiple link subsystems, protocol ID codes must be specified. The following example is one of possible solutions:

MEDIA = Ethernet

Protocol_ID = <link layer protocol ID for actual protocol>

MEDIA = PPP

Protocol_ID = <link layer protocol ID for actual protocol>

To verify protocol stack, some information can be checked inside the uncompressed header, e.g. version and protocol fields to recognize IPv4/TCP stack. For example, failure of VALUE encoding in IR packet can indicate wrong profile selection. Alternatively profile checking can be defined explicitly:

Profile_check = <list of fields and values to be checked>

3.1.2. Context selection

After the profile classification, context selection is performed.

To recognize packet flow (context), some fields inside the uncompressed header must be checked against the existing contexts for profile selected. If a match is found among the existing contexts, packet is assumed to belong to already established context (and state machine). Otherwise, a new context is created and profile is negotiated with decompressor.

Context selection parameters specify packet flow recognition granularity. Fine granularity can be achieved using e.g.:

Context_check = <sadr dadr sport dport>

while coarse granularity is possible using e.g.:

Context_check = <sadr dadr>

Flow selection granularity is a method for context reusability control.

3.2 Profile specification

The profile specification consists of three different parts:

1. Uncompressed header structure specification
2. Header format specification
3. Compressed header structure specification

The three are not totally independent and need to be interrelated carefully. The readability and compactness as well as flexibility of profile writing depends on proper notation balance.

3.2.1. Profile specification degrees of freedom

The profile writing can be seen as a process with several degrees of freedom. Profile writer can ask for maximal freedom, i.e. to specify uncompressed header structure, header format and compressed header structure independently.

However, not all that freedom is needed. Especially, it is not needed to reorder compressed header fields, because it is natural to decompress the header in reverse order than the compression was done. Another problem is the achievement of decompressablility for every compressed packet.

The following degrees of freedom are possible:

3.2.1.1. No degrees of freedom

System reads, compresses and packs fields as ordered in uncompressed header. This system can not yield optimal compression, because original field ordering for particular protocol stack is not designed for optimal compression.

3.2.1.2. Half degree of freedom on input

System reads, compresses and packs fields as ordered in the uncompressed header, but can postpone processing of particular field. This can yield optimal compression. Redundant information is present in profile. Systems like EPIC-Lite, PEBBLE and ROHC_FN belong to this category.

3.2.1.3. One degree of freedom on input with read on demand

System reads fields randomly when needed by compressor. This makes it possible to compress and pack fields in optimal order. The actual format is evaluated at the compression time, so there is no additional information exchange between the field reading and compressing processes.

3.2.1.4. One degree of freedom on input with advance read

System parses uncompressed headers to fields in advance. This makes it possible to compress and pack fields in optimal order. The actual format is partly evaluated in advance, so complex information exchange between the field reading and compressing processes takes place.

3.2.1.5. One degree of freedom on output

System reads and compresses fields as ordered in the uncompressed header, but packs compressed fields in arbitrary order. This system can not yield optimal compression, because original field ordering for particular protocol stack is not designed for optimal compression.

It must be noted that freedom on output deals with compressed fields ordering only. The placement of indicator bits can be independent of field ordering.

3.2.1.6. Some freedom on input and one degree of freedom on output

This system combines freedom on input and output. However, it is natural to decompress the compressed header in the reverse order than compression was done. Freedom on output is not absolutely needed.

From the analysis above, it can be evaluated that systems with half or with full freedom on input and read on demand are the ones acceptable.

3.2.2. Notation with field processing postponement

In systems with half degree freedom on input, fields are read as ordered in uncompressed header and then postponed for later processing (EPIC-Lite, ROHC_FN). System can be used to achieve optimal compression.

The good property of postponement system is its simplicity. The uncompressed header structure and header format can be specified simultaneously or separately.

3.2.2.1. Field postponement and simultaneous specification

In simultaneous structure and format specification, header fields are not explicitly named, and human readability suffers. To improve readability, profile writers tend to use compression functions and name them by field names. Such functions are declared later in the profile. In complex profiles like IP/TCP, function declarations are far away from the place of usage, and profile is actually human unreadable.

Another drawback in simultaneous specification is redundancy of parameters. The length of the same field is declared in all methods related to this field. For example:

```
header = function1
         function2
---------------------------------
function1 = method11(length) | method12(length)
```

where "length" is actually declared two times. More elaborate examples can be found in published EPIC-lite [2] profiles.

3.2.2.2. Field postponement and separate specification

In separate structure and format specification, header fields are explicitly named with lengths (in bits) declared. Later, field value can be compressed immediately or postponed for later compression. In the following example, some_field1 is read and compressed immediately, and some_field2 is postponed for later compression and possible multiple usage:

```
some_field1 = FIELD(length1)
some_field2 = FIELD(length2)
---------------------------------
header = some_field1 method11 | method12
         POSTPONE some_field2
---------------------------------
         some_field2 method21 | method212
```

This way, most header field compressions will be declared inline.

Decompressor can be made intelligent enough to know whether to store uncompressed value to local variable, to use it as a parameter, or to restore it to the uncompressed header field (exactly like in LABEL concept).

However, to increase readability and to make the parser simpler, explicit label usage methods can be introduced:

```
some_field1 = FIELD(length1)
some_field2 = FIELD(length2)
---------------------------------
header = some_field1 method11 | method12
         POSTPONE some_field2
---------------------------------
         method(some_field2)
         ACTIVATE some_field2 method21 | method212
```

In this example, POSTPONE method is used to indicate that the field value is to be stored in local variable when compressing, and restored from local variable when decompressing. ACTIVATE method is used to submit the value from local variable to standard parameter path for compression, and to store decompressed value to local variable after decompression.

Among the POSTPONE - ACTIVATE pair of methods, field value can be used as an input parameter for any appropriate method at will, and can be changed if needed. The functional parameter passing:

```
method(some_field2)
```

is optimal, because compression and decompression behavior is internal to the method invoked.

3.2.3. Notation with random access fields and read on demand

In systems with one degree freedom on input and read on demand, fields lengths and positions are declared and named in advance, but actual contents read takes place at the moment of compression. System can be used to achieve optimal compression.

The good property of random access is that advanced field declaration and naming avoids redundancy and improves readability However, profile must include separate declaration of all fields. Problem can be encountered with optional fields, whose positions can vary. These fields can not be freely accessed, but only within the option to which they belong. Their position is declared notrelative to the beginning of the header, but relative to the beginning of the option. For example:

```
some_field = FIELD(length, offset)
---------------------------------
header = some_field method1 | method2
---------------------------------
```

The "offset" parameter can be declared explicitly, or implicitly as a sum of previously declared fields. This way, most header field compressions will be declared inline, and only complex ones will need function declarations. Actual field contents read is performed when "some_field" is executed, and data together with length parameter are transferred to compression methods. Optional fields can be read inside the option processing structures.

Random access fields usage assumes knowledge of uncompressed header length at the decompression time. The issue of variable length options must be considered appropriately. It can be foreseen that profile writer should take care of option length communication to the decompressor, when option length is not available from the format used.

3.3. Compressed header structure

Compressed header structure consists of two specifications:

1. Order of compressed fields
2. Order of indicator bits

3.3.1. Order of compressed fields

The order of compressed fields is optimal when it is equal to the order of compression.

3.3.2. Order of indicator bits

Order of indicator bits depends on model of compressed packet chosen. At the moment, several models are proposed: ROHC3095, Ordinary Huffman, Hierarchical Huffman etc. It is possible that new models emerge in the future. At IETF-56 it was proposed to define compressed packet structure in profile using syntax like:

TCP_IP_PACKET = ROHC3095(TCP_IP_function)

This approach makes it possible to use combinations of structures

TCP_IP_PACKET = Ordinary_Huffman(IP_function)
Ordinary_Huffman(TCP_function)

or

TCP_IP_PACKET = ROHC3095 (IP_function)
Ordinary_Huffman(TCP_function)

Finally, ageneric profiles are possible:

PACKET = Ordinary_Huffman(IPV4_function | IPV6_function)
Ordinary_Huffman(TCP_function | UDP_function| SCTP_function)
Ordinary_Huffman(RTP_function | Null)

In generic profile, packet classification takes place on layer by layer basis.

The point of communication between the header format choice and compressed header packing is a list of field compression choices that accompanies the list of compressed fields. List of choices can be used to encode header format (like in Huffman) or to generate ROHC3095 compressed

3.3.3. Profile preprocessing

In complex indicator bits coding system, like Huffman is, lot of processing is needed in the offline phase of the interpreting system, or during the code generation of the conventional (hard coded) system. This processing is mainly concerned with mapping between a list of field compression choices and actual indicator bits code words. This mapping is unique for the profile. It is necessary to calculate it only once, after the profile is standardized, and before it is used (interpreted or hard coded).

It can be of benefit to submit preprocessed indicator bits encoding mapping in a standard form together with the profile itself. Interpretive systems can be made less complex, and hard coding process can be made easier to software developers. Interoperability between different implementation can easier be achieved.

3.3. Context manipulation

The context is memory which stores one or more previous values of fields in the uncompressed header. Besides header fields, context can contain some control data. Context is generally updated with each new packet. Compressor can maintain more than one context to improve robustness.

In practice, situation is more complex. The following method behavior is possible:

1. New context is formed as a copy of current one
2. Method updates context value (encoding and some control methods)
3. Method does not manipulate context (some control method)
4. Method skips context update on demand
5. Context update is skipped when field does not exist in uncompressed header

Context updating declaration must be part of method definition. This way, context behavior can be formally specified. Context update skipping can be specified when encoding method is invoked for particular field.

Context for options actually consists of all optional fields, whether present in current packet or not. For the optional field that does not exist, old option value is kept.

It is natural that fields from different options do not share the same context field. However, at least one protocol is known (DCCP) where field that carries common information (ACK) is a part of option (specific header). In two packet types of DCCP, ACK is not present. This field should use the same context value when present.

The solution is to declare context value to be "common" to all equally named fields in various options (specific headers). Context update must be skipped when field in question is not present, in a manner that all previously stored values are preserved (i.e. context"rotates", but skipped field does not). Decompressor must keep previous value.

Another issue is context field naming, which can be of benefit for context manipulation readability.

4. NOTATION EXTENSIBILITY

Formal notation should be applicable to future protocols, and thus must keep completeness as well as efficacy in profile writing. Three levels of extendibility are foreseen:

1. Profile level (the highest level)
2. Method level (encoding, control and basic)
3. Programming level (the lowest level)

Programming extendibility level is the fundamental one. Programming languages are expected to be complete. This level is used to introduce new methods (basic, encoding, or

control) from scratch. A method should be provided to declare new encoding method needed through the profile.

However, no guarantee is provided that new method will satisfy the requirement for functionality and decompressability. These issues remain responsibility of method writer.

Method level extensibility provides a method to build a new encoding method using basic methods. The difference compared to the encoding functions is that new method becomes part of set of encoding methods and is adequately treated with indicator bits. Again, no guarantee is provided for decompressability.

Finally, the highest level is profile writing itself.

5. CONCLUSION

In this paper, the header compression formal notation requirements are elaborated. The approach of an interpreting system is adopted, because it yields the most comprehensive set of requirements and provides the richest functional implementation.

The interpretive systems with download feature require a complete protocol (profile) and flow recognition parameter set. Header field declaration is analyzed in degree of freedom scope. The systems with half degree of freedom on input are recognized as optimal.

The compressed header structure is restricted to standardized models, and formal notation should be used to specify the model chosen. Combination of models can be used, which leads to the generic profile, the one that can be used for all protocol stacks.

The context manipulation is important in the scope of context field naming, and special manipulations when the same context field is used to store values from more than one header field.

Formal notation extensibility is analyzed in levels of programming language, toolbox and profile writing.

The future work will consist of proposed concepts verification and selection of most appropriate solutions.

REFERENCES

[1] Bormann C, et al, 2001, "Robust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095

[2] Price, R., Hancock, R., McCann, S., Surtees, A., Ollis, P., West, M.: "Framework for EPIC-LITE", draft-ietf-rohc-epic-lite-01.txt, 2002.

[3] Crocker D, et al: “Augmented BNF for Syntax Specifications: ABNF”, RFC 2234, 1997.

[4] Price, R., Surtees, A., West, M.: "A Formal Notation for Header Compression", draft-west-rohc-formal-notation-00.txt, 2002.

[5] Liao, H., Zhang, Q., Zhu, W.: "Generic Header Compression Notation for ROHC", draft-liao-rohc-notation-00.txt, 2002.

[6] Price, R., Surtees, A., West, M.: "Protocol-Enabled BNF-Based LanguagE (PEBBLE)", daft-ietf-rohc-formal-notation-00.txt, 2002.

[7] Price, R., Surtees, A., West, M.: " Formal Notation for Robust Header Compression (ROHC-FN)", draft-ietf-rohc-formal-notation-01.txt, 2003.

[8] Bormann C, et al, " The slides from IETF-56 ROHC meeting ", http://www.dmn.tzi.org/ietf/rohc/rohc-56.pdf, 2003.

[9] Cizmic, M., Vodopija, T., Ozegovic, J.: "EPIC Lite offline processing", SoftCOM 2002.

[10] Mornar, M., Pezelj, A., Ozegovic, J.: " Testbed for header compression implementation ", SoftCOM 2002.

[11] Price, R., Surtees, A., McCann, S., West, M., Hancock, R., Findlay, D.: "EPIC Provably Optimal Format Encoding for Compression in the Internet", SoftCOM 2002.

[12] Stula, M., Vidjak, L., Ozegovic, J.: " Program structures for EPIC-LITE experimental implementation ", SoftCOM 2002.