

Header and Signalling Compression for Cellular Systems

Abigail Surtees, Richard Price, Mark West, Robert Hancock, Paul Ollis, Stephen McCann

Siemens/Roke Manor Research Limited

Romsey, UK

E-mail: {abigail.surtees, richard.price, mark.a.west, robert.hancock, paul.ollis, stephen.mccann}@roke.co.uk

Abstract: The use of the Internet protocol suite to deliver services in cellular systems is seen as crucial to the ability to exploit the system commercially. Compared with the traditional circuit-switched model, the use of IP introduces significant overhead in the form of packet headers. Also, Internet upper layer protocols have typically paid less attention to content encoding compactness than in the traditional circuit switched world. In this paper, applicability of syntax directed compression is discussed.

1. INTRODUCTION

This paper explores solutions for header and signalling compression. It commences with the historical background of header compression developed for dial up links, and then explains the need and context for new compression approaches which have led to the formation of the ROHC (Robust Header Compression) working group within the IETF. The paper outlines the two complementary compression frameworks produced by ROHC, for header (rohc) and content (sigcomp) compression, including their current status within cellular standards.

Furthermore, the paper explains how the syntax directed compression algorithms such as EPIC and EPIC-LITE [13] can be used to instantiate these frameworks for arbitrary protocols. The conclusion enters into a discussion of how EPIC and the concepts of sigcomp can be used together to build an adaptive future proof compression solution of exceptional efficiency and applicability.

2. HISTORY

Header compression was introduced to the Internet in 1991 in Jacobson [1]. This describes how, by analysing the behaviour of TCP/IP header fields, the quantity of data that needs to be sent can be dramatically reduced by, for example, sending delta values rather than the full value and by using information from the link-layer. Although the *de facto* standard for compressing TCP/IP over dial-up links, it only supports IPv4 and does not handle TCP options or other extensions. Also, delta encoding makes it fragile in the presence of dropped packets.

Updates to header compression were discussed in Degermark et al [2, 14] and the first compression scheme for the Real-Time Protocol (RTP) was introduced in Casner and Jacobson [3]. These still provided neither the efficiency nor robustness necessary to operate in a cellular network. The Robust Header Compression (ROHC) working group in the IETF was formed in 2000 and chartered to develop new schemes for RTP and TCP compression that would be highly efficient, future proof and robust.

Bormann et al [4] is the result of this effort to deliver a solution for compressing RTP/UDP/IP in an efficient and robust way.

The ROHC working group is currently working on TCP compression. TCP is a more complex protocol than RTP and efforts are being made to create a flexible and future-proof compression scheme. The current proposal uses a novel approach for automatically generating the compressed header formats, called EPIC (Efficient Protocol Independent Compression), as described in more detail, below.

Signalling compression has been separately addressed within the ROHC working group with SigComp, see Price et al [5]. We also discuss this approach, below.

In all cases, the aim is to retain IP as the common network layer (because of its ubiquity) and to provide adaptation of the network protocols to particular links. This is preferred, with good reason, over adapting applications. Since the adaptation depends, in part, upon the characteristics of the link, flexibility is important. Being able to rapidly develop appropriate solutions, therefore, is highly desirable.

3. HEADER COMPRESSION

ROHC-RTP [4] is now the *de jure* standard for RTP header compression. The techniques that it has established will be applied to other protocols, notably TCP. The encodings designed by the rohc working group are designed to give excellent efficiency and robustness. They also allow for these two qualities to be balanced, as appropriate.

We note that the efficiency of ROHC, after intense discussion within the working group, does not come at the cost of transparency. ROHC strives to deliver packets that are bitwise identical to the packet undergoing compression.

The challenge is substantial – to reduce a 60 byte (for IPv4/UDP/RTP packets) overhead to a manageable level. However, ROHC RTP can reduce this overhead to below 2 bytes per packet, with a level of robustness appropriate to the cellular mobile environment.

In many respects ROHC acts in a similar fashion to [1], [2] and [3], in that it operates over a point-to-point link, assuming no re-ordering or duplication of packets from the link-layer. It also assumes the availability of information from the link-layer (for example, the overall packet size). The compressor chooses a ‘context identifier’ (CID) to identify the compressed flow to the decompressor (and the assignment of packets to CIDs is up to the compressor). Each flow (CID) can make use of a different compression profile.

ROHC RTP takes account of unchanging parts of the header and relationships between fields in much the same way as [1]. A key difference is the use of LSB (Least Significant Bit) encoding to replace the delta-based encoding. This works on the realisation that by sending the least significant n bits of a counter, the counter can be reconstructed correctly even where up to $2^n - 1$ values are lost. Figure 1 shows this: the first value is sent in full to initialise the decompressor but subsequently only the 3 least significant bits are sent. Even when one of these is lost the decompressor can generate the correct value (note that it has to take account of the wrap-around of the compressed value).

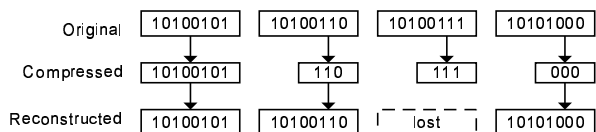


FIGURE 1- Use of LSB encoding

Other improvements include a complex series of states and modes to control the use of feedback packets and to ensure that the decompressor will be able to reliably decompress the compressed packets. Unidirectional mode requires no feedback. Optimistic mode makes use of limited feedback to reduce compressed packet size. Reliable mode makes use of more structured feedback and is less likely to propagate packet damage. The use of feedback in ROHC is more structured than in previous solutions, partly driven by the different mode behaviours, most notably R-mode.

Further, an arbitrary small number (R) of previous headers are kept and compression is carried out against all of these. This is done such that that if at least one of the R packets is received by the decompressor, decompression of the new packet will succeed. These R packets are referred to as the context history and define the guaranteed degree of robustness of the solution (i.e. up to $R - 1$ packets can be lost consecutively with no adverse effect on decompression).

4. SIGNALLING COMPRESSION

Signalling compression (for SIP, as described in Handley et al [6]) is an active area within the IETF. The need for SIP compression is clear and many approaches have been discussed. Compression of packet content (i.e. the signalling messages themselves) is the key method here, and is considered independently of header compression, which can also be applied. It could have been hard to make the decision on which compression algorithm to standardise. Instead, a novel approach has been taken which obviates the need for this decision.

The ROHC working group has defined a Universal Decompressor Virtual Machine (UDVM), see [5]. This virtual machine has a simple instruction set that is targeted at implementing decompression. The choice of algorithm then becomes a local decision at the compressor, which downloads a program to the decompressor, allowing it to decode the messages. The program persists for the whole association between a compressor and decompressor and so can be downloaded ahead of the time when it will be used.

This approach works well precisely because of the session oriented nature of the communication, where the set up cost of the decompressor can be amortised over the session lifetime. There is no need for a restricted set of standardised compressors, so the opportunity exists to develop optimised algorithms. One such approach is the use of a ‘syntax directed’ compressor – for example, EPIC, as described below.

Compressing each message independently avoids problems with message loss but cannot utilise inter-message redundancy. Better compression can be achieved with a pre-loaded static dictionary of common strings. Better yet, a history can be built up from many messages allowing a ‘dynamic’ dictionary to be used. A combined approach with both a static dictionary and dynamic history gives the best performance.

The UDVM is a critical component of the signalling compression solution. However, it should be noted that there are a number of other elements to the solution. These include a secure state-management function, to allow the retrieval of compression history (thus enabling ‘dynamic’ compression) and a feedback mechanism. Both of these are strongly interlinked with the UDVM.

In contrast to header compression, the signalling compression solution does not perform flow-separation and classification internally. Instead, these functions are provided by the entity that invokes sigcomp. As part of the security model, the invoking entity (for example a SIP user-agent), is also responsible for authenticating the messages.

5. EPIC – SYNTAX DIRECTED COMPRESSION

The following sections describe the EPIC approach to both signalling and header compression.

5.1 Syntax directed compression

Initially, we take the example of compressing a text file. General purpose compressors (such as ‘deflate’ as described in Deutsch [7]) have no *a priori* knowledge about the structure of the file. They achieve good compression by looking for redundancy (repeated substrings) and replacing the repetitions with a reference (for example, position and length values) to a previous occurrence. The advantage here, clearly, is that it works on any input data (provided that it contains sufficient redundancy). However, it is also obvious that the compressor is assuming that all possible inputs could occur.

Instead, consider the case where the file contains highly structured data that can be described with a formal grammar (typically expressed in BNF (Backus-Naur Form) as described in Crocker and Overell [8]). Some examples of such a file are HTML, SIP, or any programming language. For a given file length, there are far fewer possible valid combinations than in an unstructured file. (English is structured, but it is impractical to write a grammar that would usefully aid compression). The use of BNF to describe the syntax of information is well understood. For example, we can describe a ‘language’ in which well-formed sentences are lists of integers using the following BNF:

```
<S>          = <number> *(<number>)
<number>    = [ "+" | "-" ] <digit> *(<digit>)
<digit>     = "0" | "1" | "2" | "3" | "4" |
              "5" | "6" | "7" | "8" | "9"
```

In this example, the terminal symbols are used to match symbols in the input. By replacing these symbols with the names on the left-hand side of the rules, a well-formed input is reduced to the primary rule "<S>".

It is clear that the structure imposed by the grammar can be used to aid compression. For example, terminal symbols in the grammar can be treated as a ‘static dictionary’, since the compressor need only indicate the choice of terminal. Rules in the grammar can be thought of as identifying components of the input data (for example a URL or an HTML tag). This means that it is possible to compress a repeated instance by referring to the *n*th previous occurrence of this component. Since the position and length information is implicit in the parsed structure, the reference can be encoded more efficiently than would be possible with a general purpose compressor such as ‘deflate’. It is noted that to take advantage of this technique it is necessary to augment the grammar with information about which nodes can be repeated in this way.

Of course, in common with other compression techniques, it is important that fewer bits are used to encode the most likely outcomes. Thus, we use techniques such as Huffman (see Nelson and Gailly [9]) or arithmetic encoding to minimize the size of the data for the encoded choices. In order to do this the rules in the grammar need to have probabilities associated with them. These can either be fixed (assumed to have been measured on a representative sample of input data) or adaptive. In the former case, the probabilities can be added to the grammar. In the latter case this information will need to be stored as part of the compressed representation. There is an obvious parallel here with the approach taken by ‘deflate’. This can use either a fixed set of Huffman codes (analogous to the case described where the probabilities are fixed as part of the grammar) or dynamically calculate them per block. The threshold for making this choice is intuitively the point at which the compression gain from using adaptive codes is greater than the cost of having to include the dynamic representation.

The base syntax outlined above needs to be supplemented with additional ‘hints’ to the compressor (and decompressor), to make it suitable for use in syntax-directed compression. This augmented grammar is referred to as a ‘profile’. A profile contains sufficient information for the compressor and decompressor to be able to exchange compressed data.

5.2 EPIC and Sigcomp

The first part of this paper describes the sigcomp architecture for signalling compression. This approach, of a syntax directed compressor, can easily be used within this architecture. A key enabler is, of course, the UDVM which allows the compressor to download byte-code to instruct the decompressor how to interpret the messages that it will subsequently send. Here the profile is not only used to drive the compression process, but is also used to automatically generate the byte-code that needs to be sent to the decompressor. Taking the example of SIP, the BNF from Rosenberg et al [10] can be adapted, as described, to generate a SIP profile. The EPIC compressor can then use this to parse and efficiently compress the message. The parsing aspect is stressed, as this approach is best applied to ‘well formed’ sentences of the grammar. It is possible to allow ‘catch-all’ compression of any data, but this will not benefit from the extra information in the syntax. The profile can also be compiled into byte-code that can be sent to the decompressor.

The size of the byte-code depends upon the complexity of the grammar. The SIP grammar, for example, is relatively complex. However, it is not necessary to retain the full, fine-grained, syntax specification in order to benefit from this approach. It is interesting to consider the trade-off between the complexity of the syntax, the efficiency of compression and the size of the downloaded byte-code.

Table 1 shows the results of compressing files containing SIP messages with EPIC and with ‘deflate’. Example data files were obtained from Johnston et al [11]. Seven files (each

containing a single SIP header) were chosen, F1 ... F7, representing a complete SIP session. To more accurately assess the performance of a compressor over a complete SIP session, these data files were concatenated to produce additional files as follows:

- file TWO consists of F1 concatenated with F2;
- file THREE consists of F1, F2 and F3 concatenated together;
- files FOUR, FIVE and SIX continue this sequence; until;
- file ALL consists of all seven files concatenated together.

File Name	SIP size	DEFLATE size	EPIC size	EPIC : DEFLATE
F1	423	272	117	43%
TWO	612	293	121	41%
THREE	817	316	133	42%
FOUR	1235	366	170	46%
FIVE	1449	379	174	46%
SIX	1663	407	183	45%
ALL	1861	410	185	45%

TABLE 1 - Deflate and EPIC comparison

5.3 Header Compression

One problem with header compression, even within a well-defined framework, is defining the compression of new (possibly complex) protocols, such as TCP. The effort spent on ROHC-RTP [4], highlights the frustration of hand-crafting packet formats. Can syntax direction be exploited for producing packet formats for header compression? An obvious question is that of whether a header has a well-defined syntax. Typically, it can be seen that this is not the case, at least not in the sense that we might consider SIP has, for example.

The fact that the input is now a bit-string (rather than a character string) is not important. Although the base 'alphabet' for terminals is now just '0' and '1', the strings can be arbitrarily long and so a grammar could still be written that parsed a bit-string. A much greater concern is the fact that the traditional building blocks of syntax descriptions (choice, concatenation and repetition) do not sit comfortably on the understanding of protocol header formats.

However, recall from the discussion of syntax directed compression of text-based protocols that the profile adds information to the base syntax. One of the tools used is the ability to refer to a previous occurrence of a node in the syntax tree (a terminal within the profile). This introduces the concept of a new kind of terminal symbol. Such a symbol is not expressed as a literal string of bits, but as a function, with side-effects, that matches against the input according to some set of rules. (In this case, the rule is that the input matches some previous instance of this syntactic element.)

A grammar of such functions can be used to describe header protocols. The main requirement is to describe a

sufficiently rich set of encoding functions to allow headers to be 'parsed' and efficiently represented.

Some of the functions that are defined for this purpose, such as LSB, are familiar from current header compression techniques. The 'LSB' function is considered to match the next part of the input string if, and only if, it satisfies the definition of LSB encoding. As with parsing in general, this will consume bits from the input string. As a side-effect, it will generate the least-significant bits that need to be encoded in the compressed header. Conceptually, each set of bits consumed from the input can be regarded as a header field. There are a number of other functions that can be used in a similar way. Another simple example is the 'static' function which matches when the next part of the input bit-string is the same as the context history (as described earlier). For the common case of an unknown, but fixed-length value, 'irregular' can be used.

These functions take parameters to indicate the information needed to make the encoding decision. These can include the length of the field and the probability with which the encoding is applied (this matches directly the use of probabilities described previously).

Within a header, there may be dependencies between fields that can be exploited in compression (for example the RTP Sequence Number and Timestamp). Concepts such as these require more complex encodings to take advantage of the relationship. In this example, it is necessary to capture both that the Timestamp is typically scaled by a fixed step-size and that for each increment of the Sequence Number, the Timestamp often increases by this step-size. A 'scale' method can be used to rescale the Timestamp value. This yields not only the scaled value but also a scale-factor and an offset that also need to be compressed (these values become meta-data associated with the header and are included in the profile). An 'offset' encoding can be used to replace a value with the difference between it and a base value (in this case treating the Timestamp as an offset from the Sequence Number). The offset value then needs to be compressed, but because of the relationship between these fields this is typically more efficient than treating them independently.

A header compression profile, therefore, takes the general form of BNF syntax, but makes use of a defined set of functions in place of the more commonly expected terminal symbols. The header is parsed by traversing the syntax tree. If this is successful, then the input header will be split into fields and each of the encodings that are invoked will have generated part of the compressed representation. The path through the syntax tree (including all the choices of encoding methods) is used to select an overall encoding for the compressed packet.

The decompressor (which shares the same profile information) is able to reverse this process to reconstruct the decompressed header.

In summary, this approach can be used to specify the compression (and decompression) of arbitrary protocols (and

with arbitrary levels of robustness). We remember that ROHC-RTP [4] also specifies states and modes for compression. As these are complex and, in some respects, specific to the solution chosen, it is expected that each new profile will define its own states and modes. Thus, the key part of ROHC-RTP that can be leveraged here is the packet structure (indicating the profile to use and the context identifier, for example), preserving compatibility with current and future ROHC compression profiles. Other components, such as the use of feedback will still have to be specified, although it is hoped that simple, generic schemes will be shared by many profiles.

5.4 Notation

An overview of the notation used to write profiles for header compression can be found in Price et al [12]. It also describes a base library of encoding methods ('static', 'LSB', etc. as mentioned earlier).

The BNF input language is designed to be both human-readable and machine-readable. If only one protocol stack needs to be compressed, the input language can simply be converted by hand directly to an implementation. However, since the input language provides a complete description of the protocol stack to be compressed, it is possible to compress headers using only the information contained in the BNF description and without any additional knowledge of the protocol stack. This means that it is possible to implement a protocol-independent compressor that can download a new ROHC profile described in the BNF input language and, given a suitable set of encoding rules, immediately use it to compress headers.

An example profile can be found in [12] and also in Price et al [13] where there is further discussion of the method used to map from the path through the syntax tree to the overall encoding for the compressed packet.

6. CONCLUSIONS

This paper has given a brief overview of the historical background and development of header and more general protocol compression solutions. This paper has described recent innovations that have brought exceptional versatility and high efficiency to the deployment of compression solutions.

The EPIC approach to compression has been shown to be valid across the whole spectrum of compression applications, with specific relevance to header and signalling compression.

More importantly, it can be seen that these approaches could be harmonised to offer a highly integrated overall solution for compression. Although designed to enable compression of signalling messages, the UDVM can be programmed to decode headers encoded using EPIC. The availability of a UDVM on a host, therefore, can allow the

'compressor local choice' of both signalling and/or header compression (as appropriate).

7. REFERENCES

For RFCs and Interned Drafts see <http://www.ietf.org>.

- [1] Jacobson V, "Compressing TCP/IP Headers for Low-Speed Serial Links", RFC 1144, 1990.
- [2] Degermark M, Nordgren B, Pink S, "IP Header Compression", RFC 2507, 1999.
- [3] Casner S, Jacobson V, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, 1999.
- [4] Bormann C, et al, "Robust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, 2001.
- [5]. Price R, et al, "Signalling Compression (SigComp)", RFC 3320, 2002.
- [6]. Handley M, Schulzrinne H, Schooler E, and Rosenberg J, "SIP: Session Initiation Protocol", RFC 2543, 1999.
- [7] P. Deutsch, "Deflate", RFC 1951, 1996.
- [8] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, 1997.
- [9] Nelson M and Gailly J-L, "The Data Compression Book", M&T Books, 1995.
- [10] Rosenberg et al, "Session Initiation Protocol", draft-ietf-sip-rfc2543bis-09.txt, 2002.
- [11] Johnston et al, "SIP Telephony Call Flow Examples", draft-ietf-sip-call-flows-05.txt (*expired*), 2001.
- [12] Price et al, "A Formal Notation for Header Compression", draft-west-rohc-formal-notation-00.txt, 2002.
- [13] Price et al, "EPIC Provably Optimal Format Encoding for Compression in the Internet", SoftCOM 2002.
- [14] Degermark, M. et al, "Low-loss TCP/IP Header Compression for Wireless Networks" ACM/Baltzer Journal on Wireless Networks, vol 3, no 5, 1997.