Internet Engineering Task Force (IETF)
Request for Comments: 7400
Category: Standards Track
ISSN: 2070-1721
November 2014

C. Bormann
Universitaet Bremen TZI

6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)

Abstract

RFC 6282 defines header compression in 6LoWPAN packets (where "6LoWPAN" refers to "IPv6 over Low-Power Wireless Personal Area Network"). The present document specifies a simple addition that enables the compression of generic headers and header-like payloads, without a need to define a new header compression scheme for each such new header or header-like payload.

Table of Contents

1. Introduction

1.1. The Header Compression Coupling Problem

[RFC6282] defines a scheme for header compression in 6LoWPAN [RFC4944] packets; in this document, we refer to that scheme as 6LoWPAN Header Compression, or 6LoWPAN-HC (where "6LoWPAN" refers to "IPv6 over Low-Power Wireless Personal Area Network"). As with most header compression schemes, a new specification is necessary for every new kind of header that needs to be compressed. In addition, [RFC6282] does not define an extensibility scheme like the Robust Header Compression (ROHC) profiles defined in ROHC [RFC3095] [RFC5795]. This leads to the difficult situation in which 6LoWPAN-HC tended to be reopened and reexamined each time a new header receives consideration (or an old header is changed and reconsidered) in the 6LoWPAN/roll/CoRE cluster of IETF working groups. Although [RFC6282] was finally completed and published, the underlying problem remains unsolved.

The purpose of the present contribution is to plug into [RFC6282] as is, using its Next Header Compression (NHC) concept. We add a slightly less efficient, but vastly more general, form of compression for headers of any kind and even for header-like payloads such as those exhibited by routing protocols, DHCP, etc.: Generic Header Compression (GHC). The objective is an extremely simple specification that can be defined on a single page and implemented in a small number of lines of code, as opposed to a general data compression scheme such as that defined in [RFC1951].

1.2. Compression Approach

The basic approach of GHC's compression function is to define a bytecode for LZ77-style compression [LZ77]. The bytecode is a series of simple instructions for the decompressor to reconstitute the uncompressed payload. These instructions include:

- appending bytes to the reconstituted payload that are literally given with the instruction in the compressed data
- appending a given number of zero bytes to the reconstituted payload
- appending bytes to the reconstituted payload by copying a contiguous sequence from the payload being reconstituted ("backreferencing")
- an ancillary instruction for setting up parameters for the backreferencing instruction in "decompression variables"
- a stop code (optional; see Section 3.2)

The buffer for the reconstituted payload ("destination buffer") is prefixed by a predefined dictionary that can be used in the backreferencing as if it were a prefix of the payload. This predefined dictionary is built from the IPv6 addresses of the packet being reconstituted, followed by a static component, the "static dictionary".

As usual, this specification defines the decompressor operation in detail but leaves the detailed operation of the compressor open to implementation. The compressor can be implemented as with a classical LZ77 compressor, or it can be a simple protocol encoder that just makes use of known compression opportunities.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The term "byte" is used in its now-customary sense as a synonym for "octet".

Terms from [RFC7228] are used in Section 5.

1.4. Notation

This specification uses a trivial notation for code bytes and the bitfields in them, the meaning of which should be mostly obvious. More formally, the meaning of the notation is as follows:

Potential values for the code bytes themselves are expressed by templates that represent 8-bit most-significant-bit-first binary numbers (without any special prefix), where 0 stands for 0, 1 for 1, and variable segments in these code byte templates are indicated by sequences of the same letter, such as kkkkkkk or ssss, the length of which indicates the length of the variable segment in bits.

In the notation of values derived from the code bytes, 0b is used as a prefix for expressing binary numbers in most-significant-bit-first notation (akin to the use of 0x for most-significant-digit-first hexadecimal numbers in the C programming language). Where the above-mentioned sequences of letters are then referenced in such a binary number in the text, the intention is that the value from these bitfields in the actual code byte be inserted.

Example: The code byte template

```
101nssss
```

stands for a byte that starts (most-significant-bit-first) with the bits 1, 0, and 1, and continues with five variable bits, the first of which is referenced as "n" and the next four of which are referenced as "ssss". Based on this code byte template, a reference to

```
0b0ssss000
```

means a binary number composed from a zero bit; the four bits that are in the "ssss" field (for 101nssss, the four least significant bits) in the actual byte encountered, kept in the same order; and three more zero bits.

2. 6LoWPAN-GHC

The format of a GHC-compressed header or payload is a simple bytecode. A compressed header consists of a sequence of pieces, each of which begins with a code byte, which may be followed by zero or more bytes as its argument. Some code bytes cause bytes to be laid out in the destination buffer, and some simply modify some decompression variables.

At the start of decompressing a header or payload within an L2 packet (= fragment), the decompression variables "sa" and "na" are initialized as zero.

The code bytes are defined as follows (Table 1):

code byte	Action	Argument
0kkkkkkk	Append k = 0b0kkkkkkk bytes of data in the bytecode argument (k < 96)	k bytes of data
1000nnnn	Append 0b0000nnnn+2 bytes of zeroes	
10010000	stop code (end of compressed data; see Section 3.2)	
101nssss	Set up extended arguments for a backreference: sa += 0b0ssss000, na += 0b0000n000	
11nnnkkk	Backreference: n = na+0b00000nnn+2; s = 0b00000kkk+sa+n; append n bytes from previously output bytes, starting s bytes to the left of the current output pointer; set sa = 0, na = 0	

Table 1: Bytecodes for Generic Header Compression

Note that the following bit combinations are reserved at this time:

o 011xxxxx

o 1001nnnn (where 0b0000nnnn > 0)

For the purposes of the backreferences, the expansion buffer is initialized with a predefined dictionary, at the end of which the reconstituted payload begins. This dictionary is composed of the source and destination IPv6 addresses of the packet being reconstituted, followed by a 16-byte static dictionary (Figure 1). These 48 dictionary bytes are therefore available for backreferencing but not copied into the final reconstituted payload.

16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00

Figure 1: The 16 Bytes of Static Dictionary (in Hex)

3. Integrating 6LoWPAN-GHC into 6LoWPAN-HC

6LoWPAN-GHC plugs in as an NHC format for 6LoWPAN-HC [RFC6282].

3.1. Compressing Payloads (UDP and ICMPv6)

By definition, GHC is generic and can be applied to different kinds of packets. Many of the examples given in Appendix A are for ICMPv6 packets; a single NHC value suffices to define an NHC format for ICMPv6 based on GHC (see below).

In addition, it is useful to include an NHC format for UDP, as many header-like payloads (e.g., DHCPv6, Datagram Transport Layer Security (DTLS)) are carried in UDP. [RFC6282] already defines an NHC format for UDP (11110CPP). GHC uses an analogous NHC byte formatted as shown in Figure 2. The difference to the existing UDP NHC specification is that for 11010CPP NHC bytes, the UDP payload is not supplied literally but compressed by 6LoWPAN-GHC.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 | 1 | 0 | 1 | 0 | C |   P   |
+---+---+---+---+---+---+---+---+
```

Figure 2: NHC Byte for UDP GHC (11010CPP)

To stay in the same general numbering space, we use 11011111 as the NHC byte for ICMPv6 GHC (Figure 3).

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
+---+---+---+---+---+---+---+---+
```

Figure 3: NHC Byte for ICMPv6 GHC (11011111)

3.2. Compressing Extension Headers

Compression of specific extension headers is added in a similar way (Figure 4) (however, probably only Extension Header ID (EID) 0 to 3 [RFC6282] need to be assigned). As there is no easy way to extract the Length field from the GHC-encoded header before decoding, this would make detecting the end of the extension header somewhat complex. The easiest (and most efficient) approach is to completely elide the Length field (in the same way NHC already elides the Next Header field in certain cases) and reconstruct it only on decompression. To serve as a terminator for the extension header, the bytecode 0b10010000 has been assigned as a stop code. Note that the stop code is only needed for extension headers, not for the final payloads discussed in the previous subsection, the decompression of which is automatically stopped by the end of the packet.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 | 0 | 1 | 1 |    EID    |NH |
+---+---+---+---+---+---+---+---+
```

Figure 4: NHC Byte for Extension Header GHC

3.3. Indicating GHC Capability

The 6LoWPAN baseline includes just [RFC4944], [RFC6282], and [RFC6775] (see [Roadmap-6LoWPAN]). To enable the use of GHC towards a neighbor, a 6LoWPAN node needs to know that the neighbor implements it. While this can also simply be administratively required, a transition strategy as well as a way to support mixed networks is required.

One way to know that a neighbor does implement GHC is receiving a packet from that neighbor with GHC in it ("implicit capability detection"). However, there needs to be a way to bootstrap this, as nobody would ever start sending packets with GHC otherwise.

To minimize the impact on [RFC6775], we define a Neighbor Discovery (ND) option called the 6LoWPAN Capability Indication Option (6CIO), as illustrated in Figure 5. (For the fields marked by an underscore in Figure 5, see Section 3.4.)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |   Length = 1  |_____________________________|G|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|_______________________________________________________________|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: 6LoWPAN Capability Indication Option (6CIO)

The G bit indicates whether the node sending the option is GHC capable.

Once a node receives either an explicit or implicit indication of GHC capability from another node, it may send GHC-compressed packets to that node. Where that capability has not been recently confirmed, similar to the way Packetization Layer Path MTU Discovery (PLPMTUD)

[RFC4821] finds out about changes in the network, a node SHOULD make use of Neighbor Unreachability Detection (NUD) failures to switch back to basic 6LoWPAN header compression [RFC6282].

3.4. Using the 6CIO

The 6CIO will typically only be sent in 6LoWPAN-ND Router Solicitation (RS) packets (which cannot themselves be GHC compressed unless the host desires to limit itself to talking to GHC-capable routers). The resulting 6LoWPAN-ND Router Advertisement (RA) can then already make use of GHC and thus indicate GHC capability implicitly, which in turn allows both nodes to use GHC in the 6LoWPAN-ND Neighbor Solicitation / Neighbor Advertisement (NS/NA) exchange.

The 6CIO can also be used for future options that need to be negotiated between 6LoWPAN peers; an IANA registry is used to assign the flags. Bits marked by underscores in Figure 5 are unassigned and available for future assignment. They MUST be sent as zero and MUST be ignored on reception until assigned by IANA. Length values larger than 1 MUST be accepted by implementations in order to enable future extensions; the additional bits in the option are then deemed unassigned in the same way. For the purposes of the IANA registry, the bits are numbered in most-significant-bit-first order from the 16th bit of the option onward: the 16th bit is flag number 0, the 31st bit (the G bit) is flag number 15, up to the 63rd bit for flag number 47. (Additional bits may also be used by a follow-on version of this document if some bit combinations that have been left unassigned here are then used in an upward-compatible manner.)

Flag numbers 0 to 7 are reserved for experimental use. They MUST NOT be used for actual deployments.

Where the use of this option by other specifications or for experimental use is envisioned, the following items have to be kept in mind:

- The option can be used in any ND packet.

- Specific bits are set in the option to indicate that a capability is present in the sender. (There may be other ways to infer this information, as is the case in this specification.) Bit combinations may be used as desired. The absence of the capability _indication_ is signaled by setting these bits to zero; this does not necessarily mean that the capability is absent.

- The intention is not to modify the semantics of the specific ND packet carrying the option but to provide the general capability indication described above.

- Specifications have to be designed such that receivers that do not receive or do not process such a capability indication can still interoperate (presumably without exploiting the indicated capability).

- The option is meant to be used sparsely, i.e., once a sender has reason to believe the capability indication has been received, there is no longer a need to continue sending it.

4. IANA Considerations

IANA has added the assignments listed in Figure 6 in the "LOWPAN_NHC Header Type" registry (under "IPv6 Low Power Personal Area Network Parameters").

```
        10110EEN: Extension header GHC              [RFC7400]
        11010CPP: UDP GHC                           [RFC7400]
        11011111: ICMPv6 GHC                        [RFC7400]
```

Figure 6: IANA Assignments for the NHC Byte

IANA has allocated ND option number 36 for the "6LoWPAN Capability Indication Option (6CIO)" ND option format in the "IPv6 Neighbor Discovery Option Formats" registry [RFC4861].

IANA has created a subregistry for "6LoWPAN capability Bits" under the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry. The bits are assigned by giving their numbers as small, non-negative integers as defined in Section 3.4, in the range 0-47. The policy is "IETF Review" or "IESG Approval" [RFC5226]. The initial content of the registry is as shown in Figure 7:

```
             0-7: Reserved for Experimental Use     [RFC7400]
            8-14: Unassigned
              15: GHC capable bit (G bit)           [RFC7400]
           16-47: Unassigned
```

Figure 7: IANA Assignments for the 6LoWPAN Capability Bits

5. Security Considerations

The security considerations of [RFC4944] and [RFC6282] apply. As usual in protocols with packet parsing/construction, care must be taken in implementations to avoid buffer overflows and, in particular (with respect to the backreferencing), out-of-area references during decompression.

One additional consideration is that an attacker may send a forged packet that makes a second node believe a third victim node is GHC capable. If it is not, this may prevent packets sent by the second node from reaching the third node (at least until robustness features such as those discussed in Section 3.3 kick in).

No mitigation is proposed (or known) for this attack, except that a victim node that does implement GHC is not vulnerable. However, with unsecured ND, a number of attacks with similar outcomes are already possible, so there is little incentive to make use of this additional attack. With secured ND, the 6CIO is also secured; nodes relying on secured ND therefore should use the 6CIO bidirectionally (and limit the implicit capability detection to secured ND packets carrying GHC) instead of basing their neighbor capability assumptions on receiving any kind of unprotected packet.

As with any LZ77 scheme, decompression bombs (compressed packets crafted to expand so much that the decompressor is overloaded) are a problem. An attacker cannot send a GHC decompressor into a tight loop for too long, because the MTU will be reached quickly. Some amplification of an attack from inside the compressed link is possible, though. Using a constrained node in a constrained network as a DoS attack source is usually not very useful, though, except maybe against other nodes in that constrained network. The worst case for an attack to the outside is a not-so-constrained device using a (typically not-so-constrained) edge router to attack by forwarding out of its Ethernet interface. The worst-case amplification of GHC is 17, so an MTU-size packet can be generated from a 6LoWPAN packet of 76 bytes. The 6LoWPAN network is still constrained, so the amplification at the edge router turns an entire 250 kbit/s 802.15.4 network (assuming a theoretical upper bound of 225 kbit/s throughput to a single-hop adjacent node) into a 3.8 Mbit/s attacker.

The amplification may be more important inside the 6LoWPAN, if there is a way to obtain reflection (otherwise, the packet is likely to simply stay compressed on the way and do little damage), e.g., by pinging a node using a decompression bomb, somehow keeping that node from re-compressing the ping response (which would probably require something more complex than simple runs of zeroes, so the worst-case amplification is likely closer to 9). Or, if there are nodes that do not support GHC, those can be attacked via a router that is then forced to decompress.

All these attacks are mitigated by some form of network access control.

In a 6LoWPAN stack, sensitive information will normally be protected by transport- or application-layer (or even IP-layer) security, which are all above the adaptation layer, leaving no sensitive information to compress at the GHC level. However, a 6LoWPAN deployment that entirely depends on Media Access Control (MAC) layer security may be vulnerable to attacks that exploit redundancy information disclosed by compression to recover information about secret values. The attacker would need to be in radio range to observe the compressed packets. Since compression is stateless, the attacker would need to entice the party sending the secret value to also send some value controlled (or at least usefully varying and knowable) by the attacker in (what becomes the first adaptation-layer fragment of) the same packet. This attack is fully mitigated by not exposing secret values to the adaptation layer or by not using GHC in deployments where this is done.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007, <http://www.rfc-editor.org/info/rfc4861>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007, <http://www.rfc-editor.org/info/rfc4944>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011, <http://www.rfc-editor.org/info/rfc6282>.

[RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012, <http://www.rfc-editor.org/info/rfc6775>.

6.2. Informative References

[ICMPv6-ND] O'Flynn, C., "ICMPv6/ND Compression for 6LoWPAN Networks", Work in Progress, draft-oflynn-6lowpan-icmphc-00, July 2010.

[LZ77] Ziv, J. and A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343, May 1977.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996, <http://www.rfc-editor.org/info/rfc1951>.

[RFC3095] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., and H. Zheng, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, July 2001, <http://www.rfc-editor.org/info/rfc3095>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007, <http://www.rfc-editor.org/info/rfc4821>.

[RFC5795] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObust Header Compression (ROHC) Framework", RFC 5795, March 2010, <http://www.rfc-editor.org/info/rfc5795>.

[RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012, <http://www.rfc-editor.org/info/rfc6550>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[Roadmap-6LoWPAN]
Bormann, C., "6LoWPAN Roadmap and Implementation Guide", Work in Progress, draft-bormann-6lo-6lowpan-roadmap-00, October 2013.

Appendix A. Examples

This section demonstrates some relatively realistic examples derived from actual packet captures taken at previous interops.

For the Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6550], Figure 8 shows a Destination-Oriented Directed Acyclic Graph (DODAG) Information Solicitation (DIS), a quite short RPL message that obviously cannot be improved much.

```
IP header:
 60 00 00 00 00 08 3a ff fe 80 00 00 00 00 00 00
 02 1c da ff fe 00 20 24 ff 02 00 00 00 00 00 00
 00 00 00 00 00 00 00 1a
Payload:
 9b 00 6b de 00 00 00 00
Dictionary:
 fe 80 00 00 00 00 00 00 02 1c da ff fe 00 20 24
 ff 02 00 00 00 00 00 00 00 00 00 00 00 00 00 1a
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
copy: 04 9b 00 6b de
4 nulls: 82
Compressed:
 04 9b 00 6b de 82
Was 8 bytes; compressed to 6 bytes, compression factor 1.33
```

Figure 8: A Simple RPL Example

Figure 9 shows a RPL DODAG Information Object, a longer RPL control message that is improved a bit more. Note that the compressed output exposes an inefficiency in the simple-minded compressor used to generate it; this does not devalue the example, since constrained nodes are quite likely to make use of simple-minded compressors.

```
   IP header:
    60 00 00 00 00 5c 3a ff fe 80 00 00 00 00 00 00
    02 1c da ff fe 00 30 23 ff 02 00 00 00 00 00 00
    00 00 00 00 00 00 00 1a
   Payload:
    9b 01 7a 5f 00 f0 01 00 88 00 00 00 20 02 0d b8
    00 00 00 00 00 00 00 ff fe 00 fa ce 04 0e 00 14
    09 ff 00 00 01 00 00 00 00 00 00 00 08 1e 80 20
    ff ff ff ff ff ff ff ff 00 00 00 00 20 02 0d b8
    00 00 00 00 00 00 00 ff fe 00 fa ce 03 0e 40 00
    ff ff ff ff 20 02 0d b8 00 00 00 00
   Dictionary:
    fe 80 00 00 00 00 00 00 02 1c da ff fe 00 30 23
    ff 02 00 00 00 00 00 00 00 00 00 00 00 00 00 1a
    16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
   copy: 06 9b 01 7a 5f 00 f0
   ref(9): 01 00 -> ref 11nnnkkk 0 7: c7
   copy: 01 88
   3 nulls: 81
   copy: 04 20 02 0d b8
   7 nulls: 85
   ref(60): ff fe 00 -> ref 101nssss 0 7/11nnnkkk 1 1: a7 c9
   copy: 08 fa ce 04 0e 00 14 09 ff
   ref(39): 00 00 01 00 00 -> ref 101nssss 0 4/11nnnkkk 3 2: a4 da
   5 nulls: 83
   copy: 06 08 1e 80 20 ff ff
   ref(2): ff ff -> ref 11nnnkkk 0 0: c0
   ref(4): ff ff ff ff -> ref 11nnnkkk 2 0: d0
   4 nulls: 82
   ref(48): 20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 fa ce
    -> ref 101nssss 1 4/11nnnkkk 6 0: b4 f0
   copy: 03 03 0e 40
   ref(9): 00 ff -> ref 11nnnkkk 0 7: c7
   ref(28): ff ff ff -> ref 101nssss 0 3/11nnnkkk 1 1: a3 c9
   ref(24): 20 02 0d b8 00 00 00 00
    -> ref 101nssss 0 2/11nnnkkk 6 0: a2 f0
   Compressed:
    06 9b 01 7a 5f 00 f0 c7 01 88 81 04 20 02 0d b8
    85 a7 c9 08 fa ce 04 0e 00 14 09 ff a4 da 83 06
    08 1e 80 20 ff ff c0 d0 82 b4 f0 03 03 0e 40 c7
    a3 c9 a2 f0
   Was 92 bytes; compressed to 52 bytes, compression factor 1.77
```

Figure 9: A Longer RPL Example

Similarly, Figure 10 shows a RPL Destination Advertisement Object (DAO) message. One of the embedded addresses is copied right out of the pseudo-header; the other one is effectively converted from global to local by providing the prefix FE80 literally, inserting a number of nulls, and copying (some of) the Interface Identifier part again out of the pseudo-header. Note that a simple implementation would probably emit fewer nulls and copy the entire Interface Identifier; there are multiple ways to encode this 50-byte payload into 27 bytes.

```
IP header:
 60 00 00 00 00 32 3a ff 20 02 0d b8 00 00 00 00
 00 00 00 ff fe 00 33 44 20 02 0d b8 00 00 00 00
 00 00 00 ff fe 00 11 22
Payload:
 9b 02 58 7d 01 80 00 f1 05 12 00 80 20 02 0d b8
 00 00 00 00 00 00 00 ff fe 00 33 44 06 14 00 80
 f1 00 fe 80 00 00 00 00 00 00 00 00 00 ff fe 00
 11 22
Dictionary:
 20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 33 44
 20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 11 22
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
copy: 0c 9b 02 58 7d 01 80 00 f1 05 12 00 80
ref(60): 20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 33 44
 -> ref 101nssss 1 5/11nnnkkk 6 4: b5 f4
copy: 08 06 14 00 80 f1 00 fe 80
9 nulls: 87
ref(66): ff fe 00 11 22 -> ref 101nssss 0 7/11nnnkkk 3 5: a7 dd
Compressed:
 0c 9b 02 58 7d 01 80 00 f1 05 12 00 80 b5 f4 08
 06 14 00 80 f1 00 fe 80 87 a7 dd
Was 50 bytes; compressed to 27 bytes, compression factor 1.85
```

Figure 10: A RPL DAO Message

Figure 11 shows the effect of compressing a simple ND neighbor solicitation.

```
IP header:
 60 00 00 00 00 30 3a ff 20 02 0d b8 00 00 00 00
 00 00 00 ff fe 00 3b d3 fe 80 00 00 00 00 00 00
 02 1c da ff fe 00 30 23
Payload:
 87 00 a7 68 00 00 00 00 fe 80 00 00 00 00 00 00
 02 1c da ff fe 00 30 23 01 01 3b d3 00 00 00 00
 1f 02 00 00 00 00 00 06 00 1c da ff fe 00 20 24
Dictionary:
 20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 3b d3
 fe 80 00 00 00 00 00 00 02 1c da ff fe 00 30 23
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
copy: 04 87 00 a7 68
4 nulls: 82
ref(40): fe 80 00 00 00 00 00 00 02 1c da ff fe 00 30 23
 -> ref 101nssss 1 3/11nnnkkk 6 0: b3 f0
copy: 04 01 01 3b d3
4 nulls: 82
copy: 02 1f 02
5 nulls: 83
copy: 02 06 00
ref(24): 1c da ff fe 00 -> ref 101nssss 0 2/11nnnkkk 3 3: a2 db
copy: 02 20 24
Compressed:
 04 87 00 a7 68 82 b3 f0 04 01 01 3b d3 82 02 1f
 02 83 02 06 00 a2 db 02 20 24
Was 48 bytes; compressed to 26 bytes, compression factor 1.85
```

Figure 11: An ND Neighbor Solicitation

Figure 12 shows the compression of an ND neighbor advertisement.

```
IP header:
 60 00 00 00 00 30 3a fe fe 80 00 00 00 00 00 00
 02 1c da ff fe 00 30 23 20 02 0d b8 00 00 00 00
 00 00 00 ff fe 00 3b d3
Payload:
 88 00 26 6c c0 00 00 00 fe 80 00 00 00 00 00 00
 02 1c da ff fe 00 30 23 02 01 fa ce 00 00 00 00
 1f 02 00 00 00 00 00 06 00 1c da ff fe 00 20 24
Dictionary:
 fe 80 00 00 00 00 00 00 02 1c da ff fe 00 30 23
 20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 3b d3
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
copy: 05 88 00 26 6c c0
3 nulls: 81
ref(56): fe 80 00 00 00 00 00 00 02 1c da ff fe 00 30 23
 -> ref 101nssss 1 5/11nnnkkk 6 0: b5 f0
copy: 04 02 01 fa ce
4 nulls: 82
copy: 02 1f 02
5 nulls: 83
copy: 02 06 00
ref(24): 1c da ff fe 00 -> ref 101nssss 0 2/11nnnkkk 3 3: a2 db
copy: 02 20 24
Compressed:
 05 88 00 26 6c c0 81 b5 f0 04 02 01 fa ce 82 02
 1f 02 83 02 06 00 a2 db 02 20 24
Was 48 bytes; compressed to 27 bytes, compression factor 1.78
```

Figure 12: An ND Neighbor Advertisement

Figure 13 shows the compression of an ND router solicitation. Note that the relatively good compression is not caused by the many zero bytes in the link-layer address of this particular capture (which are unlikely to occur in practice): 7 of these 8 bytes are copied from the pseudo-header (the 8th byte cannot be copied, as the universal/local bit needs to be inverted).

```
IP header:
 60 00 00 00 00 18 3a ff fe 80 00 00 00 00 00 00
 ae de 48 00 00 00 00 01 ff 02 00 00 00 00 00 00
 00 00 00 00 00 00 00 02
Payload:
 85 00 90 65 00 00 00 00 01 02 ac de 48 00 00 00
 00 01 00 00 00 00 00 00
Dictionary:
 fe 80 00 00 00 00 00 00 ae de 48 00 00 00 00 01
 ff 02 00 00 00 00 00 00 00 00 00 00 00 00 00 02
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
copy: 04 85 00 90 65
ref(11): 00 00 00 00 01 -> ref 11nnnkkk 3 6: de
copy: 02 02 ac
ref(50): de 48 00 00 00 00 01
 -> ref 101nssss 0 5/11nnnkkk 5 3: a5 eb
6 nulls: 84
Compressed:
 04 85 00 90 65 de 02 02 ac a5 eb 84
Was 24 bytes; compressed to 12 bytes, compression factor 2.00
```

Figure 13: An ND Router Solicitation

Figure 14 shows the compression of an ND router advertisement. The indefinite lifetime is compressed to four bytes by backreferencing; this could be improved (at the cost of minor additional decompressor complexity) by including some simple runlength mechanism.

```
   IP header:
    60 00 00 00 00 60 3a ff fe 80 00 00 00 00 00 00
    10 34 00 ff fe 00 11 22 fe 80 00 00 00 00 00 00
    ae de 48 00 00 00 00 01
   Payload:
    86 00 55 c9 40 00 0f a0 1c 5a 38 17 00 00 07 d0
    01 01 11 22 00 00 00 00 03 04 40 40 ff ff ff ff
    ff ff ff ff 00 00 00 00 20 02 0d b8 00 00 00 00
    00 00 00 00 00 00 00 00 20 02 40 10 00 00 03 e8
    20 02 0d b8 00 00 00 00 21 03 00 01 00 00 00 00
    20 02 0d b8 00 00 00 00 00 00 00 ff fe 00 11 22
   Dictionary:
    fe 80 00 00 00 00 00 00 10 34 00 ff fe 00 11 22
    fe 80 00 00 00 00 00 00 ae de 48 00 00 00 00 01
    16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
   copy: 0c 86 00 55 c9 40 00 0f a0 1c 5a 38 17
   2 nulls: 80
   copy: 06 07 d0 01 01 11 22
   4 nulls: 82
   copy: 06 03 04 40 40 ff ff
   ref(2): ff ff -> ref 11nnnkkk 0 0: c0
   ref(4): ff ff ff ff -> ref 11nnnkkk 2 0: d0
   4 nulls: 82
   copy: 04 20 02 0d b8
   12 nulls: 8a
   copy: 04 20 02 40 10
   ref(38): 00 00 03 -> ref 101nssss 0 4/11nnnkkk 1 3: a4 cb
   copy: 01 e8
   ref(24): 20 02 0d b8 00 00 00 00
    -> ref 101nssss 0 2/11nnnkkk 6 0: a2 f0
   copy: 02 21 03
   ref(84): 00 01 00 00 00 00
    -> ref 101nssss 0 9/11nnnkkk 4 6: a9 e6
   ref(40): 20 02 0d b8 00 00 00 00 00 00 00
    -> ref 101nssss 1 3/11nnnkkk 1 5: b3 cd
   ref(128): ff fe 00 11 22
    -> ref 101nssss 0 15/11nnnkkk 3 3: af db
   Compressed:
    0c 86 00 55 c9 40 00 0f a0 1c 5a 38 17 80 06 07
    d0 01 01 11 22 82 06 03 04 40 40 ff ff c0 d0 82
    04 20 02 0d b8 8a 04 20 02 40 10 a4 cb 01 e8 a2
    f0 02 21 03 a9 e6 b3 cd af db
   Was 96 bytes; compressed to 58 bytes, compression factor 1.66
```

Figure 14: An ND Router Advertisement

Figure 15 shows the compression of a DTLS application data packet with a net payload of 13 bytes of cleartext and 8 bytes of authenticator (note that the IP header is not relevant for this example and has been set to 0). This makes good use of the static dictionary and is quite effective crunching out the redundancy in the TLS_PSK_WITH_AES_128_CCM_8 header, leading to a net reduction by 15 bytes.

```
IP header:
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00
Payload:
 17 fe fd 00 01 00 00 00 00 00 01 00 1d 00 01 00
 00 00 00 00 01 09 b2 0e 82 c1 6e b6 96 c5 1f 36
 8d 17 61 e2 b5 d4 22 d4 ed 2b
Dictionary:
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
ref(13): 17 fe fd 00 01 00 00 00 00 00 01 00
 -> ref 101nssss 1 0/11nnnkkk 2 1: b0 d1
copy: 01 1d
ref(10): 00 01 00 00 00 00 00 01 -> ref 11nnnkkk 6 2: f2
copy: 15 09 b2 0e 82 c1 6e b6 96 c5 1f 36 8d 17 61 e2
copy: b5 d4 22 d4 ed 2b
Compressed:
 b0 d1 01 1d f2 15 09 b2 0e 82 c1 6e b6 96 c5 1f
 36 8d 17 61 e2 b5 d4 22 d4 ed 2b
Was 42 bytes; compressed to 27 bytes, compression factor 1.56
```

Figure 15: A DTLS Application Data Packet

Figure 16 shows that the compression is slightly worse in a subsequent packet (containing 6 bytes of cleartext and 8 bytes of authenticator, yielding a net compression of 13 bytes). The total overhead does stay at a quite acceptable 8 bytes.

```
IP header:
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00
Payload:
 17 fe fd 00 01 00 00 00 00 00 05 00 16 00 01 00
 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
 cb 35 b9
Dictionary:
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
ref(13): 17 fe fd 00 01 00 00 00 00 00
 -> ref 101nssss 1 0/11nnnkkk 0 3: b0 c3
copy: 03 05 00 16
ref(10): 00 01 00 00 00 00 00 05 -> ref 11nnnkkk 6 2: f2
copy: 0e ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9
Compressed:
 b0 c3 03 05 00 16 f2 0e ae a0 15 56 67 92 4d ff
 8a 24 e4 cb 35 b9
Was 35 bytes; compressed to 22 bytes, compression factor 1.59
```

Figure 16: Another DTLS Application Data Packet

Figure 17 shows the compression of a DTLS handshake message, here a client hello. There is little that can be compressed about the 32 bytes of randomness. Still, the net reduction is by 14 bytes.

```
IP header:
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00
Payload:
 16 fe fd 00 00 00 00 00 00 00 00 00 36 01 00 00
 2a 00 00 00 00 00 00 00 2a fe fd 51 52 ed 79 a4
 20 c9 62 56 11 47 c9 39 ee 6c c0 a4 fe c6 89 2f
 32 26 9a 16 4e 31 7e 9f 20 92 92 00 00 00 02 c0
 a8 01 00
Dictionary:
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 16 fe fd 17 fe fd 00 01 00 00 00 00 00 01 00 00
ref(16): 16 fe fd -> ref 101nssss 0 1/11nnnkkk 1 5: a1 cd
9 nulls: 87
copy: 01 36
ref(16): 01 00 00 -> ref 101nssss 0 1/11nnnkkk 1 5: a1 cd
copy: 01 2a
7 nulls: 85
copy: 23 2a fe fd 51 52 ed 79 a4 20 c9 62 56 11 47 c9
copy: 39 ee 6c c0 a4 fe c6 89 2f 32 26 9a 16 4e 31 7e
copy: 9f 20 92 92
3 nulls: 81
copy: 05 02 c0 a8 01 00
Compressed:
 a1 cd 87 01 36 a1 cd 01 2a 85 23 2a fe fd 51 52
 ed 79 a4 20 c9 62 56 11 47 c9 39 ee 6c c0 a4 fe
 c6 89 2f 32 26 9a 16 4e 31 7e 9f 20 92 92 81 05
 02 c0 a8 01 00
Was 67 bytes; compressed to 53 bytes, compression factor 1.26
```

Figure 17: A DTLS Handshake Packet (Client Hello)

Acknowledgements

Colin O'Flynn has repeatedly insisted that some form of compression for ICMPv6 and ND packets might be beneficial. He actually wrote his own document, [ICMPv6-ND], which compresses better, but that document only addresses basic ICMPv6/ND and needs a much longer specification (around 17 pages of detailed specification, as compared to the single page of core specification here). This motivated the author to try something simple, yet general. Special thanks go to Colin for indicating that he indeed considers his document superseded by this one.

The examples given are based on packet capture files that Colin O'Flynn, Owen Kirby, Olaf Bergmann, and others provided.

Using these files as a corpus, the static dictionary was developed, and the bit allocations validated, based on research by Sebastian Dominik.

Erik Nordmark provided input that helped shape the 6CIO. Thomas Bjorklund proposed simplifying the predefined dictionary.

Yoshihiro Ohba insisted on clarifying the notation used for the definition of the bytecodes and their bitfields. Ulrich Herberg provided some additional review and suggested expanding the introductory material, and with Hannes Tschofenig and Brian Haberman he helped come up with the IANA policy for the "6LoWPAN capability bits" assignments in the 6CIO.

The IESG reviewers Richard Barnes and Stephen Farrell contributed topics to the Security Considerations section; they and Barry Leiba, as well as GEN-ART reviewer Vijay K. Gurbani, also provided editorial improvements.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
EMail: cabo@tzi.org