Header Compression for TLV-based Packets

ICNRG Buenos Aires (IETF 95)

Marc Mosko

April 3, 2016

“Header Compression” in TLV World

- Compress all the signaling
  - Fixed Header
  - T and L fields
  - V fields except user payload
    - KeyId
    - Public Keys
    - Name Components
    - Timestamps
    - Anything that is predictable

Motivation for something new

- Network packets are small
  - Gzip, bzip2, etc. usually expand packet because of their block encoding structure.
  - Microsoft point-to-point compress (MPPC, RFC 2118) only has minor savings, sometimes bigger.
- Dictionary and window algorithms
  - Require state exchange, lost packets result in burst errors or decoding delay.
  - Need a lot of buffer space if there are packets from mixed flows.

Why is gzip bad?

- 10 byte header, 3 byte footer.
- Back references are 3 bytes, minimum
  - But repeating T values are 2 bytes.
  - Exact patterns do not repeat much, but some fields have high redundancy that we can remove with context-dependent substitutions.
  - Won’t even work for 1/3/5 encoding with 1+1
- It will build up many short dictionary entries on cryptographic fields.
- It has to transmit the dictionary.

Why is bzip2 bad?

- Run-length encoding of 4+ byte too long
- 100k – 900k block size
- 4-byte header, 4-byte footer
- 20+ byte block header

What about window/learning

- OK between two consistent peers
  - 1-hop peer ok.
  - Otherwise, Interests can go anywhere unless you use topological name.
- Losses cause burst errors unless use ACKS
  - Leads to delay in using learned values.
  - Tradeoff between loss and burst errors.
- ICN packets might be very large
  - Need large history window, so finding longest string match might be pretty expensive.

Example (interest)

- Interest with fixed header and 2+2 TLV
  /bell/0x01020304/0x05060708/0x090a0b0c

Method	Bytes
Data (name)	**16**
Uncompressed	**48**
gzip -9	**77**
bzip2 -9	**75**
MPPC (RFC 2118)	**42**
TLV compression	**28**

Example (Content Object)

- Content object w/ 162-byte public key, 32-byte keyid, and 128-byte signature, etc.

Method	Bytes
Data (name, payload, pubkey, keyid, sig)	**372**
Uncompressed	**436**
gzip -9	**461**
bzip2 -9	**574**
MPPC (RFC 2118)	**448**
TLV compression	**396**

Overview

- Static TL compression
  - Allows reducing the overhead caused by TL encoding (2+2 and 1/3/5) *without state exchange*.
- Dictionary learned replacement
  - Learn strings like Key IDs and Public Keys. Those are long random byte strings.
  - Use delta encoding for things like Chunks or times or serial numbers.
- Byte-aligned on 'T' boundaries.

Outline of Algorithm

- Fixed header has a “compressed” flag
  - Version field is only 4 bits
  - If not set, uses 8 byte FH and 2+2 TLs
  - If set,
    - 1-byte context header (2bit flats, 3bit CID, 3bit CRC)
    - use 3, 4, or 8 byte FH and 1 – 5 byte TLs
- In “compressed” mode
  - Static TL pair or (TL)*TL string (in to 1 byte)
  - Static T, variable L (in to 1, 2, 3, 4 or 5 bytes)
  - Learned TLV replacement (in to 2, 3, or 4 bytes)
  - Learned TLV counter (only send offset from base)

Initialization

- Before using compression
  - Peers exchange willingness to compress.
  - Peers exchange capabilities
    - Maximum buffer size (used for window based dictionary definitions).
    - Name of static dictionary used, if not the default.
  - If using non-standard static dictionary
    - Exchange the dictionaries.
  - Done at link initialization or with in-band link management.
  - Determine a Context ID (CID) for this state.

State Exchange

- Out-of-band
  - Use a separate packet with FixedHeader PacketType = Dictionary
  - Sends one or more definitions.
  - Has Seqnum for reliable state exchange.
- In-band
  - Footer sends dictionary definitions, using (backwards_offset, length) back in to the packet.
  - Carries seqnum for reliable state exchange.
  - Has own CRC
- State exchange ACK

TL values

- CCNx 1.0
  - Re-uses “T” values as it’s context dependent. So, very few actual “T” values. Leads to highly-compressable packet format.
- NDN 1/3/5
  - Uses a global “T” space. Use a pre-processor to map common values in context to high-redundancy values.

Entropy examples

- Based on random source model for an Interest.
- TL + V uses 6-component name with 5 repeated.

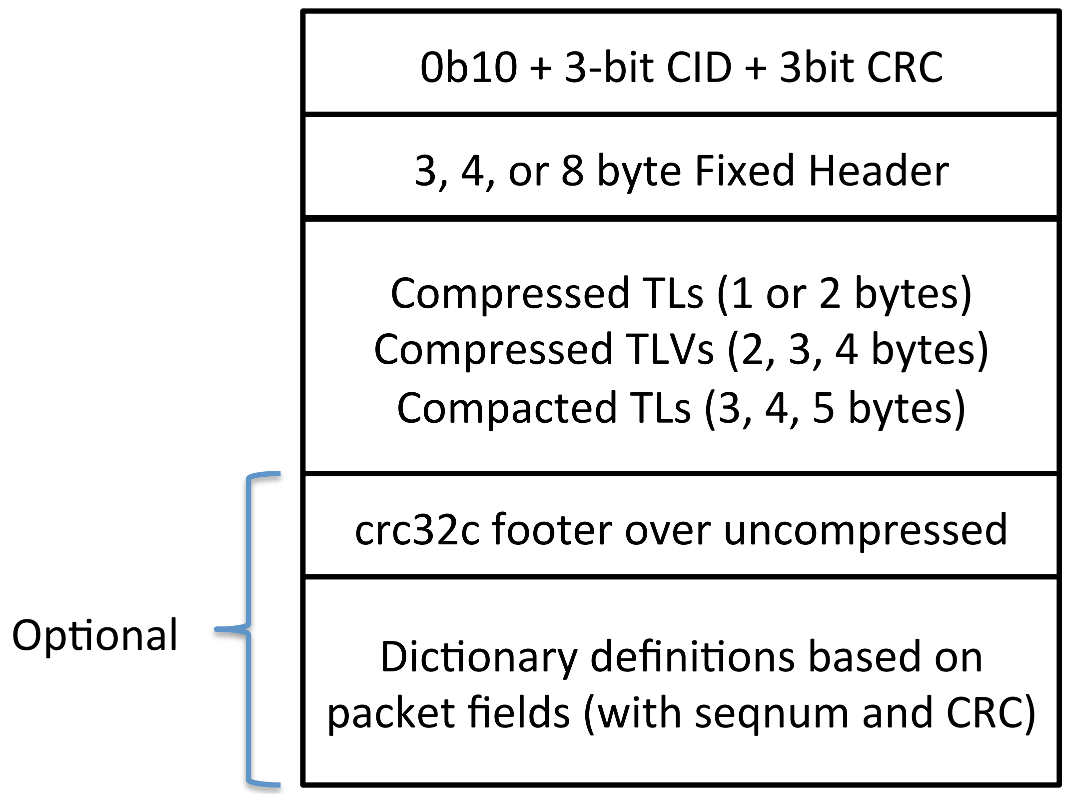
	H (bit-aligned)	H (byte-aligned)	2+2	1/3/5	TL comp.
TL only	4.9	8.0	32.0	18.9	8.0
TL + V	8.4	11.7	88.3	55.4	14.8

Conclusion

- Initialization stage
  - Use static dictionary to compress TLs.
  - Compress fixed header.
  - Can be used inside encryption envelope too.
- Learning stage
  - Use reliable state exchange to compress TLVs.
  - TLV pattern substitution.
  - Counter type for delta encoding.
- Have running code (python) for static dictionary

DETAILED BIT FIELDS

Typical compressed packet



Optional fields

- Final CRC32C
  - If the peer validating packet signatures and the packet has a ValidationAlg, can skip this.
  - Covers entire packet from CID to end of compressed body.
- In-band dictionary definitions
  - If new fields are to be learned (e.g. a KeyID), can be done in-line to avoid sending as separate state.
  - Peer must still ACK the definition before use.

FixedHeader Compression

v = version, t = packetType (PT), h = headerLen (HL),
l = packetLen (PL), m = hopLimit (HOP) c = return code (RC),
r = reserved, i = Context ID (CID)

	BYTE	HL	PL	HOP	RC	PT
Uncompressed packet						
`000vvvvr t{8} l{16} m{8} c{8} r{8} h{8}`	8	8	16	8	8	8
Compressed packet						
`10 i{3} crc{3} compressed_fh`						
`110 i{6} crc{7} compressed_fh`						
`111 reserved`						
`001vvvvr t{8} l{16} m{8} c{8} r{8} h{8}`	8	8	16	8	8	8
`010vvvvt ttllllll m{8}`	3	0	6	8	0	3
`011vvvvt tthhhhhl l{8}`	3	5	9	0	0	3
`100vvvvt tthhhhhl l{8} m{8}`	4	5	9	8	0	3

green: full len

Version field reduced to 4 bits in all packets
PacketType greater than 7 must use 8-byte fixed header

CRCs

- As per RFC 4995
  - Calculated over the preamble and CID (e.g. ‘10 i{3}’), so there are no leading 0s.
  - Initialize CRC register to all ‘1’s.
  - crc{3} = 1 + x + x^3
  - crc{7} = 1 + x + x^2 + x^3 + x^6 + x^7
  - The given combinations (i{3} crc{3}, i{6} crc{7}) will detect all bit errors over the 3 or 6 bit field.
- To verify the CRC
  - Zero the CRC bits then calculate over the 1 or 2 bytes

TL Compression

(t = type bit, l = length bit, z = compressor key)

Uncompressed Format: (“000” fixed header)

```
t{16} l{16}                              (16-bit T & 16-bit L)
```

Compressed Formats: (“1xx” fixed header)

```
0zzzllll                                 ( 3-bit Z &  4-bit L)
10zzzzzz                                 ( 6-bit Z &  fixed L)
110zzzzl l{8}                            ( 4-bit Z &  9-bit L)
1110tttt t{8} tttlllll                   (15-bit T &  5-bit L)
11110zzz z{8}                            (learned, next slide)
111110tt t{8} ttttttll l{8}              (16-bit T & 10-bit L)
1111110z z{16}                           (learned, next slide)
11111110 z{24}                           (learned, next slide)
11111111 t{16} l{16}                     (16-bit T & 16-bit L)
```

Formats with a ‘t’ encode dictionary misses.
Formats with a ‘z’ encode dictionary hits.

Learned Dictionaries

```
Variable length keys for dynamic TL + V dictionaries
11110    z{11} -- 2 bytes (2K entries)
1111110  z{17} -- 3 bytes (128K entries)
11111110 z{24} -- 4 bytes (16M entries)
```

- Used to encode TL + V tokens
  - ‘Token’ type: a fixed TLV string
  - ‘Counter’ type: a base plus an offset
- Token type: e.g. keyid, public keys, and prefix
- Counter type: e.g. times, sequence numbers

Counter Types

- A 'Z' value followed by a signed offset
  - $0 <= offset < 256$: 0bbbbbbb (1 byte)
  - $256 <= offset < 2^{15}$: 10b{14} (2 bytes)
  - $2^{15} <= offset < 2^{22}$: 110b{21} (3 bytes)
  - $2^{22} <= offset < 2^{29}$: 1110b{28} (4 bytes)
  - $2^{29} <= offset < 2^{36}$: 11110b{35} (5 bytes)
- Sign extended to length of counter

Structure

- TL compressors
  - Will always begin on a 'T' and end before a 'V'.
  - May consume multiple 'TL' pairs before first 'V', if they are all common values.
- TLV compressors
  - Will always begin on a 'T' and end with a 'V'
  - 'Token' type may consume multiple static TLV tuples.
  - 'Counter' type one TLV
- Unambiguous
  - Because all code words start on a 'T' and all 'T's are unambiguous, there is a 1:1 encode/decode.

Examples of TL Compression

```
{0x00,0x03,0x00,0x04, /* validation alg, len= 4 */
 0x00,0x02,0x00,0x00, /* CRC32C */
 0x00,0x04,0x00,0x04,
 (4-byte CRC output) } /* validation payload, len= 4 */
➔ 0b10000100 (4-byte CRC output)
➔ 12 bytes -> 1 byte

{0x00,0x09,0x00,0x20, /* type = keyid, len= 32 */
 (32-byte keyid) }
➔ 0b10000010 (32-byte keyid)
➔ 4 bytes -> 1 byte

{0x00,0x01,0x00,0x05, /* type = NameSeg, len = 5 */
 'h','e','l','l','o'}
➔  0b00010101'hello'
➔ 4 bytes -> 1 byte
```

Example of TLV Token Compression

In state exchange

```
0b11011100.00100010   // Token Definition (len = 36)
0b11111000.00000000   // z = 0xF800
{0x00,0x09,0x00,0x20, /* type = keyid, len= 32 */
 0x5c,0x23,0x4c,0x28,0x50,0xda,0x20,0x7b,
 0x88,0x25,0x8b,0xf3,0x62,0x61,0x96,0xd8,
 0xf0,0x60,0x76,0x38,0xa2,0xd4,0xe0,0xe2,
 0x49,0xb2,0xa9,0xaf,0xce,0xb8,0x85,0x59}
```

In packet

```
{0x00,0x09,0x00,0x20, /* type = keyid, len= 32 */
 0x5c,0x23,0x4c,0x28,0x50,0xda,0x20,0x7b,
 0x88,0x25,0x8b,0xf3,0x62,0x61,0x96,0xd8,
 0xf0,0x60,0x76,0x38,0xa2,0xd4,0xe0,0xe2,
 0x49,0xb2,0xa9,0xaf,0xce,0xb8,0x85,0x59}
```

➔ 0b11111000.00000000

➔ 36 bytes -> 2 bytes

Example of TLV Counter Compression

In state exchange

```
0b11011110.00001100   // Counter Definition (len = 12)
0b11111000.00000001   // z = 0xF801
{0x00,0x06}           // type 6, 2015-08-19T19:26:51.000Z
{0x00,0x00,0x01,0x4f,0x48,0xee,0x25,0xf8}
```

In packet

```
{0x00,0x06,0x00,0x08, /* type = expiry, len= 8 */
0x00,0x00,0x01,0x4f,0x49,0x25,0x14,0x78}// 2015-08-19T20:26:51.000Z
```

➔ 0b11111000.00000001

0b11100000.00110110.11101110.10000000

➔ 12 bytes -> 6 bytes

Example state exchange packet

```
11000011.01010000.0100100  // fh: ver=1, pt=5, hl=8, pl=72
0b11001010.00100000        // Dictionary Def (len = 64)
0b00010010                 // seqnum (len = 2)
{0x03,0xc8}                // seqnum
0b11000100.00100110        // Token Definition (len = 38)
0b11111000.00000000        // z = 0xF800
{0x00,0x09,0x00,0x20,      /* type = keyid, len= 32 */
 0x5c,0x23,0x4c,0x28,0x50,0xda,0x20,0x7b,
 0x88,0x25,0x8b,0xf3,0x62,0x61,0x96,0xd8,
 0xf0,0x60,0x76,0x38,0xa2,0xd4,0xe0,0xe2,
 0x49,0xb2,0xa9,0xaf,0xce,0xb8,0x85,0x59}
0b10000100                 // valalg CRC32, valpayload
{4-byte string}            // crc32c value

T_DICT = 0x0005, T_SEQNUM = 0x0001, T_TOKEN = 0x0002
Note: uses normal compression, so lengths are all in
expanded sizes.
```

Example state exchange ACK

```
11000011.01000110.00110000 // fh: ver=1, pt=5, hl=3, pl=13
0b01010011                 // ACK (len = 3)
0b00010010                 // seqnum (len = 2)
{0x03,0xc8}                // seqnum
0b10000100                 // valalg CRC32, valpayload
{0x32,0x4a,0x96,0x13}      // crc32c value

T_ACK = 0x0006, T_SEQNUM = 0x0001, T_SELECTIVE = 0x0002
13 bytes to communicate 2 bytes of data with a 4-byte CRC.
```

In-band example

```
0x0101, 0x0066, 0x2000, 0x0008,     // FixedHeader
0x0001, 0x004F,                     // Interest
0x0000, 0x0025,                     // Name
0x0001, 0x0008, ‘parc.com’          // NameSeg
0x0001, 0x0010, ‘compression.pptx’, // NameSeg
0x0013, 0x0001, {0x01},             // Chunk
0x0002, 0x0020, {32-byte string},   // KeyId restriction
0x0003, 0x0004, 0x0004, 0x0000,     // Validation Alg, CRC32C
0x0004, 0x0004, {4-byte string}     // Validation Payload
0x0005, 0x000F,                     // Dictionary Def
0b00010010, {0x03, 0xc8}            // seqnum (len = 2)
0b00100110                  // Token Definition (len = 12)
0b11111000.00000000         // z = 0xF800
0b00010001, {58}            // offset = 58 bytes back (KeyId)
0b00100001, {36}            // length = 36
0b10000100                  // valalg CRC32, valpayload
{4-byte string}             // crc32c value
T_DICT = 0x0005, T_SEQNUM=0x0001, T_TOKEN=0x0002,
T_OFFSET=0x0001, T_LENGTH = 0x0002
```

Static TL Dictionary

Z	Token	Notes
10000000	0x0002 0x0000	T_CRC32 (0)
10000001	0x0002 0x0004	T_KEYIDRESTR (4)
10000010	0x0002 0x0020	T_KEYIDRESTR (32)
10000011	0x0003 0x0004	T_VALALG (4)
10000100	0x0003 0x0004 0x0002 0x0000 0x0004 0x0004	Validation Alg w/ CRC32-C Validation Payload
10000101	0x0003 0x000C	T_INTFRAG (12)
10000110	0x0003 0x000C 0x0004 0x0008 0x0009 0x0004	Validation Alg w/ HMAC-SHA256, KeyId (4)
10000111	0x0003 0x0012	T_VALALG (18)
10001000	0x0003 0x0014 0x0004 0x0010 0x0009 0x0004	Validation Alg w/ HMAC-SHA256, KeyId (4), SigTime (8)
10001001	0x0003 0x0020	T_OBJHASHRESTR (32)
10001010	0x0003 0x0034 0x0006 0x0030 0x0009 0x0020	Validation Alg w/ RSA-SHA256 KeyId, SigTime (8)
10001011	0x0004 0x0004	T_VALPLD (4)
10001100	0x0004 0x000E	T_HMAC-SHA256
10001101	0x0004 0x0010	T_VALPLD (16)
10001110	0x0004 0x0014	T_OBJFRAG (20)
10001111	0x0005 0x0001	T_PLYTYPE (1)
10010000	0x0006 0x0008	T_EXPIRY (8)
10010001	0x0008 0x0011	T_IPID (17)
10010010	0x0009 0x0004	T_KEYID (4)
10010011	0x0009 0x0010	T_KEYID (16)
10010100	0x0009 0x0020	T_KEYID (32)
10010101	0x000B 0x00A2	T_PUBKEY (162)
10010110	0x000B 0x0126	T_PUBKEY (294)
10010111	0x000B 0x0226	T_PUBKEY (550)
10011000	0x000F 0x0008	T_SIGTIME (8)
10011001	0x0019 0x0001	T_ENDCHUNK (1)
10011010	0x0019 0x0002	T_ENDCHUNK (2)
10011011	0x0019 0x0004	T_ENDCHUNK (4)
10011100	0x0003 0x00CE 0x0006 0x00CA 0x0009 0x0020	ValAlg + RSA-SHA256 + KeyId + PubKey
10011101		

Variable Length Dictionaries

Z	Type	Length
00000000	0x0000	4-bit
00010000	0x0001	4-bit
00100000	0x000A	4-bit
00110000	0x0013	4-bit
01000000		4-bit
01010000		4-bit
01100000		4-bit
01110000		4-bit

Z	Type	Length
11000000	0x0000	9-bit
11000010	0x0001	9-bit
11000100	0x0002	9-bit
11000110	0x0003	9-bit
11001000	0x0004	9-bit
11001010	0x0005	9-bit
11001100	0x0006	9-bit
11001110		9-bit
11010000		9-bit
11010010		9-bit
11010100		9-bit
11010110		9-bit
11011000		9-bit
11011010	Dict ACK	9-bit
11011100	Token Def	9-bit
11011110	Counter Def	9-bit

Dict Act = Dictionary ACK field
Token Def = Token definition field
Counter Def = Counter definition field