Header Compression for TLV-based Packets

ICN RG 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data (name)</td>
<td>16</td>
</tr>
<tr>
<td>Uncompressed</td>
<td>48</td>
</tr>
<tr>
<td>gzip -9</td>
<td>77</td>
</tr>
<tr>
<td>bzip2 -9</td>
<td>75</td>
</tr>
<tr>
<td>MPPC (RFC 2118)</td>
<td>42</td>
</tr>
<tr>
<td>TLV compression</td>
<td>28</td>
</tr>
</tbody>
</table>
Example (Content Object)

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data (name, payload, pubkey, keyid, sig)</td>
<td>372</td>
</tr>
<tr>
<td>Uncompressed</td>
<td>436</td>
</tr>
<tr>
<td>gzip -9</td>
<td>461</td>
</tr>
<tr>
<td>bzip2 -9</td>
<td>574</td>
</tr>
<tr>
<td>MPPC (RFC 2118)</td>
<td>448</td>
</tr>
<tr>
<td>TLV compression</td>
<td>396</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th></th>
<th>H (bit-aligned)</th>
<th>H (byte-aligned)</th>
<th>2+2</th>
<th>1/3/5</th>
<th>TL comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TL only</td>
<td>4.9</td>
<td>8.0</td>
<td>32.0</td>
<td>18.9</td>
<td>8.0</td>
</tr>
<tr>
<td>TL + V</td>
<td>8.4</td>
<td>11.7</td>
<td>88.3</td>
<td>55.4</td>
<td>14.8</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Optional</th>
<th>0b10 + 3-bit CID + 3-bit CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3, 4, or 8 byte Fixed Header</td>
</tr>
<tr>
<td></td>
<td>Compressed TLs (1 or 2 bytes)</td>
</tr>
<tr>
<td></td>
<td>Compressed TLVs (2, 3, 4 bytes)</td>
</tr>
<tr>
<td></td>
<td>Compacted TLs (3, 4, 5 bytes)</td>
</tr>
<tr>
<td></td>
<td>crc32c footer over uncompressed</td>
</tr>
<tr>
<td></td>
<td>Dictionary definitions based on packet fields (with seqnum and CRC)</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Uncompressed packet</th>
<th>BYTE</th>
<th>HL</th>
<th>PL</th>
<th>HOP</th>
<th>RC</th>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000vvvvr t{8} l{16} m{8} c{8} r{8} h{8}</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compressed packet</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10 i{3} crc{3} compressed_fh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110 i{6} crc{7} compressed_fh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111 reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001vvvvr t{8} l{16} m{8} c{8} r{8} h{8}</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>010vvvvt ttlllllll m{8}</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>011vvvvt tthhhhhh l{8}</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>100vvvvt tthhhhhh l{8} m{8}</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

*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.
  – \( \text{crc}\{3\} = 1 + x + x^3 \)
  – \( \text{crc}\{7\} = 1 + x + x^2 + x^3 + x^6 + x^7 \)
  – 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

\((t = \text{type bit}, \ l = \text{length bit}, \ z = \text{compressor key})\)

**Uncompressed Format:** (“000” fixed header)
\(t\{16\} \ l\{16\}\)  \(\)  (16-bit \(T\) & 16-bit \(L\))

**Compressed Formats:** (“1xx” fixed header)

0zzzlllll  \(\)  (3-bit \(Z\) & 4-bit \(L\))
10zzzzzzz \(\)  (6-bit \(Z\) & fixed \(L\))
110zzzz0 l{8} \(\)  (4-bit \(Z\) & 9-bit \(L\))
1110tttt t{8} tttllllll \(\)  (15-bit \(T\) & 5-bit \(L\))
11110zzz z{8} \(\)  (learned, next slide)
111110tt t{8} tttttttll 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
  – $0 \leq \text{offset} < 2^{256}$: \(0\text{bbbbbbbb} \) (1 byte)
  – $2^{256} \leq \text{offset} < 2^{15}$: \(10\text{b\{14\}} \) (2 bytes)
  – $2^{15} \leq \text{offset} < 2^{22}$: \(110\text{b\{21\}} \) (3 bytes)
  – $2^{22} \leq \text{offset} < 2^{29}$: \(1110\text{b\{28\}} \) (4 bytes)
  – $2^{29} \leq \text{offset} < 2^{36}$: \(11110\text{b\{35\}} \) (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 */
\Rightarrow 0b10000100 (4-byte CRC output)
\Rightarrow 12 bytes \rightarrow 1 byte

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

{0x00,0x01,0x00,0x05, /* type = NameSeg, len = 5 */
  'h','e','l','l','o'}
\Rightarrow 0b00010101‘hello’
\Rightarrow 4 bytes \rightarrow 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, 0xda,
  0xf0, 0x60, 0x76, 0x38, 0xa2, 0xd4, 0xe0, 0xe2,
  0x49, 0xb2, 0xa9, 0xa2, 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, 0xda,
  0xf0, 0x60, 0x76, 0x38, 0xa2, 0xd4, 0xe0, 0xe2,
  0x49, 0xb2, 0xa9, 0xa2, 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
0b100000100                              // valalg CRC32, valpayload
{4-byte string}                        // crc32c value
T_DICT = 0x0005, T_SEQNUM=0x0001, T_TOKEN=0x0002,
T_OFFSET=0x0001, T_LENGTH = 0x0002
```

T_DICT = 0x0005, T_SEQNUM=0x0001, T_TOKEN=0x0002,
T_OFFSET=0x0001, T_LENGTH = 0x0002
## Static TL Dictionary

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

<table>
<thead>
<tr>
<th>Z</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0x0000</td>
<td>4-bit</td>
</tr>
<tr>
<td>00010000</td>
<td>0x0001</td>
<td>4-bit</td>
</tr>
<tr>
<td>00100000</td>
<td>0x000A</td>
<td>4-bit</td>
</tr>
<tr>
<td>00110000</td>
<td>0x0013</td>
<td>4-bit</td>
</tr>
<tr>
<td>01000000</td>
<td></td>
<td>4-bit</td>
</tr>
<tr>
<td>01010000</td>
<td></td>
<td>4-bit</td>
</tr>
<tr>
<td>01100000</td>
<td></td>
<td>4-bit</td>
</tr>
<tr>
<td>01110000</td>
<td></td>
<td>4-bit</td>
</tr>
<tr>
<td>11000000</td>
<td>0x0000</td>
<td>9-bit</td>
</tr>
<tr>
<td>11000100</td>
<td>0x0002</td>
<td>9-bit</td>
</tr>
<tr>
<td>11000110</td>
<td>0x0003</td>
<td>9-bit</td>
</tr>
<tr>
<td>11001000</td>
<td>0x0004</td>
<td>9-bit</td>
</tr>
<tr>
<td>11001010</td>
<td>0x0005</td>
<td>9-bit</td>
</tr>
<tr>
<td>11001100</td>
<td>0x0006</td>
<td>9-bit</td>
</tr>
<tr>
<td>11001110</td>
<td></td>
<td>9-bit</td>
</tr>
<tr>
<td>11010000</td>
<td></td>
<td>9-bit</td>
</tr>
<tr>
<td>11010100</td>
<td></td>
<td>9-bit</td>
</tr>
<tr>
<td>11010110</td>
<td></td>
<td>9-bit</td>
</tr>
<tr>
<td>11011000</td>
<td></td>
<td>9-bit</td>
</tr>
<tr>
<td>11011010</td>
<td>Dict ACK</td>
<td>9-bit</td>
</tr>
<tr>
<td>11011100</td>
<td>Token Def</td>
<td>9-bit</td>
</tr>
<tr>
<td>11011110</td>
<td>Counter Def</td>
<td>9-bit</td>
</tr>
</tbody>
</table>

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