Robust Header Compression for CCNx 1.0

Draft 3

Marc Mosko

Aug 26, 2015

ROHC Overview (RFC 5795)

- Modes of operation
  - Unidirectional mode (U-mode)
  - Bidirectional Optimistic (O-mode)
  - Bidirectional Reliable (R-mode)
- Compressor/decompressor states
  - Initialization and Refresh (IR) state
  - First Order (FO) state
  - Second Order (SO) state

Modes

- Unidirectional
  - Only from compressor to decompressor
- Bidirectional Optimistic
  - Reverse channel used for error and recovery of optional ack of significant updates.
- Bidirectional Reliable
  - More use of feedback for updates and prevent loss of synch between compressor/decompressor.

States

- Initialization and Refresh
  - Full packet headers sent
- First Order
  - Detected and stored static fields
  - Some dynamic packet field differences
- Second Order
  - Suppress all dynamic fields (e.g. sequence numbers)

ROHC Mode Applicability to CCN

- CCN relies on bidirectional channels
  - O-mode and R-mode most applicable
  - U-mode not necessary
- Initialization and Refresh
  - CCN encoding has some static structures that could be compressed even before First Order mode.

ROHC State Applicability To CCNx

- Initialization and Refresh State
  - Compress common TL pairs
  - Compress FixedHeader
- First Order State
  - Compress common values
    - Name prefixes, KeyIDs, EndChunkNumber, etc.
- Second Order State
  - Compress larger dynamic structures
    - ValidationAlg, Chunked Name, time fields

Motivation

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

What we want to develop

- Static TL compression
  - Allows reducing the overhead caused by TL encoding (2+2 and 1/3/5) *without state exchange*.
  - Streaming operation, does not require going back to fix-up values.
- Dictionary learned replacement
  - Learn strings like Key IDs and Public Keys. Those are random byte strings.
  - Use (base, offset) encoding for things like Chunks or times or serial numbers.

Outline of Algorithm

- Fixed header has a "compressed" flag
  - If not set, uses 8 byte FH and 2+2 TLs
  - If set, used 3, 4, or 8 byte FH and 1 – 5 byte TLs
- In "compressed" mode
  - Uses dictionary replacement
    - 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 in-band 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.

Dictionary Exchange

```
Exchange := DICT Validation
DICT := SEQNUM 1*ENTRY
ENTRY := RESET / DEF
DEF := TOKEN / COUNTER
TOKEN := LEARNED_Z / STATIC_Z
LEARNED_Z := Z_STRING / Z_OFFSET
Z_STRING := Z STRING
Z_OFFSET := Z OFFSET
OFFSET := BACK LENGTH
STATIC_Z := Z STRING LENGTH
COUNTER := Z BASE

RESET := <0-length token>
Z := <Defined Z value>
STRING := *OCTET
BASE := Number
BACK := Number
LENGTH := Number
Validation := <normal Validation section>
```

- A dictionary is a SEQNUM and one or more entries.
- A RESET entry clears all previous definitions.
- A definition can be a TOKEN or a COUNTER.
- A COUNTER is a Z value taken from the 'Learned' types followed by the base to apply an offset to.
- A TOKEN is Learned or Static.
  - Static used to transfer non-standard initial dictionaries.
- Learned TOKENs can be defined immediately (out-of-band) or in footer of packet (in-band).

State Exchange

- Out-of-band
  - Use a separate packet with FixedHeader PacketType = Dictionary
  - Sends one or more definitions of Token Type and Counter Type.
- In-band
  - To facilitate streaming, a packet is sent with learned values uncoded.
  - Footer sends dictionary definitions, using (backwards_offset, length) back in to the packet.
  - Still carries seqnum for reliable state exchange.
  - Has own CRC

Bi-directional Exchanges

- Each dictionary is one-way
  - So there is bi-directional state exchange.
  - Can piggyback ACK on token definitions.
- TBD:
  - Is this the right approach?
  - Fields like KeyID and the name prefix will always be symmetric.

State Exchange (out of band)

- Uses FixedHeader with PacketType=Dictionary
  - A link-local packet, not forwarded
  - Includes sequence number for reliable delivery
  - Peer sends ACKs
  - Has Validation section
    - CRC32C, HMAC, or signature
  - Can use one of the encrypted formats
    - AES-GCM authenticated encryption
    - See separate encryption document

State exchange ACK

```
Packet := FixedHeader ACK Validation
ACK := SEQNUM [ACCEPTED / NACK_FLAG]
SEQNUM := number
ACCEPTED := <list of Z values>
NACK_FLAG := <0-length TLV>
Validation := <normal, e.g. HMAC or CRC or signature)
```

When using backwards references with in-band state exchange, it is the responsibility of the sender to not go backwards beyond the receivers buffer size (defined in initialization).

A receiver can ACK all Z values defined by sending ACK without a SELECTIVE section. It can REJECT all Z values by sending an empty ACCEPTED section. It can ACK some and REJECT some by including a subset of Z values in the ACCEPTED section.

If a receiver rejects a SEQNUM, it SHOULD send a REJECT rather than time out.

If the receiver rejects a SEQNUM because of validation failure, it MAY send a NACK, though if it is an attack and remaining silent could be better. A NACK does not REJECT Z values, it only indicates a reception error.

State exchange processing

- SEQNUM is monotonically ordered
- Definitions applied in order
  - In order by SEQNUM and in order as listed.
- Withdrawing a definition

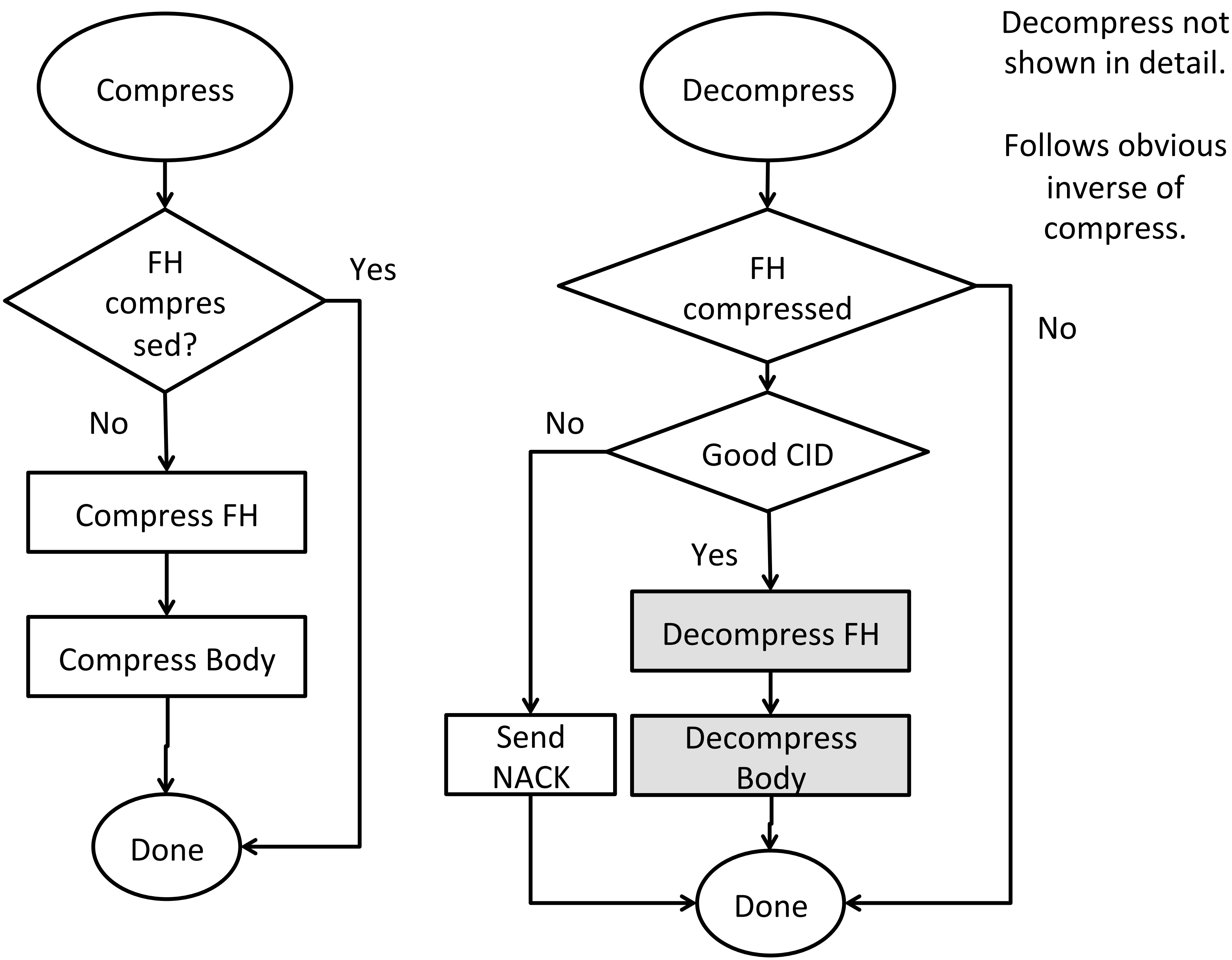
In-band exchange

- Add a footer of type T_DICT
  - Same type as used in out-of-band
  - Comes after the packet's Validation section
- The 'V' is always compressed
  - Contains seqnum (for ACK)
  - Contains one or more T_TOKEN or T_COUNTER definitions.
  - Each definition uses {back_offset, length} pair to indicate which part of packet is the string to match.

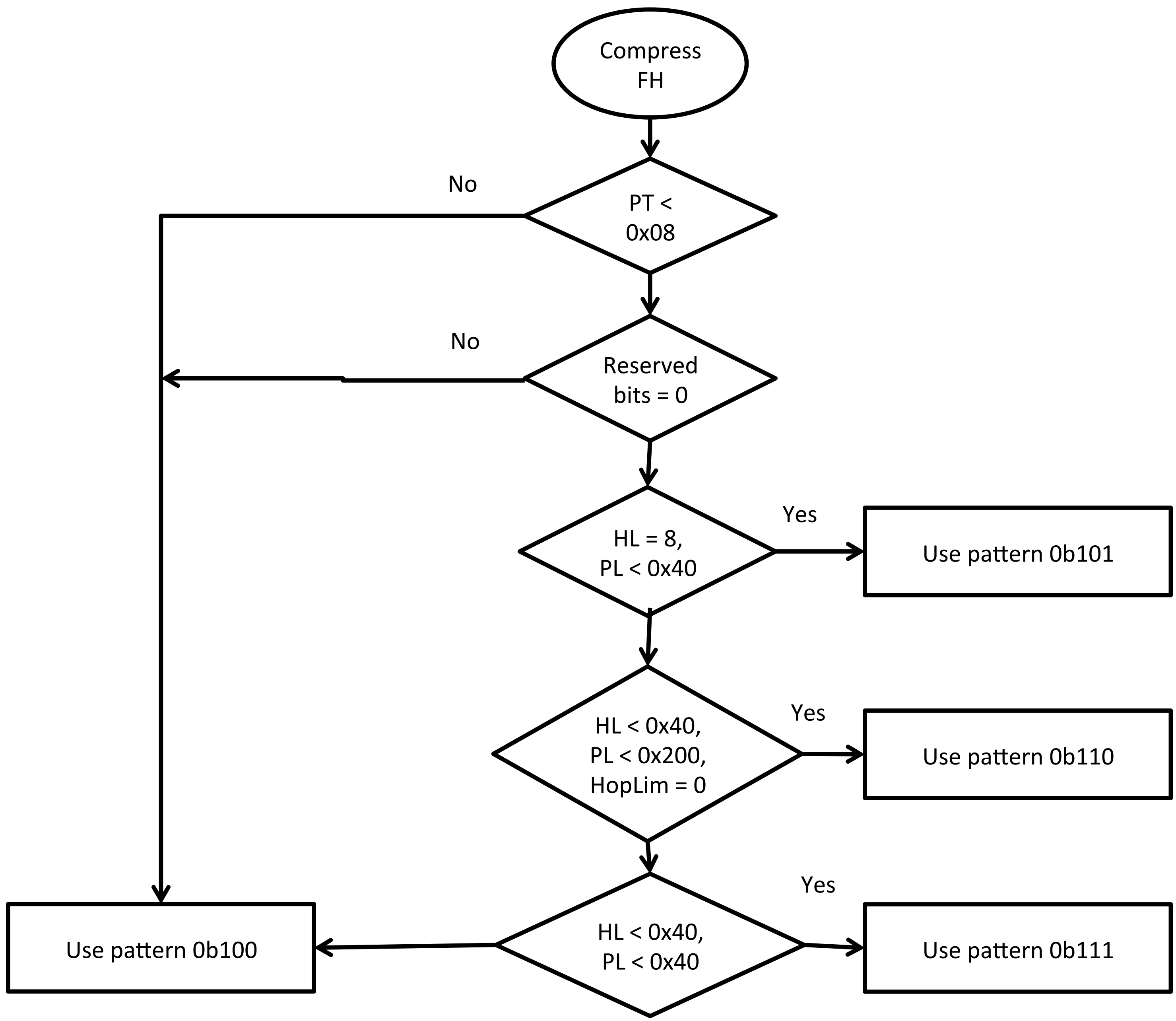
Why backward offsets

- Allows different implementations to keep different size buffers.
  - Can throw away head of packet, if desired.
  - Use Selective ACK to only accept what could be buffered.

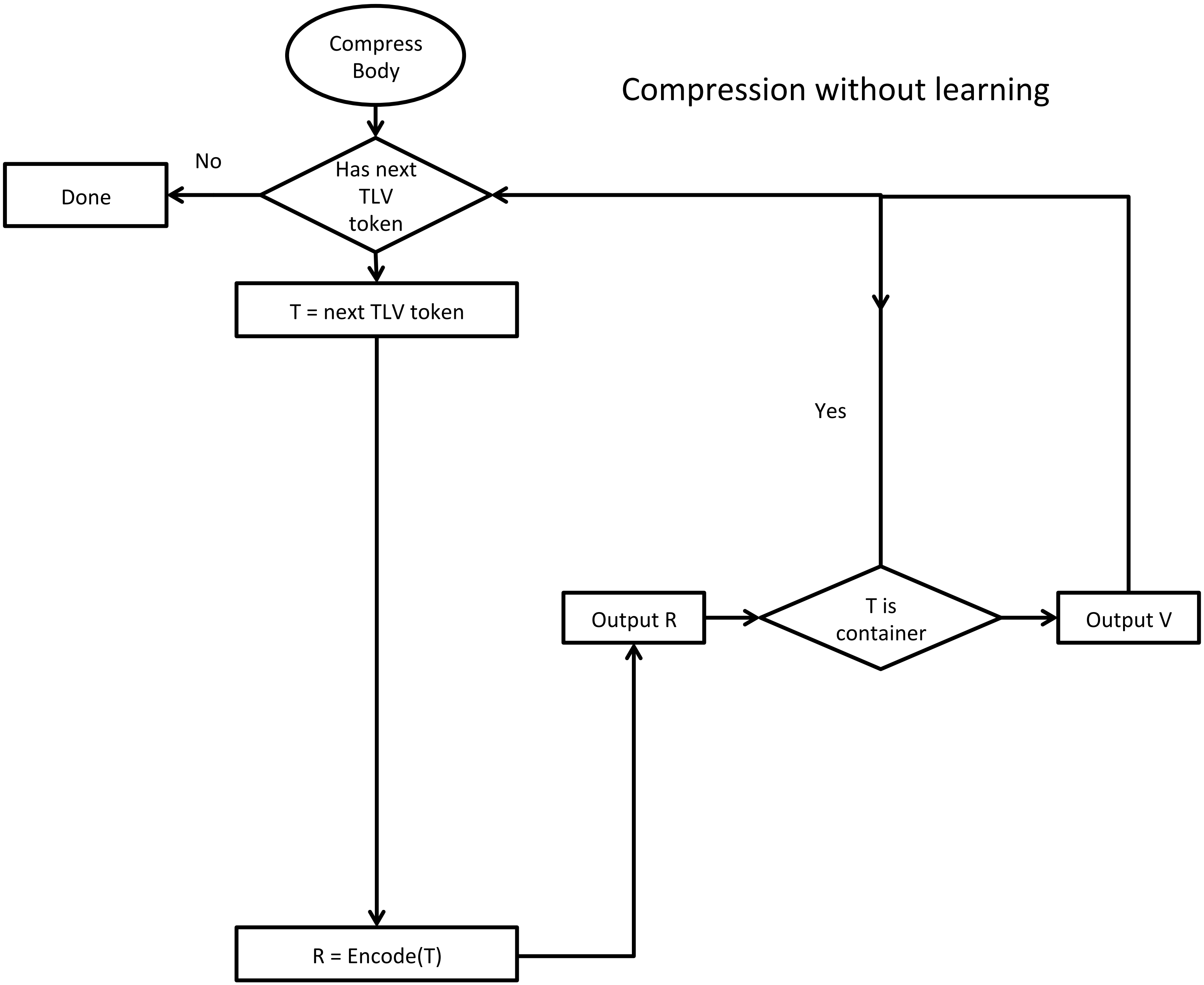
ALGORITHM FLOWCHARTS



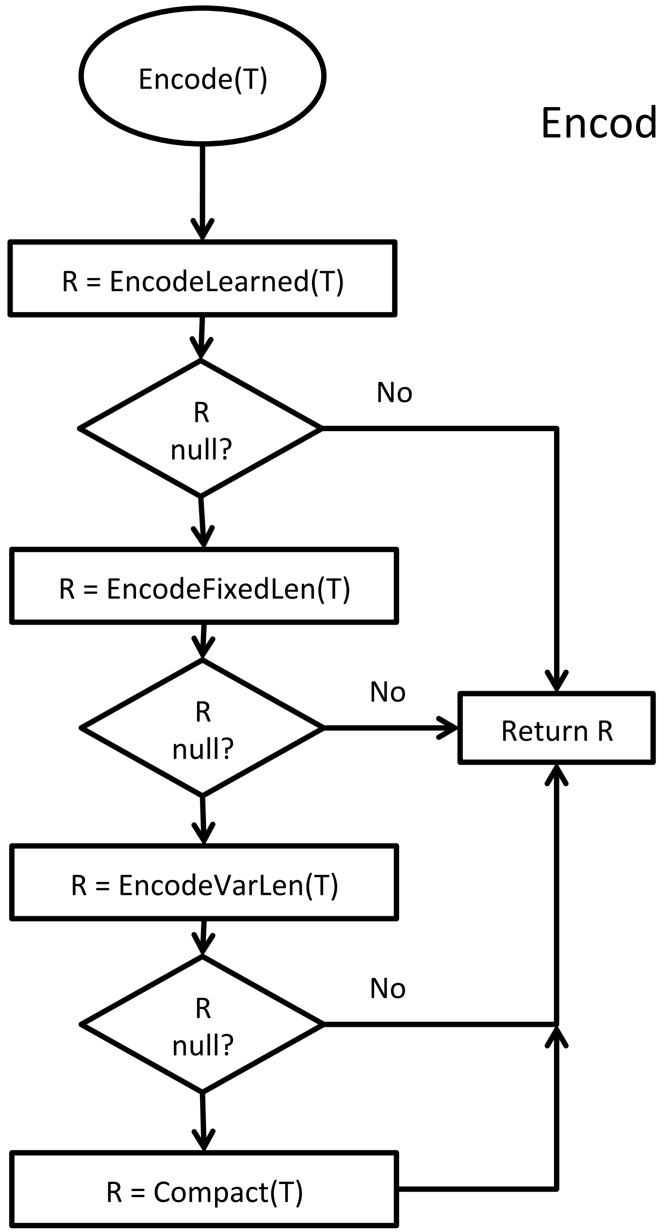
Compress
FH compres sed?
Yes
No
Compress FH
Compress Body
Done
Decompress
Decompress not shown in detail.
Follows obvious inverse of compress.
FH compressed
No
Good CID
No
Yes
Decompress FH
Decompress Body
Send NACK
Done

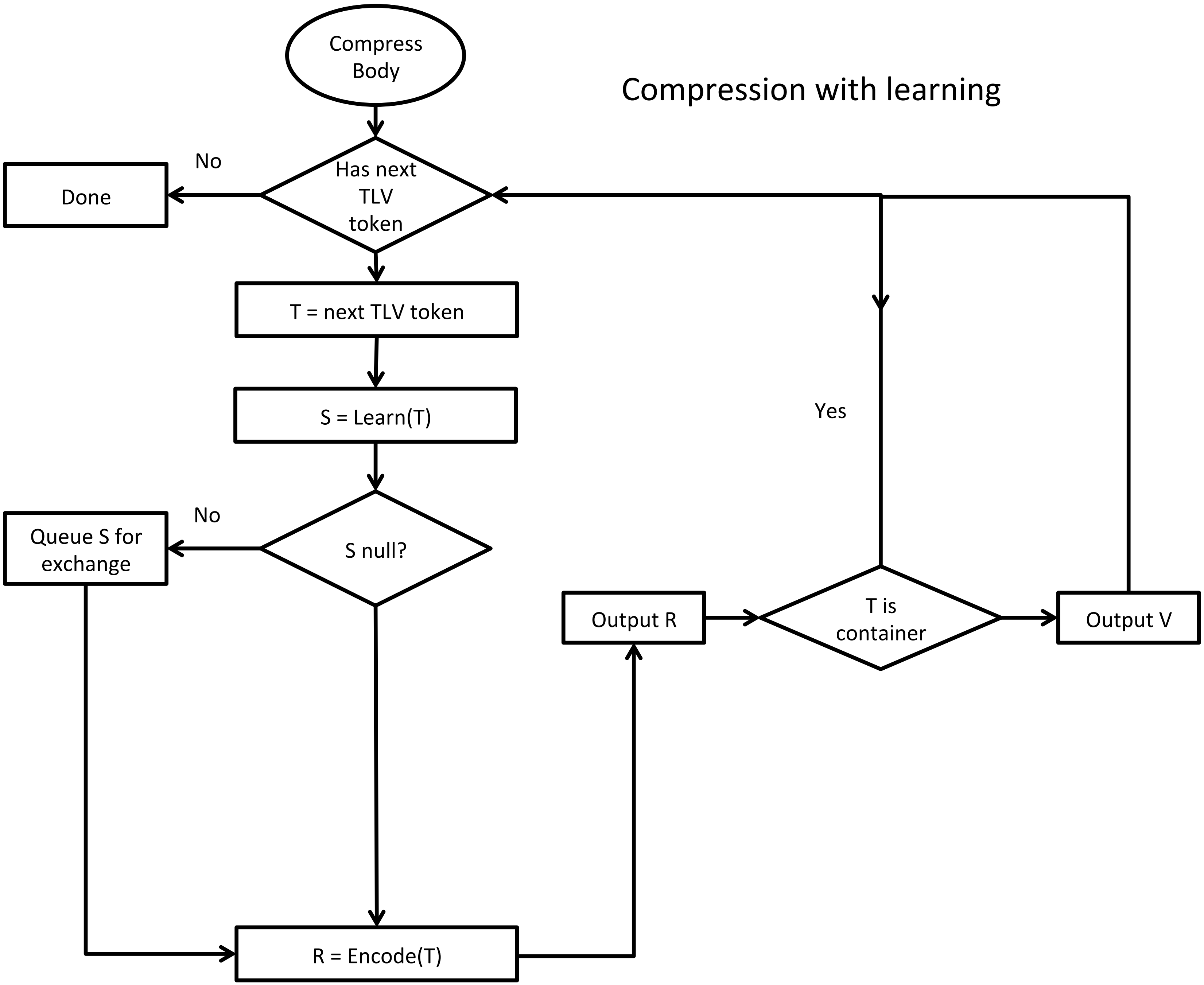


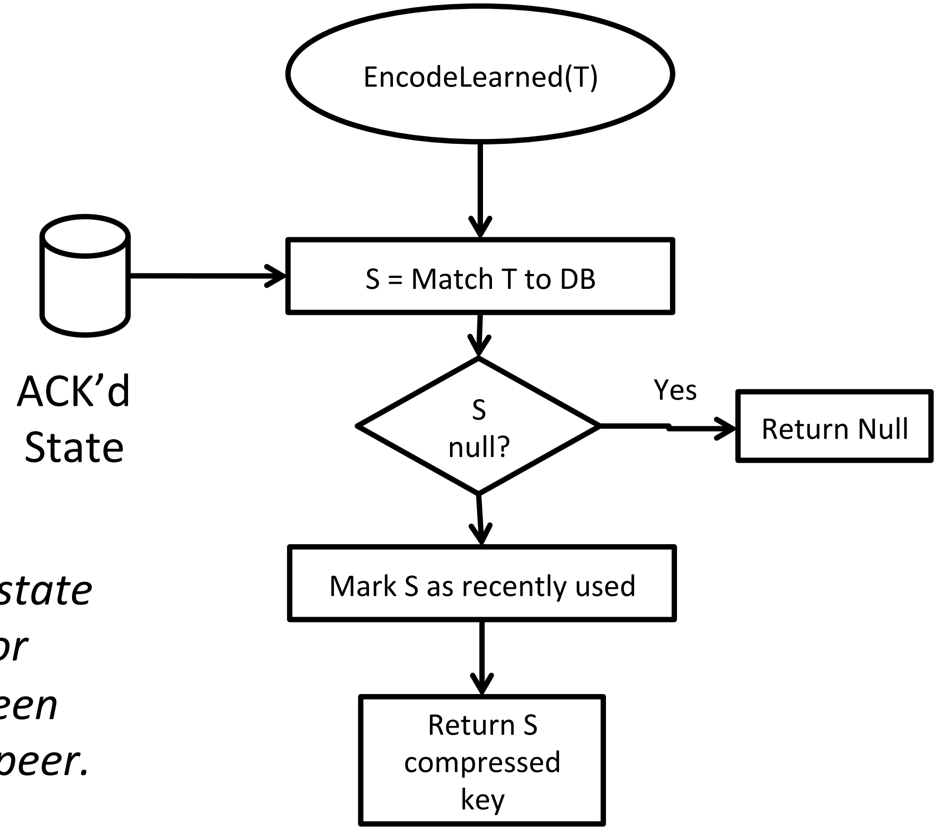
Compress FH
PT < 0x08
No
Reserved bits = 0
No
HL = 8, PL < 0x40
Yes
Use pattern 0b101
HL < 0x40, PL < 0x200, HopLim = 0
Yes
Use pattern 0b110
HL < 0x40, PL < 0x40
Yes
Use pattern 0b111
Use pattern 0b100


Compress Body
Compression without learning
No
Has next TLV token
Done
T = next TLV token
Yes
Output R
T is container
Output V
R = Encode(T)

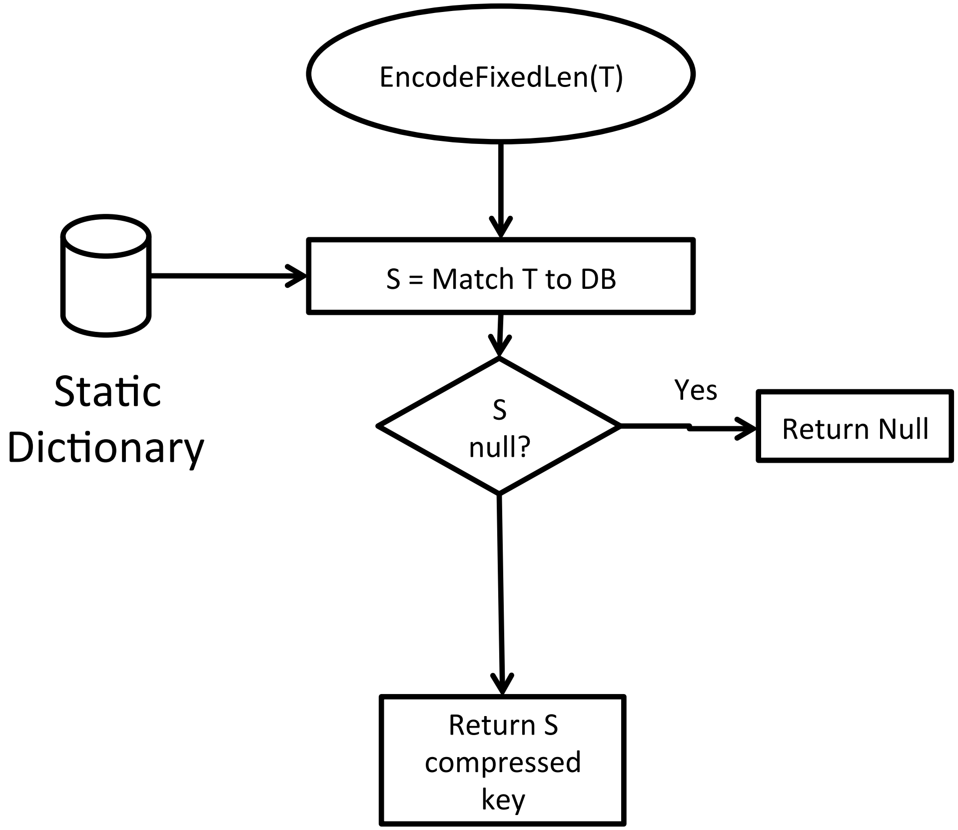
Encoding

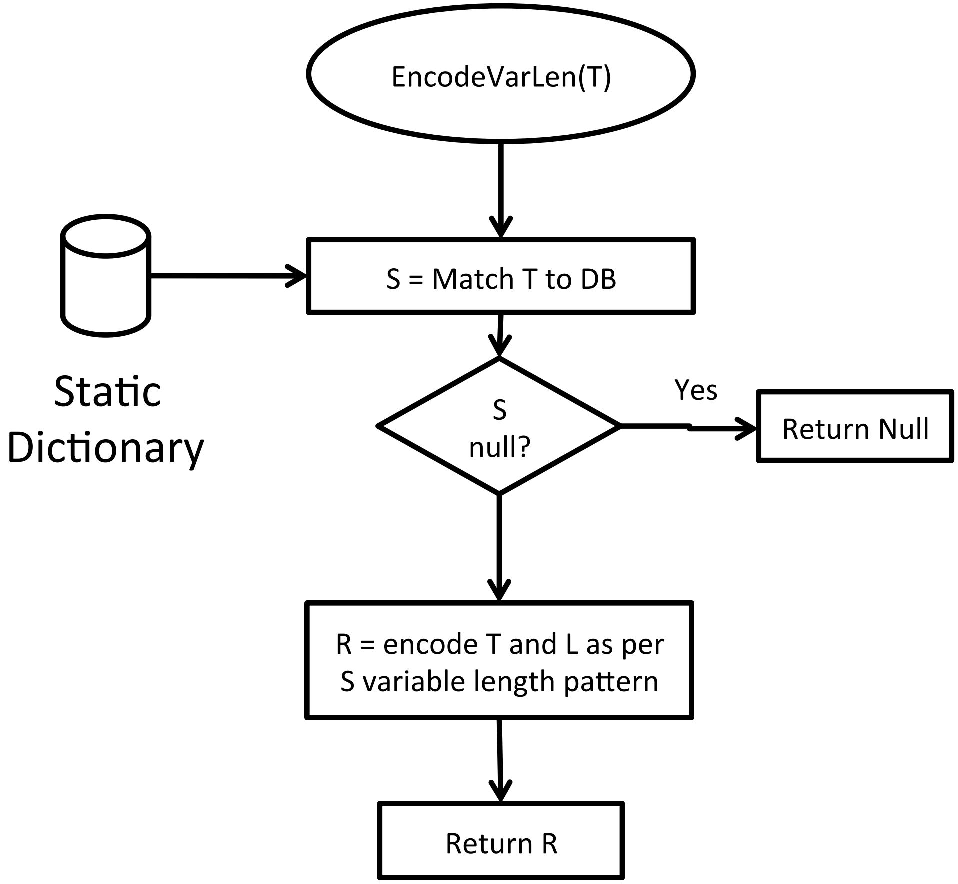

Encode(T)
R = EncodeLearned(T)
R
null?
No
R = EncodeFixedLen(T)
R
null?
No
Return R
R = EncodeVarLen(T)
R
null?
No
R = Compact(T)


Compression with learning
Compress Body
Has next TLV token
No
Done
T = next TLV token
S = Learn(T)
S null?
No
Queue S for exchange
R = Encode(T)
Output R
T is container
Yes
Output V



ACK'd state are those state vectors S queued for transfer that have been acknowledged by the peer.


EncodeFixedLen(T)
S = Match T to DB
Static Dictionary
S null?
Yes
Return Null
Return S compressed key


EncodeVarLen(T)
S = Match T to DB
Static
Dictionary
S
null?
Yes
Return Null
R = encode T and L as per
S variable length pattern
Return R

ANALYSIS

Information Theory Analysis

- Static dictionary
  - Does not require any state exchange.
  - Only analyze it for TL compression, not V.
  - There is no state transfer and is immune to packet loss or synchronization loss.
- Learned dictionary
  - Compacts TL and V
  - Analyze over whole TLV

A typical Interest

T_IntLifetime:	4 + 8 bytes, (4 bits, excluding V)
T_Interest:	4 bytes, (11 bits, assume < 1024 bytes)
T_Name:	4 bytes, (12 bits, assume < 1024 bytes)
T_NameSeg:	6x 4 bytes TL, plus 6x 8 bytes data (average) (6x 5 bits)
T_KeyIdRestr:	4 bytes + 32 bytes (5 bits, excluding V)
T_HashRestr:	4 bytes + 32 bytes (5 bits, excluding V)
T_ValAlg:	4 bytes (0 bits)
T_CRC32C:	4 bytes (2 bits)
T_ValPay:	4 bytes + 4 bytes (0 bits)

Shannon: H = E(# bits per symbol) ➔ H = 4.93 for bit-aligned data

Our coding would use E(L) = 8, so about 1.62x worse *for bit-aligned data*.
This is the price of staying byte-aligned for T and V fields.

(2+2 encoding H = 32, NDN 1/3/5 encoding H = 18.9)

Byte-aligned T and V

If we require the TL encoding be byte aligned (so V is byte aligned), then

Shannon: H = E(# bits per symbol) ➔ H = 8

Our coding would use E(L) = 8, so it is an optimal encoding of the packet.

In terms of overhead (TL) compression:

2+2 H = 32, so we should see 75% reduction in TL overhead

1/3/5 H = 18.9, so we should see 57.6% reduction in TL overhead

Including V fields

Using same Interest as previously, but including bits for V fields

Assume InterestLifetime is a 3-byte counter (encoded as 5 bytes total)
Assume first 5 name components are common prefix (encoded as 3 bytes total)
Assume last name component is chunk # (encoded as counter, 3 bytes total)
All these "predictable" fields have 0 bits of Information.

H = E(# bits per symbol) ➔ H = 8.38 (bit-aligned) or 11.69 (byte-aligned)

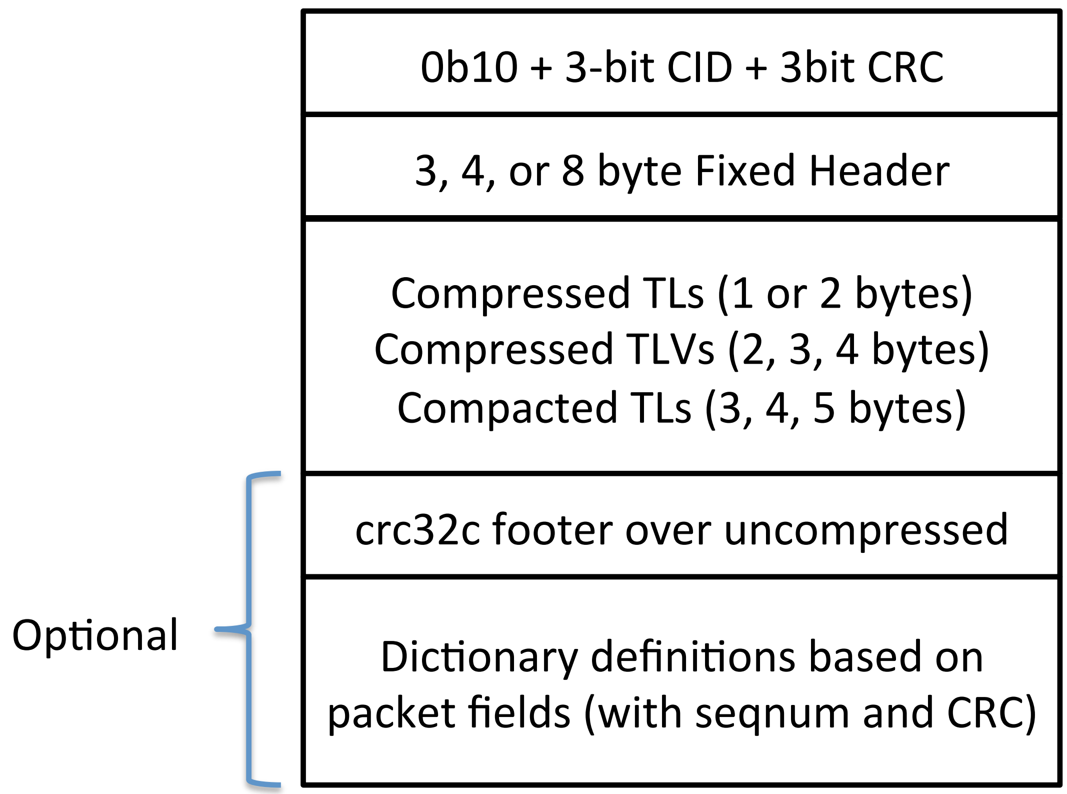
Our encoding: H = 14.77, about 1.26x over the limit (byte-aligned)

Compared to:

- 2+2 has H = 88.3, so expect 83% compression
- 1/3/5 has H = 55.4, so expect 73% compression

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{10} crc{3} 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

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

Conclusion

- Initialization stage
  - Use static dictionary to compress TLs.
  - Compress fixed header.
  - Can be used as 'native compressed' format too.
- Learning stage
  - Use reliable state exchange to compress TLVs.
  - Token type static pattern substitution.
  - Counter type for signed offset from a base.