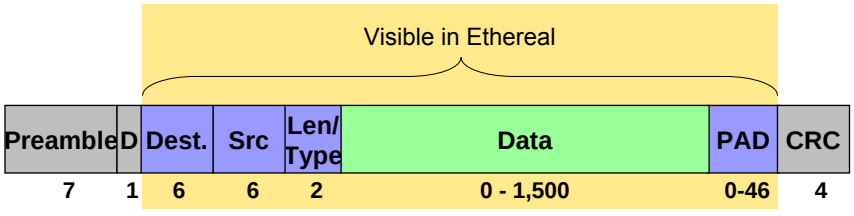
Protocol Encoding

Protocol Design – S-38.3157

Looking at a Packet: L2 frame (IPv4)

```
0000   00 a0 c5 e3 96 4e 00 0d 60 ff 7e 1a 08 00 45 00   .....N..`.~...E.
0010   00 30 8f 1a 40 00 80 06 65 49 c0 a8 00 05 41 72   .0..@...eI....Ar
0020   04 45 0b cd 00 50 e0 dc fd 0b 00 00 00 00 70 02   .E...P........p.
0030   ff ff 92 b5 00 00 02 04 05 b4 01 01 04 02         ..............
```

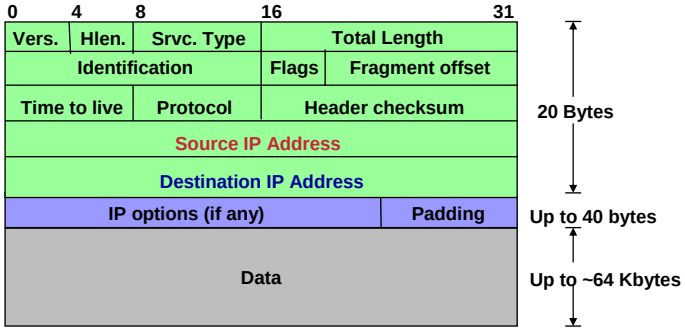


L2 Frame: fixed field sizes and offsets; optimized for “serial” processing

Looking at a Packet: L3 IPv4 Packet

```
0000   00 a0 c5 e3 96 4e 00 0d 60 ff 7e 1a 08 00 45 00   .....N..`.~...E.
0010   00 30 8f 1a 40 00 80 06 65 49 c0 a8 00 05 41 72   .0..@...eI....Ar
0020   04 45 0b cd 00 50 e0 dc fd 0b 00 00 00 00 70 02   .E...P........p.
0030   ff ff 92 b5 00 00 02 04 05 b4 01 01 04 02         ..............
```

- Fixed offsets for base IP header fields
- TLV encoding for IP option fields

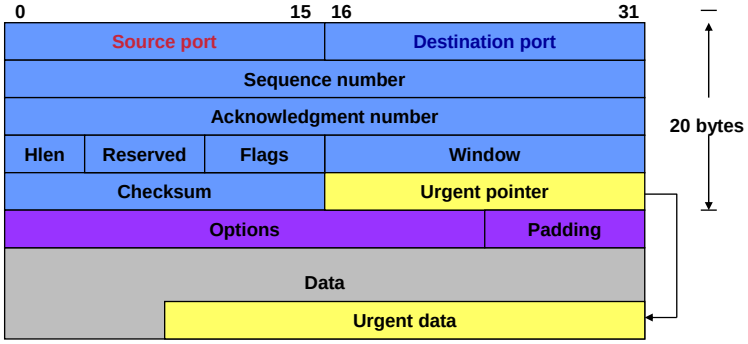


Looking at a Packet: L4 (TCP SYN)

```
0000   00 a0 c5 e3 96 4e 00 0d 60 ff 7e 1a 08 00 45 00   .....N..`.~...E.
0010   00 30 8f 1a 40 00 80 06 65 49 c0 a8 00 05 41 72   .0..@...eI....Ar
0020   04 45 0b cd 00 50 e0 dc fd 0b 00 00 00 00 70 02   .E...P........p.
0030   ff ff 92 b5 00 00 02 04 05 b4 01 01 04 02         ..............
```

MSS = 1460 2 x NOP Sack-Permitted

- Fixed offsets for base TCP header fields
- Fixed length options (NOP, End of options)
- TLV encoded options
- Option types 0 and 1 need to be known



Looking at another Packet: L4 (TCP Data)

```
0000   00 0d 60 ff 7e 1a 00 a0 c5 e3 96 4e 08 00 45 00   ..`.~......N..E.
0010   05 dc 75 09 40 00 31 06 c8 ae 41 72 04 45 c0 a8   ..u.@.1...Ar.E..
0020   00 05 00 50 0b cd 0a 39 f4 f4 e0 dc ff 1a 50 10   ...P...9......P.
0030   83 2c d7 62 00 00 48 54 54 50 2f 31 2e 31 20 32   .,.b..HTTP/1.1 2
0040   30 30 20 4f 4b 0d 0a 44 61 74 65 3a 20 53 75 6e   00 OK..Date: Sun
0050   2c 20 30 32 20 41 70 72 20 32 30 30 36 20 31 30   , 02 Apr 2006 10
0060   3a 35 38 3a 35 33 20 47 4d 54 0d 0a 53 65 72 76   :58:53 GMT..Serv
0070   65 72 3a 20 41 70 61 63 68 65 2f 31 2e 33 2e 32   er: Apache/1.3.2
0080   37 20 28 55 6e 69 78 29 20 52 65 73 69 6e 2f 32   7 (Unix) Resin/2
0090   2e 31 2e 73 30 33 30 35 30 35 20 6d 6f 64 5f 73   .1.s030505 mod_s
00a0   73 6c 2f 32 2e 38 2e 31 34 20 4f 70 65 6e 53 53   sl/2.8.14 OpenSS
00b0   4c 2f 30 2e 39 2e 37 62 0d 0a 4c 61 73 74 2d 4d   L/0.9.7b..Last-M
```


Looking at another Packet: L7 (HTTP 200 OK)

```
0000   00 0d 60 ff 7e 1a 00 a0 c5 e3 96 4e 08 00 45 00   ..`.~......N..E.
0010   05 dc 75 09 40 00 31 06 c8 ae 41 72 04 45 c0 a8   ..u.@.1...Ar.E..
0020   00 05 00 50 0b cd 0a 39 f4 f4 e0 dc ff 1a 50 10   ...P...9......P.
0030   83 2c d7 62 00 00 48 54 54 50 2f 31 2e 31 20 32   .,.b..HTTP/1.1 2
0040   30 30 20 4f 4b 0d 0a 44 61 74 65 3a 20 53 75 6e   00 OK..Date: Sun
0050   2c 20 30 32 20 41 70 72 20 32 30 30 36 20 31 30   , 02 Apr 2006 10
0060   3a 35 38 3a 35 33 20 47 4d 54 0d 0a 53 65 72 76   :58:53 GMT..Serv
0070   65 72 3a
0080   37 20 28
0090   2e 31 2e
00a0   73 6c 2f
00b0   4c 2f 30
...
```

- HTTP: Text enc
- Single start line,
- Type ":" Value re

Syntax definition by means of Augmented Back-Naur Form (ABNF):

```
Request       = Request-Line
                *(( general-header
                | request-header
                | entity-header ) CRLF)
                CRLF
                [ message-body ]
```

Looking at another Packet: L7 (HTTP Body: HTML)

```
...
0160   65 65 70 2d 41 6c 69 76 65 0d 0a 43 6f 6e 74 65   eep-Alive..Conte
0170   6e 74 2d 54 79 70 65 3a 20 74 65 78 74 2f 68 74   nt-Type: text/ht
0180   6d 6c 0d 0a 0d 0a 3c 21 44 4f 43 54 59 50 45 20   ml....<!DOCTYPE
0190   48 54 4d 4c 20 50 55 42 4c 49 43 20 22 2d 2f 2f   HTML PUBLIC "-//
01a0   57 33 43 2f 2f 44 54 44 20 48 54 4d 4c 20 34 2e   W3C//DTD HTML 4.
01b0   30 31 20
01c0   2f 45 4e
01d0   41 44 3e
01e0   71 75 69
01f0   70 65 22
0200   74 2f 68
0210   77 69 6e
```

Syntax definition by means of HTML DTD:

```
<!ENTITY % html.content "HEAD, BODY">

<!ELEMENT HTML O O (%html.content;)
                   -- document root element -->
<!ATTLIST HTML
        %i18n; -- lang, dir --
        %version;
>
```

- HTML: Text encc
- Elements may cc
- Document valida

Last Year's Course: UDP File Transfer

Examples from some theoretical outlines

- Box notation with 16-bit word width
- Box notation with 32-bit word width
- Box notation with unknown word width
- Line-oriented text protocol (a bit FTP-style)

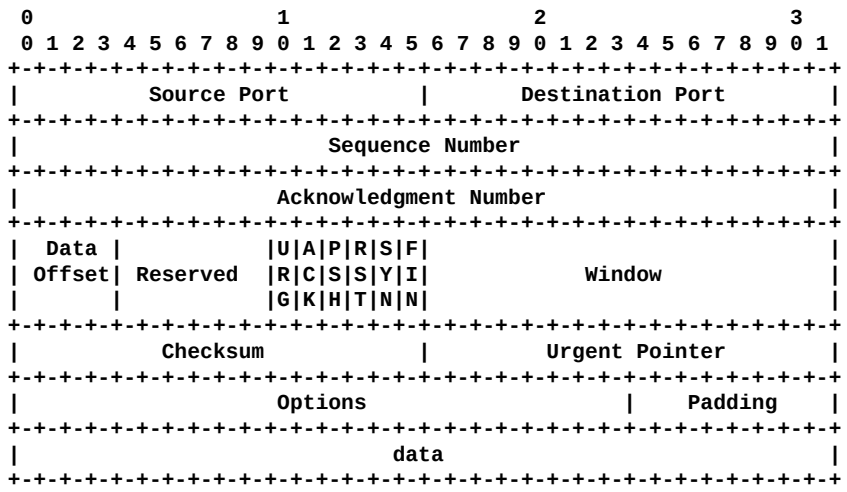
Protocol Encoding

- Objectives
  - Represent information on the wire so that it is equally understood by all peers
    - Typically requires conversion into some local data representation and language
    - "Marshalling", "encoding", "coding", "serialization"
  - Deal with different machine-dependent or otherwise possible representations
    - Binary representation: big endian vs. little endian, floating point representation, …
    - Text representation: EBCDIC vs. ASCII, UTF-8 vs. ISO 8859-1
- Meta goals
  - Debuggability
  - Diagnostic support
  - Principle of least surprise
  - Extensibility, evolvability
  - Efficiency
  - Robustness
- Sample non-objective: replicating a programming language or paradigm "on the wire"

Protocol Encoding – Why Bother?

- Yes, we finally need some representation on the medium in the end – but this comes last…
- True: Concepts, semantics, design decisions, etc. should not be guided by encoding
  - And from this perspective, you should really not bother too early
  - In particular, get agreement on the protocol goals and requirements first
  - Text or binary encoding is *not a requirement* per se
- But: Need to write down messages using *some notation* during the discussion
  - Even strawman notations may govern thinking about and acceptance of a protocol
- Worse: bad encoding choice may ruin (acceptance of) a protocol

Simple Example: Box Notation



HELSINKI UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMMUNICATIONS AND NETWORKING

Simple Example: Box Notation

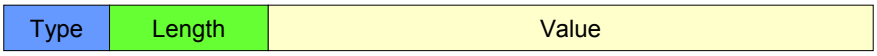
- Some thirty years of ASCII-based protocol specs
- Few common rules
  - Alignment on natural boundaries (32 bit, 16 bit values)
    - Exception: 128 bit IPv6 addresses (64 bit alignment)
  - Good for use of fixed offsets
- Intuitive! Little room for misunderstandings. No learning curve.
- Requires – and encourages! – modesty
  - What you can't write down, you won't get interoperable anyway ;-)
- Implicit conventions assumed
  - E.g., network byte order encoding, 2's-complement for signed numbers, …
  - Consistent: most significant bit first numbering in legend above packet box

But: TCP Options

- Variable number of variable length options
- Explicit encoding of type and length
  - Length found in fixed position
  - Receivers are able to skip unknown options
- Exception: well-known options of kind 0 and 1
- "NOP" / "End of options" to pad TCP header to a 32-bit boundary

```
End of Option List

        +--------+
        |00000000|
        +--------+
         Kind=0

No-Operation

        +--------+
        |00000001|
        +--------+
         Kind=1

Maximum Segment Size

        +--------+--------+---------+--------+
        |00000010|00000100|   max seg size   |
        +--------+--------+---------+--------+
         Kind=2   Length=4
```


TLV: Type Length Value



- Questions
  - What is the optimal length of "type" and "length"?
  - When does the list of TLV items end?
  - What does length include? (value, length+value, all, …)
  - Who manages the type space?
- Issue: implied "length" information from type
  - Skipping unknown values should be possible with minimal knowledge / parsing effort
- Simple for TCP options: almost arbitrary combinations possible
  - Some options restricted to SYN packets for feature negotiation
- General questions
  - How to define which combinations of (TLV-encoded) values are possible?
  - In which order may they occur?
  - Does the ordering mean anything?

Schema vs. Encoding Rules

- Schema
  - What are the top-level information units exchanged?
    - Messages, PDUs, packets, documents, streams, …
  - How are they structured
    - Which items are allowed?
    - How often may they occur?
    - Is their ordering meaningful?
      - Within a single type?
      - Across types?
- Encoding rules
  - Mapping the abstract syntax onto bits
    - Implementation: Transformation of the local into the transfer encoding and vice versa
  - How to represent individual items on the “wire”
  - How to aggregate them to form a larger information unit
  - How to delineate different information units
  - How to achieve data transparency

XDR: External Data Representation (RFC 1832)

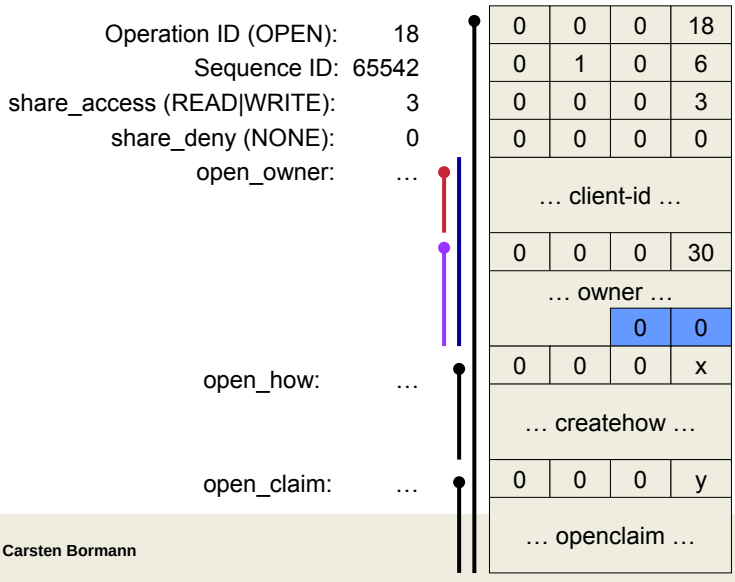
- Encoding for Sun RPC mechanism
  - NFS (RFC 3530)
- Schema language
  - Pseudo C-style notation for function calls and data structures
- Encoding rules
  - Types implied from sequence
  - Exception: choices identified by “switch ()” statement
  - 32-bit alignment of all length and value fields

Schema Example: NFS OPEN

```
enum nfs_opnum4 {
        ...
        OP_OPEN = 18,
        ...
};

union nfs_argop4 switch (nfs_opnum4 argop) {
        ...
        case OP_OPEN:  OPEN4args opopen;
        ...
};

struct OPEN4args {
        seqid4          seqid;
        uint32_t        share_access;
        uint32_t        share_deny;
        open_owner4     owner;
        openflag4       openhow;
        open_claim4     claim;
};

enum createmode4 {
        UNCHECKED4      = 0,
        GUARDED4        = 1,
        EXCLUSIVE4      = 2
};
```

```
union createhow4 switch (createmode4 mode) {
    case UNCHECKED4:
    case GUARDED4:
          fattr4         createattrs;
    case EXCLUSIVE4:
          verifier4      createverf;
};

enum opentype4 {
    OPEN4_NOCREATE  = 0,
    OPEN4_CREATE    = 1
};

union openflag4 switch (opentype4 opentype) {
    case OPEN4_CREATE:
        createhow4     how;
    default:
        void;
};
```


Encoding Example: NFS OPEN



Example: SDNV

- Self-delimiting numeric value (SDNV)
- Motivations
  - Unpredictable scaling requirements (from very small to very large)
  - Limited efficiency requirements for processing (no fixed offsets)
    - E.g., not 10s of Gbit/s wire speed
- Schema: used with box notation
- Encoding
  - Byte-wise extensible: 1 extension bit + 7 data bits per “octet”
  - Sample value representations

0	001 1010	= 001 1010 = 42
1	000 0001	0 000 0000 = 1000 0000 = 128
1	111 1111	1 111 1111 0 111 1111 = 11111 11111111 11111111 = 2 097 152

ASN.1

- Abstract Syntax Notation One (X.68[0-3], X.69[0-4])
  - Formal ITU-T language for describing abstract protocol syntax
  - Originally extracted from X.400 (email the OSI way) (X.409)
    - X.208 (schema) + X.209 (Basic Encoding Rules)
  - Countless revisions and extensions since (superseding X.208/9)
  - Many different encoding rules: Basic, Packed, XML
    - Variants: Distinguished, Canonical
  - RFC 3641: Generic String Encoding Rules (GSER)
- Data type definitions and constraints
  - Basic types: Integer, Boolean, enumeration, many string types, ANY
  - Compound types: SEQUENCE, SET, SEQUENCE OF, SET OF, CHOICE
  - Notation also carries encoding-rule-specific information (“tags”, “IMPLICIT”)

ASN.1 Schema Example: X.509 Certificate

```
Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING
}

TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1,
        serialNumber         CertificateSerialNumber,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
        subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
        extensions      [3]  EXPLICIT Extensions OPTIONAL
}
```


ASN.1 Schema Example: X.509 Certificate

```
<30 82 02 BB>   0  699: SEQUENCE {
<30 82 02 7B>   4  635:   SEQUENCE {
<A0 03>         8    3:     [0] {
<02 01>        10    1:       INTEGER 2
                 :       }
<02 01>        13    1:     INTEGER 17
<30 09>        16    9:     SEQUENCE {
<06 07>        18    7:       OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
                 :       }
<30 2A>        27   42:     SEQUENCE {
<31 0B>        29   11:       SET {
<30 09>        31    9:         SEQUENCE {
<06 03>        33    3:           OBJECT IDENTIFIER countryName (2 5 4 6)
<13 02>        38    2:           PrintableString 'US'
                 :           }
                 :         }
<31 0C>        42   12:       SET {
<30 0A>        44   10:         SEQUENCE {
<06 03>        46    3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
<13 03>        51    3:           PrintableString 'gov'
                 :           }
                 :         }
<31 0D>        56   13:       SET {
<30 0B>        58   11:         SEQUENCE {
<06 03>        60    3:           OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
<13 04>        65    4:           PrintableString 'NIST'
                 :           }
                 :         }
                 :       }
<30 1E>        71   30:     SEQUENCE {
<17 0D>        73   13:       UTCTime 30/06/1997 00:00:00 GMT
<17 0D>        88   13:       UTCTime 31/12/1997 00:00:00 GMT
                 :       }
```

and so on…

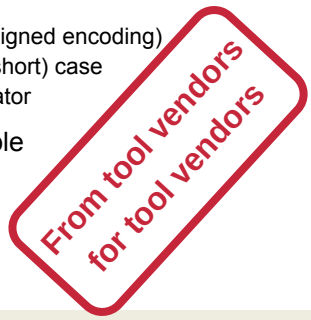
"Example": SNMP

- ASN.1 was a moving target, so this was simplified to an IETF specification "Structure of Management Information" (SMIv2, RFC 2578)
- Reduce base types to INTEGER, OCTET STRING, OBJECT IDENTIFIER
- Build MIB from:
  - Scalars, identified by ASN.1 OIDs (address, name, uptime, ...)
  - Tables, where the cells are indexed by composite ASN.1 OIDs of the form:

    OID-prefix.column.index-val
  - Where OID-prefix usually is, and index-val can be, a sequence of OID nodes.
- Using GET-NEXT, an entire table (or the whole MIB) can be accessed sequentially.
- Result: Solved the problem for SNMP at the time.
  - Very limited means for expressing structure
  - SNMP would have been better off with a language tailored to the problem

ASN.1 Packed Encoding Rules (PER)

- "Perceived" issue with BER: lengthy encoding
- Provide more bit-efficient encoding (not just) for low speed links
  - One motivation: PSTN
- PER removes all explicit encodings and redundancies
  - Minimize the number of bits required per item
  - Heuristic optimization: "normally small numbers"
  - Octet alignment only for strings larger than two octets (aligned encoding)
  - Different length encodings to match the most common (short) case
  - Bit maps for OPTIONALs and one bit as extension indicator
- Protocol definitions need to be known to be parseable
  - Basic extension mechanism introduced into the notation
  - Extensions again encoded in BER (otherwise they could not be skipped)
- Adds by far too much complexity
  - How to kill a standard: H.323, H.245, …

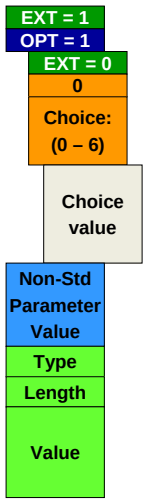


ASN.1 Example 2: A Bit of H.323 / H.225.0

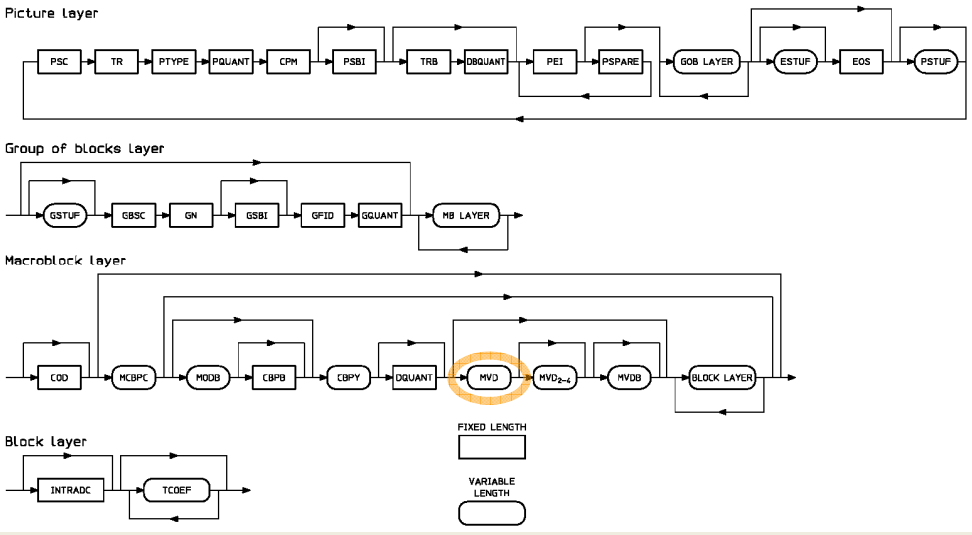
```
H323-UU-PDU ::=         SEQUENCE {
    h323-message-body    CHOICE {
        setup                   Setup-UUIE,
        callProceeding          CallProceeding-UUIE,
        connect                 Connect-UUIE,
        alerting                Alerting-UUIE,
        userInformation         UI-UUIE,
        releaseComplete         ReleaseComplete-UUIE,
        facility                Facility-UUIE,
        ...
    },
    nonStandardData             NonStandardParameter OPTIONAL,
    ...,
    h4501SupplementaryService SEQUENCE OF OCTET STRING OPTIONAL
}

AliasAddress            ::= CHOICE
{
    e164        IA5String (SIZE (1..128)) (FROM ("0123456789#*,")),
    h323-ID     BMPString (SIZE (1..256)),
    ...
}
```

Size encoded in 7, 8 bits fixed

Each char encoded in 4 bits:
"#"→0000, "*"→0001, "0"→0011



Schema Example: H.263



Encoding Example: H.263 (Huffman)

- Excerpt from motion vector coding table in H.263
- Huffman coding: most likely values are coded in shorter code words
  - Predefined coding table
  - No dynamic calculation based upon actual frequencies
- Synchronization words (containing many '0's) for "locating" larger structures
  - For skipping without decoding
  - For recovery after (bit) errors or erasures

22	-5		10	0000 0100 11
23	-4.5		10	0000 0101 01
24	-4		10	0000 0101 11
25	-3.5		8	0000 0111
26	-3		8	0000 1001
27	-2.5		8	0000 1011
28	-2		7	0000 111
29	-1.5		5	0001 1
30	-1		4	0011
31	-0.5		3	011
32	0		1	1
33	0.5		3	010
34	1		4	0010
35	1.5		5	0001 0
36	2		7	0000 110
37	2.5		8	0000 1010
38	3		8	0000 1000
39	3.5		8	0000 0110
40	4		10	0000 0101 10
41	4.5		10	0000 0101 00
42	5		10	0000 0100 10

DVB / MPEG: Multi-Protocol Encapsulation

```
Syntax                                  No. of bits     Mnemonic

PES_data_packet () {
  data_identifier                           8           uimsbf
  sub_stream_id                             8           uimsbf
  PTS_extension_flag                        1           bslbf
  output_data_rate_flag                     1           bslbf
  Reserved                                  2           bslbf
  PES_data_packet_header_length             4           uimsbf
  if (PTS_extension_flag=="1") {
       Reserved                             7           bslbf
       PTS_extension                        9           bslbf
  }
  if (output_data_rate_flag=="1") {
       Reserved                             4           bslbf
       output_data_rate                    28           uimsbf
  }
  for (i=0;i<N1;i++) {
       PES_data_private_data_byte           8           bslbf
  }
  for (i=0;i<N2;i++) {
       PES_data_byte                        8           bslbf
  }
}
```

General Aspects of Binary Encodings

- Field lengths
  - Fixed vs. variable length (BER integer encoding, length of length, SDNV)
  - Efficient access vs. efficient representation
- Implementation issues with unconstrained field sizes
  - Calls for constraints whenever they are known
    - Otherwise implementors will introduce "convenience constraints" (→ bugs)
- Avoid "impossible" encodings
  - To avoid confusion, need for error handling, core dumps
  - Side effect: increases expressible range
  - Violators: IPv4 header length, TCP options
- Extensibility
  - Keep your reserved "bit" to signal extensions
- Type assignment and registration of types in constrained spaces
  - Unconstrained "object type" fields are variable length (e.g., ASN.1 OBJECT IDENTIFIER)
    - Also: OIDs bear the risk of uncontrolled allocation of identifiers

Text-based Encodings

RFC (2)822

To: cabo@tzi.org
From: jo@netlab.tkk.fi
Subject: PD course

- Header consists of zero or more header fields
  - HTTP (SIP, RTSP, MSRP, …) request/status lines don't belong here
- Each header field consists of "Type: Value"
  - Followed by CRLF
  - Typically, one header field per line
  - Header folding possible (header continued on next line)
- Sequence generally not relevant
  - Headers of different types may appear in arbitrary order
  - Multiple headers of same type may imply ordering
  - Canonical encoding to be defined, e.g. for signing
- Beware of "`char line [512];`"
  - Subtle differences across protocols
    - SMTP defines upper limit for line length! (998 chars + CRLF)
    - HTTP, RTSP, SIP, and others do not
- Header-body delimiter: <empty line> CRLF

Subject: Protocol Design course
≈
Subject: Protocol Design
course

Via: host-a, host-b, host-c
==
Via: host-a
Via: host-b
Via: host-c

Getting Formal: ABNF (RFC 2234, 4234)

- Integrates schema and encoding rules
  - Standard notation format for text-based encodings in the IETF
    - Examples: HTTP, SIP, RTSP, SDP, URI
  - (would work for IPv4 header, too, but restricted to characters)
- Based upon Backus-Naur Form (BNF)
  - Syntax to express context-free grammars
  - Define a language by means of production rules using terminal and non-terminal symbols
  - Semi-example: <tcp-packet> ::= <header> <data> | <header>
- Augmented BNF
  - Different from the one your CS neighborhood would expect
  - Examples: [k]*[n]<expr>, "[" <expr> "]", "/" instead of "|", case-insensitive
- Programmers often work by example rather than by (A)BNF

ABNF Example: URIs

```
URI         = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part   = "//" authority path-abempty
            / path-absolute
            / path-rootless
            / path-empty

path-absolute = "/" [ segment-nz *( "/" segment ) ]

segment       = *pchar
segment-nz    = 1*pchar
segment-nz-nc = 1*( unreserved / pct-encoded / sub-delims / "@" )
              ; non-zero-length segment without any colon ":"

pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"
```

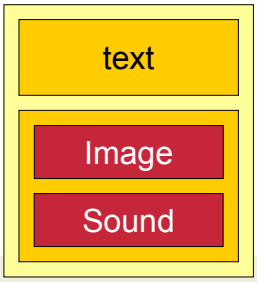

ABNF Example: URIs

```
URI         = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part   = "//" authority path-abempty
            / path-absolute
            / path-rootless
            / pat

path-absolute = "

segment       = *
segment-nz    = 1
segment-nz-nc = 1
              ; n

pchar         = u
```

```
Examples for URIs

ftp://ftp.is.co.za/rfc/rfc1808.txt
http://www.ietf.org/rfc/rfc2396.txt
ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:John.Doe@example.com
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

MIME

- Multipurpose Internet Mail Extensions
  - Not just mail: used with HTTP, SIP, and many other application protocols
- Traditional: plain text follows RFC(2)822-style headers
  - No identification of contents possible
  - Transfer (and content) encoding limited to ASCII
  - Remember: uuencode (“begin 644 filename … end”)
- Define the purpose of a piece of content (in a message body)
  - Type, encodings
  - Intended interpretation
  - Specify additional parameters
  - Allow for references
- Allow for multipart contents
  - Arbitrarily nested pieces of contents
  - Specify the above for each part individually



MIME

```
Content-Type: image/jpeg
Content-Length: 5489
Content-Transfer-Encoding: base64
Content-ID:  42
Content-Description: an image of a tree
Content-Disposition: attachment; ...
```

```
Content-Type: multipart/alternative
Content-Type: multipart/mixed
Context-Type: multipart/related
```

What does MIME bring to the table?

- Internet-wide scheme to identify different types of coded objects
  - Registry
  - Reaches into operating systems today...
    - Instead of ".txt" vs. ".doc" vs. ".pdf"
- Provide for multiple Transfer-Encodings of combinations of such coded objects allowing to carry binary in text etc.
- Combination rules (multipart-mixed, multipart-alternative)
- Container for more complex domain-specific techniques (multipart-signed)
- Does not **quite** obviate ZIP...

Unconstrained numbers, again

- Content-Length: 10987654321
- Was unreasonable when HTTP was widely implemented
- Implementers put the decoded value into a 32-bit number
  - E.g., IIS and IE in certain versions just use the value modulo 2^{32}
- Today, videos etc. are often larger than that
- Text encoding makes integer sizes a non-issue ***on the surface***, only
  - Specification should constrain or provide implementer guidance!

Text Encoding Gone Terribly Wrong: SDP

- Original intention: announcing parameters of a simple mc session
  - Session level: when, who, what about?
  - Media level: which codecs, addresses, and other parameters
- Bent and extended over time
  - Moving from one-way announcements to two-way negotiation
  - Demand for richer expressiveness
- Neither is supported by the trivial syntax
  - Encoding all additional semantics as attributes (a=)
- Lesson to be learned: beware of too much “simplicity”
  - No evolution path
  - But change is perceived to hurt

Text Encoding Gone Terribly Wrong: SDP

- Original intention: announcing parameters of a simple mc session
  - Session level: when, who, what about?
  - Media level: which codecs, addresses, and other parameters
- Bent and
  - Moving
  - Deman
- Neither i
  - Encodi
- Lesson t
  - No evo
  - But cha

```
v=0
o=Laura 289083124 289083124 IN IP4 one.example.com
t=0 0
c=IN IP4 224.2.17.12/127
a=group:LS 1 2
m=audio 30000 RTP/AVP 0
a=mid:1
m=video 30002 RTP/AVP 31
a=mid:2
m=audio 30004 RTP/AVP 0
c=IN IP4 224.2.17.13/127
i=This media stream contains the Spanish translation
a=mid:3
```

XML

- XML: Text-based notation derived from SGML
  - Adapted to current thinking (e.g., uses UTF-8 by default)
- Elements, Attributes
  - Content of elements can be further elements and/or text
- Markup: Start/End Tags (and Empty Tags)
  - Start tag contains attributes, if any
- Three widely used Schema Notations:
  - DTD (inherited from SGML)
  - W3C Schema (“XSD”)
  - RELAX-NG, often in compact format (“RNC”)

DTD

- Focused on element structure of document
  - Attributes can choose form a small set of types
  - Legacy: Does not address XML namespaces
- Syntax: designed for inclusion in SGML documents
  - (needed special syntax to set it apart from document itself)

```
<!ELEMENT book (page)+>
<!ATTLIST book
  authors-blog CDATA #IMPLIED>

<!ELEMENT page (#PCDATA)>
```

- Verdict: The old way of doing it, limited expressiveness

W3C Schema ("XSD", "WXS", ...)

- Increase expressiveness greatly
  - Influenced by database schema definition requirements
  - Full control over both elements and attributes
  - Large predefined set of datatypes
- Syntax: Attempt to use the power of XML itself (at the cost of limited readability)
- Verdict:

W3C Schema ("XSD", "WXS", ...)

- Increase expressiveness greatly
  - Influenced by database schema definition requirements
  - Full control over both elements and attributes
  - Large predefined set of datatypes

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="page"/>
      </xs:sequence>
      <xs:attribute name="authors-blog" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="page" type="xs:string"/>
</xs:schema>
```

From tool vendors for tool vendors

RELAX-NG

- Focus on simplicity
  - re-uses W3C Schema datatype collection
  - eschews PSVI (Post-Schema Validation Instance)
    (no default values that need to be obtained outside the document itself — all documents are “standalone”)
- Syntax
  - There is an XML syntax
  - Most designers use more readable compact syntax (“RNC”) today
    - (there is a one-to-one transformation between both syntaxes)

```
start = element book {
  attribute authors-blog { xsd:anyURI }?,
  page+
}
page = element page { text }
```

- Verdict: What XML schema designers use today
  - (Compilers can generate XSD and DTD from RNC)

When to use XML?

- Default choice! (At least at L7)
- Excels most as interchange format for complex structures
  - Namespaces facilitate extensions by multiple groups in different places
- Stable toolset (innovation ≠ mucking around)
  - User driven, not toolset vendor driven
  - Open source available (and there is a second source for most anything)
  - Available for the strangest programming environments
- When not to use XML?
  - Problem is extremely unlikely to get more complicated over time
  - Needs to run on a toaster (i.e., carrying an XML library to the target is way too heavy) — or has to be implemented directly in silicon…
  - Has to be handled at 10 Gbit/s by a washing machine processor

Simple non-XML example: Bencoding

- **Strings**: length (base 10); a colon; the string. 4:spam ⇔ 'spam'
- **Integers**: an 'i'; the number in base 10; an 'e'. i3e ⇔ 3, i-3e ⇔ -3. Integers have no size limitation. i-0e is invalid. All encodings with a leading zero, such as i03e, are invalid, other than i0e, which of course corresponds to 0.
- **Lists**: an 'l'; the elements; an 'e'.
  l4:spam4:eggse ⇔ ['spam', 'eggs']
- **Dictionaries**: a 'd'; alternating keys and their corresponding values; an 'e'. Keys must be strings and appear sorted (as raw strings).
  d3:cow3:moo4:spam4:eggse ⇔ {'cow': 'moo', 'spam': 'eggs'}
  d4:spaml1:a1:bee ⇔ {'spam': ['a', 'b']}

- In use as the protocol encoding for Bittorrent.

Another non-XML example: JSON

- JavaScript Object Notation
  - RFC 4627, MIME type: application/json (war: text/json)
  - null, true/false, Number, String, Array, Hash ("Object")
- Originally intended for feeding data into JavaScript programs (e.g., in a Browser environment)
  - Used e.g. in JSON-RPC
- Has become popular as a more data-oriented alternative to XML
  - Cf. Apple PLIST format, Bencoding, YAML
  - All these have no Schema languages!

```
[
  {
    "Latitude":  37.7668,
    "Longitude": -122.3959,
    "City":      "San Francisco",
    "State":     "CA",
    "Zip":       "94107",
    "Country":   "US"
  },
  {
    "Latitude":  37.371991,
    "Longitude": -122.02602,
    "City":      "Sunnyvale",
    "State":     "CA",
    "Zip":       "94085",
    "Country":   "US"
  }
]
```

General Issues with Representation of Text

- In the beginning, there was NVT (ASCII + CRLF)
  - OS-specific variants (Unix: LF, Mac: CR)
- Zillions of character codes
  - Pre-ASCII: TELEX (Baudot), EBCDIC
  - ASCII variants: DIN 66003 et al. †
  - ASCII extensions: ISO 8859-1 et al., Windows 1252, ISO-2022-JP, …
  - Unicode in its transfer encodings: UTF-8, UTF-16
- Gateways
  - Often (believe they have to) transcode and lose some information
- Data transparency
- Binary contents: Base64 as transfer encoding

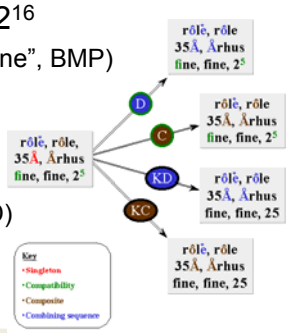
The history of character sets

- Pre-history: Baudot (Telex), EBCDIC
- 1960s: **ASCII** (7 bits, 95 characters)
  - + national variants (“{“ ➔ “ä”)
- 1980s: Regional **unibyte** character sets (8 bits)
  - ISO 8859-1 ≡ **Latin-1** (Default character set of the Web)
  - 14 more regional ISO-8859 Variants (Latin 2 etc., Greek, Thai, ...)
- 1980s+: **multibyte** (CJK)
  - JIS-X0208 (.jp), plus three different *Encodings* (SJIS, EUC-JP, …)
  - GB-2312 (.cn), Big5 (.tw), KSC-5601 (.kr)
  - ISO 2022: Switching between different character sets

This was not sustainable.

Unicode

- Unicode ≡ ISO/IEC 10646
- Universal Character Set (UCS)
- Allocates a number (“code point”) to each character
  - Building on ASCII and Latin-1 (ISO/IEC 8859-1)
- Number space: 0–0x10FFFF (0–1114111), $17*2^{16}$
  - “Important” characters in first 2^{16} (“Basic Multilingual Plane”, BMP)
  - Required politically difficult “Han unification” (CJK)
- Issue: Normalization
  - Composed vs. decomposed (NFC vs. NFD)
  - Compatibility formats (NFKC vs. NFC vs. NFKD vs. NFD)



UTF: Unicode Transformation Formats

Three main representations (“transformation formats”):

- UTF-32: One 32-bit word per character
- UTF-16: Most characters in 16 bits, some in 32 bits
  - BMP is encoded as 16-bit number
  - Escape for ≥ 2^{16}: surrogate characters 110110xxxxxxxxxx 110111xxxxxxxxxx
- UTF-8 (RFC 3629): ASCII-compatible (ASCII characters in 8 bits)
  - 0x0..0x7F 0xxxxxxx
  - 0x80..0x7FF 110xxxxx 10xxxxxx
  - 0x800..0xFFFF 1110xxxx 10xxxxxx 10xxxxxx
  - 0x10000..0x10FFFF 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

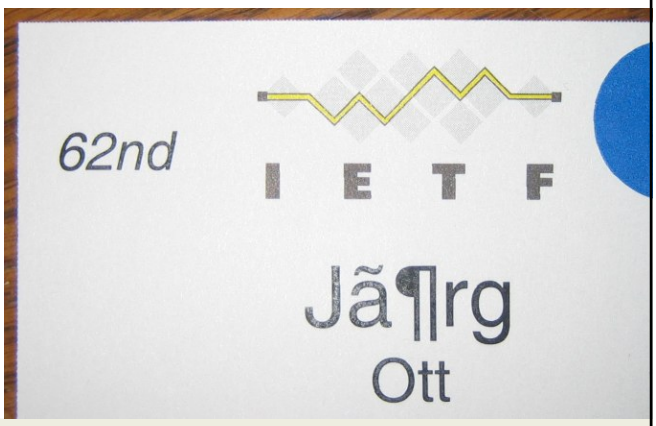
Byte order issues

Cf. SNDV

+ Hacks (SCSU, GB-18030)

Summarizing 45 Years of Pain

Just use UTF-8!



General Issues with Text-based Encoding

- Obviously: character set issue (for user-visible text)
  - Often ignored so that things just work by chance (or don't)
- Perceived (and sometimes real) message size
  - Text encoding can be less efficient than binary
  - Longer identifiers for types, longer value representations (except for strings), delimiters
- Not that great to carry binary blobs around (base64)
  - Mixtures: e.g., SMTP allows transparent 8 bit encoding
  - FTP, RTSP, SIP use separate transports for data
- Illusion of well-defined semantics
  - Implied from "human readability"
  - Tendency for underspecification, may lead to false implied semantics

Some Further Aspects

- Canonical Encoding
  - Example: Comparison of URIs
  - Example: Security (digital signatures)
- Encoding & Security
  - S/MIME
  - OpenPGP
  - XML DSig
- Encoding and Evolvability
  - Organizational framework
  - Protocol numbers & registry
  - Need to carry at least one bit indicating “base line” (i.e. not yet evolved)
  - Tradeoff: …
- Performance
  - Implementation complexity (incl. code size)
  - Computational requirements
  - Bits on the wire

Short Cases Studies

Case Study: MSRP

```
MSRP bla4711 SEND
To-path:msrp://b.dom.org:9876/abc;tcp
From-path:msrp://a.dom.org:8888/xyz;tcp
Message-ID: 123
Content-Type: text/plain
Success-Report: yes

Hi! How are you doing?
-------bla4711$
```

"Record Marking"

- Message Session Relay Protocol
  - A text-based transport protocol to carry instant messages (of arbitrary size) between two endpoints
  - RFC822-style encoding
  - Significant overhead
- Why not just TCP?
  - Need to be able to work through relays (because of NATs)
  - Want to support multiplexing of multiple conversations
  - (Can easily incorporate MIME types)
- Why not use BEEP?
  - Good question: need to be able to work through relays…
  - "Not invented here" (real aspect: change control of the protocol)
- Notes on the historic evolution
  - Today: Base MSRP does no longer support relays
  - Tomorrow: ICE TCP will eliminate the need for them
  - All the time: Multiplexing could have done more efficiently with STCP
  - But since we got that far, MSRP stays the way it is…

Case Study: H.323

- Original idea: Extending ISDN-based video conferencing into LANs
- Combination of slightly modified Q.931 PDUs
  - Encoding in some octet-oriented TLV-style variant (relatively light processing)
  - Re-using and profiling readily defined Q.931 and related PDUs
- and User-to-User Information Elements (UUIE)
  - Encoding in ASN.1 PER to carry all the information that do not fit the ISDN-tailored Q.931 aspects
- Internet use gained importance: UUIE's role grew
  - Virtually all relevant information now encoded in ASN.1
    - Partially redundant to Q.931 IEs (what takes precedence?)
  - Q.931 primarily overhead that also needs to be done
  - Seemed a good idea at the time – if you were a company in the ISDN and conferencing business and wanted to re-use code and knowledge
    - Probably also helped acceptance in the ITU-T back in 1995/1996

Case Study: Binary Floor Control Protocol (BFCP)

- Developed in the IETF for use with SIP
  - Request-response + asynchronous notifications
  - Binary TLV encoding
- Usage with SIP — Why is BFCP binary rather than text?
  - Strong motivation from 3GPP: must work for mobile nodes with minimal overhead
  - Simple protocol without much extension requirements
    - Designed to the task at hand
  - Straightforward data structures, limited degree of nesting
- Historical notes
  - First protocol that actually got done in XCON WG
    - Benefits from “straight-to-the-point” design (well: and simple tasks)
    - All other protocols caught in endless modeling, design, and encoding discussions
  - Now considered as one possible base for general conference control protocol
    - As are several text-based proposals: fierce discussions to come
    - Past arguments circle both around concepts and also to a significant degree about text vs. binary

Excursion: Turning Text into Binary

- Compression
  - Reduce redundancy in encoding after message creation
  - If optional: need to distinguish uncompressed from compressed from random messages
- Simple application-specific example: DNS
- Typical general example: DEFLATE (RFC 1951)
- Often little is known about the payload
  → content/application-specific compression needed
- Header compression to address common part of (text) protocols
  - Specific compression schemes: VJ TCP HC, IP/UDP/RTP header compression
  - Compression framework: ROHC

Early Header Compression (HC)

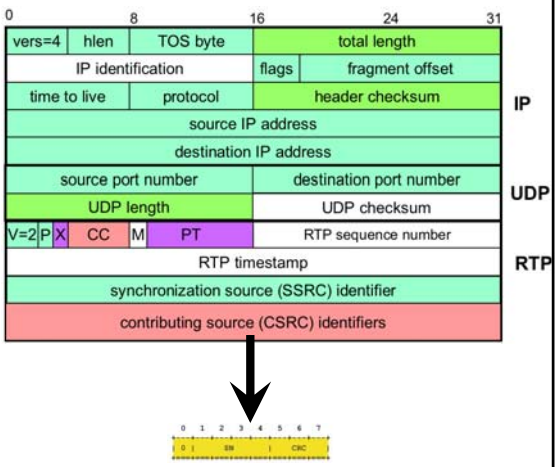
- TCP/IP Header Compression was pioneered in 1990
  - Van Jacobson, RFC 1144
  - TELNET access over very low bandwidth vs. 40 bytes header overhead
- Little advantage for Web traffic (large packets)
- Renewed interest with IPv6 (RFC2507: IP Header Compression)
  - Can compress IP header chains
- Real-time, conversational traffic (VoIP): small packets
- RFC 2508: Compressed RTP
- 1990s: delta coding technology

Robust Header Compression

- The problem with delta coding: error propagation
  - No errors on wired links
  - RFC 2507/2508: Errors can be repaired in one RTT
- Significant performance impairment with wireless links
  - High loss rate
  - High RTT (interleaving!) in 2G/3G
- 1999/2000:
  - LSB encoding instead of delta encoding
  - Optimistic compression, enhanced by checksum checks

RFC 3095

- Published in July 2001
  - Robust header compression for IP/UDP/RTP, IP/ESP, IP/UDP
  - Part of 3GPP since Release 4
- Typically reduces 40 bytes of IP/UDP/RTP header to one byte
  - Zero-byte variant possible with link-layer assist (LLA, RFC 3242)
- Recently complemented by IP-only and UDP-lite variants



Requirements and Issues

- Requirements
  - Transparency — HC is hop-by-hop; hosts don‘t get to know
  - Performance — within design bracket
  - Error Tolerance — does not break when used outside design bracket
- Issues
  - Header compression is “organized layer violation”
    - Need to track L3-L7 protocols
  - Headers get bigger (IPv4 ↯ IPv6)
  - New headers are introduced (IPsec, tunneling/mobility, ...)
  - New options are invented for existing protocols (e.g., for TCP)
  - New protocols (e.g., DCCP)

HC: Current Work

- Complement UDP/RTP ROHC by a TCP ROHC
- TCP has changed since RFC 1144 (and RFC 2507)
  - Large Windows, Timestamps; SACK; ECN
- Assumption: Lower error rates (see RFC 3819!)
- Various approaches for combining header compression and lower-layer protocols (e.g., MPLS)
- New protocols are being designed with HC in mind
  - New transport protocol DCCP was reviewed for compressibility
  - SRTP security scheme was designed to allow compressibility
  - Self-describing packets are desirable!

Syntax = Bikeshed Color

- Everybody can join a discussion about the color of a bikeshed to be built
  - Everyone has an opinion
  - The decision is pretty much inconsequential, so it's not dangerous to voice it
  - There is no good way to terminate the discussion
- Bikeshed issues can consume a significant part of the decision making bandwidth
- Bikeshed issues tend to be revisited even after decision making
  - Cf. Megaco (H.248): A coin was tossed to decide "Text or Binary"
  - The result was "Text"
  - The Binary faction raised a stink
  - The final standard says "Binary mandatory, Text optional"
  - Actual interoperation uses Text

This is often Religion!

- Many people have previous experience in situations where there were indeed reasons that, e.g., made Text a better choice than Binary
- It is only human to try to re-use this experience in new situations
- But the people on a committee don't share the same experience!

➡ Religious issues

Concluding Remarks: Rules of Thumb

- There are rich resources to choose from…
  - And there is usually not a single "best choice" identifiable
- Cultural compatibility to group
  - The group of people developing a protocol
  - The target group of implementers, operational people, …
- In non-trivial cases, formal notations can help, if
  - The notation is **stable**
  - The actual designers are **familiar** (and productive) with it
  - It is not taken as license to introduce rampant **complexity**
  - Tools don't get in the way (e.g., are readily available and do **work right**)
    - Even better if the tools actually contribute, e.g., consistency checks