University of Glasgow

Presentation & Application Layers

Networked Systems 3
Lecture 15

Lecture Outline

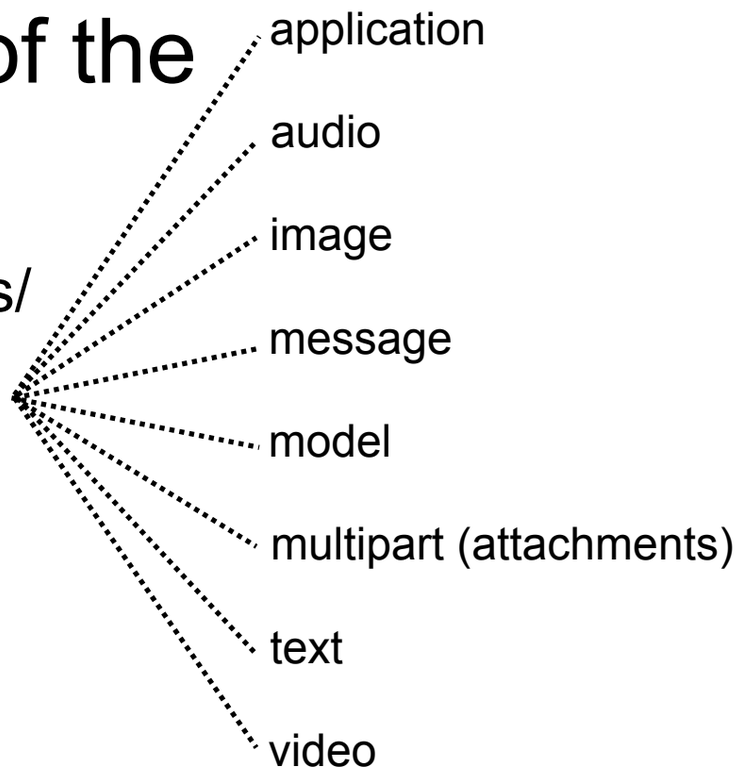
- Presentation layer
  - Media types and content negotiation
  - Channel encoding
  - Internationalisation
- Application layer
  - Message syntax and framing
  - Interaction style
  - Signalling error conditions

The Presentation Layer

- Managing the presentation, representation, and conversion of data:
  - Media types and content negotiation
  - Channel encoding and format conversion
  - Internationalisation, languages, and character sets
- Common services used by many applications

Media Types

- Data formats often not self-describing
- *Media types* identify the format of the data
  - http://www.iana.org/assignments/media-types/
  - Categorise formats into eight top-level types
  - Each has many sub-types
  - Each sub-type may have parameters:



text/plain; charset=iso-8859-1

Sub-type Parameters

- Media types included in protocol headers to describe format of included data

Content Negotiation

- Many protocols negotiate the media formats used
  - Ensure sender and receiver have common format both understand
- Typically some version of an offer-answer exchange
  - The *offer* lists supported formats in order of preference
  - Receiver picks highest preference format it understands, includes this in its *answer*
  - Negotiates common format in one round-trip time



Channel Encoding

- If the protocol is textual, how do you transport binary data?
  - Encode binary data in a textual format for transfer
    - What is binary data? What is an appropriate textual format?
  - Signal that the content has been encoded
    - The MIME `Content-Transfer-Encoding:` header
    - May require negotiation of an appropriate transfer encoding, if data passing through several systems

What is Binary Data?

- Data that cannot be represented within the textual character set in use
  - If using 7 bit ASCII text, any data using all eight bits
    - Example: very old versions of `sendmail` used the 8th bit to signal that quoted data was present, stripping it off data on input, since email was guaranteed to be 7 bit ASCII only
  - If using EBCDIC, any unassigned character
  - If using UTF-8, invalid multi-byte sequences
- Must be *encoded* to fit the character set in use

Sending Binary Data

- Many protocols send binary directly, not encoded in textual format
  - E.g. TCP/IP headers, RTP, audio-visual data
- Two issues to consider:
  - Byte ordering – the Internet is big endian, must convert from little-endian PC format
  - Word size – how big is an integer (e.g., 16, 32, or 64 bit)? how is a floating point value represented?

```
#include <arpa/inet.h>

uint16_t htons(uint16_t hs);
uint16_t ntohs(uint16_t ns);
uint32_t htonl(uint32_t hl);
uint32_t ntohl(uint32_t nl);
```

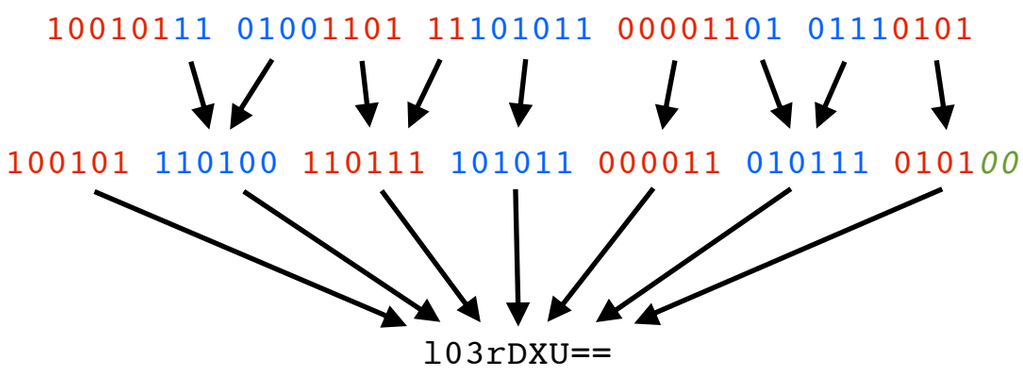
Coding Binary Data for a Textual Channel

- Issues when designing a binary coding scheme:
  - *Must be backwards compatible with text-only systems*
  - Some systems only support 7-bit ASCII
  - Some systems enforce a maximum line length
  - Must survive translation between character sets
    - Legacy systems using ASCII, national extended ASCII variants, EBCDIC, etc.
  - Must not use non-printing characters
  - Must avoid escape characters that might be interpreted by the channel (e.g., $ \ # ; & “ ”)
    - If might use escape characters to convert 8-bit values into format suitable for the channel, if 8-bit values are rare
    - E.g., quoted-printable encoding uses = as escape character, so that the string straße is quoted as stra=dfe (an = is represented as =3d)

Base 64 Encoding

000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011110	f	101111	v	111111	/
						(pad)	=

- Textual encoding of binary
  - Split each group of 3 bytes (24 bits) into four 6-bit values, and encode as text using lookup table shown
  - Use = characters to pad if needed
  - Encode no-more than 76 characters per line



  - Average 33% increase in data size (3 bytes → 4)

Internationalisation (i18n)

- What character set to use?
  - A national character set? ASCII, iso-8859-1, koi-8, etc.
    - Need to identify the character set and the language
    - Complex to convert between character sets
  - Unicode?
    - A single character set that can represent (almost?) all characters, from (almost?) all languages
    - 21 bits per character (0x000000 – 0x10FFFF)
    - Several representations (e.g. UTF-8, UTF-32)
    - Just represents characters – still need to identify the language

Unicode and UTF-8

- *Strong recommendation*: Unicode in UTF-8 format
  - UTF-8 is a variable-length coding of unicode characters

Unicode character bit pattern:		UTF-8 encoding:
`00000000 00000000 0zzzzzzz`	→	`0zzzzzzz`
`00000000 00000yyy yyzzzzzz`	→	`110yyyyy 10zzzzzz`
`00000000 xxxxyyyy yyzzzzzz`	→	`1110xxxx 10yyyyyy 10zzzzzz`
`000wwwxx xxxxyyyy yyzzzzzz`	→	`11110www 10xxxxxx 10yyyyyy 10zzzzzz`

  - Backwards compatible with 7-bit ASCII characters
    - Codes in the ASCII range coded identically, all non-ASCII values are coded with high bit set
    - No zero octets occur within UTF-8, so it can be represented as a string in C
  - Widely used in Internet standard protocols

Unicode: Things to Remember

- Unicode just codes the characters, need to code the language separately
  - Different languages have very different rules!
    - Is text written left-to-right or right-to-left?
    - How to sort? e.g. in German, ä sorts after a, in Swedish, ä sorts after z
    - How to do case conversion and case insensitive comparison? e.g., in German, `toupper("straße") = "STRASSE"`
    - How to handle accents? ligatures? ideograms? etc.
  - At the protocol level:
    - Code the characters as UTF-8 and specify the language
    - Let the application-layer programmer worry about using the data!

The Application Layer

- Protocol functions specific to the application logic
  - Deliver email
  - Retrieve a web page
  - Stream video
  - …

Application Protocol Style

- How is the application protocol data structured?
  - Textual or binary?
  - Framing mechanism?
- How do the interactions occur?
  - Explicit request-response, or potentially unsolicited?
  - Degree of chatter?
- How are errors reported?

Textual or Binary?

- Does the protocol exchange textual or binary messages?
  - Textual – flexible and extensible
    - See http://www.ietf.org/rfc/rfc3252.txt – “Binary Lexical Octet Ad-hoc Transport” – for a counter example (and note the publication date!)
    - High-level application layer protocols (e.g., email, web, instant messaging, …)
  - Binary – highly optimised and efficient
    - Audio and video data (e.g., JPEG, MPEG, Vorbis, …)
    - Low-level or multimedia transport protocols (e.g., TCP/IP, RTP, …)
- Design for extensibility, rather than optimality

Framing over TCP

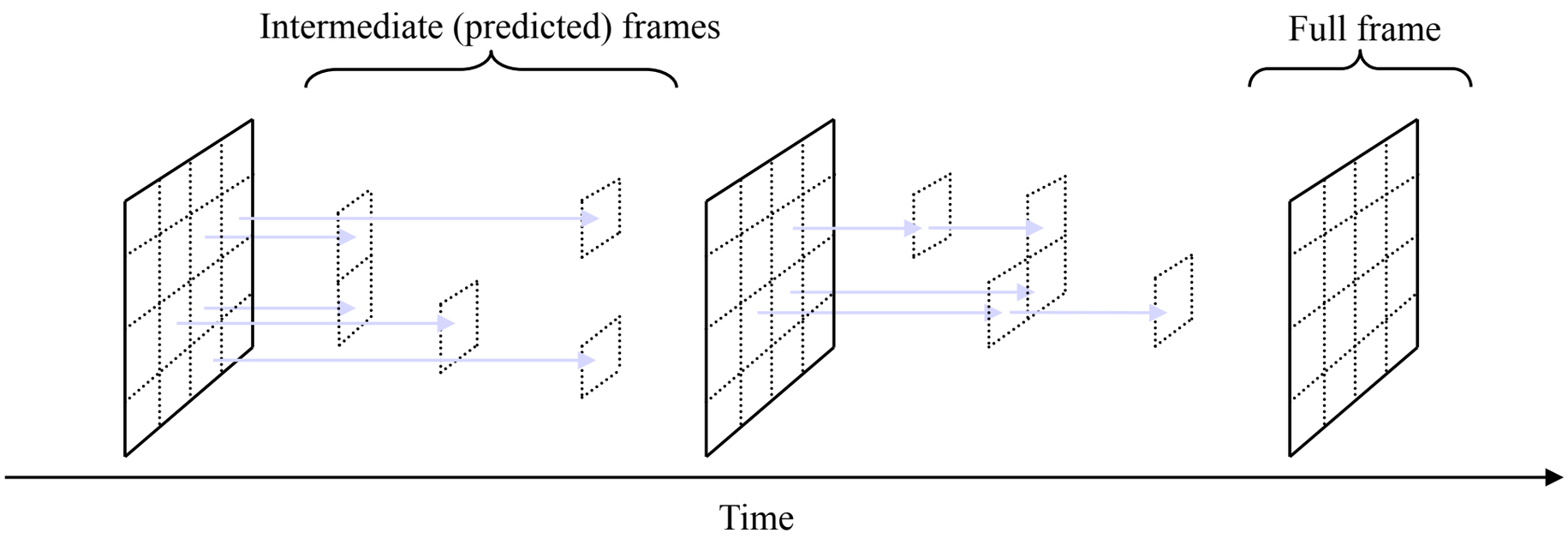
- How to denote record boundaries?
  - TCP connection is reliable, but doesn't frame the data; must parse the byte stream
  - Requires structured protocol:
    - Textual request-response format: send request and headers giving details, and receive a structured response (e.g., HTTP, SMTP)
    - Tag-stream protocols parse the stream until appropriate closing tag seen (e.g., Jabber)
    - Binary protocols – TLV structure

```
C: GET /index.html HTTP/1.1
C: Accept-Language: en-gb
C: Accept-Encoding: gzip, deflate
C: Accept: text/xml, text/html, text/plain
C: User-Agent: Mozilla/5.0 (Macintosh; U; Mac OS X; en-gb)
C: Cache-Control: max-age=0
C: Connection: keep-alive
C: Host: www.dcs.gla.ac.uk
C:
S: HTTP/1.1 200 OK
S: Server: Apache/2.0.46 (Red Hat)
S: Last-Modified: Mon, 17 Nov 2003 08:06:50 GMT
S: Accept-Ranges: bytes
S: Content-Length: 3646
S: Connection: close
S: Content-Type: text/html; charset=UTF-8
S:
S: <HTML>
S: <HEAD>
S: <TITLE>Computing Science, University of Glasgow</TITLE>
S: ...remainder of page elided...
```

- Trade-off flexibility, extensibility, ease of parsing

Framing over UDP

- UDP provides framing – data is delivered a packet at a time – but is unreliable
- Application must organise the data so it's useful if some packets lost
  - E.g. streaming video with I and P frames



How do Interactions Occur?

- How does communication proceed?
  - Does the server announce its presence on the initial connection? Or does it wait for the client to start?
  - Is there an explicit request for every response? Can the server send unsolicited data?
  - Is there a lot of chatter, or does the communication complete within a single round-trip?

Reducing Protocol Chatter

- The more “chatty” protocols take many round trips to complete a transaction
  - RTT fixed by speed-of-light irrespective of network bandwidth → often limiting factor in response time
- Want to reduce number of round trips before the transaction completes → send transaction in single request, get a single response

How are Responses Signalled?

- Useful to have an extensible framework for response codes
- Many applications settled on a three digit numeric code
  - First digit indicates response type
  - Last two digits give specific error (or other response)

Error Code	Meaning
1xx	In progress
2xx	Ok
3xx	Redirect
4xx	Client error
5xx	Server error

- Allows signalling new error types, with meaningful response from existing clients
  - Backwards compatibility

Questions?