University of Glasgow

# Higher-layer Protocols

## Networked Systems (H)
## Lecture 19

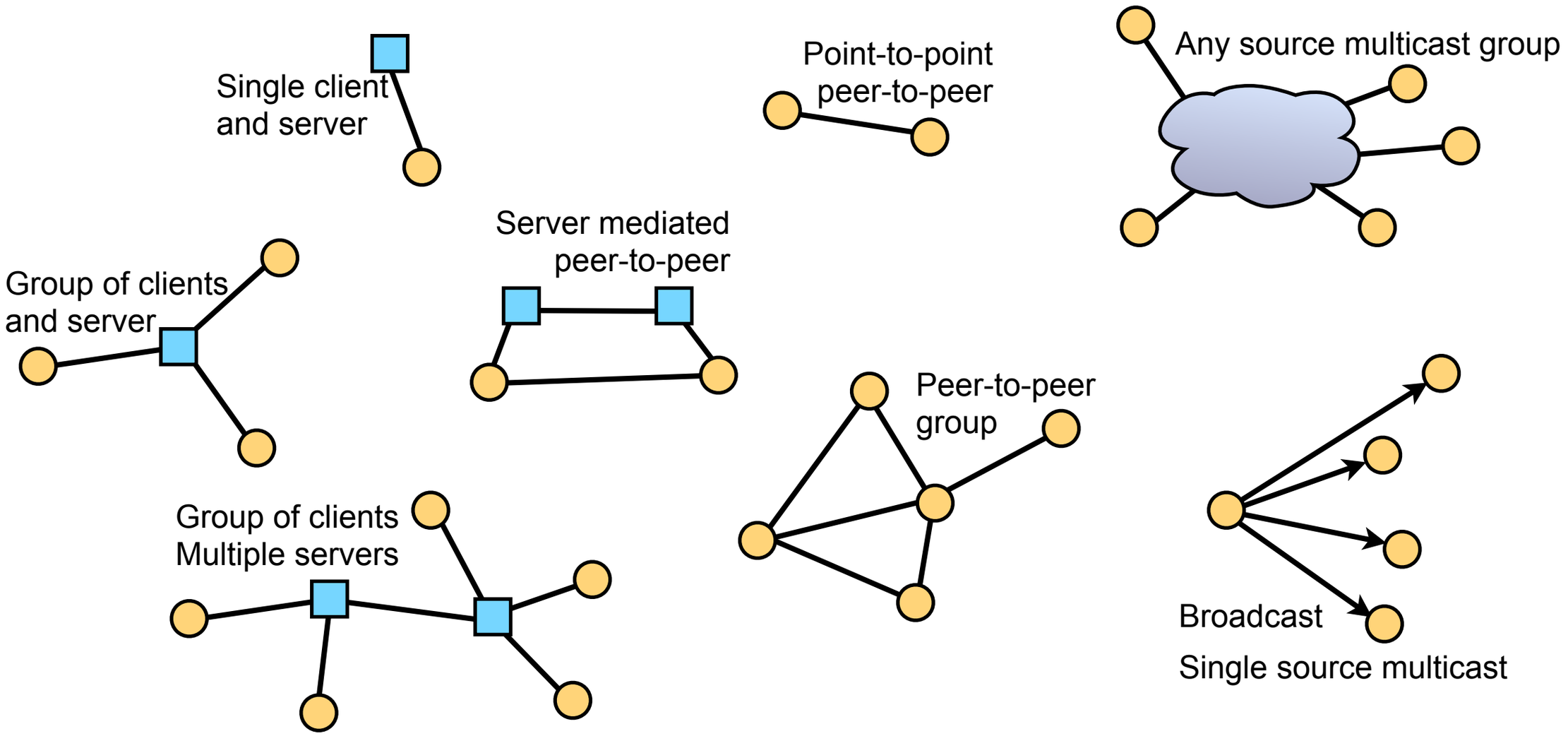
# Higher Layer Protocols

- The OSI reference model defines three layers above the transport:
  - Session layer
  - Presentation layer
  - Application layer
- Typically implemented within an application or library, rather than within the kernel
- Relatively ill-defined boundaries between layers

- Goal – support application needs:
  - Setup/manage transport layer connections
  - Name and locate application-level resources
  - Negotiate data formats, and perform format conversion if needed
  - Present data in appropriate manner
  - Implement application-level semantics

# Session Layer: Managing Connections

- What connections does the application need?



# Managing Connections

- How to find participants?
  - Look-up name in a directory (e.g. DNS, web search engine)
  - Server mediated connection (e.g. instant messenger, VoIP call)
- How to setup connections?
  - Direct connection to named host (→ NAT issues)
  - Mediated service discovery, followed by peer-to-peer connection
    - E.g. VoIP using SIP and RTP with ICE
- How does session membership change?
  - Does the group size vary greatly? How rapidly do participants join and leave? Are all participants aware of other group members?

# User and Resource Mobility

- IP addresses encode location → mobility breaks transport layer connections
- Session layer must find new location, establish new connections
  - Might be redirected by the old location – e.g., HTTP redirect
  - Mobile devices may update a directory with new location
- Complexity is pushed up from the network to the higher layers

HTTP request

```
GET /index.html HTTP/1.1
Host: www.google.com
```

HTTP response

```
HTTP/1.1 302 Moved Temporarily
Location: http://www.google.co.uk/index.html
Cache-Control: private
Content-Type: text/html
Server: gws
Content-Length: 231
Date: Sun, 17 Feb 2008 23:23:30 GMT

<HTML>
 <HEAD>
  <TITLE>302 Moved</TITLE>
 </HEAD>
 <BODY>
  <H1>302 Moved</H1>
  The document has moved
  <A HREF="http://www.google.co.uk/index.html">here</A>.
 </BODY>
</HTML>
```

# Multiple Connections

- A single session may span multiple transport connections
  - E.g., retrieving a web page containing images – one connection for the page, then one per image
  - E.g., a peer-to-peer file sharing application, building a distributed hash table
- Session layer responsible for co-ordinating the connections

# Middleboxes and Caches

- Some protocols rely on middleboxes or caches
  - Web cache – optimise performance, moving popular content closer to hosts
  - Email server – supports disconnected operation by holding mail until user connects
  - SIP proxy servers and instant messaging servers – locate users, respond for offline users
- The end-to-end argument applies, once again
  - Only add middleboxes when absolutely necessary

# The Presentation Layer

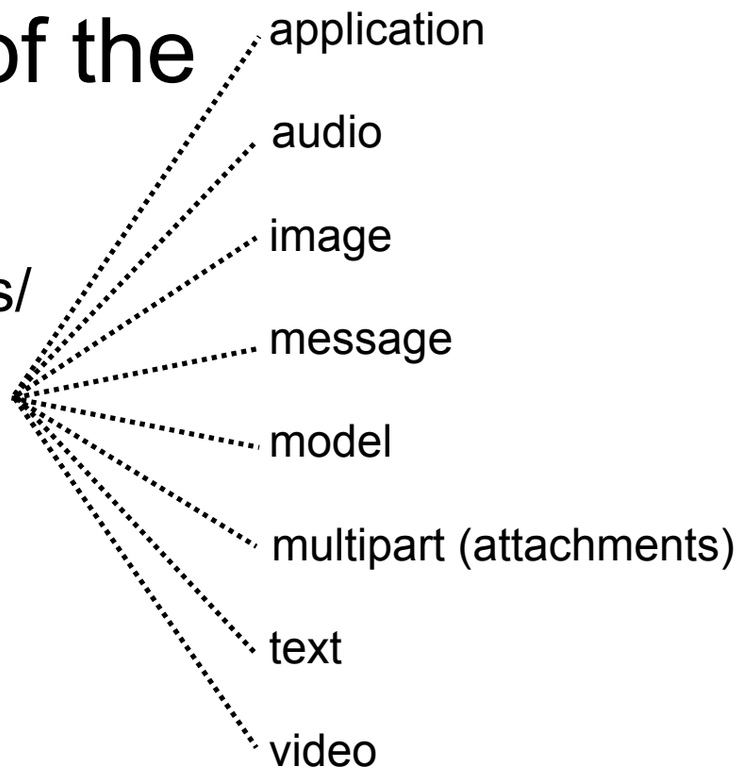
- Managing the presentation, representation, and conversion of data:
  - Media types and content negotiation
  - Channel encoding and format conversion
  - Internationalisation, languages, and character sets
- Common services used by many applications

# 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

- Does the protocol exchange text or binary data?
  - Text – flexible and extensible
    - 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, …)

- Recommendation: prefer extensibility, rather than performance, unless profiling shows performance is a concern

# Channel Encoding

- Text-based protocols, can't directly send binary data
  - Example: old versions of `sendmail` used 8th bit to mark quoted data, and stripped it from data on input since email was guaranteed to be 7 bit ASCII only – hence must now encode binary files sent as attachments
- Data must be *encoded* to fit the character set in use and the encoding must be signalled
  - The MIME `Content-Transfer-Encoding:` header
  - May require negotiation of an appropriate transfer encoding, if data passing through several systems

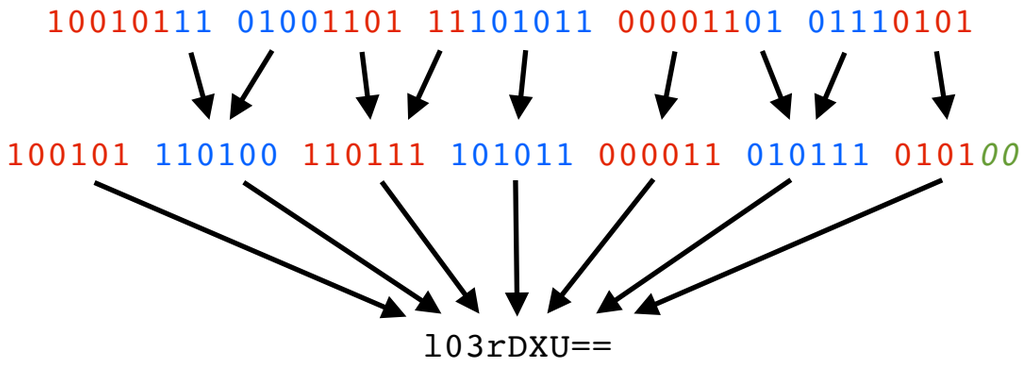
# Channel Encoding for Binary Data

- 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)

# Sending Unencoded 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);
```

# 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
  - ...
- Issues to consider:
  - What types of message are needed?
    - Highly application dependent – difficult to give general guidelines
  - How do interactions occur?
  - How are errors reported?

# Interaction Styles

- 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?

# Interaction Styles: Reducing 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 Errors Reported?

- Useful to have an extensible framework for error reporting
- 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, but lets older clients give meaningful response – backwards compatible

# Summary

- Higher-layers are ill-defined in the OSI model – not clear layer boundaries
- Often implemented in user applications, rather than in the operating system
- Consider layering as a design aid, if writing such applications

# Wrap-up

- Exam is worth 80% of course mark
  - Duration: 1.5 hours
  - Rubric: answer all 3 questions
  - Copies of past exam papers are on Moodle

- No revision lecture – send questions by email

# Networked Systems in Level 4

- Two taught modules cover networked systems:
  - Advanced Networking and Communications H
  - Distributed Algorithms and Systems H
- Individual projects in networked systems:
  - Look for projects supervised by members of the Embedded, Networked, and Distributed Systems research group
  - Talk to us if you're interested in networking-related projects – we generally have more project ideas than proposed, and can often suggest something that fits with your interests
  - Level 4 projects in this area can lead to MSci/PhD work, if interested