

Application Layer Protocols

Networked Systems 3
Lecture 17

Lecture Outline

- Application logic and protocol style
  - How is the application protocol data structured?
  - How do the interactions occur?
  - How are errors signalled?
- Application protocol examples
  - SMTP, POP3, HTTP, Jabber

Application Logic

- Session layer conveys data between applications
- The presentation and application layers impose meaning on that data to perform an application-level task
  - Deliver email
  - Retrieve web pages
  - Stream video
  - Etc.

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?

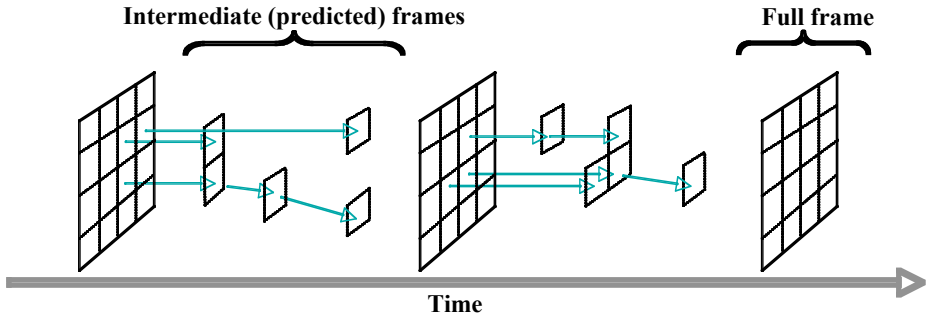
- Does the protocol exchange textual or binary messages?
  - Textual – flexible and extensible
    - See http://www.ietf.org/rfc/rfc3252.txt – “Binary Lexical Octet Ad-hoc Transport” – for a counter example (and note the publication date!)
    - High-level application layer protocols (e.g. email, web, 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, …)

Framing over TCP

- How to denote record boundaries?
  - TCP connection is reliable, but doesn't frame the data; must parse the byte stream
  - Requires a structured protocol:
    - HTTP/RTSP/SIP – text based messages, comprising an initial request, followed by headers, one per line, ending with a blank line
    - XML-based protocols (e.g. Jabber) – parse data until the appropriate closing tag is seen
    - Binary protocols – begin with a length field, telling how much data to read

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

Application Protocol Examples

- Wide range of application protocols used today
- Four common examples:
  - SMTP – sending email
  - POP3 – retrieving email
  - HTTP – world wide web
  - Jabber – open standard for instant messaging

Email

- One of the oldest Internet applications
  - Simple Mail Transfer Protocol (SMTP)
    - http://www.ietf.org/rfc/rfc5321.txt
    - http://www.ietf.org/rfc/rfc5322.txt

    Mail sending
    Original version: RFCs 821 and 822
  - Post Office Protocol, v3
    - http://www.ietf.org/rfc/rfc1939.txt

    Mail download from server
  - Internet Message Access Protocol, v4rev1
    - http://www.ietf.org/rfc/rfc3501.txt

    Remote mailbox manipulation

Sending Email: SMTP

```
S: 220 mr1.dcs.gla.ac.uk ESMTP Exim 4.42 Wed, 27 Feb 2008 10:31:18 +0000
C: HELO bo720-1-01.dcs.gla.ac.uk
S: 250 mr1.dcs.gla.ac.uk Hello bo720-1-01.dcs.gla.ac.uk [130.209.250.151]
C: MAIL FROM:csp@dcs.gla.ac.uk
S: 250 OK
C: RCPT TO:csp@csperkins.org
S: 250 Accepted
C: DATA
S: 354 Enter message, ending with "." on a line by itself
C: From: Colin Perkins <csp@dcs.gla.ac.uk>
C: To:   Colin Perkins <csp@csperkins.org>
C: Date: Wed 27 Feb 2008 10:32:45
C: Subject: Test
C:
C: This is a test
C: .
S: 250 OK id=1JUJa1-00073j-22
C: QUIT
S: 221 mr1.dcs.gla.ac.uk closing connection
```

Line-by-line request-response; very chatty
All commands are four characters + data
All responses are numeric + explanatory text

Structure of message: inspiration for HTTP design
Headers, blank line, then body
Many headers re-used identically in HTTP

Retrieving Email: POP3

```
S: +OK POP3 mr1 v2003.83rh server ready
C: USER csp
S: +OK User name accepted, password please
C: PASS ...password elided...
S: +OK Mailbox open, 4 messages
C: STAT
S: +OK 4 21142
C: LIST
S: +OK Mailbox scan listing follows
S: 1 1626
S: 2 7384
S: 3 6101
S: 4 6031
S: .
C: RETR 1
S: +OK 1626 octets
S: Return-path: <jcz@vxu.se>
S: Envelope-to: csp@dcs.gla.ac.uk
S: Delivery-date: Wed, 13 Feb 2008 18:40:07 +0000
S: ...email message elided...
S: .
C: DELE 1
S: +OK Message deleted
C: QUIT
S: +OK Sayonara
```

Line-by-line request-response; very chatty
Follows style of SMTP

World Wide Web: HTTP

- Hypertext Transport Protocol – HTTP/1.1
  - http://www.ietf.org/rfc/rfc2616.txt
- Flexible, textual, client-server protocol, with no unsolicited responses
  - Range of request types (`GET`, `PUT`, `OPTIONS`, …)
  - *Extremely* flexible headers → hard to parse, validate
  - All information needed to answer a request sent at once – response can be provided within single RTT

World Wide Web: HTTP

```
C: GET /index.html HTTP/1.1
C: Accept-Language: en-gb
C: Accept-Encoding: gzip, deflate
C: Accept: text/xml, application/xml, application/xhtml+xml, text/html, text/plain
C: User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-gb)
C:     AppleWebKit/523.12.2 (KHTML, like Gecko) Version/3.0.4 Safari/523.12.2
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: Date: Wed, 27 Feb 2008 22:44:25 GMT
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...
```

Initial request line ("GET…")
Headers, one per line
Blank line indicates end of request

Initial response code ("HTTP/1.1 200 OK")
Headers, one per line
The "Content-Length:" header indicates body size
Blank line indicates end of headers

Unstructured body data follows, with specified size

Instant Messaging

- Many proprietary instant messaging protocols
  - MSN, AIM, etc.
  - Poorly documented, trying to achieve lock-in
- Two open standards
  - Extensible Messaging and Presence Protocol (XMPP)
    - http://www.ietf.org/rfc/rfc3920.txt (a.k.a. “Jabber”)
  - SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)
    - Extremely complex; driven by telcos; not widely used

Instant Messaging: Jabber

```
C: <?xml version='1.0'?>
   <stream:stream to='example.com' xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
S: <?xml version='1.0'?>
   <stream:stream from='example.com' id='someid'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
C:   <message from='juliet@example.com' to='romeo@example.net'
              xml:lang='en'>
C:     <body>Art thou not Romeo, and a Montague?</body>
C:   </message>
S:   <message from='romeo@example.net' to='juliet@example.com'
              xml:lang='en'>
S:     <body>Neither, fair saint, if either thee dislike.</body>
S:   </message>
C: </stream:stream>
S: </stream:stream>
```

Source: RFC 3920

Data structured as an XML stream
Must be incrementally parsed, watching for closing tags
Easy to validate correctness, due to formal XML syntax

Inefficient, due to XML bloat → compresses well

Lots of open source tools: www.jabber.org

Application Protocol Examples

- Only given brief overview of these protocols – many details omitted
  - If implementing, read the standards documents, to understand the details!
- Internet applications traditionally built on very flexible, text-based, protocols
  - Open standards, open source, rapid evolution

Questions?