Services and Applications

G54ACC – IP and Up

Lecture 6

Contents

- Higher layers
- HTTP
- XMPP
- Radically different

Contents

- Higher layers
  - Session, Presentation
- HTTP
- XMPP
- Radically different

Session & Presentation Layers

- Session: open/close semi-permanent dialogue
  - SSH, RFC4251 *et al*
  - Floor control via, e.g., RTCP, RFC3550
- Presentation: conversion of data encodings
  - ASCII —Unicode UTF8
  - Unix (LF) — MS-DOS (CRLF)
    - Crazy filesystem shenanigans
  - MPEG
- Rather forced
  - Not traditionally part of TCP/IP stack
  - Functions still exist, just not explicitly layered
  - Generally lumped together in "application layer"

Contents

- Higher layers
- HTTP
  - Objects
  - Requests, Responses
  - State
  - Network usage
- XMPP
- Radically different

HTTP

- HyperText Transport Protocol
  - Evolution through three versions: 0.9, 1.0, 1.1
  - Client-server request-response protocol
- Actors
  - *Objects*, named via URIs (URLs)
  - *Clients* (or *user agents*), retrieve objects from ...
  - *Servers*, which store or generate objects
  - *Proxies*, sometimes get in the way
    - Provide features like caching, logging, transcoding, &c.

Objects

- Used to be HTML pages
  - HyperText Markup Language
- Might now be just about anything
  - Page, image, endpoint, computation
- Labelled via a Uniform Resource Identifier
  - In the web, this is a Uniform Resource Locator
    - scheme://host:port/path/to/resource/?query
  - Rarely, might be Uniform Resource Name
    - urn:ietf:rfc:3986

Requests

- HTTP/1.0
  - Connect TCP/80
  - Issue request method
    - GET, POST, HEAD
  - Issue headers
    - Language, &c.
  - Process result
- Issue further requests as required
  - Images, &c.
  - Separate connections

```
$ telnet google.com 80
Trying 173.194.37.104...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 302 Found
Location: http://www.google.co.uk/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie:
PREF=ID=76b0229e458ed281:TM=1283786147:LM=1283786147:S
=YChkvG74Grq1FOnS; expires=Wed, 05-Sep-2012 15:15:47
GMT; path=/; domain=.google.com
Set-Cookie: NID=38=E2-
NE5vYotdt6QPD8ENPiOrLaI3DJUS635jvNkw8AkMIRFp37i1jV8G6j
Pik3wvrWdMQRvw2BI1PKLp-
WS3bhZuRZ6lHZZfgfQDqXje6gb5BIXgBxATV_N1Glh-Lkqj3;
expires=Tue, 08-Mar-2011 15:15:47 GMT; path=/;
domain=.google.com; HttpOnly
Date: Mon, 06 Sep 2010 15:15:47 GMT
Server: gws
Content-Length: 221
X-XSS-Protection: 1; mode=block

<HTML><HEAD><meta http-equiv="content-type"
content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.co.uk/">here</A>.
</BODY></HTML>
```

Responses

- Status line
- *Response headers*
  - Provide meta-data
  - Location, type, mtime, length, &c.
- **Content**
  - Generated, read from file, &c.
  - Stateless: no server-side link between requests

```
$ telnet google.com 80
Trying 173.194.37.104...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 302 Found
Location: http://www.google.co.uk/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie:
PREF=ID=76b0229e458ed281:TM=1283786147:LM=1283786147:S
=YChkvG74Grq1FOnS; expires=Wed, 05-Sep-2012 15:15:47
GMT; path=/; domain=.google.com
Set-Cookie: NID=38=E2-
NE5vYotdt6QPD8ENPiOrLaI3DJUS635jvNkw8AkMIRFp37i1jV8G6j
Pik3wvrWdMQRvw2BI1PKLp-
WS3bhZuRZ6lHZZfgfQDqXje6gb5BIXgBxATV_N1Glh-Lkqj3;
expires=Tue, 08-Mar-2011 15:15:47 GMT; path=/;
domain=.google.com; HttpOnly
Date: Mon, 06 Sep 2010 15:15:47 GMT
Server: gws
Content-Length: 221
X-XSS-Protection: 1; mode=block

<HTML><HEAD><meta http-equiv="content-type"
content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.co.uk/">here</A>.
</BODY></HTML>
```

Handling State

- Necessary for some applications
  - E.g., Shopping carts, preferences, usage tracking, &c.
- Let the client do it
  - Avoids server scaling issues
  - Needs care with authentication though
- Cookies
  - Intended to be small
  - Server headers *Set-Cookie: <cookie-value>*
  - Client can then use *Cookie: <cookie-value>* in subsequent requests

Network Usage

- Connection per request is problematic
  - Inefficient if objects retrieved one at a time
  - Unfair if requests handled in parallel
    - In a TCP-fair sense as go through slow-start every time
    - Somewhat inefficient too due to TCP overheads
  - E.g., Old Netscape “limit to 4 connections” behaviour
- HTTP/1.1
  - Added persistent connections
  - Multiple requests on a single connection
  - ...although this still leads to scheduling issues

Contents

- Higher layers
- HTTP
- XMPP
  - Actors
  - Protocol
  - Extensions: BOSH
- Radically different

XMPP

- EXtensible Messaging & Presence Protocol
  - Basis for Jabber and Google Talk
  - Core provides XML streaming
    - Messages, Presence,
  - Extensions for HTTP binding, real-time media signalling, multi-user chat, publish-subscribe, &c.
  - RFC 3920, 3921; XEP series

Actors

- *Clients*, connect to ...
- *Servers*, which may interconnect directly
  - Servers are networked and can route messages
  - Typically multiplexed over single TCP connection
- *Gateways*, connect foreign clients to XMPP network
  - E.g., gateway Skype into a Google Talk session
- Address format is JID: `node@domain/resource`
  - Bare JID drops the /resource
- Security via TLS

Protocol

- Exchange *XML streams*
  - Multiple *XML stanzas*
  - ...encapsulated in `stream`
- Three stanzas defined
  - `message`
    - Push information
  - `presence`
    - Express availability
    - Negotiate subscriptions
  - `iq`
    - Request-response

```
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'>
...  encryption, authentication, and resource binding ...
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>
```

Extensions

- Administered by the XMPP Foundation
  - http://xmpp.org/extensions/
  - E.g., XEP-0124, BOSH; XEP-0206, XMPP over BOSH
- Bidirectional Streams Over Synchronous HTTP
  - Firewall friendly (TCP/80 vs. TCP/{5222-3, 5269})
  - Free compression (most servers support Gzip)
  - Hides unreliability
    - Emulates long-lived connection by a sequence of request-responses

BOSH

- Naive approach
  - Client polls server periodically for data
  - High latency and wastes bandwidth and battery
  - Matters especially on mobile clients
- Better
  - Client sends new request on receipt of response
  - Server always has outstanding connection down which it can push data
  - Works well with HTTP/1.1 but not so bad with 1.0

Corner Cases

- Client gets new data
  - Existing connection is blocked at the server
  - So open new connection to send, causing server to close old one
- Nothing happens for several minutes
  - Need a *keepalive*
  - Server returns empty and client sends a new empty request
- Constrained client
  - Can’t do HTTP/1.1 or multiple connections
  - Revert to naive polling mode

Contents

- Higher layers
- HTTP
- XMPP
- Radically different
  - Peer-to-peer
  - BitTorrent
  - Active networks

Peer-to-Peer (P2P)

- Both previous protocols were client-server
  - What if the server is the bottleneck?
  - ...whether through CPU, network, management...
- Alternative: peer-to-peer systems
  - E.g., CAN, CHORD, Pastry, BitTorrent, KaZaA, &c.
  - No designated central point (but consider BT *tracker*)
  - Typically self-organizing
- Often provide *distributed hash table* abstraction
  - Structured *vs*. unstructured
  - Usually scale as *N*.log(*N*) with network size *N*

BitTorrent

- Distributes load away from a single source site
  - (Approx.) proportional to popularity
- File divided into *pieces,* obtained separately
  - In random order, trying to keep *file* live
  - Each piece has a hash to provide integrity
- *Torrent descriptor* file made available to a *seed*
- *Tracker* knows who's participating in torrent
  - Can itself be distributed, e.g., DHT methods
- Client contacts tracker to obtain list of peers
  - Connect to peers, start downloading pieces
  - "Fair" to use many TCP connections to download?

Common P2P Issues

- Seeding the swarm: problems of flash crowds
- Anger of netadmins: breaks usage models
- No anonymity: leaves you open to attack
- Validity of metadata: bad torrents
- Leeching
  - Tit-for-tat schemes penalise newcomers
  - ...but what are the incentives for peers to share?

Active Networks

- And now for something completely different
  - Depending on how you look at it
- The network considers packets to be passive
  - Routers and middleboxes just forward
  - ...with a bit of rewriting, possibly triggering response
- What about if each packet was a bit of code?
  - Routers execute packet (header?) instead
  - Interesting research idea, never really took off
  - Lots of cool stuff about constraining runtime environment, proving properties on the code, &c.

Summary

- In the IP stack, session and presentation layers are generally subsumed into application layer
- Two widely used and interesting application protocols are HTTP and XMPP
- Boundless possibility, e.g., peer-to-peer active networks