Introduction to Computer Networks

Application Layer Overview

Where we are in the Course

• We are finally at the Application Layer!
  – Builds distributed “network services” (DNS, Web) on Transport services
Topic

• The DNS (Domain Name System)
  – Human-readable host names, and more
  – Distributed namespace & resolution

Names and Addresses

• **Names** are higher-level identifiers for resources
• **Addresses** are lower-level locators for resources
  – Multiple levels, e.g. full name → email → IP address → Ethernet address
• **Resolution** (or lookup) is mapping a name to an address

Directory Name, e.g. “Andy Tanenbaum,”
or “flits.cs.vu.nl”

Lookup

Name, e.g. “Vrijie Universiteit, Amsterdam”
or IPv4 “130.30.27.38”

Address, e.g.

Directory

Network
Before the DNS – HOSTS.TXT

- Directory was a file HOSTS.TXT regularly retrieved for all hosts from a central machine at the NIC (Network Information Center)
- Names were initially flat, became hierarchical (e.g., lcs.mit.edu) ~85
- Neither manageable nor efficient as the ARPANET grew ...

DNS

- A naming service to map between host names and their IP addresses (and more)
  - www.uwa.edu.au → 130.95.128.140
- Goals:
  - Easy to manage (esp. with multiple parties)
  - Efficient (good performance, few resources)
- Approach:
  - Distributed directory based on a hierarchical namespace
  - Automated protocol to tie pieces together
DNS Namespace

- Hierarchical, starting from “.” (dot, typically omitted)

TLDs (Top-Level Domains)

- Run by ICANN (Internet Corp. for Assigned Names and Numbers)
  - Starting in ’98; naming is financial, political, and international 😊

- 22+ generic TLDs
  - Initially .com, .edu, .gov, .mil, .org, .net
  - Added .aero, .museum, etc. from ’01 through .xxx in ’11
  - Different TLDs have different usage policies

- ~250 country code TLDs
  - Two letters, e.g., “.au”, plus international characters since 2010
  - Widely commercialized, e.g., .tv (Tuvalu)
  - Many domain hacks, e.g., instagr.am (Armenia), goo.gl (Greenland)
DNS Zones

• A **zone** is a contiguous portion of the namespace

    ![DNS Zones Diagram]

DNS Zones (2)

• Zones are the basis for distribution
  – EDU Registrar administers .edu
  – UW administers washington.edu
  – CS&E administers cs.washington.edu

• Each zone has a **nameserver** to contact for information about it
  – Zone must include contacts for delegations, e.g., .edu knows nameserver for washington.edu
DNS Resource Records

• A zone is comprised of DNS resource records that give information for its domain names

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA</td>
<td>Start of authority, has key zone parameters</td>
</tr>
<tr>
<td>A</td>
<td>IPv4 address of a host</td>
</tr>
<tr>
<td>AAAA (“quad A”)</td>
<td>IPv6 address of a host</td>
</tr>
<tr>
<td>CNAME</td>
<td>Canonical name for an alias</td>
</tr>
<tr>
<td>MX</td>
<td>Mail exchanger for the domain</td>
</tr>
<tr>
<td>NS</td>
<td>Nameserver of domain or delegated subdomain</td>
</tr>
</tbody>
</table>

DNS Resource Records (2)
DNS Resolution

- DNS protocol lets a host resolve any host name (domain) to IP address
- If unknown, can start with the root nameserver and work down zones
- Let’s see an example first …

DNS Resolution (2)

- flits.cs.vu.nl resolves robot.cs.washington.edu
Iterative vs. Recursive Queries

• Recursive query
  – Nameserver completes resolution and returns the final answer
  – E.g., flits → local nameserver

• Iterative query
  – Nameserver returns the answer or who to contact next for the answer
  – E.g., local nameserver → all others

DNS Resolution

• What are the implications of the resolution process presented above?
Iterative vs. Recursive Queries (2)

- Recursive query
  - Lets server offload client burden (simple resolver) for manageability
  - Lets server cache over a pool of clients for better performance

- Iterative query
  - Lets server “file and forget”
  - Easy to build high load servers

Caching

- Resolution latency should be low
  - Adds delay to web browsing

- Cache query/responses to answer future queries immediately
  - Including partial (iterative) answers
  - Responses carry a TTL for caching
Caching (2)

- flits.cs.vu.nl now resolves eng.washington.edu
  - And previous resolutions cut out most of the process

Local Nameservers

- Local nameservers typically run by IT (enterprise, ISP)
  - But may be your host or AP
  - Or alternatives e.g., Google public DNS

- Clients need to be able to contact their local nameservers
  - Typically configured via DHCP
Root Nameservers

- Root (dot) is served by 13 server names
  - a.root-servers.net to m.root-servers.net
  - All nameservers need root IP addresses
  - Handled via configuration file (named.ca)

- There are >250 distributed server instances
  - Highly reachable, reliable service
  - Most servers are reached by IP anycast
    (Multiple locations advertise same IP! Routes take client to the closest one.)
  - Servers are IPv4 and IPv6 reachable

DNS Protocol

- Query and response messages
  - Built on UDP messages, port 53
  - ARQ for reliability; server is stateless!
  - Messages linked by a 16-bit ID field

![](image1)

DNS Protocol (2)

- Service reliability via replicas
  - Run multiple nameservers for domain
  - Return the list; clients use one answer
  - Helps distribute load too

![](image2)
DNS Issues

• Are there any security issues with DNS?

DNS Issues

• Recall that CDNs allow you to replicate content at multiple locations

• DNS can be used to redirect clients to CDN nodes based on the resolver IP

• But this can go wrong sometimes…
Introduction to Computer Networks

HTTP, the HyperText Transfer Protocol (§7.3.1-7.3.4)

Web Context

Page as a set of related HTTP transactions
Web Protocol Context

- HTTP is a request/response protocol for fetching Web resources
  - Runs on TCP, typically port 80
  - Part of browser/server app

**Diagram**

```
<table>
<thead>
<tr>
<th>browser</th>
<th>server</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>HTTP</td>
</tr>
<tr>
<td>TCP</td>
<td>TCP</td>
</tr>
<tr>
<td>IP</td>
<td>IP</td>
</tr>
<tr>
<td>802.11</td>
<td>802.11</td>
</tr>
</tbody>
</table>
```

Fetching a Web page with HTTP

- Start with the page URL:
  http://en.wikipedia.org/wiki/Vegemite

- Steps:
  - Resolve the server to IP address (DNS)
  - Set up TCP connection to the server
  - Send HTTP request for the page
  - (Await HTTP response for the page)
  - Execute / fetch other Web resources / render
  - Clean up any idle TCP connections
Static vs Dynamic Web pages

- Static web page is a file contents, e.g., image
- Dynamic web page is the result of program execution
  - Javascript on client, PHP on server, or both

HTTP Protocol

- Originally a simple protocol, with many options added over time
  - Text-based commands, headers

- Try it yourself:
  - As a “browser” fetching a URL
  - Run “telnet en.wikipedia.org 80”
  - Type “GET /wiki/Vegemite HTTP/1.0” to server followed by a blank line
  - Server will return HTTP response with the page contents (or other info)
HTTP Protocol (2)

• Commands used in the request

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Read a Web page</td>
</tr>
<tr>
<td>HEAD</td>
<td>Read a Web page's header</td>
</tr>
<tr>
<td>POST</td>
<td>Append to a Web page</td>
</tr>
<tr>
<td>PUT</td>
<td>Store a Web page</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the Web page</td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the incoming request</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Connect through a proxy</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Query options for a page</td>
</tr>
</tbody>
</table>

HTTP Protocol (3)

• Codes returned with the response

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client's request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>
HTTP Protocol (4)

- Many header fields specify capabilities and content
  - E.g., Content-Type: text/html, Cookie: lect=8-4-http

<table>
<thead>
<tr>
<th>Function</th>
<th>Example Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser capabilities</td>
<td>User-Agent, Accept, Accept-Charset, Accept-Encoding,</td>
</tr>
<tr>
<td>(client → server)</td>
<td>Accept-Language</td>
</tr>
<tr>
<td>Caching related</td>
<td>If-Modified-Since, If-None-Match, Date, Last-Modified,</td>
</tr>
<tr>
<td>(mixed directions)</td>
<td>Expires, Cache-Control, ETag</td>
</tr>
<tr>
<td>Browser context</td>
<td>Cookie, Referer, Authorization, Host</td>
</tr>
<tr>
<td>(client → server)</td>
<td></td>
</tr>
<tr>
<td>Content delivery</td>
<td>Content-Encoding, Content-Length, Content-Type,</td>
</tr>
<tr>
<td>(server → client)</td>
<td>Content-Language, Content-Range, Set-Cookie</td>
</tr>
</tbody>
</table>

Introduction to Computer Networks

HTTP Performance (§7.3.4, §7.5.2)
PLT (Page Load Time)

• PLT is the key measure of web performance
  – From click until user sees page
  – Small increases in PLT decrease sales

• PLT depends on many factors
  – Structure of page/content
  – HTTP (and TCP!) protocol
  – Network RTT and bandwidth

Page Load Time

• How can we optimize page load time?
  – Consider all layers of the stack
  – Consider different kinds of web pages and what they contain
Early Performance

• HTTP/1.0 used one TCP connection to fetch one web resource
  – Made HTTP very easy to build
  – But gave fairly poor PLT...

Early Performance (2)

• Many reasons why PLT is larger than necessary
  – Sequential request/responses, even when to different servers
  – Multiple TCP connection setups to the same server
  – Multiple TCP slow-start phases

• Network is not used effectively
  – Worse with many small resources / page
Parallel Connections

- One simple way to reduce PLT
  - Browser runs multiple (8, say) HTTP instances in parallel
  - Server is unchanged; already handled concurrent requests for many clients
- How does this help?
  - Single HTTP wasn’t using network much ...
  - So parallel connections aren’t slowed much
  - Pulls in completion time of last fetch

Persistent Connections

- Parallel connections compete with each other for network resources
  - 1 parallel client ≈ 8 sequential clients?
  - Exacerbates network bursts, and loss
- Persistent connection alternative
  - Make 1 TCP connection to 1 server
  - Use it for multiple HTTP requests
Persistent Connections (2)

One request per connection
Sequential requests per connection
Pipelined requests per connection

Persistent Connections (3)

• Widely used as part of HTTP/1.1
  – Supports optional pipelining
  – PLT benefits depending on page structure, but easy on network

• Issues with persistent connections
  – How long to keep TCP connection?
  – Can it be slower? (Yes. But why?)
Web Caching

- Users often revisit web pages
  - Big win from reusing local copy!
  - This is caching

- Key question:
  - When is it OK to reuse local copy?

Web Caching (2)

- Locally determine copy is still valid
  - Based on expiry information such as “Expires” header from server
  - Or use a heuristic to guess (cacheable, freshly valid, not modified recently)
  - Content is then available right away
Web Caching (3)

- Revalidate copy with server
  - Based on timestamp of copy such as “Last-Modified” header from server
  - Or based on content of copy such as “Etag” header from server
  - Content is available after 1 RTT

Web Caching (4)

- Putting the pieces together:

Web Proxies

- Place intermediary between pool of clients and external web servers
  - Benefits for clients include greater caching and security checking
  - Organizational access policies too!

- Proxy caching
  - Clients benefit from a larger, shared cache
  - Benefits limited by secure and dynamic content, as well as “long tail”

Web Proxies (2)

- Clients contact proxy; proxy contacts server
Introduction to Computer Networks

CDNs (Content Delivery Networks) (§7.5.3)

Context

• As the web took off in the 90s, traffic volumes grew and grew. This:
  1. Concentrated load on popular servers
  2. Led to congested networks and need to provision more bandwidth
  3. Gave a poor user experience

• Idea:
  – Place popular content near clients
  – Helps with all three issues above
Questions

• What are good locations to establish CDN nodes?

Before CDNs

• Sending content from the source to 4 users takes $4 \times 3 = 12$ “network hops” in the example
After CDNs

- Sending content via replicas takes only $4 + 2 = 6$ “network hops”

![Diagram of CDNs]

Popularity of Content

- Zipf’s Law: few popular items, many unpopular ones; both matter

![Graph of Zipf's Law]

George Zipf (1902-1950)

How to place content near clients?

• Use browser and proxy caches
  – Helps, but limited to one client or clients in one organization

• Want to place replicas across the Internet for use by all nearby clients
  – Done by clever use of DNS
Content Delivery Network (2)

- DNS resolution of site gives different answers to clients
  - Tell each client the site is the nearest replica (map client IP)

![Diagram of Content Delivery Network]

Business Model

- Clever model pioneered by Akamai
  - Placing site replica at an ISP is win-win
  - Improves site experience and reduces bandwidth usage of ISP

![Diagram of Business Model]
Topic

- The Future of HTTP
  - How will we make the web faster?
  - A brief look at some approaches

Modern Web Pages

- Waterfall diagram shows progression of page load

webpagetest tool for http://coursera.org (Firefox, 5/1 Mbps, from VA, 3/1/13)
Modern Web Pages (2)

Yikes!
-23 requests
-1 Mb data
-2.6 secs

webpagetest tool for http://coursera.org (Firefox, 5/1 Mbps, from VA, 3/1/13)

Modern Web Pages (3)

Yay! (Network used well)

- Waterfall and PLT depends on many factors
  - Very different for different browsers
  - Very different for repeat page views
  - Depends on local computation as well as network
Recent work to reduce PLT

Pages grow ever more complex!
   - Larger, more dynamic, and secure
   - How will we reduce PLT?

1. Better use of the network
   - HTTP/2 effort based on SPDY

2. Better content structures
   - mod_pagespeed server extension

SPDY (“speedy”)

• A set of HTTP improvements
  - Multiplexed (parallel) HTTP requests on one TCP connection
  - Client priorities for parallel requests
  - Compressed HTTP headers
  - Server push of resources

• Now being tested and improved
  - Default in Chrome, Firefox
  - Basis for an HTTP/2 effort
mod_pagespeed

• Observation:
  – The way pages are written affects how quickly they load
  – Many books on best practices for page authors and developers

• Key idea:
  – Have server re-write (compile) pages to help them load quickly!
  – mod_pagespeed is an example

mod_pagespeed (2)

• Apache server extension
  – Software installed with web server
  – Rewrites pages “on the fly” with rules based on best practices

• Example rewrite rules:
  – Minify Javascript
  – Flatten multi-level CSS files
  – Resize images for client
  – And much more (100s of specific rules)