CSCI-1680
DNS II + WWW

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti
TCP milestone II: Schedule meeting by Friday, April 22 (tomorrow)
   - If you have trouble finding a slot, let us know
   - Questions?

HW3: Due Tuesday, April 26
   - Problem 1 should help with Milestone II

Office hours today
   - Remote: 3-5pm
   - Will probably add group-style hours 5-7pm, look for announcement
   - Whenever my door is open, or if the sign says so
$ dig cs.brown.edu @10.1.1.10
; <<< DiG 9.10.6 <<<> cs.brown.edu @10.1.1.10
; ; global options: +cmd
; ; Got answer:
; ; ->>HEADER<<<- opcode: QUERY, status: NOERROR, id: 8536
; ; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

; ; OPT PSEUDOSECTION:
; ; EDNS: version: 0, flags:; udp: 1220
; ; QUESTION SECTION:
;cs.brown.edu. IN A

; ; ANSWER SECTION:
 cs.brown.edu. 1800 IN A 128.148.32.12

; ; Query time: 69 msec
; ; SERVER: 10.1.1.10#53(10.1.1.10)
; ; WHEN: Tue Apr 19 09:03:39 EDT 2022
; ; MSG SIZE  rcvd: 57
# DNS record types

<table>
<thead>
<tr>
<th>RR Type</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>IPv4 Address</td>
<td>128.148.56.2</td>
</tr>
<tr>
<td><strong>AAAA</strong></td>
<td>IPv6 Address</td>
<td>2001:470:8956:20::1</td>
</tr>
<tr>
<td><strong>CNAME</strong></td>
<td>Specifies an alias (&quot;Canonical name&quot;)</td>
<td>systems.cs.brown.edu. 86400 IN CNAME systems-v3.cs.brown.edu. systems-v3.cs.brown.edu. 86400 IN A 128.148.36.51</td>
</tr>
<tr>
<td><strong>MX</strong></td>
<td>Mail servers</td>
<td>MX <code>&lt;priority&gt;</code> <code>&lt;ip&gt;</code> eg. MX 10 1.2.3.4</td>
</tr>
<tr>
<td><strong>SOA</strong></td>
<td>Start of authority</td>
<td>Information about who owns a zone</td>
</tr>
<tr>
<td><strong>PTR</strong></td>
<td>Reverse IP lookup</td>
<td>7.34.148.128.in-addr.arpa. 86400 IN PTR quanto.cs.brown.edu.</td>
</tr>
<tr>
<td><strong>SRV</strong></td>
<td>How to reach specific services (eg. host, port)</td>
<td><code>_minecraft._tcp.example.net</code> 3600 SRV <code>&lt;priority&gt;</code> <code>&lt;weight&gt;</code> <code>&lt;port&gt;</code> <code>&lt;server IP&gt;</code></td>
</tr>
</tbody>
</table>

Great Firewall of CIT

If attacker is on the path (say, it is the ISP, or a malicious version of TStaff), what could they do?

- If attacker controls server, C can:
  - Forge/alter responses
  - Block traffic

[Diagram showing network flow and DNS resolution issues with IP and domain names marked in red.]
Great Firewall of CIT

If attacker is on the path (say, it is the ISP, or a malicious version of TStaff), what could they do?

– Can sniff all DNS queries
– Send fake responses back first
– Could do this selectively, to direct facebook.com to cs.brown.edu, for example...
“Helpful” ISPs

- Many ISPs hijack NXDOMAIN responses to “help” by offering search and advertisement related to the domain
  - Redirect to WiFi login
  - Respond to bogus domains with "HELPFUL INFO" (ie. ADS)

  e.g., Google.com → Ad-based search page (typo)

→ This was once more common, but some ISPs still may do it.
DEFENSES AGAINST DOS HIJACKING/SPOOFING

- DNSSEC: Protocol to verify whole hierarchy
  More info at
  of DNS lookups
  END OF LECTURE 21

- HIERARCHY MUST SUPPORT
  AT ALL LEVELS
  CSLAB, CS. BROWN. EDU.

- DNS OVER HTTPS
  DNS OVER TLS
  TUNNEL DNS TRAFFIC
  THROUGH A MORE SECURE PROTOCOL

- GREAT, BUT
  POSSIBLE PRIVACY CONCERN
  (TRAFFIC GOES ELSEWHERE)

- MIGHT INTERFERE W/ LOCAL DNS
“Helpful” ISPs

• Many ISPs hijack NXDOMAIN responses to “help” by offering search and advertisement related to the domain

• E.g., www.bicycleisntadomain.com doesn’t (currently) exist
  – Could return a page with search and ads on bicycles (or domain registrations?)
HTTP: Hypertext Transfer Protocol
Precursors

• 1945, Vannevar Bush, Memex:
  – “a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility”

• Precursors to hypertext
  – “The human mind [...] operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain”

• His essay, “As we may think”, is worth reading!
Tim Berners-Lee

- Physicist at CERN, trying to solve real problem
  - Distributed access to data
- WWW: distributed database of pages linked through the Hypertext Transfer Protocol
  - First HTTP implementation: 1990
  - HTTP/0.9 – 1991
    - Simple GET command
  - HTTP/1.0 – 1992
    - Client/server information, simple caching
  - HTTP/1.1 – 1996
    - Extensive caching support
    - Host identification
    - Pipelined, persistent connections, …

The first webserver
• **HTTP/2 – 2015**
  - Main goal: reduce latency
  - True multiplexing of messages
  - Binary encoding, compression

• **HTTP/3 – Draft as of 2022**
  - Same goals as HTTP/2
  - Integrates security via TLS (next class…)
  - Replace transport layer with **QUIC**
  - Already supported in >70% of browsers

http://httpwg.org/specs/rfc7540.html
Why so successful?

• Ability to self publish
  – Like youtube for video

• But...
  – Mechanism is easy
  – Independent, open
  – Free

• Current debate
  – Is it easy enough? Why is facebook so popular, even though it is not open?
Components

- **Content**
  - Objects (may be static or dynamically generated)

- **Clients**
  - Send requests / Receive responses

- **Servers**
  - Receive requests / Send responses
  - Store or generate content

- **Proxies/Middleboxes**
  - Placed between clients and servers
  - Provide extra functions
    - Caching, anonymization, logging, transcoding, filtering access
  - Explicit or transparent

---

\(1 \text{ IN THE MIDDLE: NETWORK DEVICES}\)

\(2 \text{ CLIENT} \)

\(3 \text{ SERVER} \)

\(\text{GET} \quad \text{R} \)

\(2\)

\(3\)

\(\text{EXAMPLES: CACHES, FILTERING}\)
Ingredients

- HTTP - Fetch/Upload Data
  - Hypertext Transfer Protocol
- HTML
  - Language for description of content
- Names (mostly URLs)
  - Won’t talk about URIs, URNs

J - Base for describing pages
- Just one type of content
- Can be any file, HTML, CSS, JS, images, binary data, JSON...
How to find stuff?

- DNS: names for one or more hosts
  - eg. cs.brown.edu

- How do we ask for a specific resource from this host?

URL: Uniform Resource Locator
How to find stuff: URLs

protocol://[name@]hostname[:port]/directory/resource?k1=v1&k2=v2#tag

- **Name:** can identify a client
- **Hostname:** FQDN or IP address
- **Port number:** defaults to common protocol port (e.g., 80, 22)
- **Directory:** path to the resource
- **Resource:** name of the object

After that, various delimiters to specify further, common examples:
- ?parameters are passed to the server for execution
- #tag allows jumps to named tags within document
HTTP

- Client-server protocol
- Protocol (but not data) in ASCII (before HTTP/2)
- Stateless
- Extensible (header fields)
- Server typically listens on port 80 (or 443, with TLS)
- Server sends response, may close connection (client may ask it to stay open)

Lots of HTTP development to optimize this (multiplex/reuse connections)
Steps in HTTP\(^{1.0}\) Request

- Open TCP connection to server
- Send request
- Receive response
- TCP connection terminates
  - How many RTTs for a single request? \textit{Many}
- You may also need to do a DNS lookup first!
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 24 Mar 2011 12:58:46 GMT
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g
ETag: "840a88b-236c-49f3992853bc0"
Accept-Ranges: bytes
Content-Length: 9068
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
HTTP Request

- **Method:**
  - GET: current value of resource, run program
  - HEAD: return metadata associated with a resource
  - POST: update a resource, provide input for a program

- **Headers:** useful info for proxies or the server
  - E.g., desired language

[Diagram of HTTP request structure with columns for method, URL, version, headers, and body.]
Sample Browser Request

GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh ... 
Accept: text/xml,application/xml ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

In your browser: Inspect element -> Network view
HTTP Response

- Status Codes:
  - 1xx: Information  e.g., 100 Continue
  - 2xx: Success    e.g., 200 OK
  - 3xx: Redirection e.g., 302 Found (elsewhere)
  - 4xx: Client Error e.g., 404 Not Found
  - 5xx: Server Error e.g., 503 Service Unavailable
HTTP is Stateless

- Each request/response treated independently
- Servers not required to maintain state

Client sends info about state via cookies
HTTP is Stateless

• Each request/response treated independently
• Servers not required to maintain state
• This is good!
  – Improves server scalability
HTTP is Stateless

- Each request/response treated independently
- Servers not required to maintain state
- This is good!
  - Improves server scalability
- This is also bad...
  - Some applications need persistent state
  - Need to uniquely identify user to customize content
  - E.g., shopping cart, web-mail, usage tracking, (most sites today!)
HTTP Cookies

- Client-side state maintenance
  - Client stores small state on behalf of server
  - Sends request in future requests to the server
  - Cookie value is meaningful to the server (e.g., session id)

- Can provide authentication
Anatomy of a Web Page

• HTML content
• A number of additional resources
  – Images
  – Scripts
  – Frames
• Browser makes one HTTP request for each object
  – Course web page: 14 objects
  – Modern web pages: hundreds of objects
Modern web pages and HTTP

- Web APIs: HTTP response/requests are a standard way to ask for anything
- Modern web pages: use Javascript to make lots of requests without reloading page
  - And can use APIs for all kinds of other stuff
Example: Github public API

$ curl https://api.github.com/users/ndemarinis
{
  "login": "ndemarinis",
  "id": 1191319,
  "node_id": "MDQ6VXNlcjExOTEzMTk=",
  "gravatar_id": "",
  "url": "https://api.github.com/users/ndemarinis",
  "type": "User",
  "site_admin": false,
  "name": "Nick DeMarinis",
  "blog": "https://vty.sh",
  "twitter_username": null,
  "public_repos": 10,
  ...
}
HTTP Performance

• What matters for performance?
• Depends on type of request
  – Lots of small requests (objects in a page)
  – Some big requests (large download or video)
Small Requests
Small Requests

• Latency matters
Small Requests

- Latency matters
- RTT dominates
Small Requests

- Latency matters
- RTT dominates
- Two major causes:
  - Opening a TCP connection
  - Actually sending the request and receiving response
  - And a third one: DNS lookup!
How can we reduce the number of connection setups?

- Keep the connection open and request all objects serially
  - Works for all objects coming from the same server
  - Which also means you don’t have to “open” the window each time
- Persistent connections (HTTP/1.1)
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh ... 
Accept: text/xml,application/xml ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Small Requests (cont)
Second problem is that requests are serialized
  – Similar to stop-and-wait protocols!
Small Requests (cont)

• Second problem is that requests are serialized
  – Similar to stop-and-wait protocols!

• Two solutions
  – Pipelined requests (similar to sliding windows)
  – Parallel Connections
    • Browsers implement this differently—see “Inspect element”
  – How are these two approaches different?
HTTP/2

• Adds more options to trade off:
• Multiplexed streams on same connection
  – Plus stream weights, dependencies
• No head of line blocking!
  – But what happens if there is packet loss?

https://www.twilio.com/blog/2017/10/http2-issues.html
HTTP/3

- Mapping of HTTP semantics onto QUIC
  - E.g., QUIC already implements multiple streams, and HTTP doesn’t need to do it
- QUIC: Another transport-layer protocol, intended to replace TCP
  - RFC9000
  - Same goals as TCP, but…
    - Integrates security by default (TLS, next class)
    - Supports multiple streams at once
    - Various tricks to reduce message size and latency

- By moving multiplexing into the transport layer, can do so in a way that benefits HTTP (no head of line blocking!)
Comparison: QUIC’s handshake
Larger Objects

- Problem is throughput in bottleneck link
- Solution: HTTP Proxy Caching
  - Also improves latency, and reduces server load
How to Control Caching?

• Server sets options
  – **Expires** header
  – No-Cache header

• Client can do a conditional request:
  – Header option: if-modified-since
  – Server can reply with 304 NOT MODIFIED
Caching

• Where to cache content?
Caching

- Where to cache content?
  - Client (browser): avoid extra network transfers
Caching

• Where to cache content?
  – Client (browser): avoid extra network transfers
  – Server: reduce load on the server
Caching

- Where to cache content?
  - Client (browser): avoid extra network transfers
  - Server: reduce load on the server
  - Service Provider: reduce external traffic
Caching

- Why caching works?
  - Locality of reference:
    - Users tend to request the same object in succession
    - Some objects are popular: requested by many users
How well does caching work?

• Very well, up to a point
  – Large overlap in requested objects
  – Objects with one access place upper bound on hit ratio
  – Dynamic objects not cacheable*

• Example: Wikipedia
  – About 400 servers, 100 are HTTP Caches (Squid)
  – 85% Hit ratio for text, 98% for media

* But can cache portions and run special code on edges to reconstruct
Reverse Proxies

• Close to the server
  – Also called Accelerators
  – Only work for static content
Forward Proxies

• Typically done by ISPs or Enterprises
  – Reduce network traffic and decrease latency
  – May be transparent or configured
Content Distribution Networks

• Integrate forward and reverse caching
  – One network generally administered by one entity
  – E.g. Akamai

• Provide document caching
  – Pull: result from client requests
  – Push: expectation of high access rates to some objects

• Can also do some processing
  – Deploy code to handle some dynamic requests
  – Can do other things, such as transcoding
Example CDN

- ISP-1
- Forward proxies
- ISP-2
- Backbone ISP
- Server
- CDN
- Clients
How Akamai works

• Akamai has cache servers deployed close to clients
  – Co-located with many ISPs
• Challenge: make same domain name resolve to a proxy close to the client
• Lots of DNS tricks. BestBuy is a customer
  – Delegate name resolution to Akamai (via a CNAME)
• From Brown:
  dig www.bestbuy.com
  ;; ANSWER SECTION:
  www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
  – Ping time: 2.53ms
  a1105.b.akamai.net. 20 IN A 198.7.236.240
  – Ping time: 3.20ms
• From Berkeley, CA:
  a1105.b.akamai.net. 20 IN A 198.189.255.200
  a1105.b.akamai.net. 20 IN A 198.189.255.207
  – Ping time: 3.20ms
dig www.bestbuy.com

;; ANSWER SECTION:
www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240

;; AUTHORITY SECTION:
b.akamai.net. 1101 IN NS nlb.akamai.net.
b.akamai.net. 1101 IN NS n0b.akamai.net.

;; ADDITIONAL SECTION:
n0b.akamai.net. 1267 IN A 24.143.194.45
n1b.akamai.net. 2196 IN A 198.7.236.236
### DNS Resolution

```plaintext
dig www.bestbuy.com

;; ANSWER SECTION:
www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240

;; AUTHORITY SECTION:
b.akamai.net. 1101 IN NS n1b.akamai.net.
b.akamai.net. 1101 IN NS n0b.akamai.net.

;; ADDITIONAL SECTION:
n0b.akamai.net. 1267 IN A 24.143.194.45
n1b.akamai.net. 2196 IN A 198.7.236.236
```

- `n1b.akamai.net` finds an edge server close to the client’s local resolver
  - Uses knowledge of network: BGP feeds, traceroutes. Their secret sauce…
Example

dig www.bestbuy.com
@109.69.8.51

www.bestbuy.com. 12 IN A 23.60.221.144

tracert to 23.60.221.144 (23.60.221.144), 64 hops max, 52 byte packets
1 router (192.168.1.1)  44.072 ms  1.572 ms  1.154 ms
2 138.16.160.253 (138.16.160.253)  4.276 ms  3.460 ms  2.722 ms
3 10.1.18.5 (10.1.18.5)  1.854 ms  1.649 ms  1.354 ms
4 10.1.80.5 (10.1.80.5)  2.342 ms  2.183 ms  2.132 ms
5 lab-net-r-230.net.brown.edu (128.148.230.6)  4.740 ms  4.244 ms  3.844 ms
6 131.109.200.1 (131.109.200.1)  3.581 ms  3.286 ms  3.238 ms
7 host-198-7-224-105.oshean.org (198.7.224.105)  4.288 ms  6.218 ms  8.332 ms
8 5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.209 ms  6.103 ms  5.031 ms
9 ee-4.00.bstmn07.us.bb.gin.ntt.net (129.250.66.93)  3.982 ms  5.824 ms  4.514 ms
10 ee-6.24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  4.883 ms  10.205 ms  10.477 ms
11 ae-6.00.nycmny01.us.bb.gin.ntt.net (129.250.5.62)  9.256 ms
12 ee-6.00.nycmny01.us.bb.gin.ntt.net (129.250.3.94)  8.538 ms  7.955 ms
13 a23-60-221-144.deploy.static.akamai Technologies.com (23.60.221.144)  94.884 ms  92.779 ms  53.291 ms
Other CDNs

- Akamai, Limelight, Cloudflare
- Amazon, Facebook, Google, Microsoft
- Netflix

- Where to place content?
- Which content to place? Pre-fetch or cache?
How to Control Caching?

- Server sets options
  - **Expires** header
  - No-Cache header
- Client can do a conditional request:
  - Header option: if-modified-since
  - Server can reply with 304 NOT MODIFIED
- More when we talk about Content Distribution
Next Class

- Global data distribution
  - CDN and P2P

- How to create your own application layer protocol!
  - Data / RPC