CSCI-1680
DNS II + WWW

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti
• TCP grading: sign up for a grading meeting
  – Let us know if you don’t see any slots
• Final project: you should have received an email about teams
• Project proposal: due by Monday, 12/5
  – Really not much required, just sketch what you want to do and your plan
  – I’ll review these daily: submit earlier => earlier feedback!
• My office hours today: 3-4pm (CIT316, zoom), 5-7pm (location TBA)
More on DNS
$ 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
"Helpful" ISPs

Some ISPs hijack DNS for “helpful” purposes

• Could rewrite NXDOMAIN responses => search page with ads
  – google.com => ISP search page
Some ISPs hijack DNS for “helpful” purposes

• Could rewrite NXDOMAIN responses => search page with ads
  – google.com => ISP search page

• Captive portals: When joining public Wifi, respond to all DNS queries with IP of login page
  – Most OSes/browsers have mechanisms to detect this
What can be done?

Some defenses against DNS spoofing/hijacking

- **DNSSEC**: protocol to sign/verify hierarchy of DNS lookups
  - Expensive to deploy, hierarchy must support at all levels
  - APNIC DNSSEC monitor: [https://stats.labs.apnic.net/dnssec](https://stats.labs.apnic.net/dnssec)

- **Tunneling DNS**: client uses DNS via more secure protocol
  - DNS over HTTPS
  - DNS over TLS
HTTP: Hypertext Transfer Protocol
HTTP

• “Application protocol for distributed, collaborative hypermedia information systems”

• Fundamental protocol behind “the web”

• Today, HTTP is fundamental of most things we do on the Internet… and thus most modern applications

But what is hypertext?
Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia
(Redirected from HTTP)

The Hypertext Transfer Protocol (HTTP) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP protocol version that was named 0.9.[2]

That first version of HTTP protocol soon evolved into a more elaborated version that was the first draft toward a future version 1.0.[3]

Development of early HTTP Requests for Comments (RFCs) started a few years later and it was a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium.
Hypertext predates HTTP

1945: Vannevar Bush envisions the “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, …
• HTTP/2 – 2015
  – Main goal: reduce latency
  – True multiplexing of messages
  – Binary encoding, compression

• HTTP/3 – 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
Ingredients

- **HTTP**
  - Hypertext Transfer Protocol

- **HTML**
  - Language for description of content

- Names (mostly URLs)
  - Won’t talk about URIs, URNs

- **BASIS FOR DESCRIBING**
  - **HTML, CSS, JS**
  - **DIRECT DATA**

- **FETCH/UPLOAD DATA**
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 (eg. 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
How to find stuff: URLs

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

HTTP/HTTPS/ZOOY

INTERNET

APPLICATION TO REQ.

RESOURCE.
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 dev. to optimize this.
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?
- 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
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
(EMPTY LINE)

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

\[ \text{Client sends info about itself with cookies.} \]
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...
  - Most applications need some 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

"Ask server for resource - server can respond"
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

> 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">
...