CMSC417 Spring 2016 Leture #20 4/20/2016

Agenda

⇒ p4 due Friday

⇒ finish up checksums / MAC
- □ jamming signals — no magic signal, just any signal
- □ why catch errors at L2?
- □ using only one resister — jnz, not, jnz in assembly
- □ polynomial division rules

⇒ HTTP / REST

⇒ Software-Defined Networking (SDN)

⇒ caches, proxies, middleboxes

⇒ rich headers: user-agent, last-modified, etc.

CMS417 Spring 2016 Lecture #20 4/20/2016

HTTP - HyperText Transfer Protocol

⇒ protocol used to access web pages

⇒ generally fetches HTML and the other content needed to display it

- □ images
- □ audio
- □ video
- □ code - javascript

⇒ can transfer nearly anything

Protocol Outline

⇒ request/response, client/server protocol

⇒ spoken over TCP

⇒ text-based and thus easy for humans to read

⇒ message format:

```
<start-line> <crlf>
<headers-one-per-line> <crlf>
<crlf>
<optional-body>
```

- □ start-line: either <action> <URL> <protocol> (request)
  or <protocol> <code> <status> (response)
- □ headers: colon-separated key-value pairs, e.g.,
  "Content-Length: 4096"
- □ body: empty for MOST requests, data encoded using MIME for messages w/ bodies

CMSC417 Spring 2016 Lecture #20 4/20/2016

HTTP actions|

⇒ GET - give me the object at the URL
⇒ HEAD - just give me the metadata about the object
⇒ POST - put the body under the URL
⇒ PUT - put the body at the URL
⇒ DELETE - delete the object at the URL
⇒ TRACE - send me the body back (sees how/if modified)
⇒ CONNECT - used to set up connections through others
⇒ OPTIONS - figure out available options

□ GET, HEAD, TRACE, CONNECT and OPTIONS are "safe" b/c they don't change server state

status codes|

⇒ 1xx - informational
⇒ 2xx - success
⇒ 3xx - redirection
⇒ 4xx - client error
⇒ 5xx - server error

example|

http://www.cs.umd.edu/index.htm

→ protocol → DNS to IP → resounce to talk about

1) establish TCP connection to the IP address
2) send "GET /index.htm HTTP/1.1"
2) recv "HTTP/1.1 200 OK

⋮

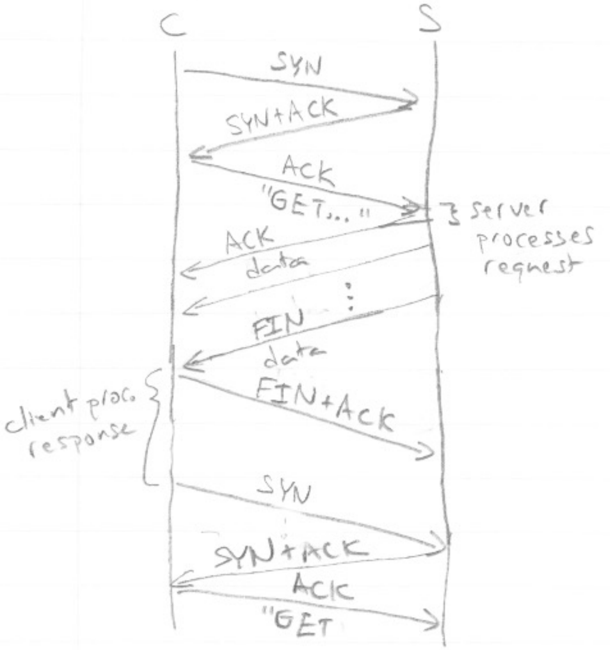
<HTML>

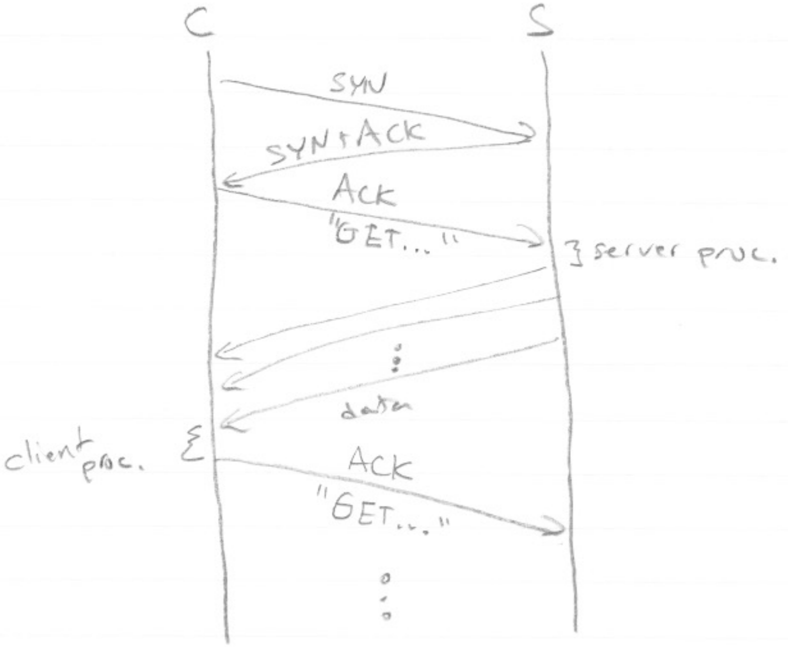
⋮

</HTML>"

CMSC417 Spring 2016 Lecture #20 4/20/2016

HTTP 1.0 | HTTP 1.1 | "persistent connections"





⇒ one TCP connection per item needed

⇒ one TCP connection for many items

persistent connections |

⇒ reduce load on server ~ fewer connection establishments

⇒ congestion window works b/c connections last long enough for it to matter

⇒ save an RTT from every request after the first

⇒ Bad: server doesn't as easily know when it can close a connection b/c they don't know when a client is "done"

□ expire them after a timeout and deal

CMSC 417 Spring 2016 Lecture #20 4/20/2016

REST - REpresentative State Transfer

⇒ Allow for Remote Procedure Calls (RPCs) over HTTP using existing actions

- ⇒ GET ≈ getters in object oriented prog.
- ⇒ PUT ≈ setters in " " "
- ⇒ POST ≈ function call in " " "
- ⇒ DELETE has the obvious meaning
- ⇒ URLs become the objects

Good things about REST

⇒ HTTP is everywhere and has tons of good tools and libraries to work with it

⇒ Human readability makes troubleshooting, debugging, playing easy

⇒ pretty much <u>everything</u> has a REST API

- □ github
- □ facebook
- □ twitter
- □ ...

Bad things about REST

⇒ relatively high-overhead and expensive to parse, encode, and decode operations

⇒ deep application semantics can be hard to easlly provide, and describe using only GET, PUT, POST and DELETE

CMSC417 Spring 2016 Lecture #20 4/20/2016

Software-Defined Networking (SDN)

⇒ traditional routers/switches:

control plane
forwarding plane

⇒ runs on commodity CPUs
⇒ speaks STP, OSPF, RIP, etc.

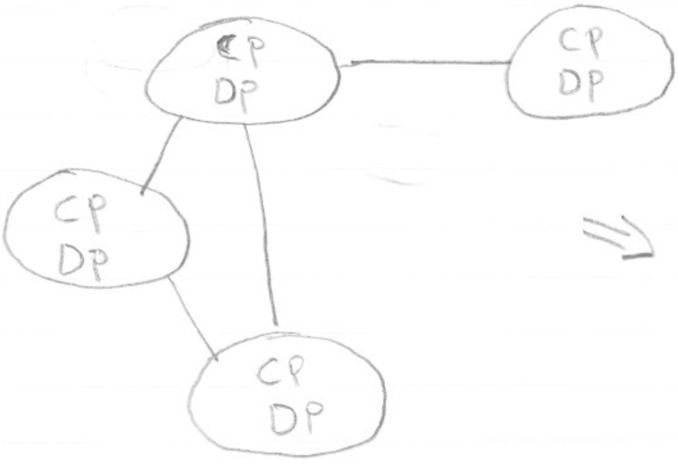
⇒ runs on custom ASICs
⇒ uses table-lookups on tables populated by the control plane

⇒ key SDN realization:

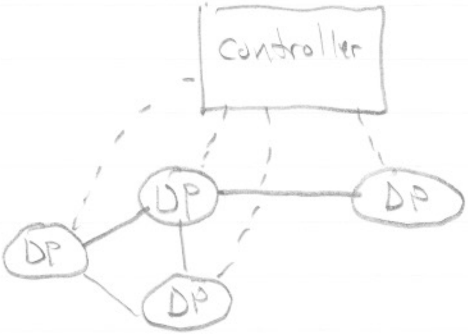
□ most of the new ideas in networking vere modifying the control plane

□ this is hard and slow b/c you need to do it to all switches/routers in a network and likely develop standards so it will work between diff. companies' switches/routers

□ if we instead standardized the interface between the control and data planes, we could develop new stuff faster



⇒ control plane & data plane on every device



⇒ single, centralized control plane == controller

⇒ network devices only have data planes

CMSC 417 Spring 2016 Lecture # 20 4/20/2016

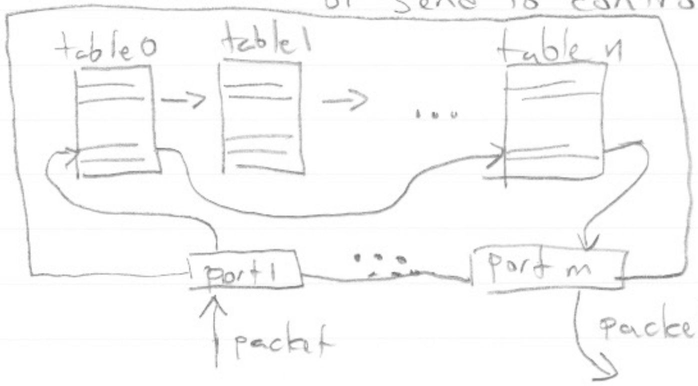
OpenFlow:

⇒ one standard protocol to speak between the control plane (controller) and data plane

⇒ model of a device is a sequence of tables with match-action rules as table entries

□ match: e.g., destination IP addr = 192.168.0.1, or IP-proto = TCP or TCP port = 80, or any combination

□ action: e.g., output port 4, or set src MAC to Y, or send to controller



⇒ can also capture packets via "send to controller" action and then a packet-in message from the switch/router to controller

⇒ cand send packets from controller to network using a packet-out messege to the switch/router with the port(s) to send it out

⇒ controller can use capture/sending to replicate features of control planes

□ capture ARP to learn host IP, MAC, location

□ capture/send routing protocols to interopefate with non-SDN devices

□ ...