Networking Attacks

THE UNIVERSITY OF CHICAGO

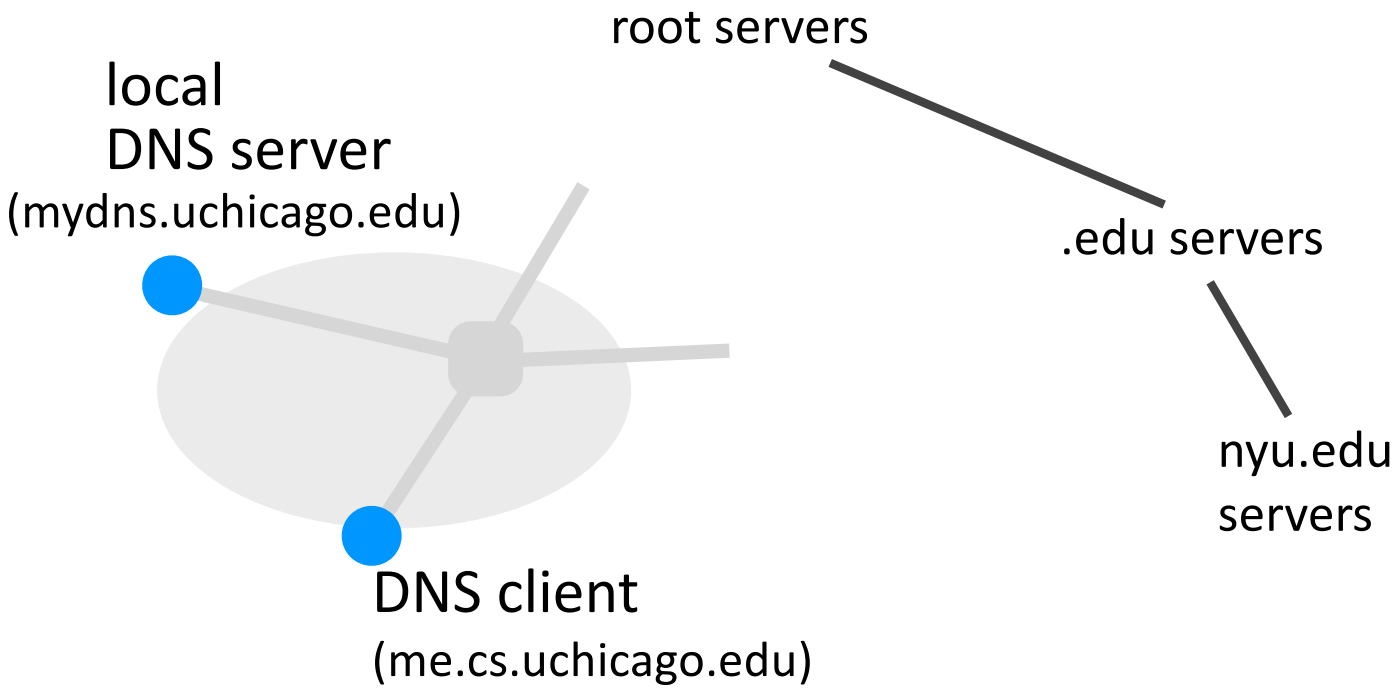
Ben Zhao

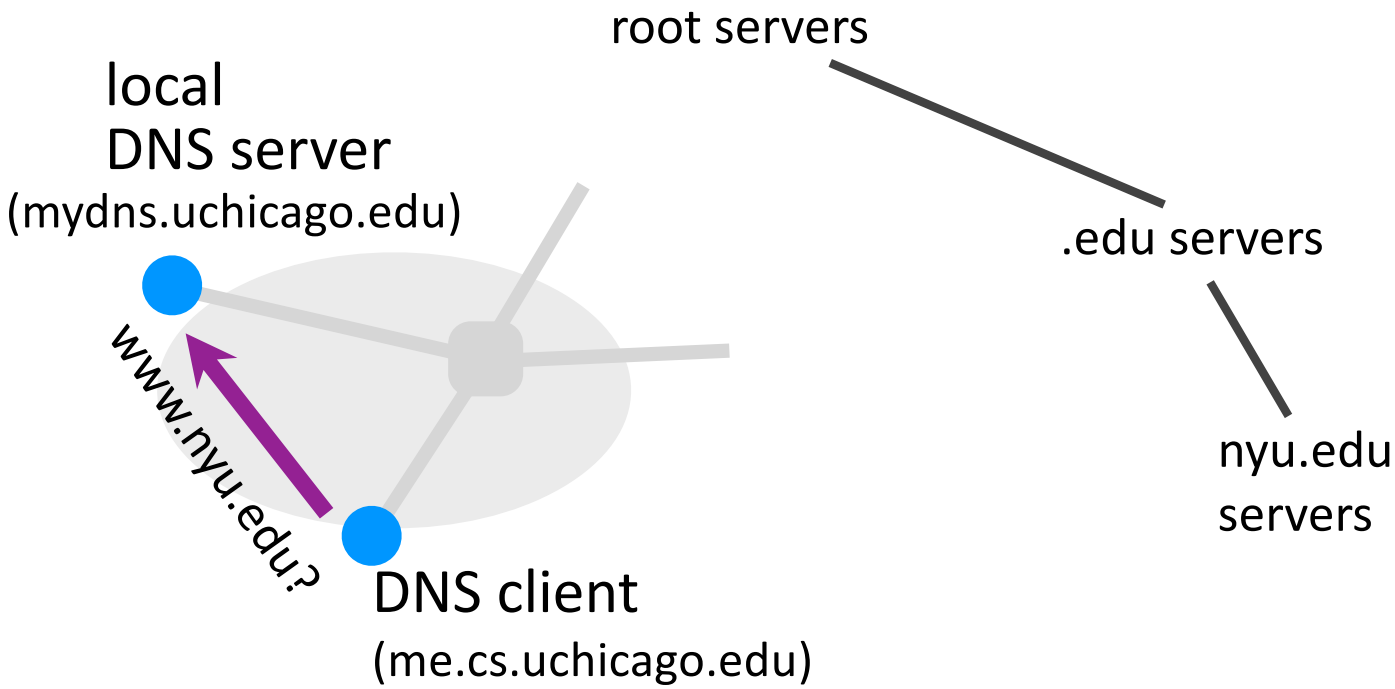
Oct 24, 2018

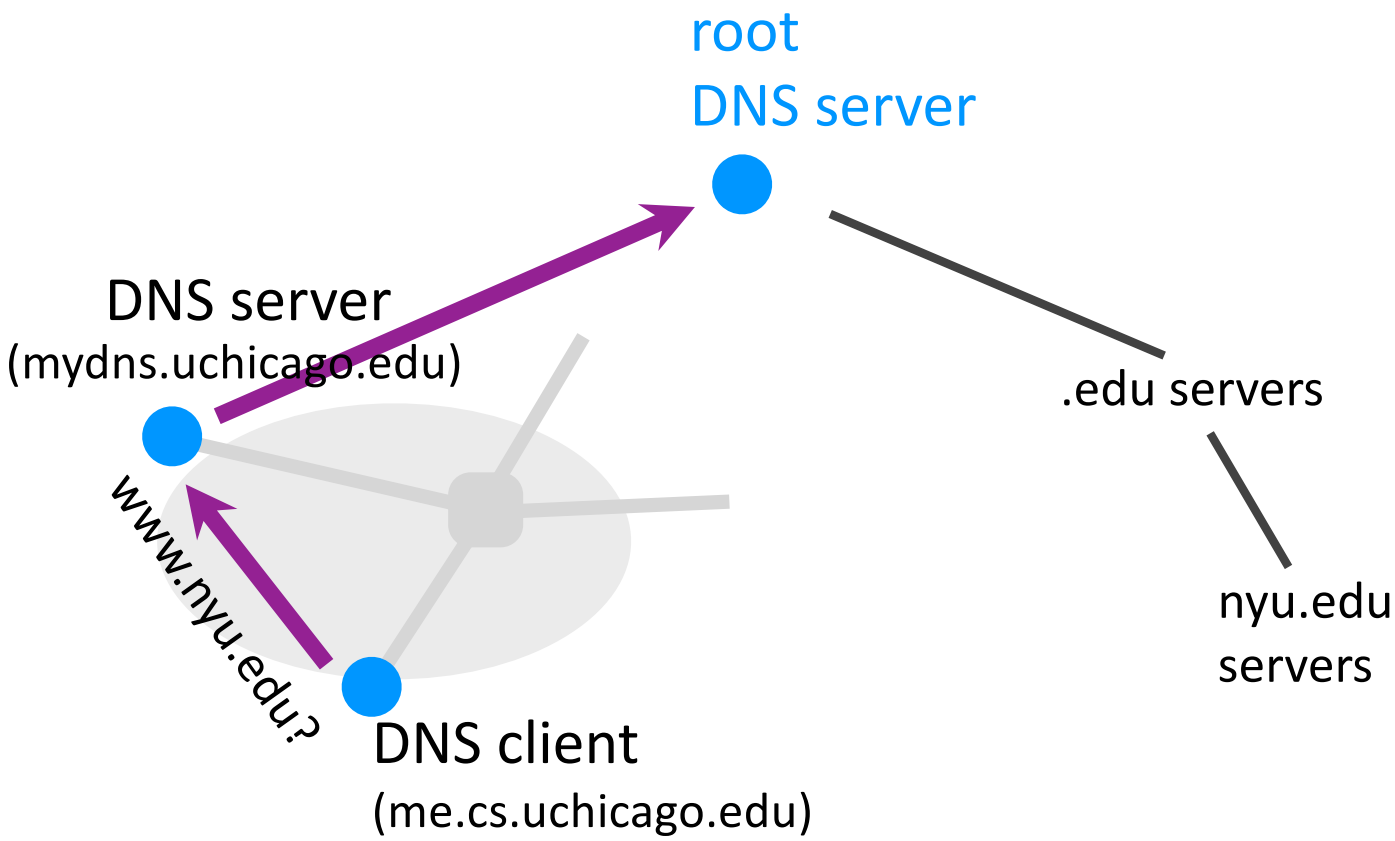
CS 232/332

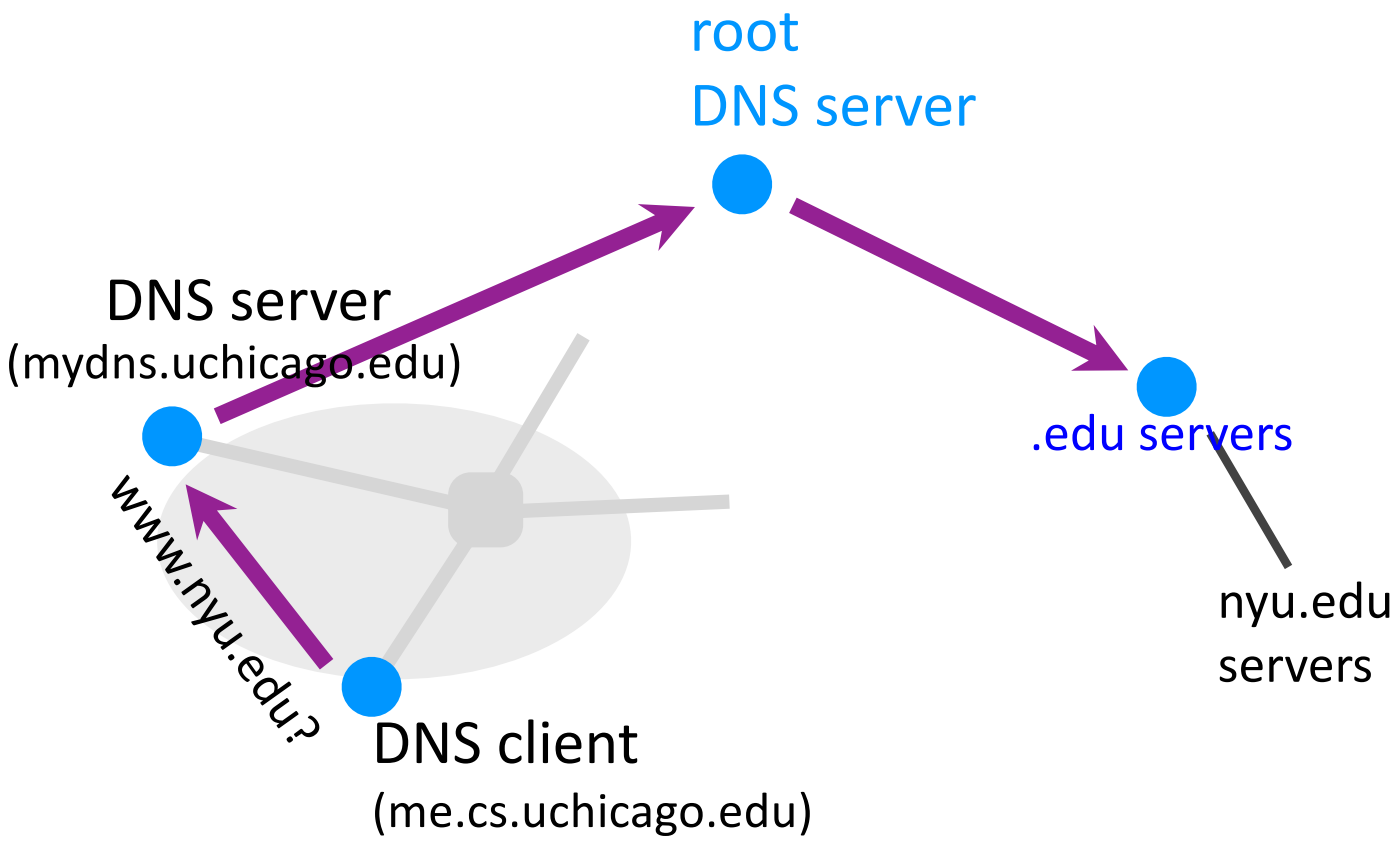
Inserting Resource Records into DNS

- Example: you just created company “FooBar”
- You get a block of IP addresses from your ISP
  - say 212.44.9.128/25
- Register foobar.com at registrar (e.g., Go Daddy)
  - Provide registrar with names and IP addresses of your authoritative name server(s)
  - Registrar inserts RR pairs into the .com TLD server:
    - (foobar.com, dns1.foobar.com, NS)
    - (dns1.foobar.com, 212.44.9.129, A)
- Store resource records in your server dns1.foobar.com
  - e.g., type A record for www.foobar.com
  - e.g., type MX record for foobar.com

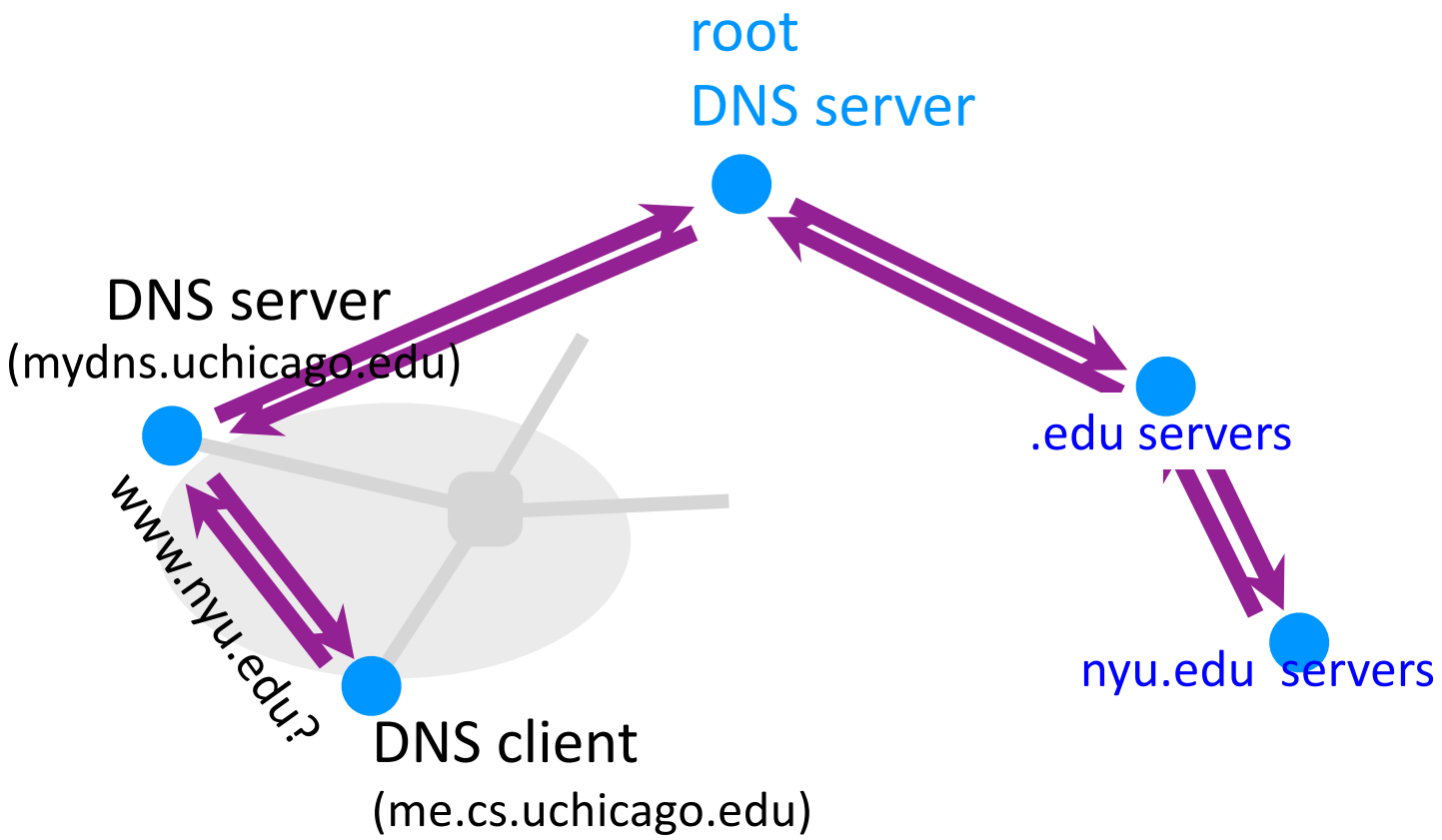

root servers
local
DNS server
(mydns.uchicago.edu)
.edu servers
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)

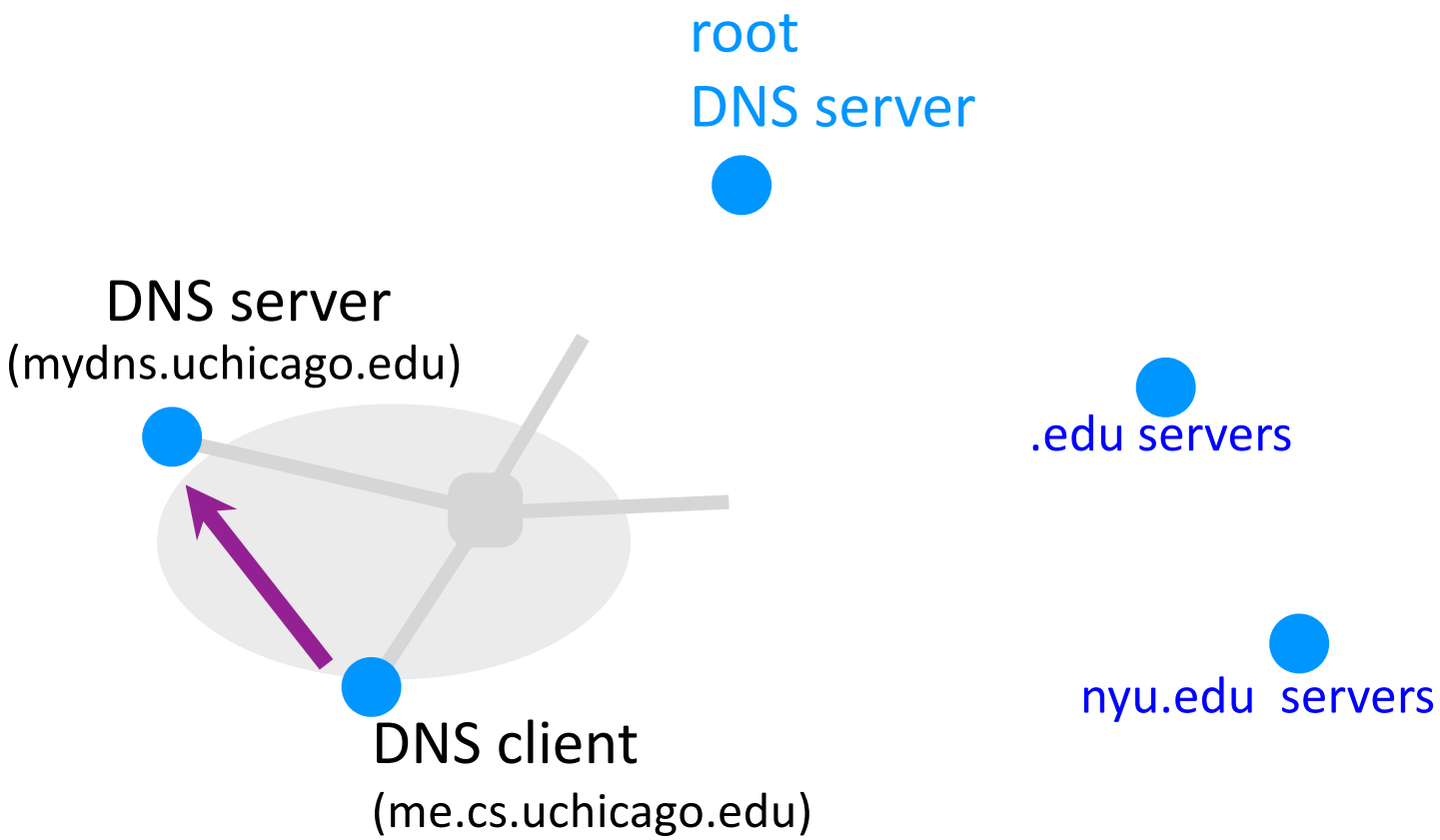

root servers
local
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)

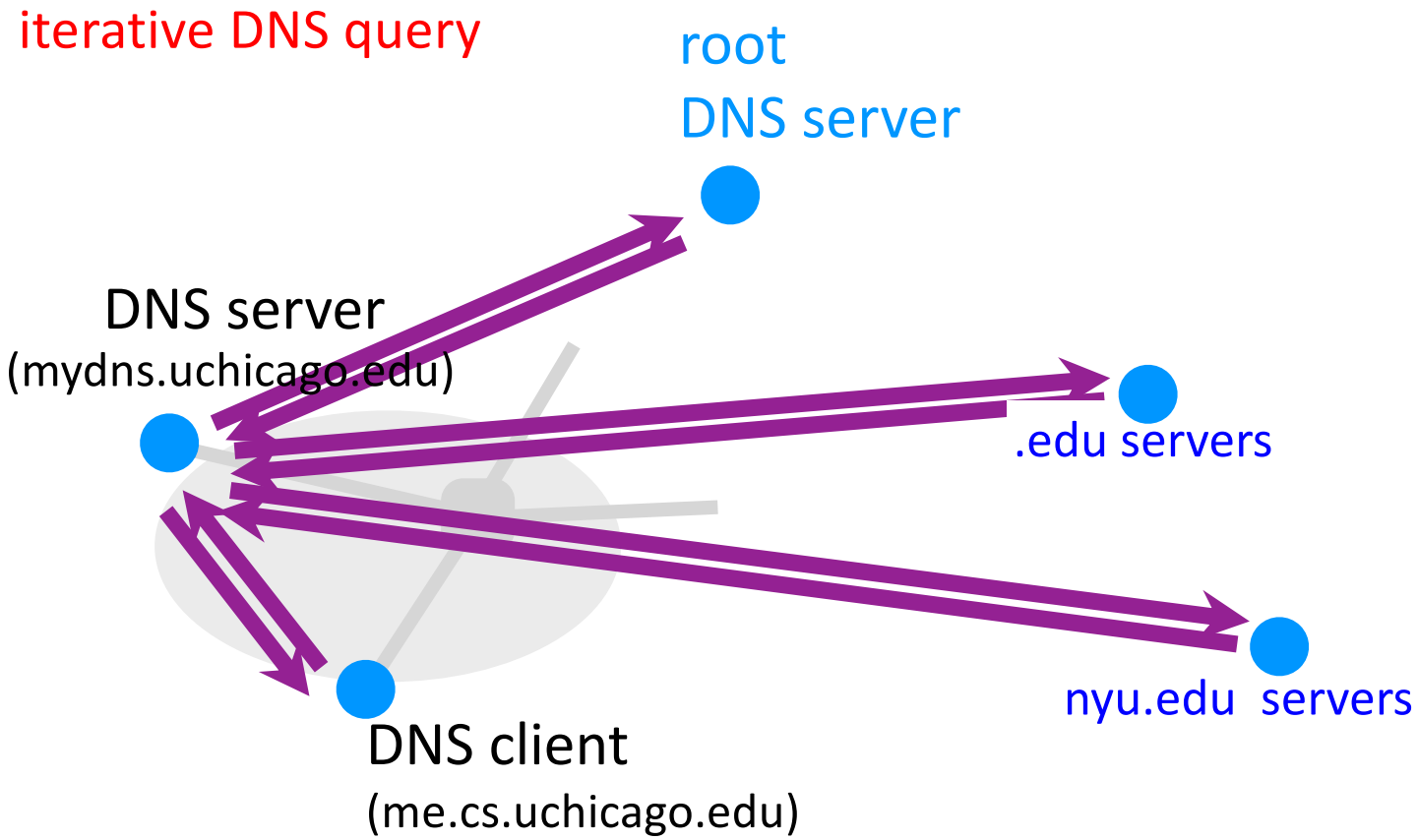

root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)


root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)

recursive DNS query




root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
nyu.edu servers
DNS client
(me.cs.uchicago.edu)


iterative DNS query
root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
nyu.edu servers
DNS client
(me.cs.uchicago.edu)

The Tour Continues…

- IP Addressing and Allocation
- DNS
- IP Routing
- Transport layer (TCP, congestion control)

Addressing, Forwarding, Routing

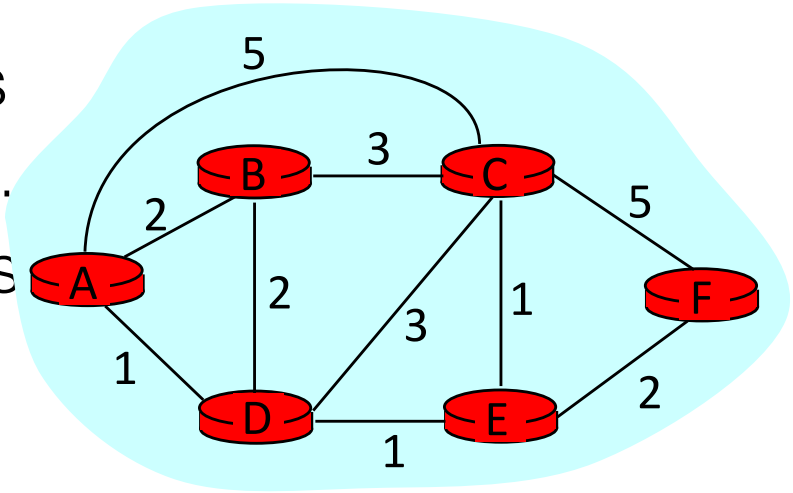
- Addressing: we covered already
- Forwarding: Local router process determines output link (a.k.a “next hop”) for each packet
  - *read address from packet’s network layer header*
  - *search forwarding table*
- Routing: Network-wide process that determines the content of forwarding tables

  → *determines the end-to-end path for each destination*

Routing

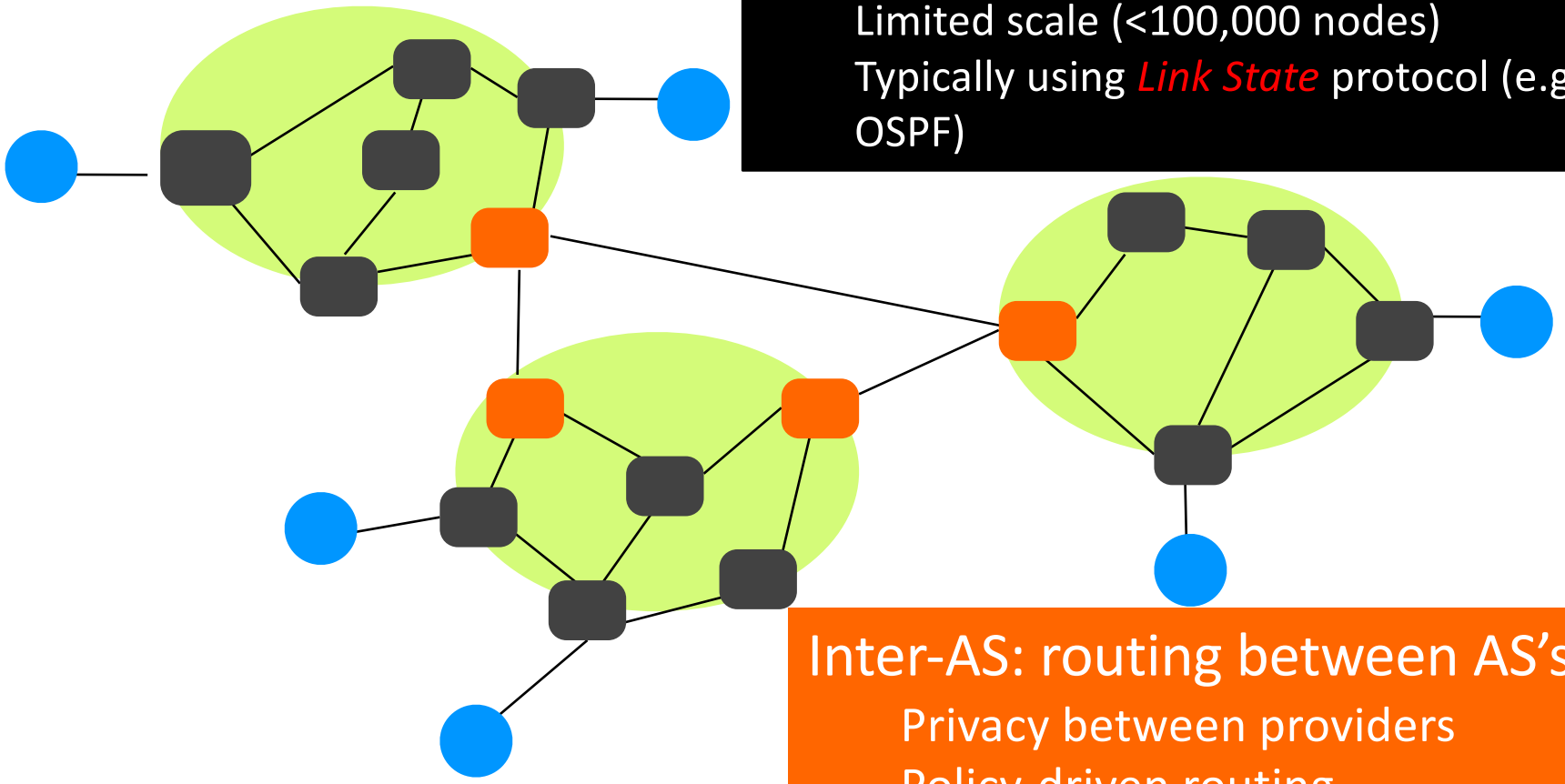
- Goal: determine "good" path through network from source to destination

- Network modeled as a graph
  - Routers → nodes, Link →edges
    - Edge cost: delay, congestion level,.
  - A node knows only its neighbors and the cost to reach them



- How does each node learns how to reach every other node along the shortest path?

Intra-AS & Inter-AS Routing



Intra-AS: routing within a single AS

- Trusted domain (within one company)
- Limited scale (<100,000 nodes)
- Typically using *Link State* protocol (e.g. OSPF)

Inter-AS: routing between AS's

- Privacy between providers
- Policy-driven routing
- BGP, a *Path Vector* protocol
  - Variant of *Distance Vector* routing

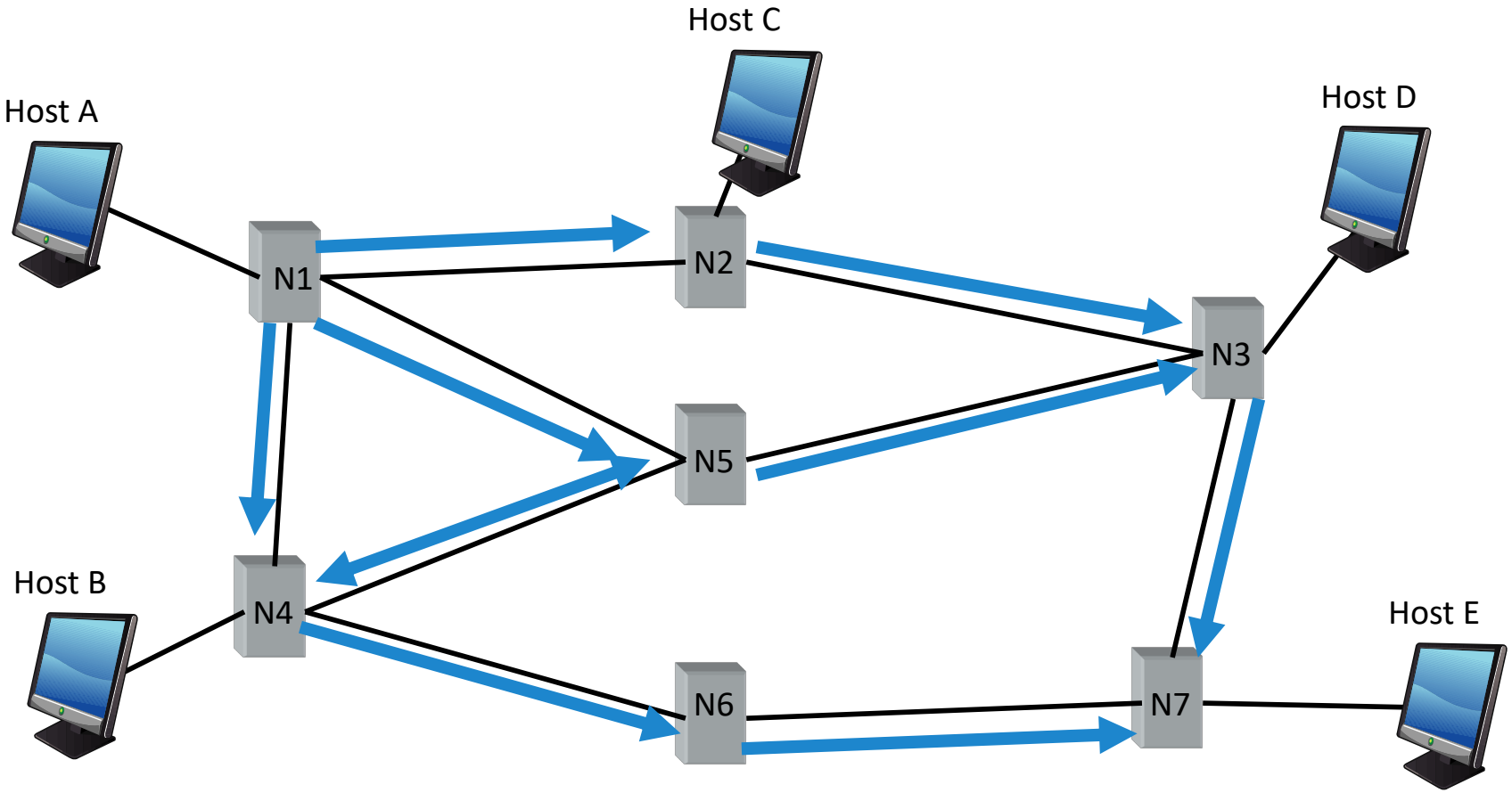
Intra-AS & Inter-AS Routing

- Intra-AS: routing within a single AS
  - Trusted domain (within one company)
  - Limited scale (<100,000 nodes)
  - Typically using *Link State* protocol (e.g. OSPF)

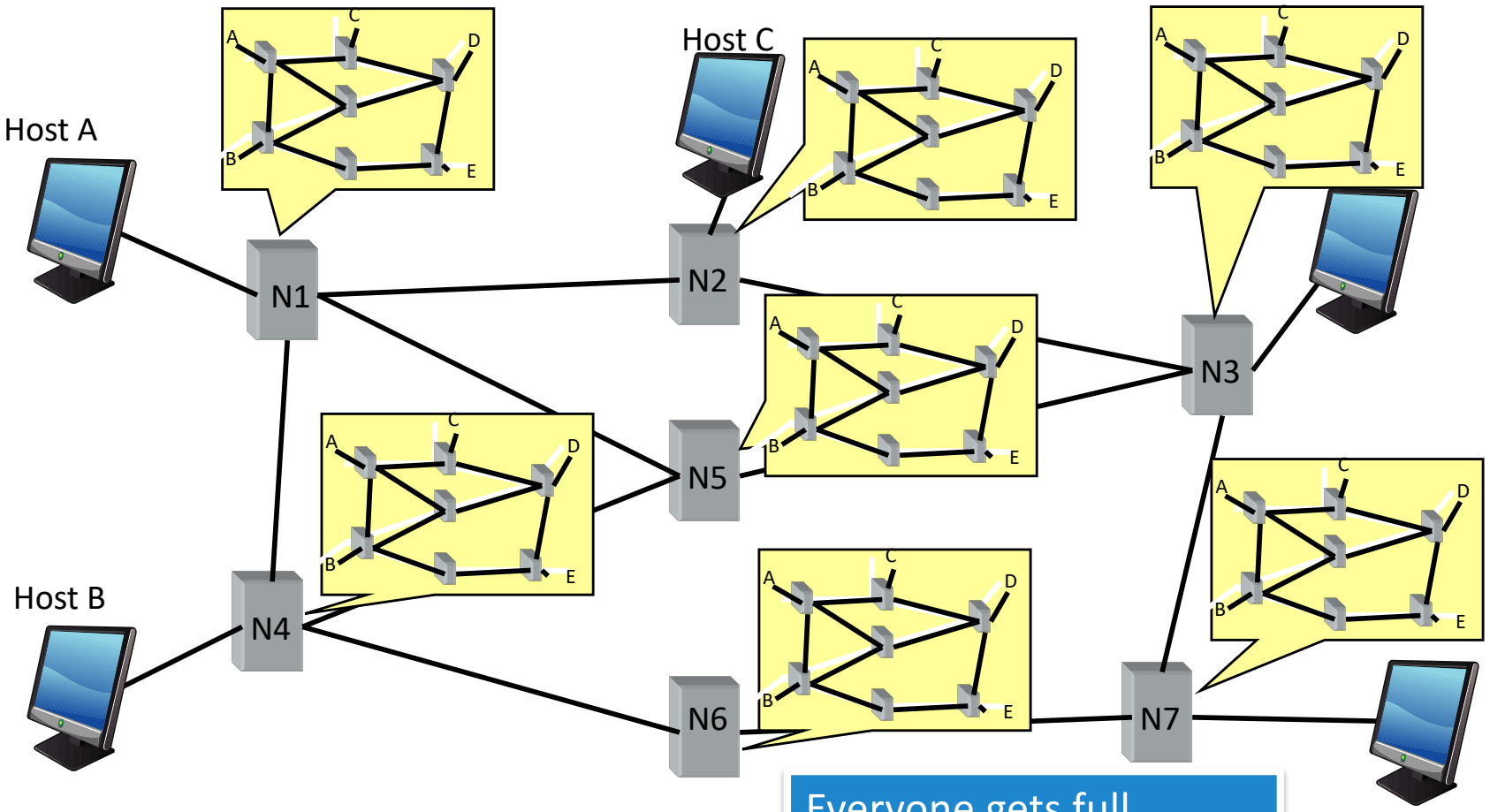
- Inter-AS: routing between AS's
  - Privacy between providers
  - Policy-driven routing
  - BGP, a *Path Vector* protocol
    - Variant of *Distance Vector* routing

Link State: Control Traffic

- Each node floods its local information to every other node in network
- Each node ends up knowing entire network topology
  → use Dijkstra to compute shortest path to every other node

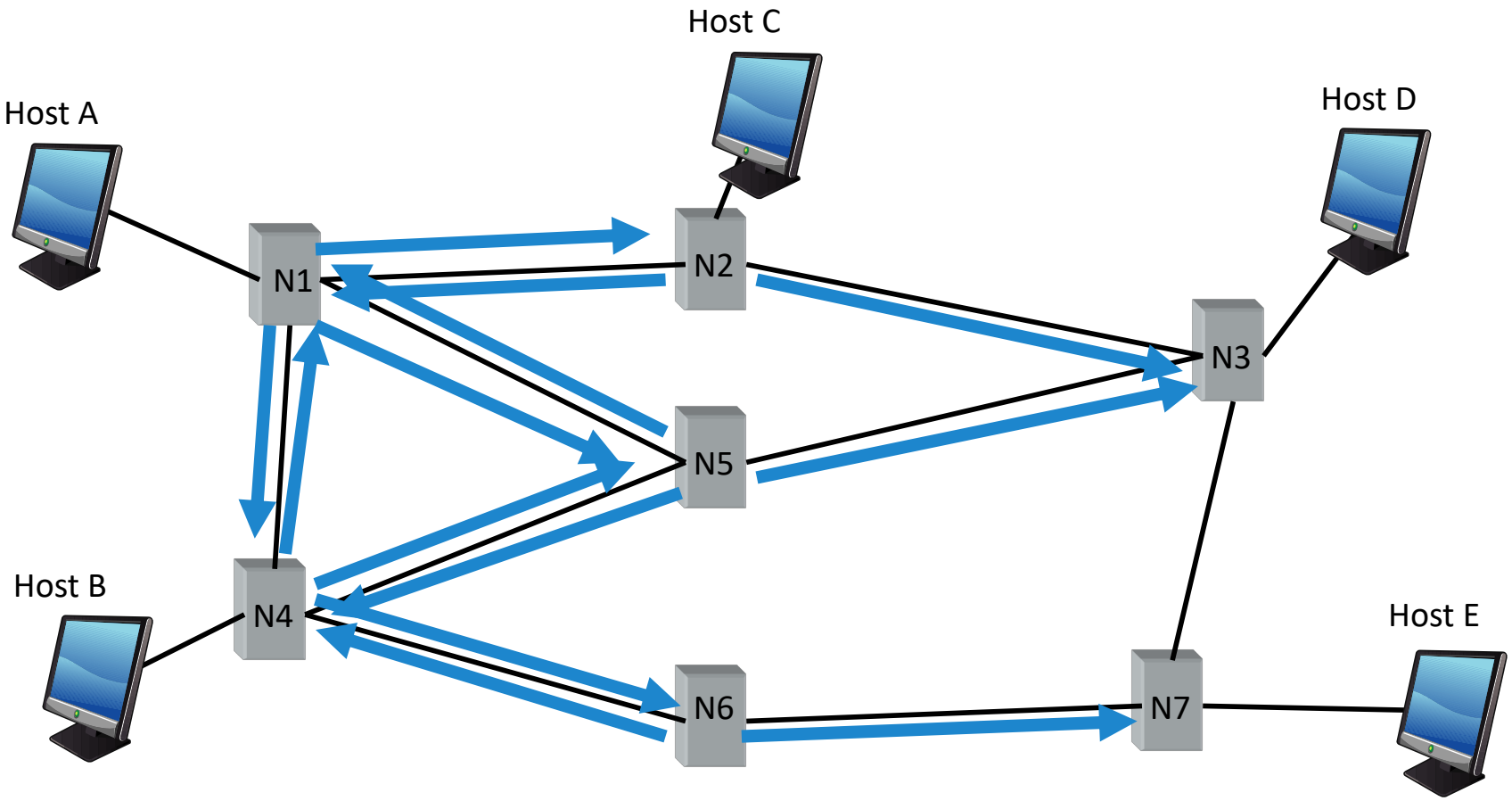


Link State: Node State

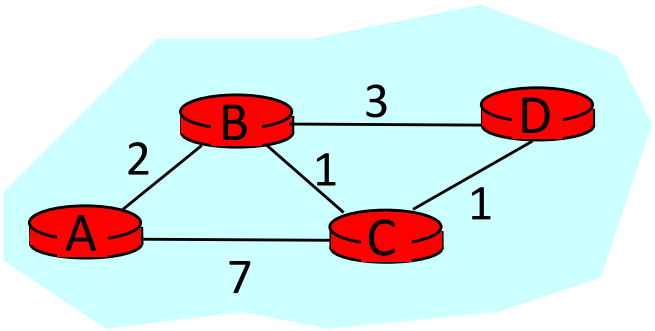


Distance Vector: Control Traffic

- When the routing table of a node changes, it sends table to neighbors
  - A node updates its table with information received from neighbors



Example: Distance Vector Algorithm



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

1 ***Initialization:***
2 **for all** neighbors *V* **do**
3 **if** *V* adjacent to *A*
4 D(*A, V*) = c(*A,V*);
5 **else**
6 D(*A, V*) = ∞;
…

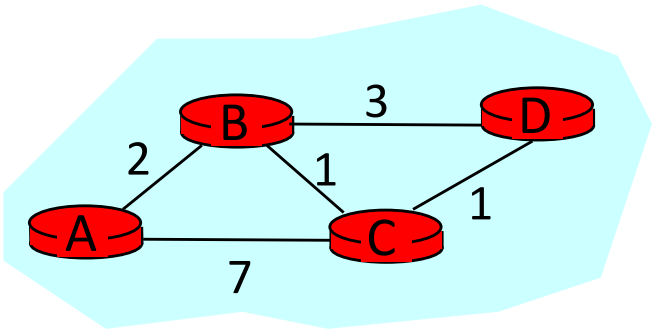
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

(D(C,A), D(C,B), D(C,D))

Node C

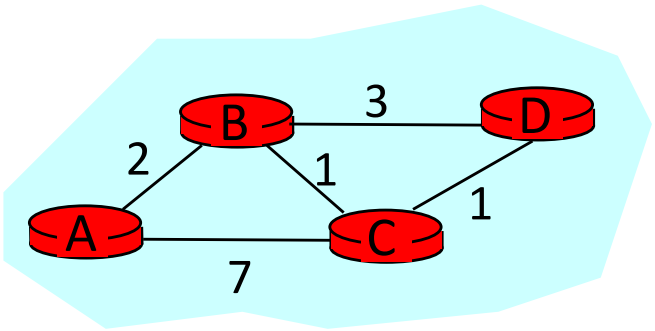
Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

```
...
7 loop:
...
12  else if (update D(V, Y) received from V)
13    for all destinations Y do
14      if (destination Y through V)
15        D(A,Y) = D(A,V) + D(V, Y);
16      else
17        D(A, Y) = min(D(A, Y),
                        D(A, V) + D(V, Y));
18  if (there is a new minimum for dest. Y)
19    send D(A, Y) to all neighbors
20  forever
```

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	**8**	**C**

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A,D)= min(D(A, D),D(A,C)+D(C,D)
= min(∞ , 7 + 1) = 8

(D(C,A), D(C,B), D(C,D))

Node C

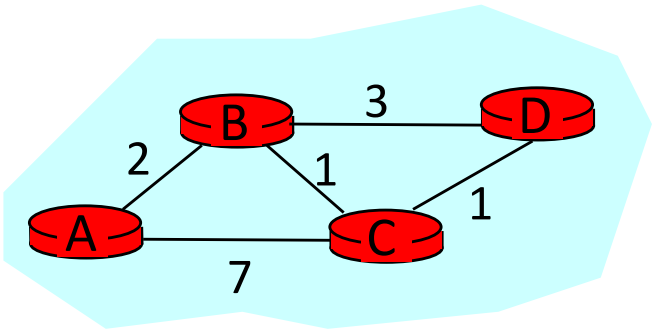
Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

…
7 ***loop:***
…
12 **else if** (update D(*V, Y*) received from *V*)
13 **for all** destinations Y **do**
14 **if** (destination *Y* through *V*)
15 D(*A,Y*) = D(*A,V*) + D(*V, Y*);
16 **else**
17 D(A, Y) = min(D(*A, Y*),
D(*A, V*) + D(*V, Y*));
18 **if** (there is a new minimum for dest. *Y*)
19 **send** D(*A, Y*) to all neighbors
20 **forever**

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	8	C

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

…
7 ***loop:***
…

```
12  else if (update D(V, Y) received from V)
13    for all destinations Y do
14      if (destination Y through V)
15        D(A,Y) = D(A,V) + D(V, Y);
16      else
17        D(A, Y) = min(D(A, Y),
                        D(A, V) + D(V, Y));
18  if (there is a new minimum for dest. Y)
19    send D(A, Y) to all neighbors
20  forever
```

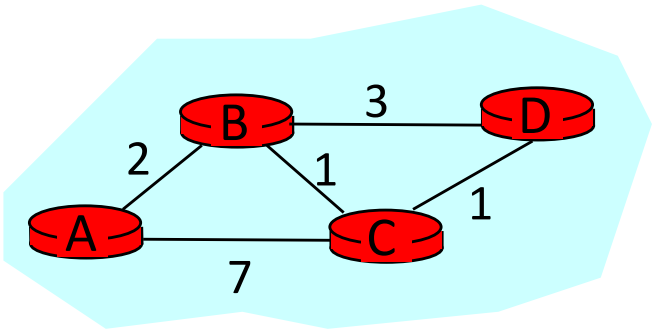
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (B→A, C→A)



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A,D) = min(D(A,D), D(A,B) + D(B,D))
= min(8, 2 + 3) = 5

D(A,C) = min(D(A,C), D(A,B) + D(B,C))
= min(7, 2 + 1) = 3

…
7 ***loop:***
…
12 **else if** (update D(*V, Y*) received from *V*)
13 **for all** destinations Y **do**
14 **if** (destination *Y* through *V*)
15 D(*A,Y*) = D(*A,V*) + D(*V, Y*);
16 **else**
17 D(A, Y) = min(D(*A, Y*),
D(*A, V*) + D(*V, Y*));
18 **if** (there is a new minimum for dest. *Y*)
19 **send** D(*A, Y*) to all neighbors
20 **forever**

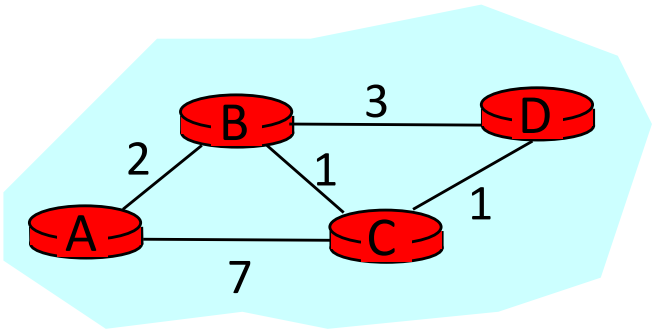
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: End of 1st Iteration



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

Node C

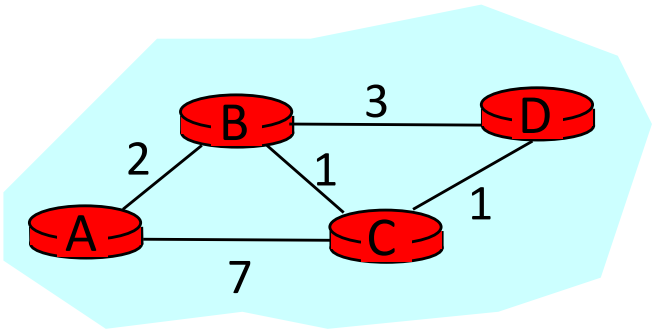
Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	4	B
B	3	B
C	1	C

...
7 ***loop:***
...
12 **else if** (update D(*V, Y*) received from *V*)
13 **for all** destinations Y **do**
14 **if** (destination *Y* through *V*)
15 D(*A,Y*) = D(*A,V*) + D(*V, Y*);
16 **else**
17 D(A, Y) = min(D(*A, Y*),
D(*A, V*) + D(*V, Y*));
18 **if** (there is a new minimum for dest. *Y*)
19 **send** D(*A, Y*) to all neighbors
20 **forever**

Example: End of 3nd Iteration



...
7 ***loop:***
...
12 **else if** (update D(*V, Y*) received from *V*)
13 **for all** destinations Y **do**
14 **if** (destination *Y* through *V*)
15 D(*A,Y*) = D(*A,V*) + D(*V, Y*);
16 **else**
17 D(A, Y) = min(D(*A, Y*),
D(*A, V*) + D(*V, Y*));
18 **if** (there is a new minimum for dest. *Y*)
19 **send** D(*A, Y*) to all neighbors
20 **forever**

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

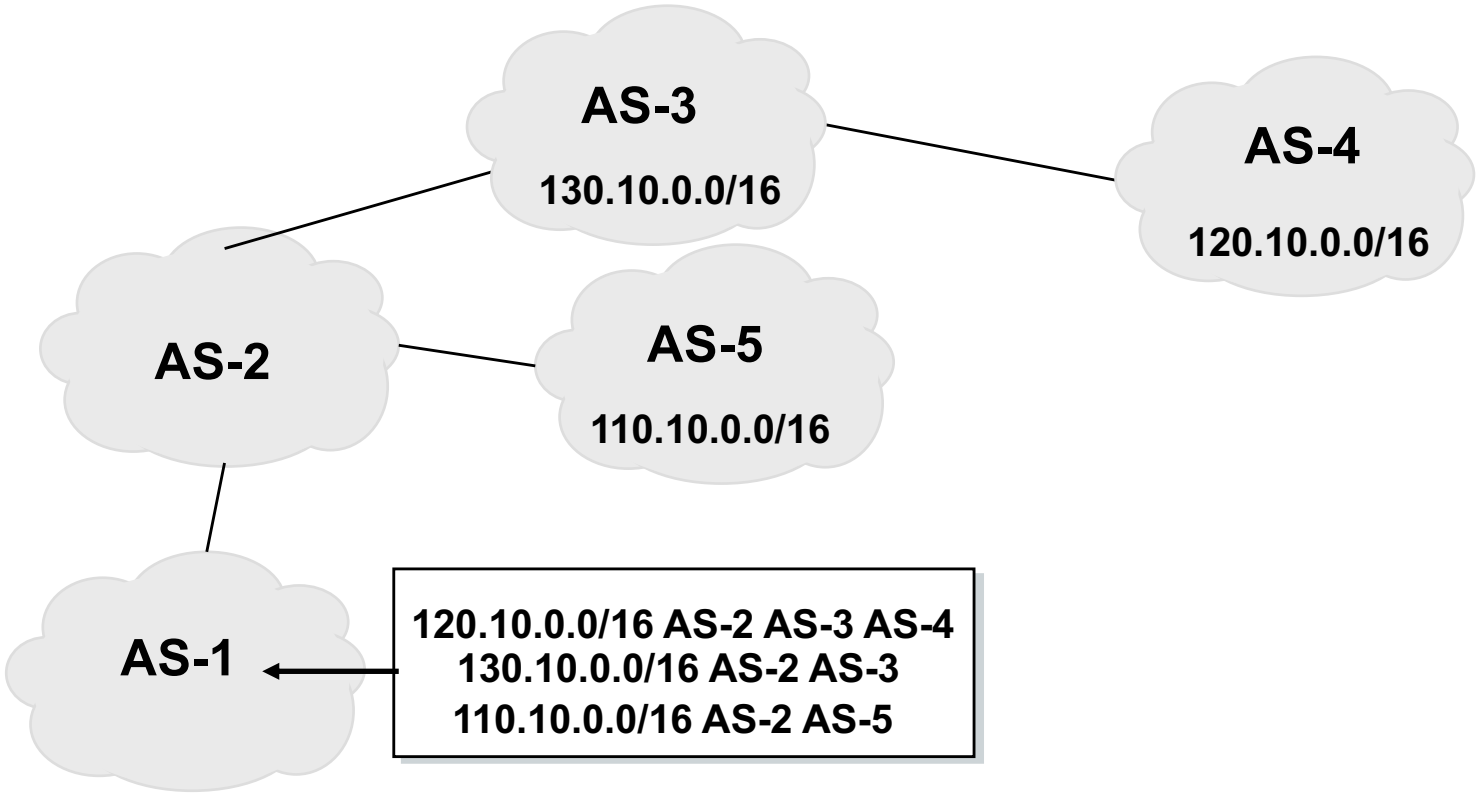
Node D

Dest.	Cost	NextHop
A	4	C
B	2	C
C	1	C

Nothing changes → algorithm terminates

BGP: a Path-Vector Protocol

- An AS-path: sequence of AS's a route traverses
- Used for loop detection and to apply policy
- Default choice: route with fewest # of AS's



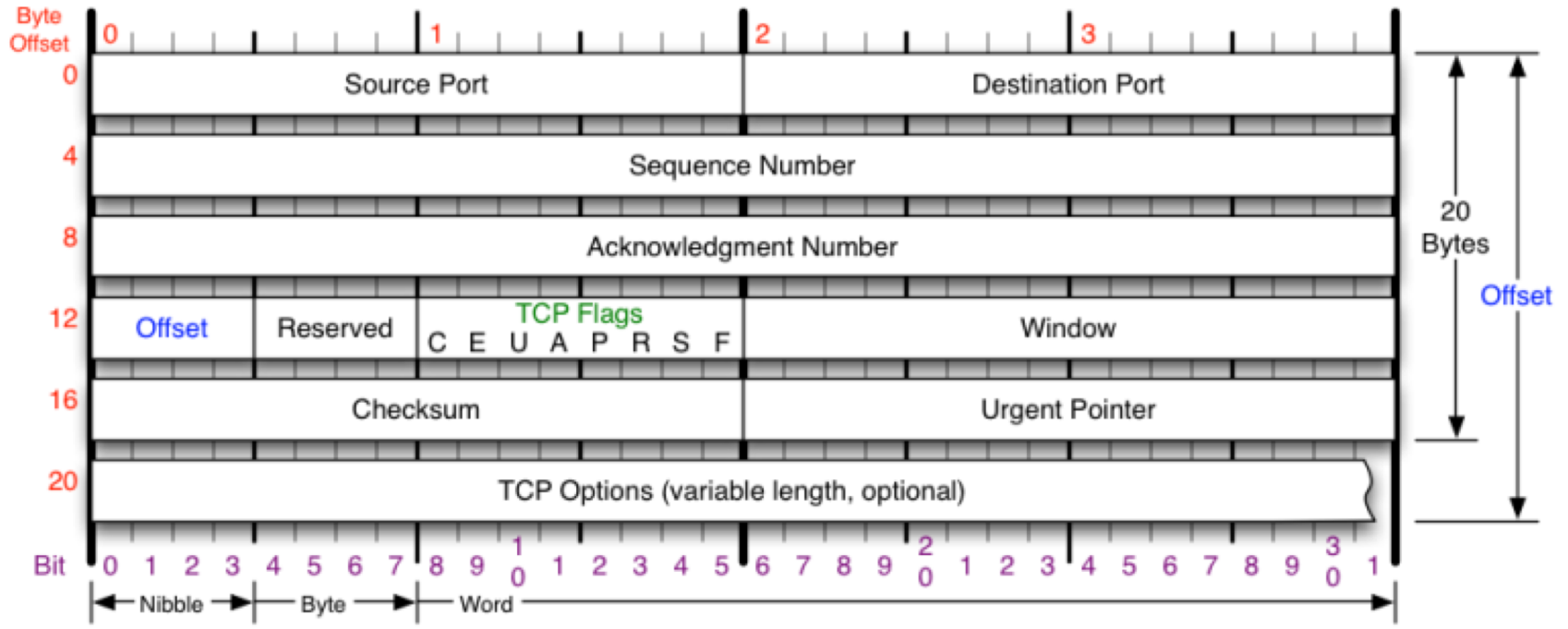
The Tour Continues…

- IP Addressing and Allocation
- DNS
- IP Routing
- Transport layer (TCP, congestion control)

TCP (Transmission Control Protocol)

- Multiplexes between services
- Multi-packet connections
- Handles loss, duplication, & out-of-order delivery
  — all received data ACKnowledged
- Flow control
  — sender doesn’t overwhelm recipient
- Congestion control
  — sender doesn’t overwhelm network

TCP header



TCP Flags

C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Receiver Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

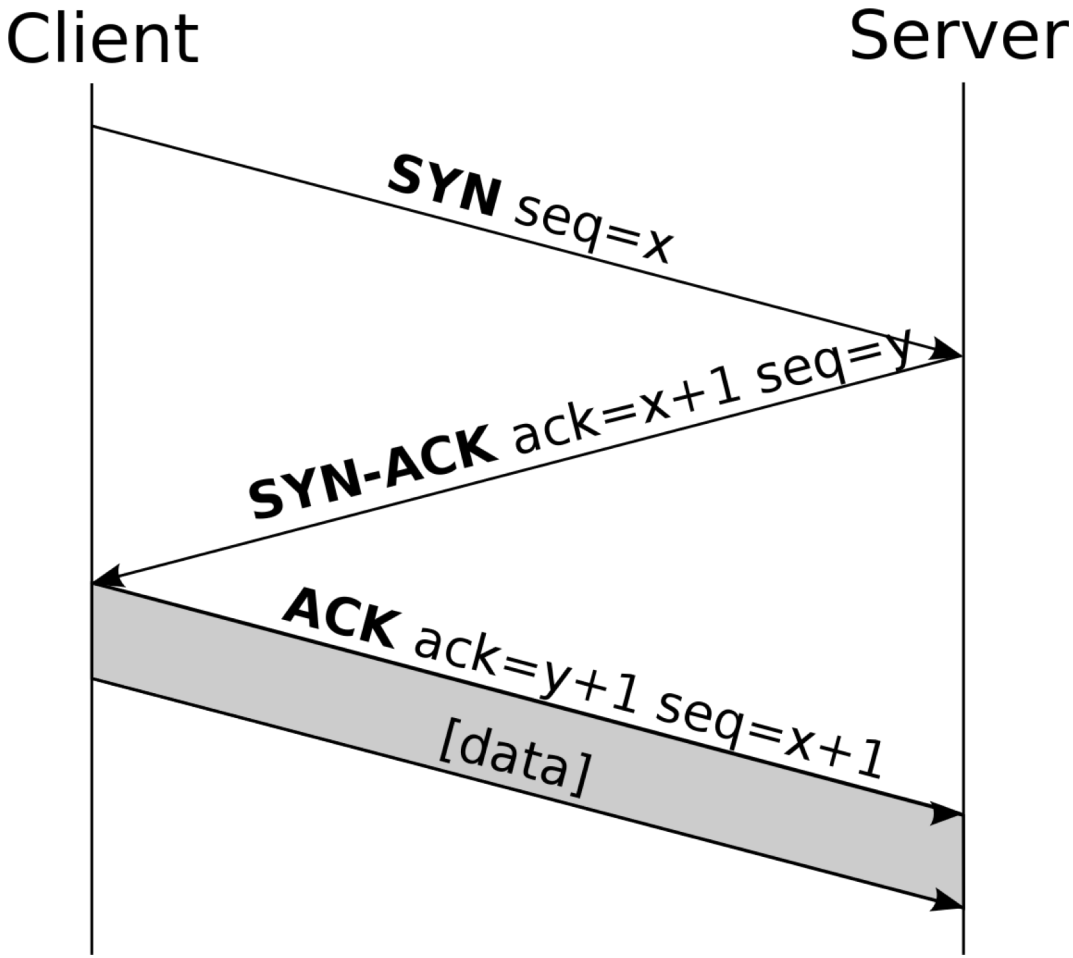
RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

TCP connections

- Explicit connection setup & teardown
- Explicit control flags (e.g., SYN, ACK, FIN, RST)
- Sequence numbers — reliability & ordering

Setup: 3-way handshake



Source: Wikimedia commons

Let’s Talk About Attacks…

Network threat model

- Network scanning
- Attacks on confidentiality
  (e.g., eavesdropping)
- Attacks on integrity
  (e.g., spoofing, packet injection)
- Attacks on availability
  (e.g., denial of service (DoS))
  - Resource exhaustion (e.g., CPU, memory, B/W)
  - Easy to perform, very difficult to defend

Network Scanning: Ping

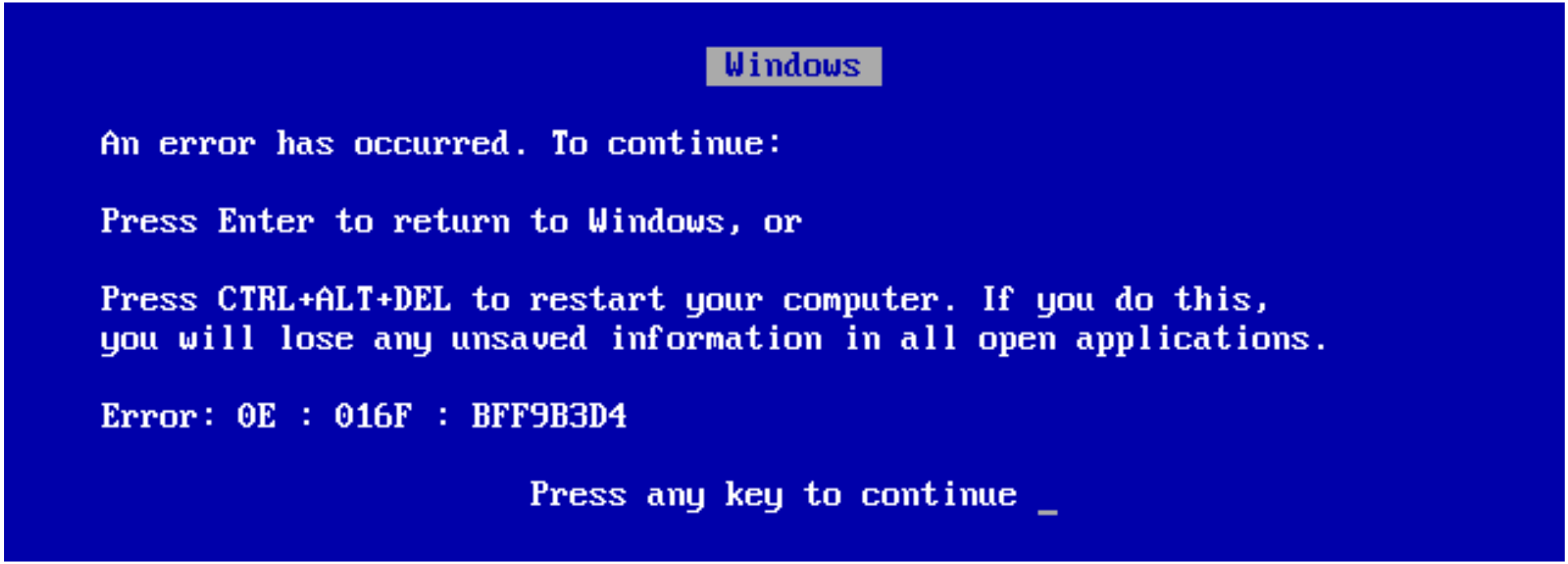
- Essential, low-level network utility
- Sends a “ping” ICMP message to a host on the internet

```
$ ping 66.66.0.255
PING 66.66.0.255 (66.66.0.255) 56(84) bytes of data.
64 bytes from 66.66.0.255: icmp_seq=1 ttl=58 time=41.2 ms
```

- Destination host is supposed to respond with a “pong”
  - Indicating that it can receive packets
- By default, ping messages are 56 bytes long (+ some header bytes)
  - Maximum size 65535 bytes
- What if you send a ping that is >65535 bytes long?

Ping of Death

- $ ping –s 65535 66.66.0.255
  - Attack identified in 1997
  - IPv6 version identified/fixed in 2013



Network Scanning: Traceroute

- traceroute — hops between me and host
  - Sends repeated ICMP reqs w/ increasing TTL

```
thor Wed Oct 24(12:51am)[~]:-> traceroute www.slack.com
traceroute to www.slack.com (52.85.115.213), 64 hops max, 52 byte packets
 1  v11router (128.135.11.1)  1.265 ms  0.788 ms  0.778 ms
 2  a06-021-100-to-d19-07-200.p2p.uchicago.net (10.5.1.186)  1.292 ms  0.749 ms  0.833 ms
 3  d19-07-200-to-h01-391-300.p2p.uchicago.net (10.5.1.46)  2.124 ms  2.435 ms  2.072 ms
 4  192.170.192.34 (192.170.192.34)  0.755 ms
    192.170.192.32 (192.170.192.32)  0.810 ms  0.701 ms
 5  192.170.192.36 (192.170.192.36)  0.887 ms  0.918 ms  0.877 ms
 6  r-equinix-isp-ae2-2213.wiscnet.net (216.56.50.45)  1.625 ms  1.803 ms  1.866 ms
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  178.236.3.103 (178.236.3.103)  4.516 ms  4.326 ms  4.320 ms
12  * * *
13  * * *
14  * * *
15  server-52-85-115-213.ind6.r.cloudfront.net (52.85.115.213)  4.554 ms  4.398 ms  4.757 ms
thor Wed Oct 24(12:52am)[~]:->
```

Port Scanning

- What services are running on a server? Nmap

```
linux3 Wed Oct 24(12:54am)[~]:-> nmap www.cs.uchicago.edu

Starting Nmap 7.01 ( https://nmap.org ) at 2018-10-24 00:55 CDT
Nmap scan report for www.cs.uchicago.edu (34.203.108.171)
Host is up (0.019s latency).
Other addresses for www.cs.uchicago.edu (not scanned): 54.164.17.80 54.85.61.218
rDNS record for 34.203.108.171: ec2-34-203-108-171.compute-1.amazonaws.com
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 4.99 seconds
linux3 Wed Oct 24(12:55am)[~]:->
```

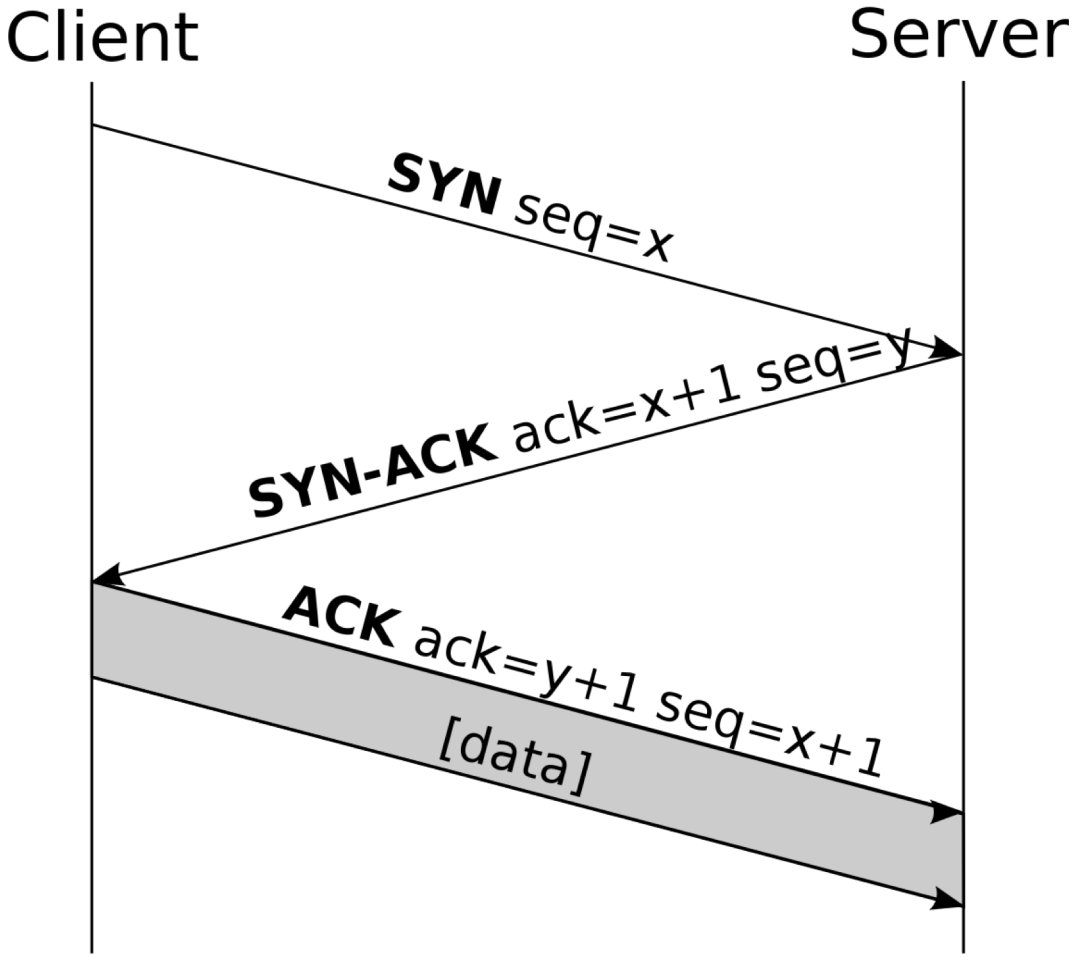
- 5 seconds to scan a single machine!!

Port Scanning on Steroids

- How do you speed up scans for all IPv4?
  - Don't wait for responses; pipeline
  - Parallelize: divide & conquer IPv4 ranges
  - Randomize permutations w/o collisions
- Result: the zmap tool
  - Scan all of IPv4 in 45mins (w/ GigE cxn)
  - IPv4 in 5 mins w/ 10GigE

SYN scan

Only send SYN



Responses:

- SYN-ACK — port open
- RST — port closed
- Nothing — filtered (e.g., firewall)

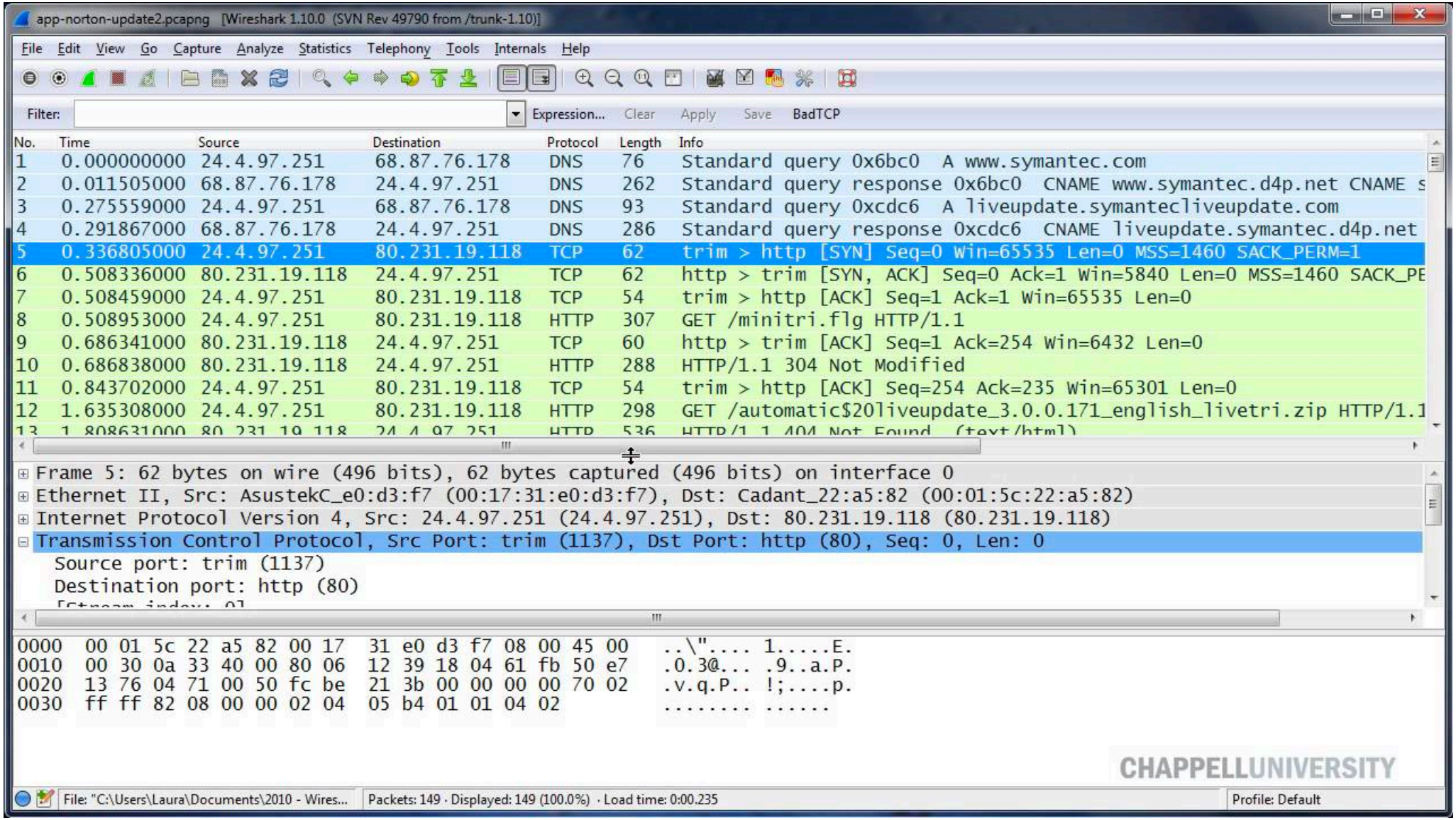
Eavesdropping

Tools: Wireshark, tcpdump, Bro, …

Steps:

1. Parse data link layer frames
2. Identify network flows
3. Reconstruct IP packet fragments
4. Reconstruct TCP connections
5. Parse app protocol messages

Wireshark, Detailed Protocol Analyzer



(TS//SI//NF) FAA702 Operations

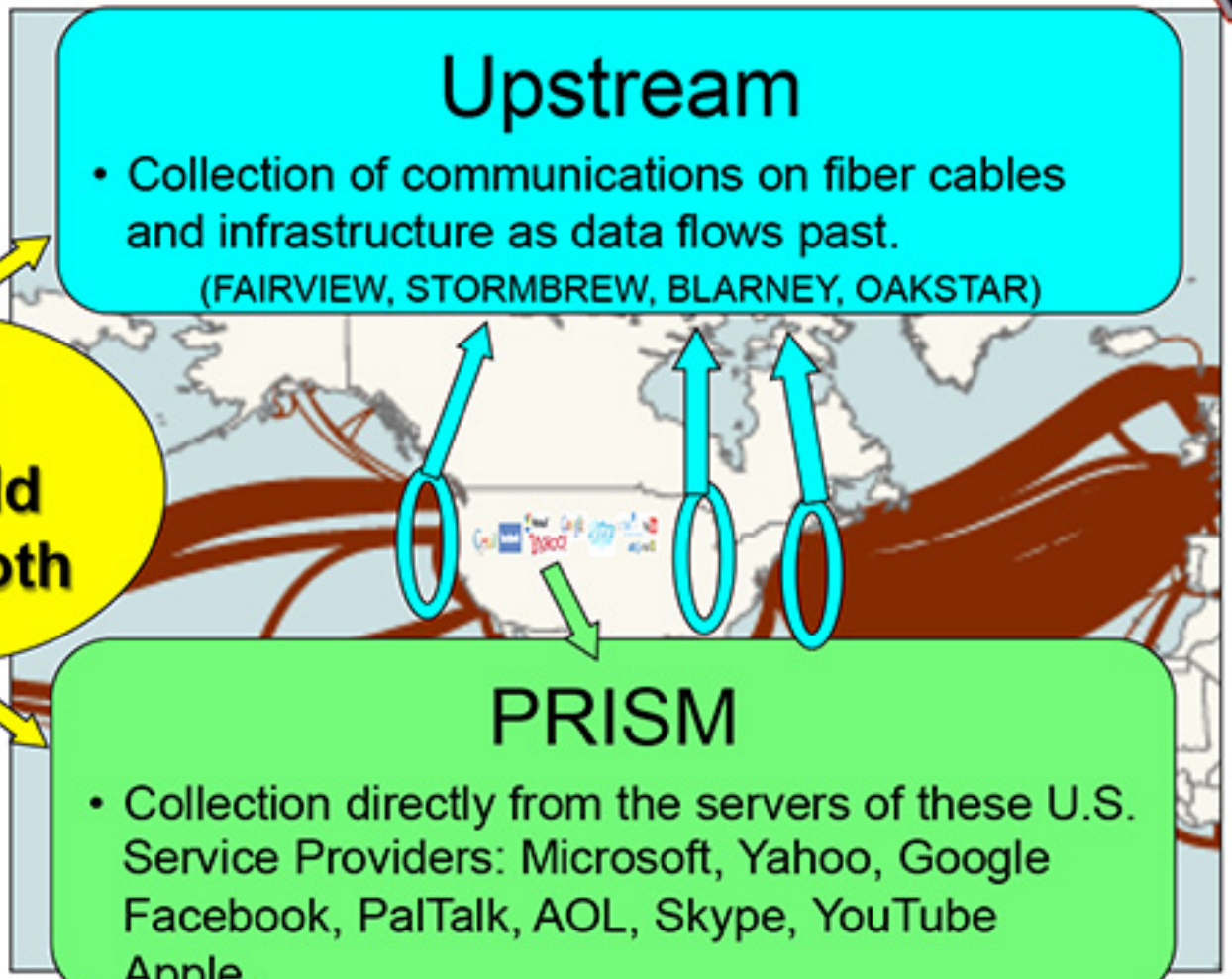
Two Types of Collection

Upstream

- Collection of communications on fiber cables and infrastructure as data flows past.

(FAIRVIEW, STORMBREW, BLARNEY, OAKSTAR)

You Should Use Both



PRISM

- Collection directly from the servers of these U.S. Service Providers: Microsoft, Yahoo, Google Facebook, PalTalk, AOL, Skype, YouTube Apple.

From Snowden archives, dated April 2013

(TS//SI//NF) PRISM Collection Details

Current Providers

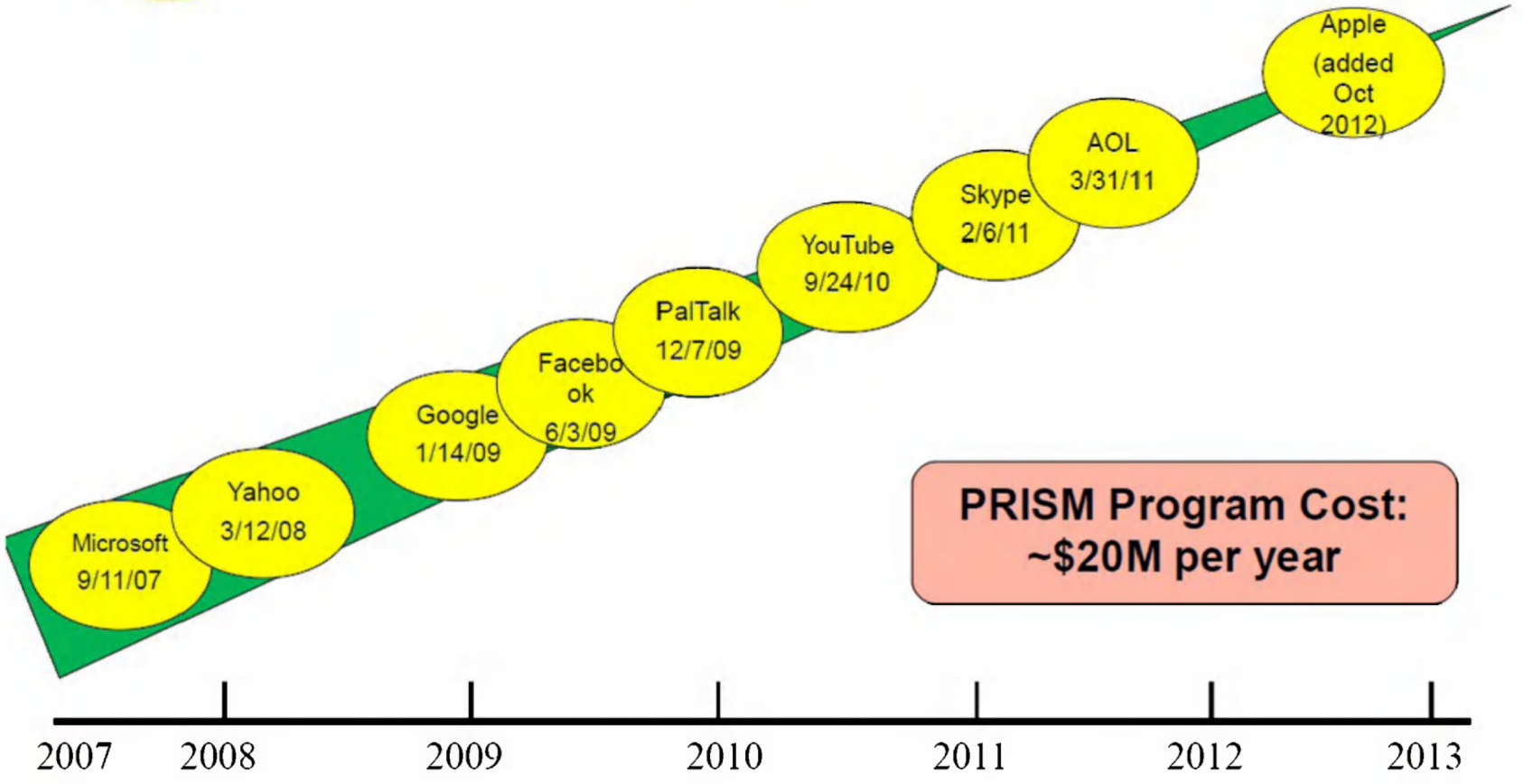
- Microsoft (Hotmail, etc.)
- Google
- Yahoo!
- Facebook
- PalTalk
- YouTube
- Skype
- AOL
- Apple

What Will You Receive in Collection (Surveillance and Stored Comms)? It varies by provider. In general:

- E-mail
- Chat – video, voice
- Videos
- Photos
- Stored data
- VoIP
- File transfers
- Video Conferencing
- Notifications of target activity – logins, etc.
- Online Social Networking details
- **Special Requests**

Complete list and details on PRISM web page:
Go PRISMFAA

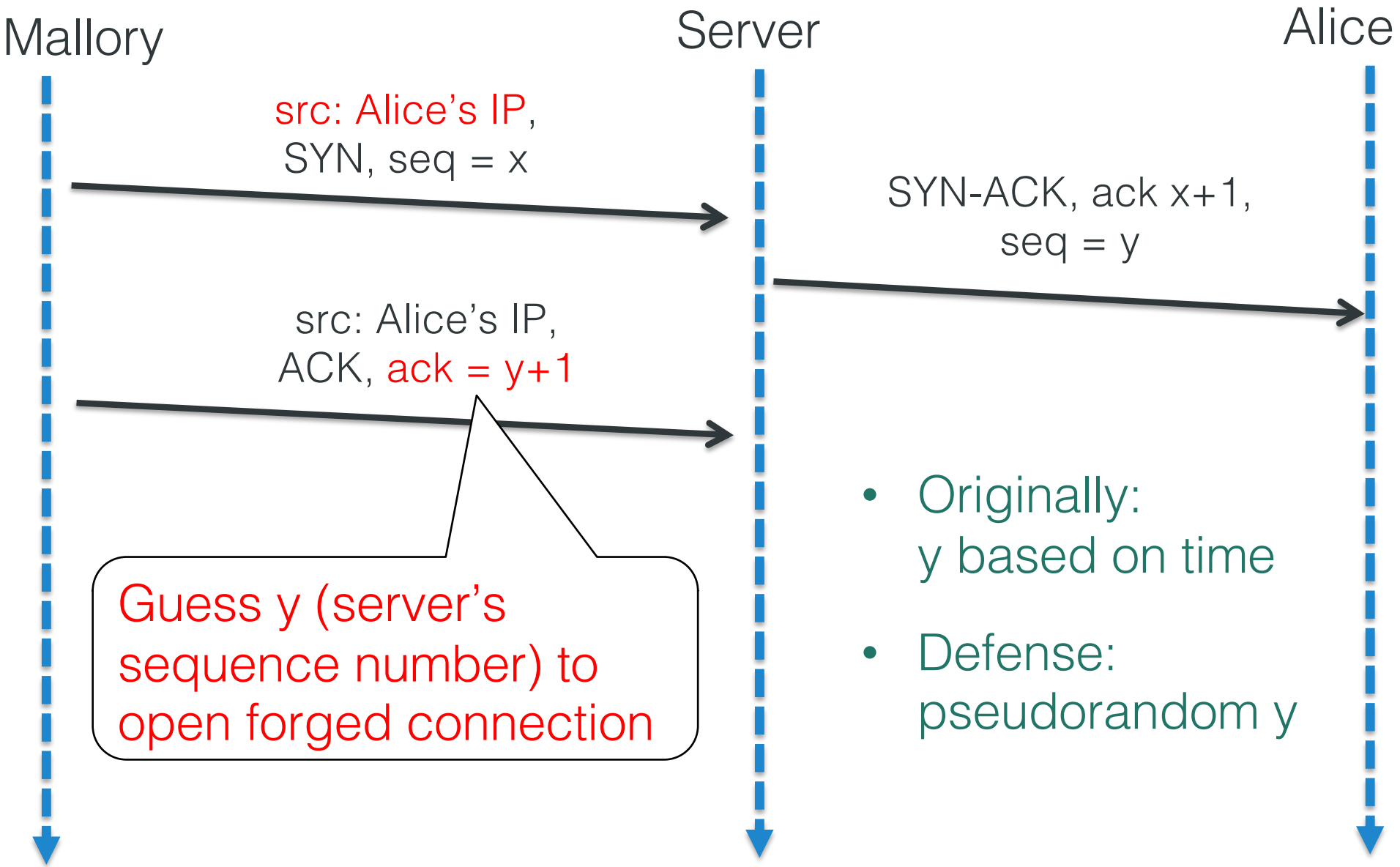
(TS//SI//NF) Dates When PRISM Collection Began For Each Provider



PRISM Program Cost: ~$20M per year

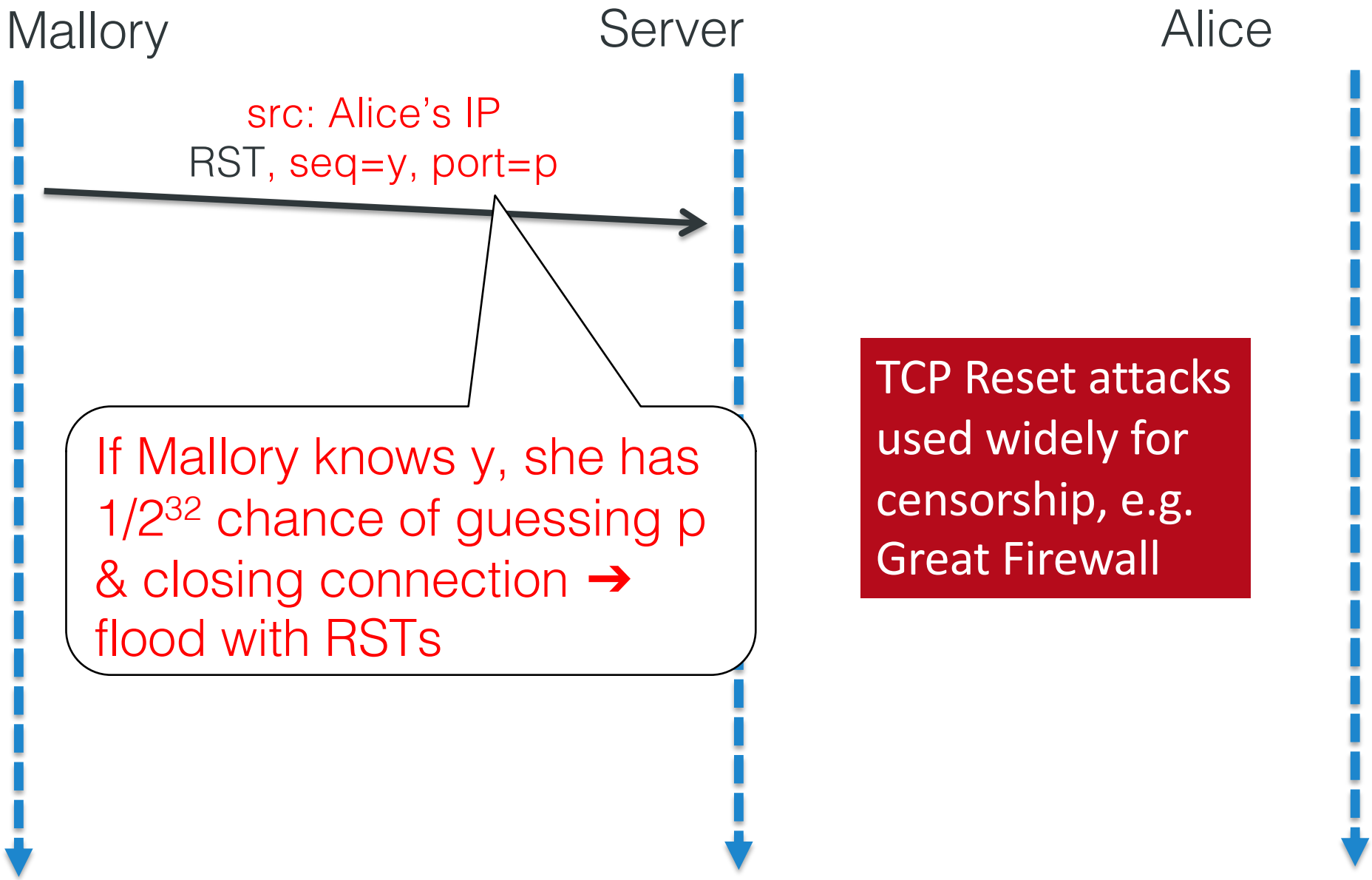
Active Attacks

Active Attacks: Blind Spoofing



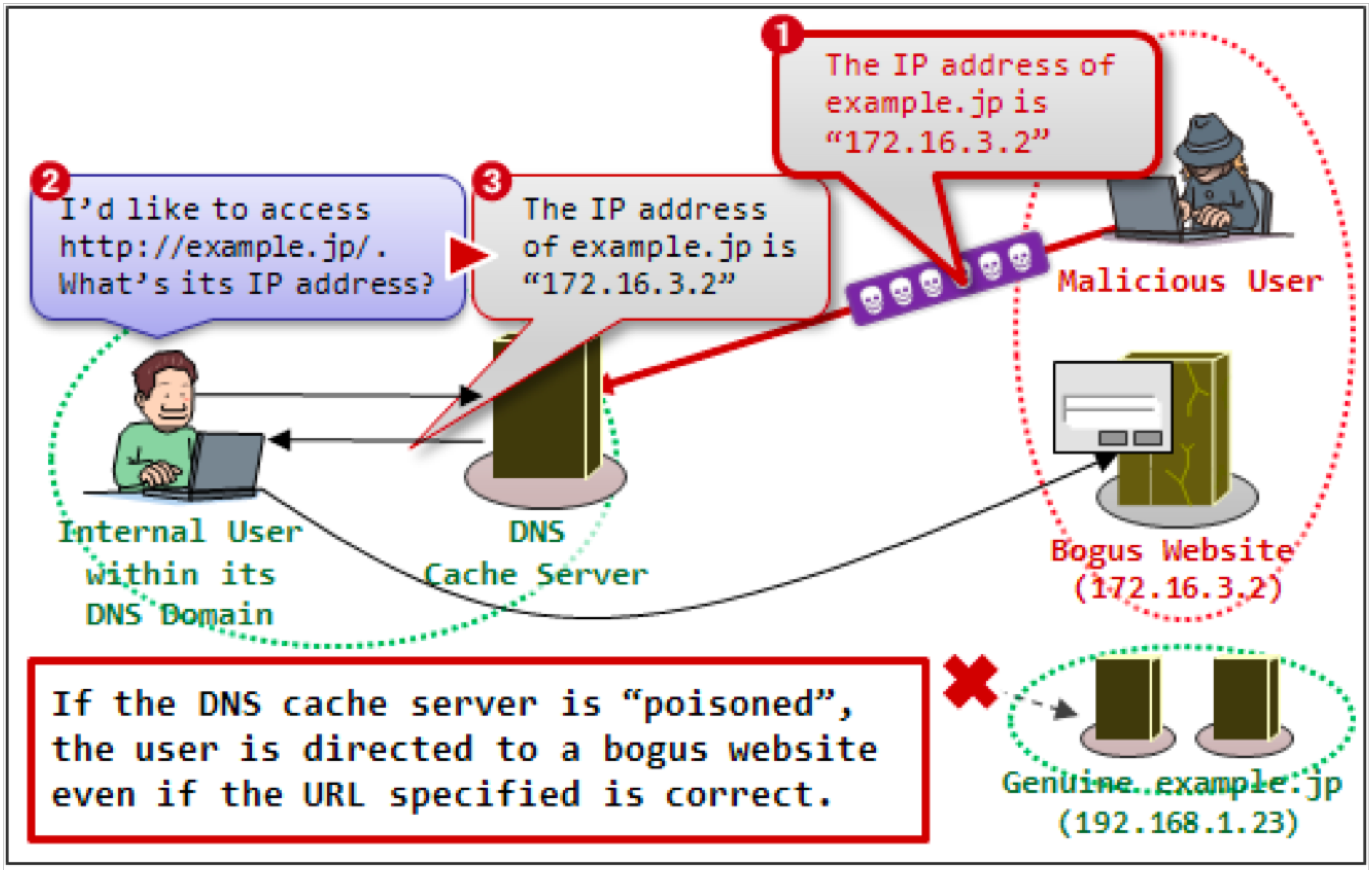
- Originally: y based on time
- Defense: pseudorandom y

RST Hijacking



TCP Reset attacks used widely for censorship, e.g. Great Firewall

DNS Cache Poisoning



DNS Cache Poisoning (cont.)

Defense: randomize 16-bit QID