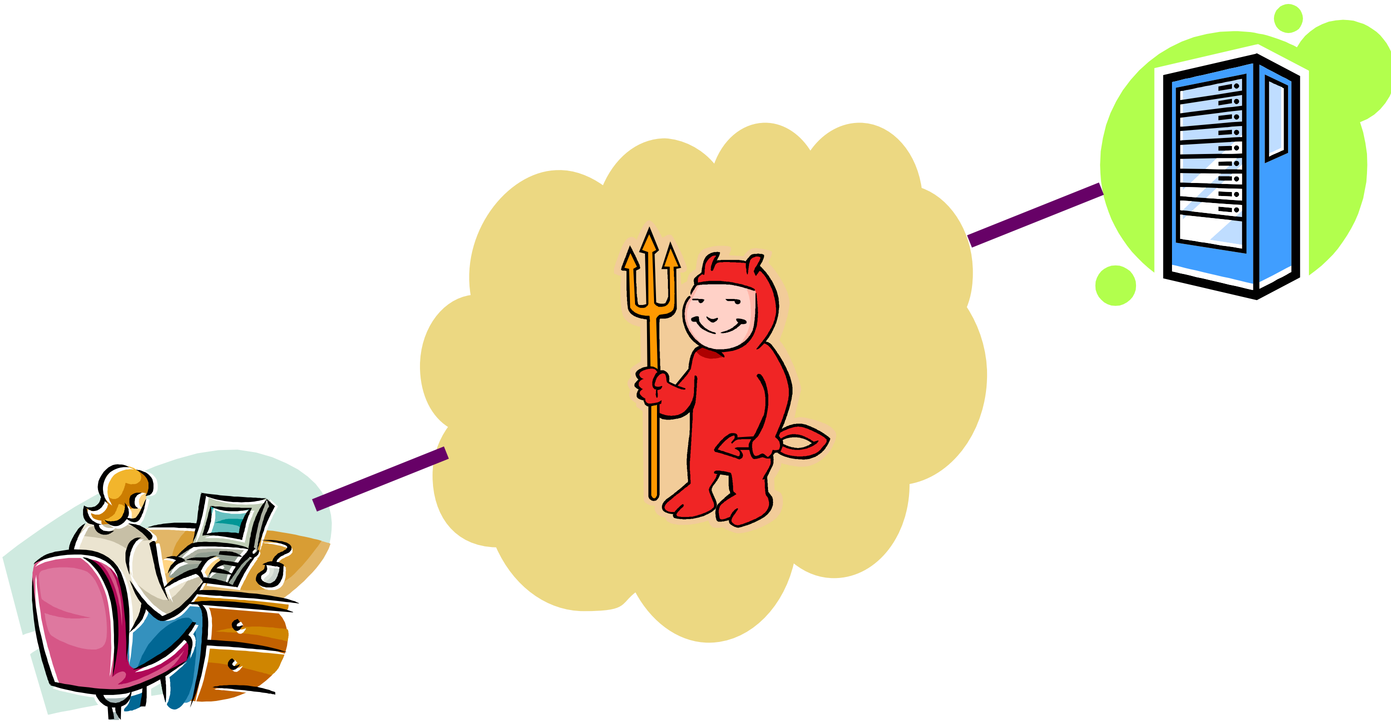
Recall from crypto lecture

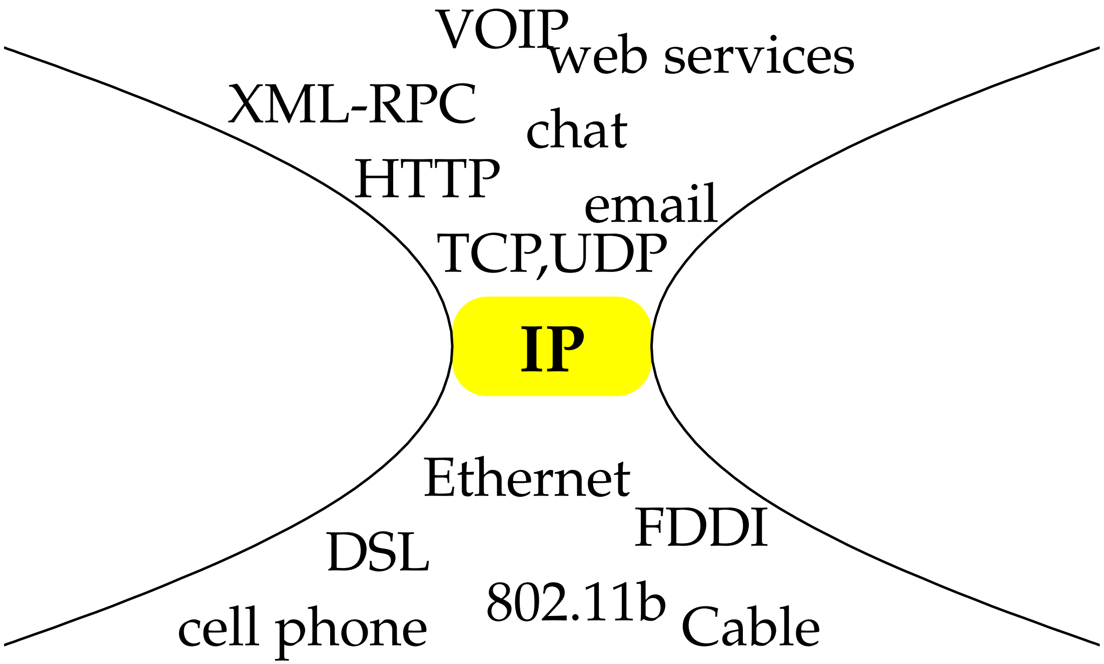


- **We basically assume bad guys control the network**
- **Now we will make this more precise**

The medium-term plan

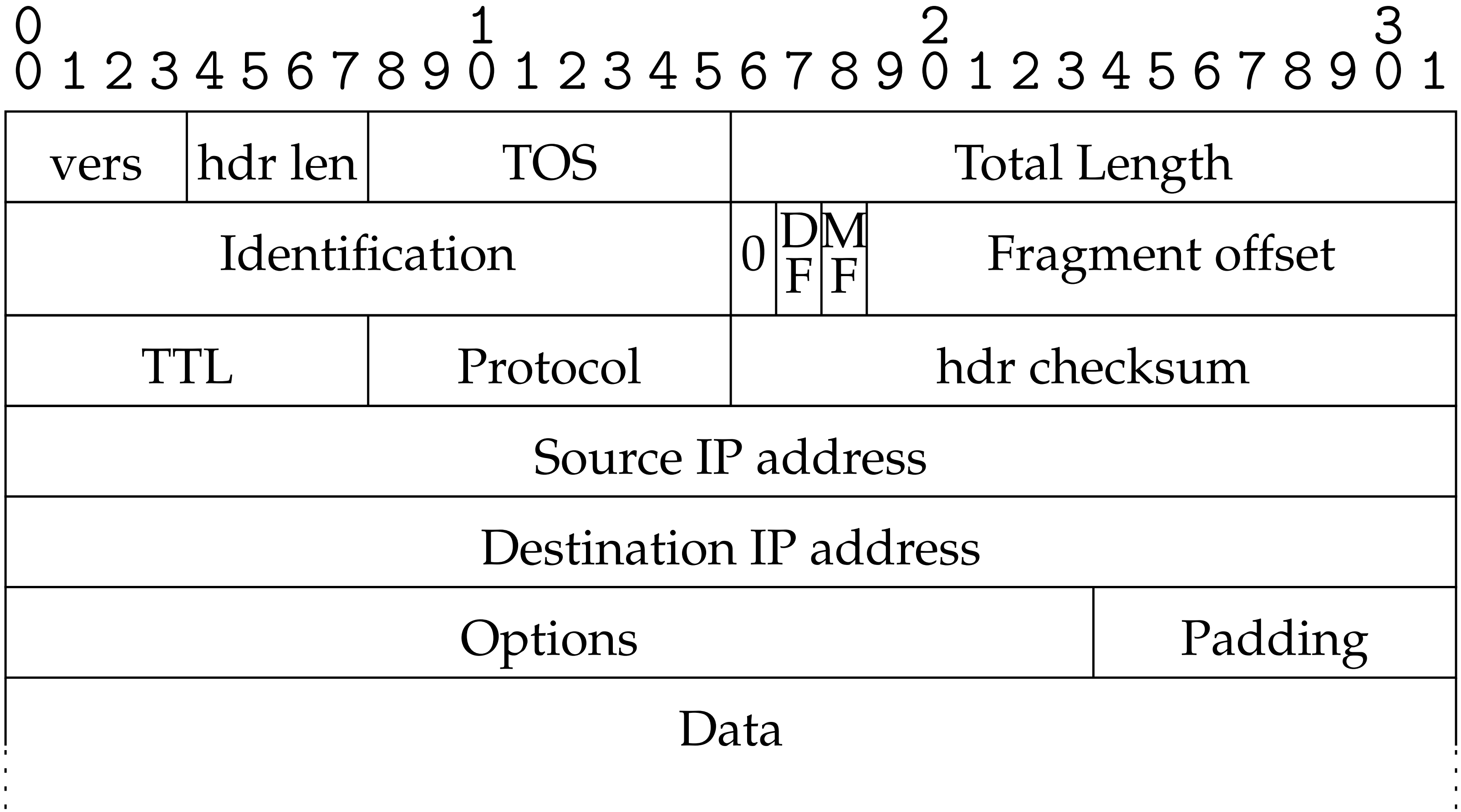
- **Today: How Internet works & how to attack it**
  - How attackers can realize picture on previous slide
- **Thursday: Defense mechanisms**
- **Next Tuesday: Denial of service**
- **Next Thursday: Automated attacks & defenses**
- **Following Tuesday: Privacy & anonymity**

Internet protocol (IP)



- **Many different physical networks**
- **Many different network applications**
- **Idea: Inter-operate through narrow IP protocol**
  - Often referred to as "hourglass model"

IP packet format



IP header details

- **Routing is based on destination address**
- **TTL (time to live) decremented at each hop (avoids loops)**
  - TTL mostly saves from routing loops
  - But other cool uses…
- **Fragmentation possible for large packets**
  - Fragmented in network if crosses link w. small frame size
  - MF bit means more fragments for this IP packet
  - DF bit says “don’t fragment” (returns error to sender)
- **Following IP header is “payload” data**
  - Typically beginning with TCP or UDP header

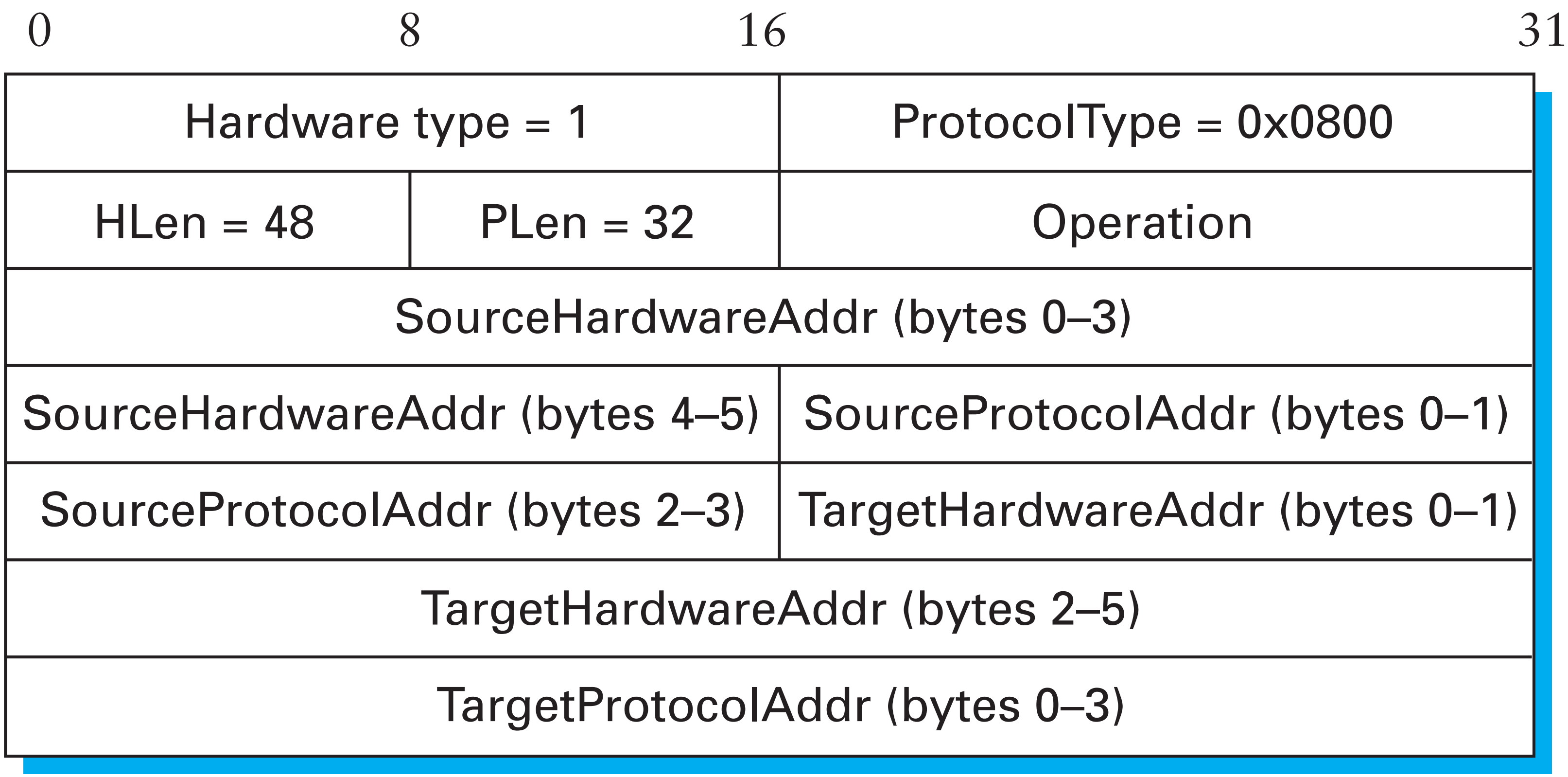
Simple protocol: ICMP

- **Internet Control Message Protocol (ICMP)**
  - Echo (ping)
  - Redirect (from router to source host)
  - Destination unreachable (protocol, port, or host)
  - TTL exceeded (so datagrams don't cycle forever)
  - Checksum failed
  - Reassembly failed
  - Cannot fragment
  - Many ICMP messages include part of packet that triggered them
- **Example use: Traceroute**

IP vs. lower-level net addresses

- **Must map IP addresses into physical addresses**
  - E.g., Ethernet address of destination host or next hop router
  - Often called *Medium Access Control* (MAC) address (not message authentication code or mandatory access control)
- **Could encode MAC address in IP address [IPv6]**
- **Usually use ARP – *address resolution protocol***
  - Table of IP to physical address bindings
  - Broadcast request if IP address not in table
  - Everybody learns physical address of requesting node (broadcast)
  - Target machine responds with its physical address
  - Table entries are discarded if not refreshed

ARP Ethernet packet format



[figures from Peterson & Davie]

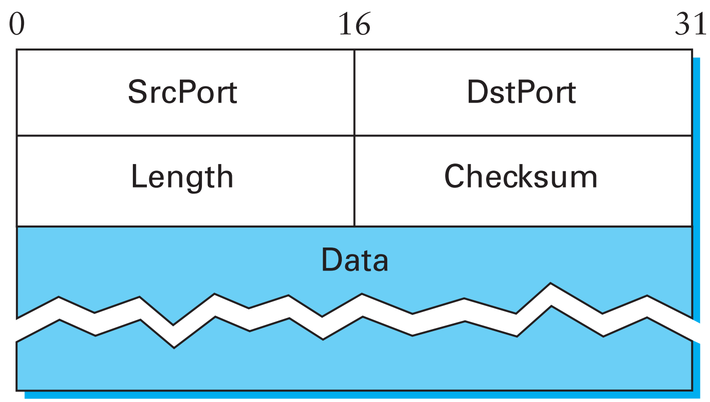
LAN Eavesdropping

- **Most network cards support "promiscuous mode"**
  - Return all packets, not just those address to your MAC addr.
  - Useful for network debugging, software Ethernet switches
  - Also useful for eavesdropping
- **It used to be all Ethernets were broadcast networks**
  - All hosts tapped into same coaxial cable
  - Any host could see all other hosts' packets
- **Today still the case with 802.11b**
  - What web pages do people surf during lecture?
    [wireshark demo]
- **But *switched Ethernet* solves the problem**

Wrong: Eavesdropping w. switches

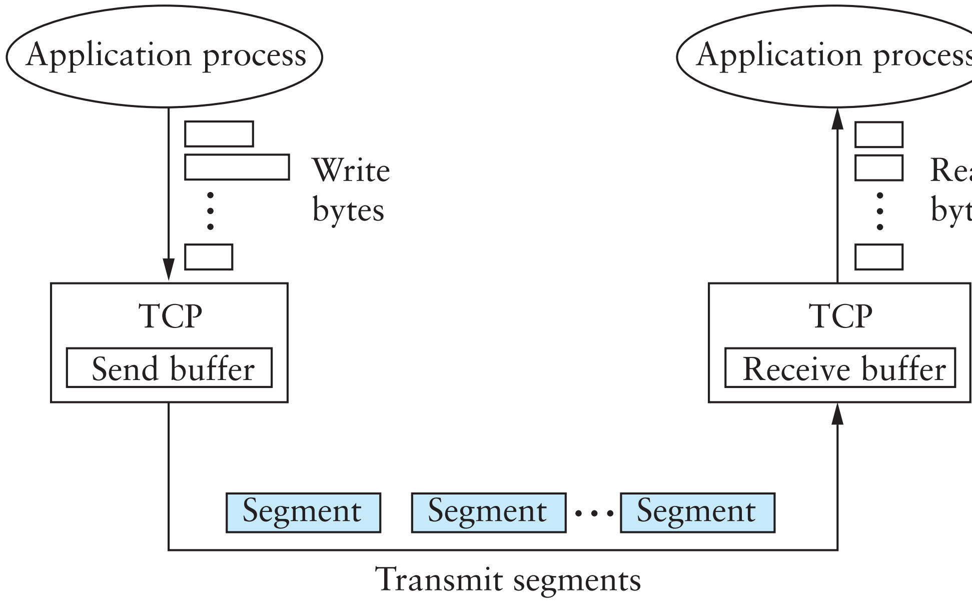
- **Old switches "fail open" on MAC table overflow**
  - Attacker just generates packets from tons of MAC addresses
  - Ethernet switch then reverts to broadcast-style network
- **ARP spoofing**
  - Broadcast an ARP request "from" target's IP address
  - Insert your MAC address for target IP in everyone's ARP table
  - (Note: May generate log messages)
- **ICMP redirect abuse**
- **RIP routing protocol abuse**
- **BGP routing protocol abuse**
- **DHCP abuse (give bogus default router)**

UDP – *user datagram protocol*



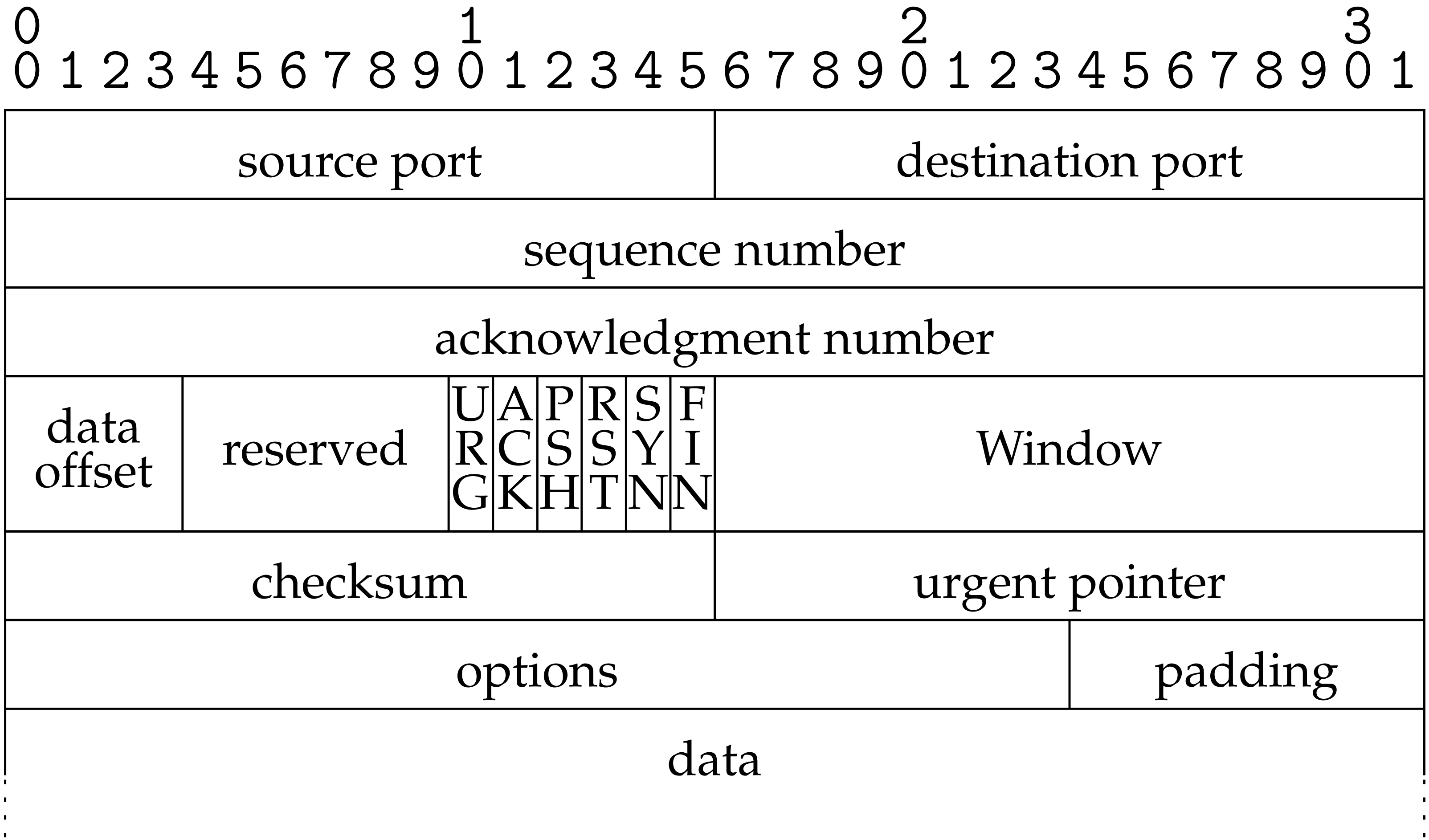
- **Unreliable and unordered datagram service**
- **Adds multiplexing, checksum on whole packet**
- **No flow control, reliability, or order guarantees**
- **Endpoints identified by ports**
  - servers have well-known ports (e.g., 53 for DNS)
- **Checksum includes "pseudo-header" w. IP addresses**

TCP – *Transmission Control Protocol*



- **Full duplex, connection-oriented byte stream**
- **Flow control**
  - If one end stops reading, writes at other eventuall block/fail
- **Congestion control**
  - Keeps sender from overrunning network

TCP segment


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
source port
destination port
sequence number
acknowledgment number
data offset
reserved
URG
ACK
PSH
RST
SYN
FIN
Window
checksum
urgent pointer
options
padding
data

TCP fields

- **Ports**
- **Seq no. – segment position in byte stream**
- **Ack no. – seq no. sender expects to receive next**
- **Data offset – # of 4-byte header & option words**
- **Window – willing to receive (flow control)**
- **Checksum**
- **Urgent pointer**

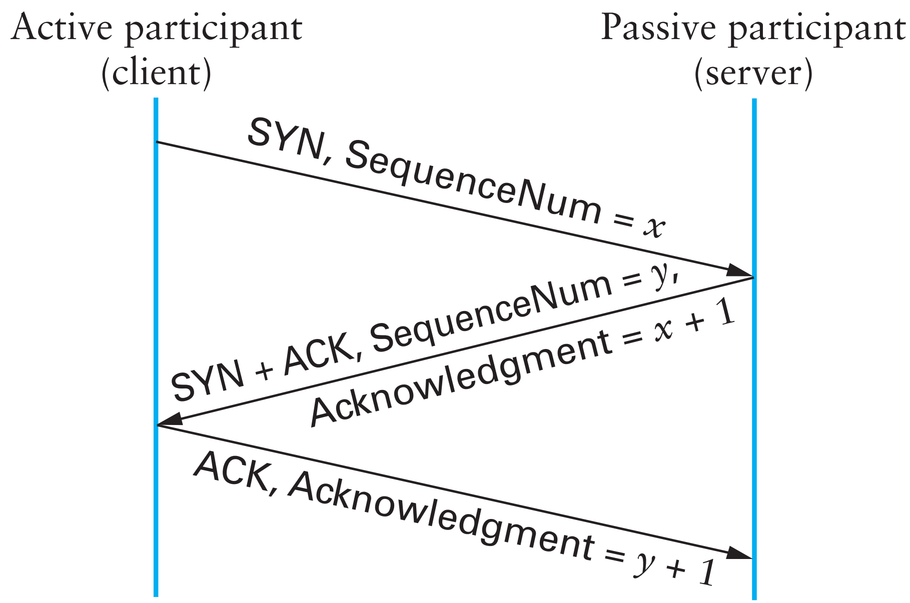
TCP Flags

- URG – urgent data present
- ACK – ack no. valid (all but first segment)
- PSH – push data up to application immediately
- RST – reset connection
- SYN – “synchronize” establishes connection
- FIN – close connection

A TCP Connection (no data)

```
orchard.48150 > essex.discard:
        S 1871560457:1871560457(0) win 16384
essex.discard > orchard.48150:
        S 3249357518:3249357518(0) ack 1871560458 win 17376
orchard.48150 > essex.discard: . ack 1 win 17376
orchard.48150 > essex.discard: F 1:1(0) ack 1 win 17376
essex.discard > orchard.48150: . ack 2 win 17376
essex.discard > orchard.48150: F 1:1(0) ack 2 win 17376
orchard.48150 > essex.discard: . ack 2 win 17375
```

Connection establishment



- **Need SYN packet in each direction**
  - Typically second SYN also acknowledges first
  - Supports “simultaneous open,” seldom used in practice
- **If no program listening: server sends RST**
- **If server backlog exceeded: ignore SYN**
- **If no SYN-ACK received: retry, timeout**

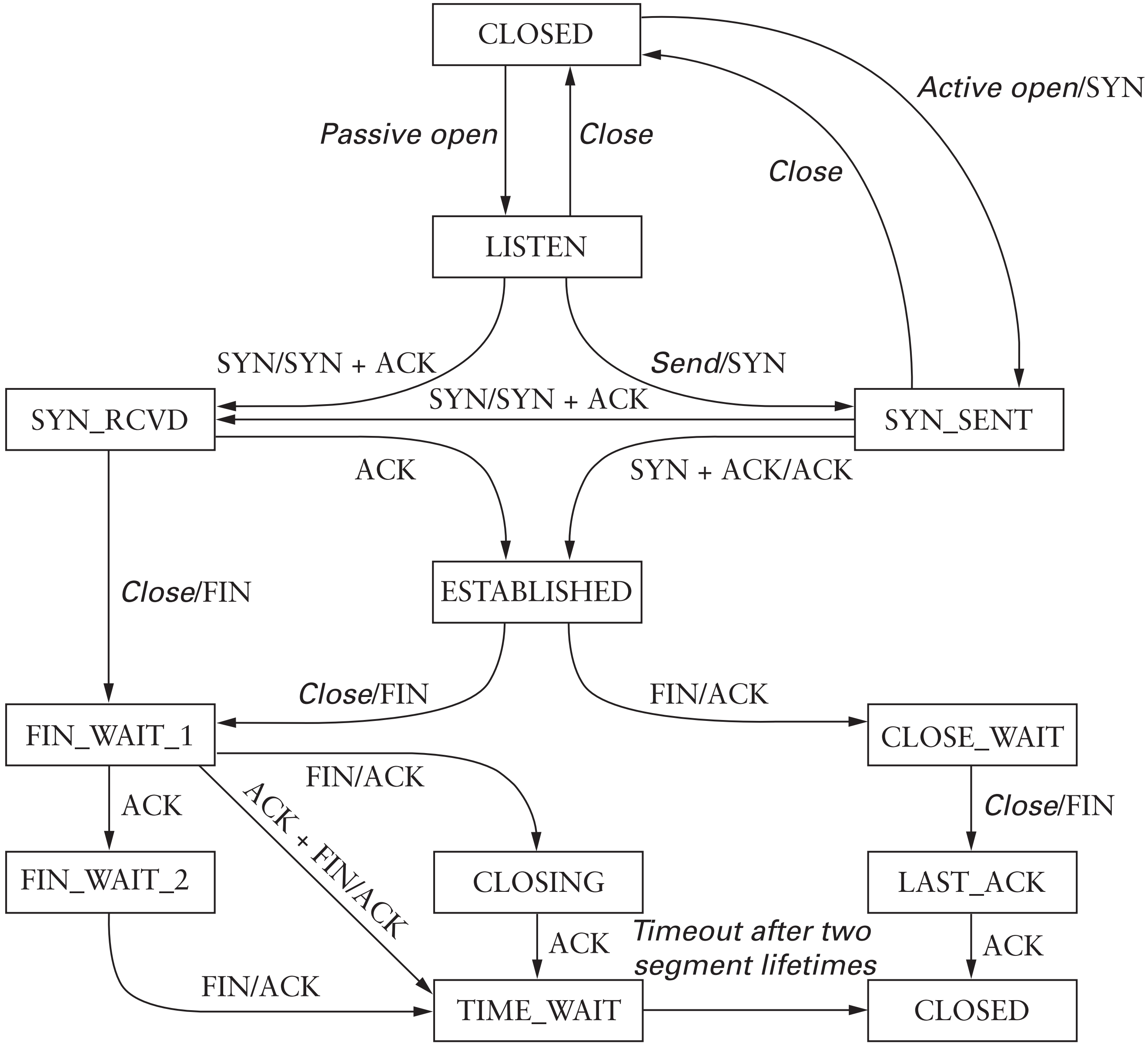
Connection termination

- **FIN bit says no more data to send**
  - Caused by close or shutdown on sending end
  - Both sides must send FIN to close a connection
- **Typical close:**
  - $A \rightarrow B$: FIN, seq S_A, ack S_B
  - $B \rightarrow A$: ack $S_A + 1$
  - $B \rightarrow A$: FIN, seq S_B, ack $S_A + 1$
  - $A \rightarrow B$: ack $S_B + 1$
- **Can also have simultaneous close**
- **After last message, can A and B forget about closed socket?**

TIME_WAIT

- **Problems with closed socket**
  - What if final ack is lost in the network?
  - What if the same port pair is immediately reused for a new connection? (Old packets might still be floating around.)
- **Solution: "active" closer goes into TIME_WAIT**
  - Active close is sending FIN before receiving one
  - After receiving ACK and FIN, keep socket around for 2MSL (twice the "maximum segment lifetime")
- **Can pose problems with servers**
  - OS has too many sockets in TIME_WAIT, slows things down
  - Hack: Can send RST and delete socket, set SO_LINGER socket option to time 0 (useful for benchmark programs)

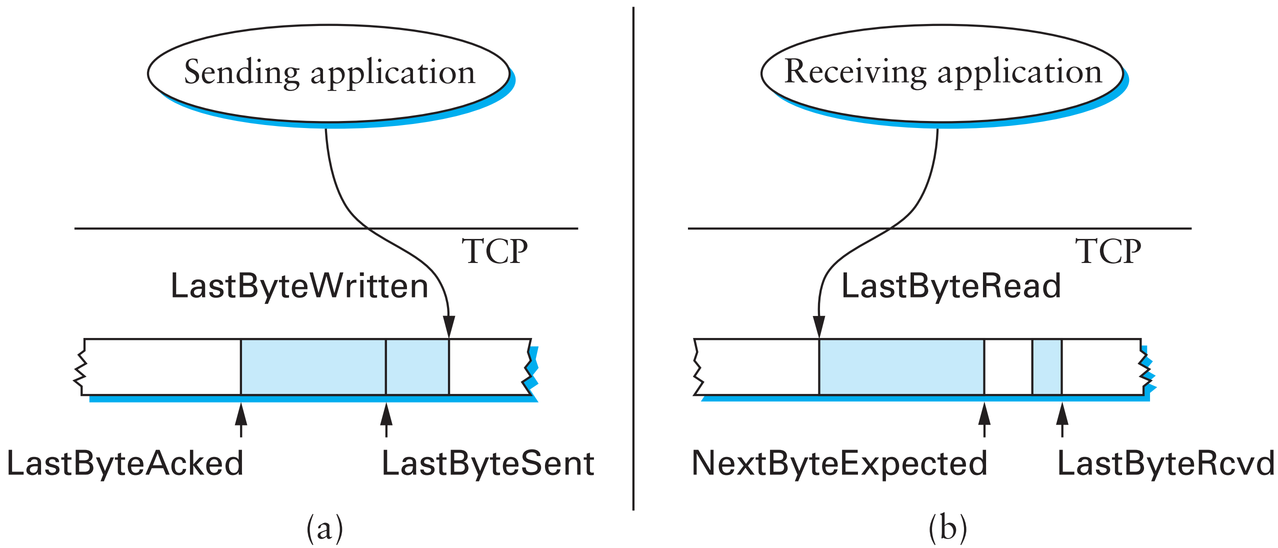
State summary…



Sending data

- **Data sent in MSS-sized segments**
  - Chosen to avoid fragmentation (e.g., 1460 on ethernet LAN)
  - Write of 8K might use 6 segments—PSH set on last one
  - PSH avoids unnecessary context switches on receiver
- **Sender's OS can delay sends to get full segments**
  - Nagle algorithm: Only one unacknowledged short segment
  - TCP_NODELAY option avoids this behavior
- **Segments may arrive out of order**
  - Sequence number used to reassemble in order
- **Window achieves flow control**
  - If window 0 and sender's buffer full, write will block or return EAGAIN

Sliding window



- **Used to guarantee reliable & in-order delivery**
- **Also used for flow control**
  - Instead of fixed window size, receiver sends AdvertisedWindow

A TCP connection (3 byte echo)

```
orchard.38497 > essex.echo:
        S 1968414760:1968414760(0) win 16384
essex.echo > orchard.38497:
        S 3349542637:3349542637(0) ack 1968414761 win 17376
orchard.38497 > essex.echo: . ack 1 win 17376
orchard.38497 > essex.echo: P 1:4(3) ack 1 win 17376
essex.echo > orchard.38497: . ack 4 win 17376
essex.echo > orchard.38497: P 1:4(3) ack 4 win 17376
orchard.38497 > essex.echo: . ack 4 win 17376
orchard.38497 > essex.echo: F 4:4(0) ack 4 win 17376
essex.echo > orchard.38497: . ack 5 win 17376
essex.echo > orchard.38497: F 4:4(0) ack 5 win 17376
orchard.38497 > essex.echo: . ack 5 win 17375
```

Retransmission

- **TCP dynamically estimates round trip time**
- **If segment goes unacknowledged, must retransmit**
- **Use exponential backoff (in case loss from congestion)**
- **After ∼10 minutes, give up and reset connection**
- **Many optimizations in TCP**
  - E.g., Don't necessarily halt everything for one lost packet
  - Just reduce window by half, then slowly augment

Congestion avoidance

- **Transmit at just the right rate to avoid congestion**
  - Slowly increase transmission rate to find maximum
  - One lost packet means too fast, cut rate
  - Use additive increase, multiplicative decrease
- **Sender-maintained congestion window limits rate**
  - Maximum amount of outstanding data:
    min(congestion-window, flow-control-window)
- **Cut rate in half after 3 duplicate ACKs**
  - Fewer duplicates may just have resulted from reordering
  - Fast retransmit: resend only lost packet
- **If timeout, cut cong. window back to 1 segment**
  - Slow start – exponentially increase to ss thresh

Access control

- **Many services base access control on IP addresses**
  - E.g., mail servers allow relaying
  - NFS servers allow you to mount file systems
  - X-windows can rely on IP address
  - Old BSD “rlogin/rsh” services
  - Many clients assume they are talking to right server based in part on IP address (e.g., DNS, NTP, rsync, etc)
- **Very poor assumption to make**

Spoofing TCP source [Morris]

- **Suppose can't eavesdrop but can forge packets**
- **Can send forged SYN, not get SYN-ACK, but then send data anyway**
  - E.g., data might be "`tcpserver 0.0.0.0 2323 /bin/sh -i`"
  - Allows attacker to get shell on machine
- **Problem: What server Initial SeqNo to ACK?**

Spoofing TCP source [Morris]

- **Suppose can't eavesdrop but can forge packets**
- **Can send forged SYN, not get SYN-ACK, but then send data anyway**
  - E.g., data might be "`tcpserver 0.0.0.0 2323 /bin/sh -i`"
  - Allows attacker to get shell on machine
- **Problem: What server Initial SeqNo to ACK?**
  - In many OSes, very ISNs very predictable
  - Base guess on previous probe from real IP addr
- **Problem: Real client may RST unexpected SYN-ACK**

Spoofing TCP source [Morris]

- **Suppose can't eavesdrop but can forge packets**
- **Can send forged SYN, not get SYN-ACK, but then send data anyway**
  - E.g., data might be "`tcpserver 0.0.0.0 2323 /bin/sh -i`"
  - Allows attacker to get shell on machine
- **Problem: What server Initial SeqNo to ACK?**
  - In many OSes, very ISNs very predictable
  - Base guess on previous probe from real IP addr
- **Problem: Real client may RST unexpected SYN-ACK**
  - Spoof target may be running a server on some TCP port
  - Overwhelm that port with SYN packets until it ignores them
  - Will likewise ignore the victim server's SYN-ACK packet

Spoofing TCP [Joncheray]

- **Say you can eavesdrop, want to tamper w. connection**
  - E.g., system uses challenge-response authentication
  - Want to hijack already authenticated TCP connection
- **Recall each end of TCP has flow-control window**
- **Idea: *Desynchronize* the TCP connection**
  - E.g., usually $C_{\text{ACK}} \leq S_{\text{SEQ}} \leq C_{\text{ACK}} + C_{\text{WIN}}$ and $S_{\text{ACK}} \leq C_{\text{SEQ}} \leq S_{\text{ACK}} + S_{\text{WIN}}$
  - If no data to send and sequence numbers outside of range, TCP connection is *desynchronized*
- **Q: How to desynchronize a TCP connection?**

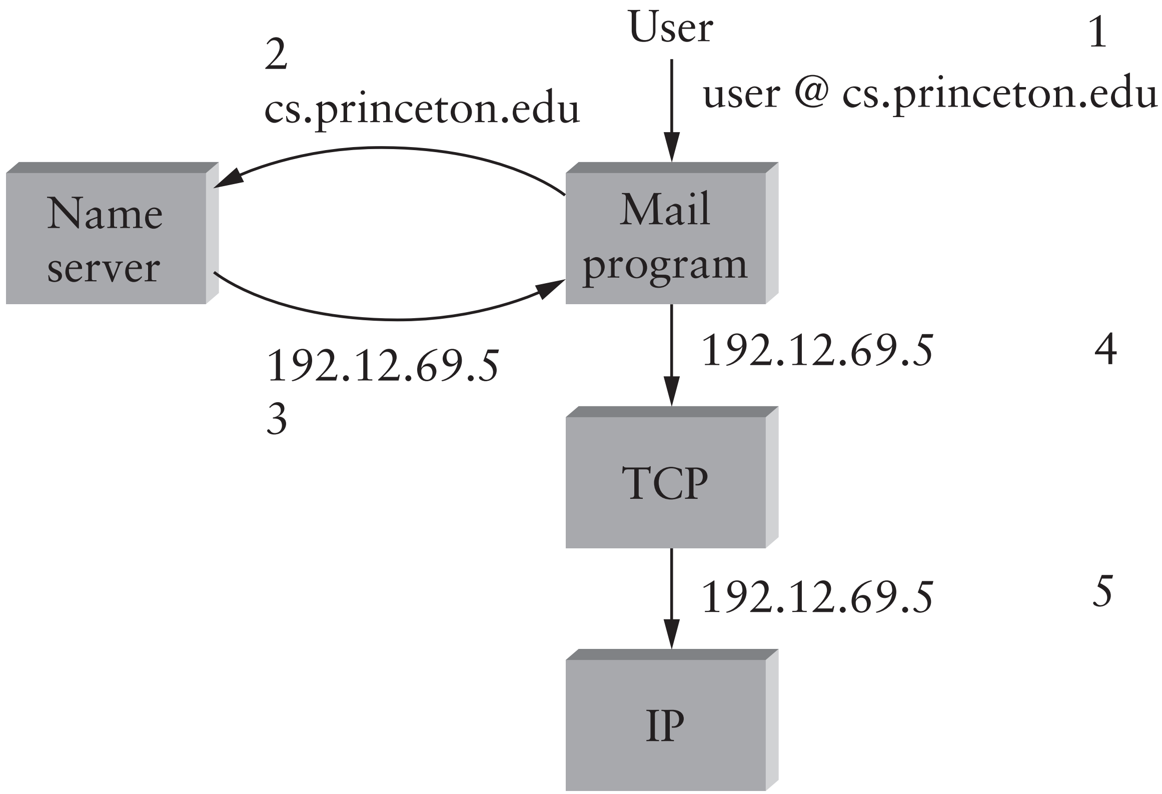
Desynchronizing TCP

- **Early desynchronization**
  - Client connects to server
  - Attacker sents RST, then forged SYN to server
  - Server has connection w. same ports, different S_{ACK}
- **Null data desynchronization**
  - Attacker generates a lot of data that will be ignored by app.
  - Sends NULL data to both client and server
  - Drives up C_{ACK} and S_{ACK} so out of range
- **How to exploit this for hijacking?**

Exploiting desynchronized TCP

- **Packets with SeqNo outside of window are ignored**
  - After all, old, retransmitted packets might still be bouncing around the network
  - Can't just RST a connection because you see an old packet
- **As long as desynchronized, just inject data**
  - Data sent by real nodes will be ignored
  - Injected data will cause ACKs that get ignored
  - So attacker determines what each side receives
- **ACK Storms**
  - Out of window packet does cause an ACK to be generated
  - ACK itself out of window, causes other side to generate ACK
  - Ping-pong continues until a packet is lost
  - Bad for network, but not so bad for attacker

Domain Name System (DNS)



- **Users can't remember IP addresses**
  - Need to map symbolic names (`www.stanford.edu`)→IP addr
- **Implemented by library functions & servers**
  - `gethostbyname()` talks to *name server* over UDP

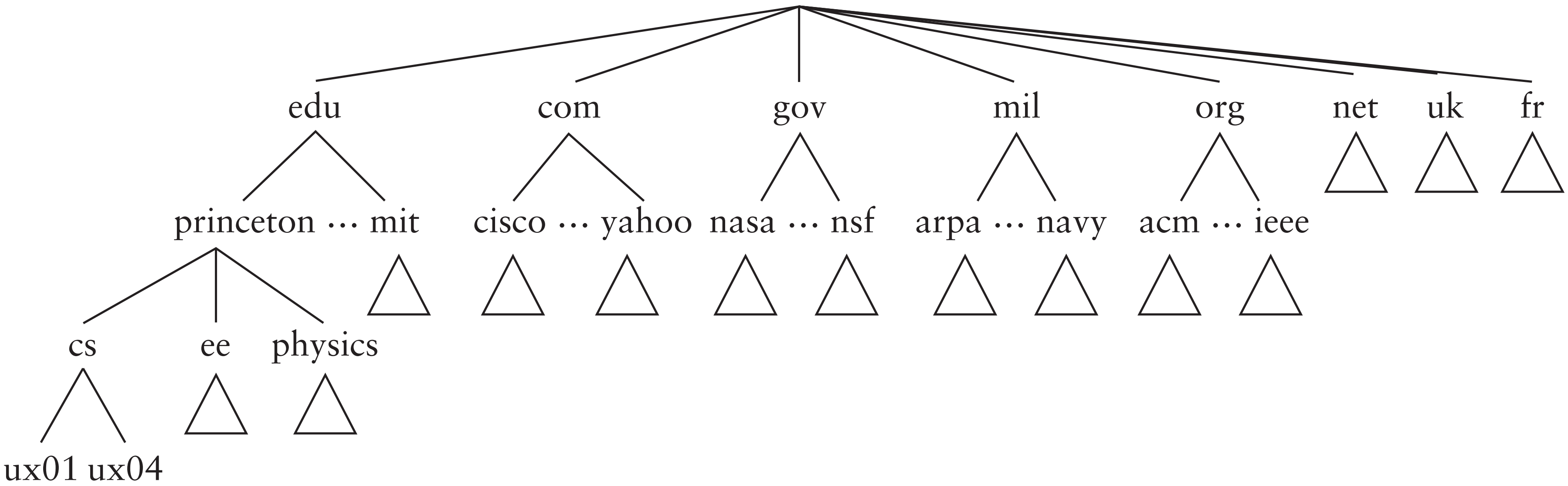
Goals of DNS

- **Scalability**
  - Must handle huge number of records
  - Potentially *exponential* in name size—because custom software may synthesize names on-the-fly
- **Distributed control**
  - Let people control their own names
- **Fault-tolerance**
  - Old software assumed all addresses in `hosts.txt` file
  - Bad potential failure modes when name lookups fail
  - Minimize lookup failures in the face of other network problems
- **Security? Not so much**

The good news

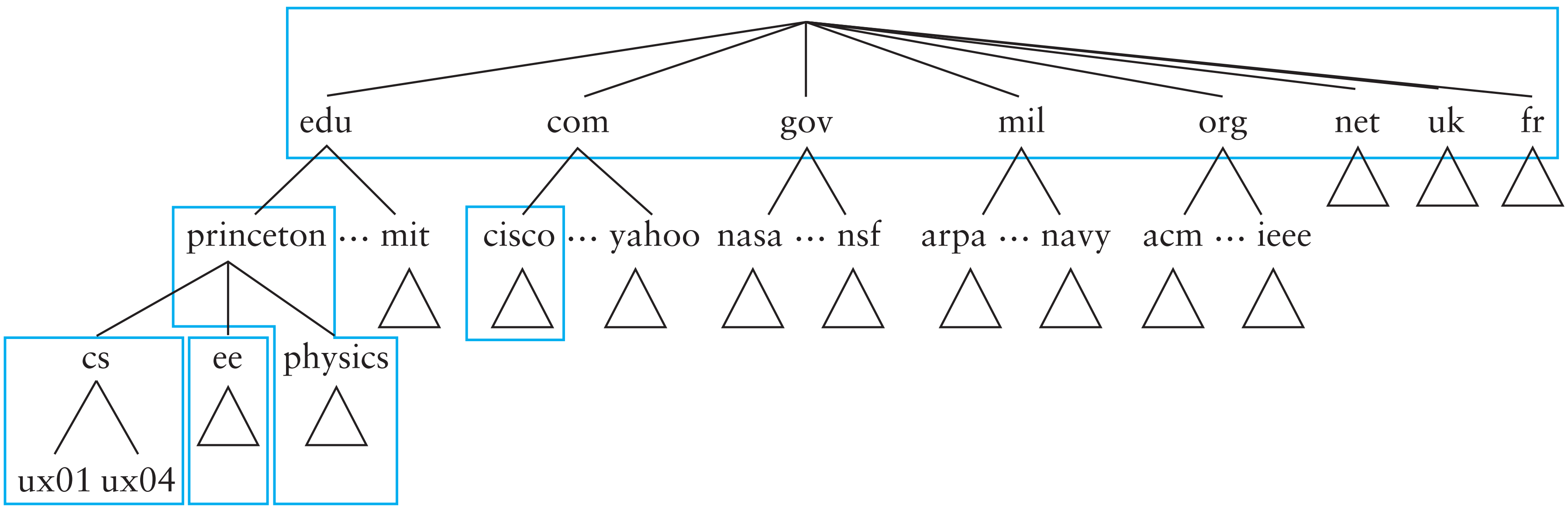
- **Properties that make DNS goals easier to achieve:**
  1. **Read-only or read-mostly database**
     - People typically look up hostnames much more often than they are updated
  2. **Loose consistency**
     - When adding a machine, may be okay if info takes minutes or hours to propagate
- **These suggest approach w. aggressive caching**
  - Once you have looked up hostname, remember result
  - Don't need to look it up again in near future

DNS Names



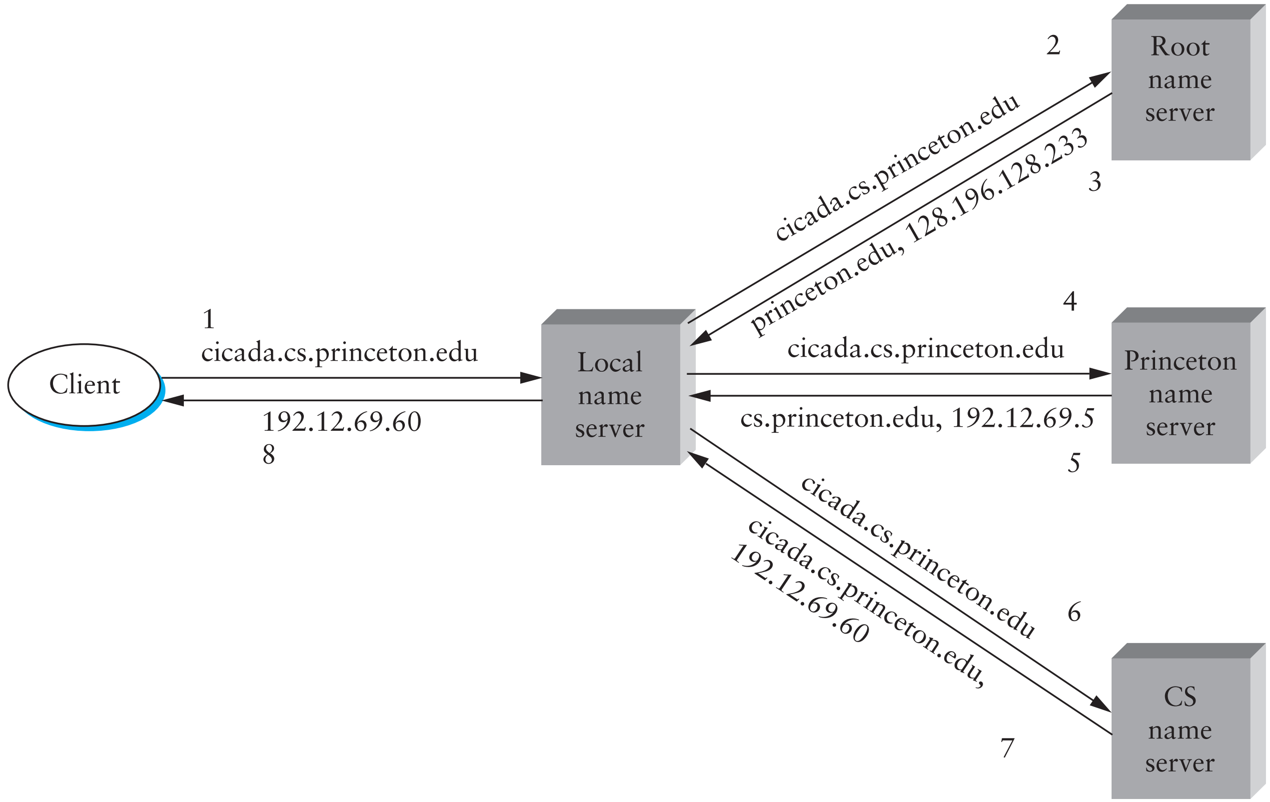
- **Use hierarchical naming scheme**

DNS Names



- **Break namespace into a bunch of zones**
  - root (.), `edu.`, `stanford.edu.`, `cs.stanford.edu.`, ...
  - Zones separately administered $\Longrightarrow$ delegation
  - Parent zones tell you how to find servers for dubdomains.
- **Each zone served from several replicated servers**

DNS software architecture



- **Apps make recursive queries to local DNS server**
- **Local server queries remote servers non-recursively**
  - Aggressively caches result
  - E.g., only contact root on first query ending `.stanford.edu`

DNS protocol

- **TCP/UDP port 53**
- **Most traffic uses UDP**
  - Lightweight protocol has 512 byte UDP message limit
  - retry w. TCP if UDP fails (e.g., reply truncated)
- **TCP requires message boundaries**
  - Prefix all messages w. 16-bit length
- **Bit in query determines if query is recursive**

Resource records

- **All DNS info represented as resource records (RR):**
  name* [TTL] [*class*] *type rdata
  - *name* – domain name (e.g., `www.nyu.edu`)
  - TTL – time to live in seconds
  - *class* – for extensibility, usually IN (1) "Internet"
  - *type* – type of the record
  - *rdata* – resource data dependent on the *type*
- **Some important DNS RR types:**
  - A – Internet address (IPv4)
  - NS – name server
  - MX – mail exchanger

Resource record examples

- **Example resource records**

```
stanford.edu.  2603    IN  A     171.67.20.37
stanford.edu.  152554  IN  NS    Avallone.stanford.edu.
stanford.edu.  172800  IN  NS    AUTHDNS4.NETCOM.DUKE.edu.
stanford.edu.  3595    IN  MX    20 mx1.stanford.edu.
```

- **[Demo of dig program]**

Mapping addresses to names

- **Sometimes want to find DNS name given address**
- **PTR records specify names**

  ***name* [TTL] [IN] PTR "*ptrdname*"**

  - *name* – somehow encode address… how?
  - *ptrdname* – domain name for this address
- **IPv4 addrs stored under** `in-addr.arpa` **domain**
  - Reverse name, append `in-addr.arpa`
  - To look up 216.165.108.10 → `10.108.165.216.in-addr.arpa.`
  - Why reversed? Delegation!
- **IPv6 under** `ip6.arpa`
  - Historical note: ARPA funded original Internet

Access control based on hostnames

- **Weak access control frequently based on hostname**
  - E.g., allow clients matching `*.stanford.edu` to see web page
- **Is it safe to trust the PTR records you get back?**

Access control based on hostnames

- **Weak access control frequently based on hostname**
  - E.g., allow clients matching `*.stanford.edu` to see web page
- **Is it safe to trust the PTR records you get back?**
- **No: PTR records controlled by network owner**
  - E.g., My machine serves `3.66.171.in-addr.arpa.`
  - I can serve `11.3.66.171.in-addr.arpa. IN PTR www.berkeley.edu.`
  - Don't believe I own Berkeley's web server!
- **How to solve problem?**

Access control based on hostnames

- **Weak access control frequently based on hostname**
  - E.g., allow clients matching `*.stanford.edu` to see web page
- **Is it safe to trust the PTR records you get back?**
- **No: PTR records controlled by network owner**
  - E.g., My machine serves `3.66.171.in-addr.arpa.`
  - I can serve `11.3.66.171.in-addr.arpa. IN PTR www.berkeley.edu.`
  - Don't believe I own Berkeley's web server!
- **How to solve problem?**
  - Always do forward lookup on PTRs you get back
  - `www.berkeley.edu. 600 IN A 169.229.131.92`
  - Doesn't match my IP (171.66.3.11), so reject

Some implementation details

- **How does local name server know root servers?**
  - Need to configure name server with *root cache* file
  - Contains root name servers and their addresses

```
.                       3600000  NS  A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000  A   198.41.0.4
.                       3600000  NS  B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000  A   128.9.0.107
...
```

Some implementation details

- **How do you get addresses of other name servers?**
  - To lookup names ending `.stanford.edu.`, ask `Avallone.stanford.edu.`
  - But how to get `Avallone.stanford.edu.`'s address?
- **Solution: glue records – A records in parent zone**
  - Name servers for `edu.` have A record of `Avallone.stanford.edu.`
  - [Check using `dig +norec`]

Structure of a DNS message

```
+---------------------+
|        Header       |
+---------------------+
|       Question      | the question for the name server
+---------------------+
|        Answer       | RRs answering the question
+---------------------+
|      Authority      | RRs pointing toward an authority
+---------------------+
|      Additional     | RRs holding additional information
+---------------------+
```

- **Same message format for queries and replies**
  - Query has zero RRs in Answer/Authority/Additional sections
  - Reply includes question, plus has RRs
- **Authority allows for delegation**
- **Additional for glue + other RRs client might need**

Header format

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- **QR – 0=query, 1=response**
- **RCODE – error code**
- **AA=authoritative answer, TC=truncated, RD=recursion desired, RA=recursion available**

Encoding of RRs

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                      NAME                     /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Using DNS for load-balancing

- **Can have multiple RR of most types for one name**
  - Required for NS records (for availability)
  - Useful for A records
  - (Not legal for CNAME records)
- **Servers rotate order in which records returned**
  - Most apps just use first address returned ("`#define h_addr h_addr_list[0]`")
  - Even if your name server caches results, clients will be spread amonst servers
- **Example:** `dig cnn.com` **multiple times**

Secondary servers

- **Availability requires geographically disperate replicas**
  - E.g., Stanford asks Duke to serve `stanford.edu`
- **Typical setup: One master many slave servers**
- **How often to sync up servers? Trade-off**
  - All the time $\Longrightarrow$ high overhead
  - Rarely $\Longrightarrow$ stale data
- **Put trade-off under domain owner's control**
  - Fields in SOA record control secondary's behavior
  - Primary can change SOA without asking human operator of secondary

The SOA record

- **Every delegated zone has one SOA record**
  name* [TTL] [IN] SOA *mname rname
  serial refresh retry expire minimum
  - *name* – Name of zone (e.g., `nyu.edu`
  - *mname* – DNS name of main name server
  - *rname* – E-mail address of contact (`@`→`.`)
  - *serial* – Increases each time zone changes
  - *refresh* – How often secondary servers should sync
  - *retry* – How soon to re-try sync after a failure
  - *expire* – When to discard data after repeated failures
  - *minimum* – How long to cache negative results

Cache issues

- **How do you know you can trust glue records?**

Cache poisoning

- **How do you know you can trust glue records?**
  - You can't really
- **I lied when saying forward lookups can check PTRs**
- **Consider the following attack:**
  - I connect to your server from 171.66.3.11, and serve you:
    `11.3.66.171.in-addr.arpa. IN NS www.berkeley.edu.`
    `www.berkeley.edu. 600 IN A 171.66.3.11`
  - Looks like `www.berkeley.edu.` is name server for PTR
  - Therefore, you must use glue record I supply you with
- **For a long time BIND wouldn't fix problem**
  - Probably worried about decreased cache efficiency

DNS poisoning in the wild

- **January 2005, the domain name for a large New York ISP, Panix, was hijacked to a site in Australia.**
- **In November 2004, Google and Amazon users were sent to Med Network Inc., an online pharmacy**
- **In March 2003, a group dubbed the "Freedom Cyber Force Militia" hijacked visitors to the Al-Jazeera Web site and presented them with the message "God Bless Our Troops"**

TXT records

- **Can place arbitrary text in DNS**

  ***name* [TTL] [IN] TXT "*text*" ...**

  - *text* – whatever you want it to mean
- **Great for prototyping new services**
  - Don't need to change DNS infrastructure
- **Example:** `dig aol.com txt`
  - What's this? SPF = "sender permitted from"
  - SPF specifies IP addresses allowed to send mail from `@aol.com`
  - Allows for low-security whitelisting
  - Nice for whitelisting because attacks like DNS poisoning and Joncheray may be too hard for spammers to do at high rates
  - But doesn't directly address spam problem

Same Origin Principle revisited

- **Recall Same Origin Principle for Java/Javascript**
  - Can only connect to server
- **“Origin” defined in terms of server name in URL**
- **Can you see a problem?**

Exploiting DNS to violate S.O.