



Application Layer – DNS

Lec 12

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g.,
www.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
 - note: core Internet function, implemented as application-layer protocol
 - complexity at network’s “edge”

DNS



DNS services

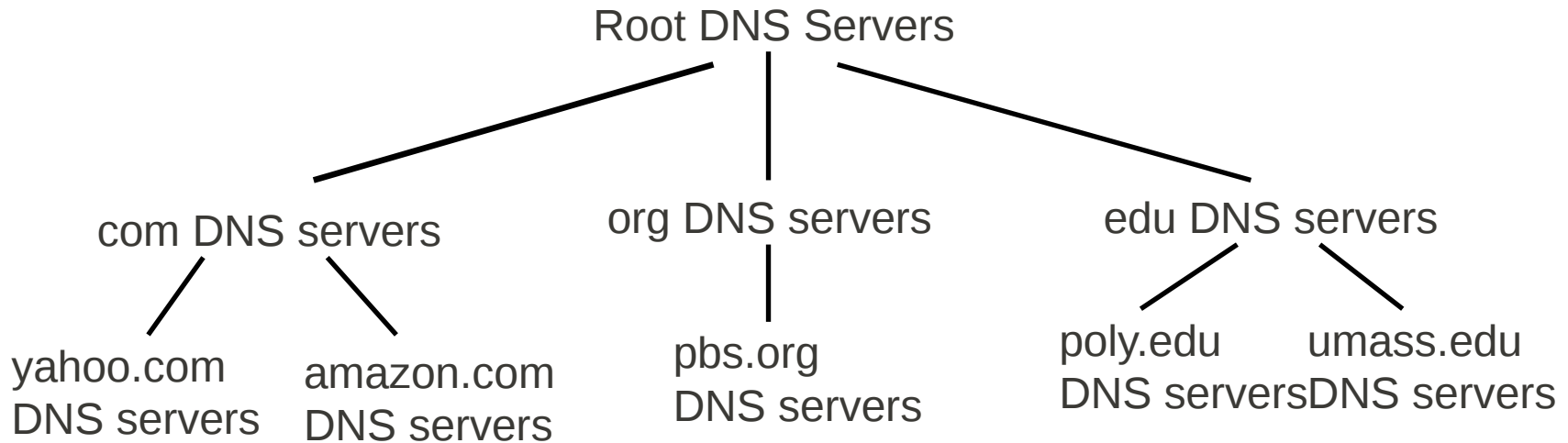
- Hostname to IP address translation
- Host aliasing
 - Canonical and alias names
- Mail server aliasing
- Load distribution
 - Replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

doesn't *scale*!

Distributed, Hierarchical Database

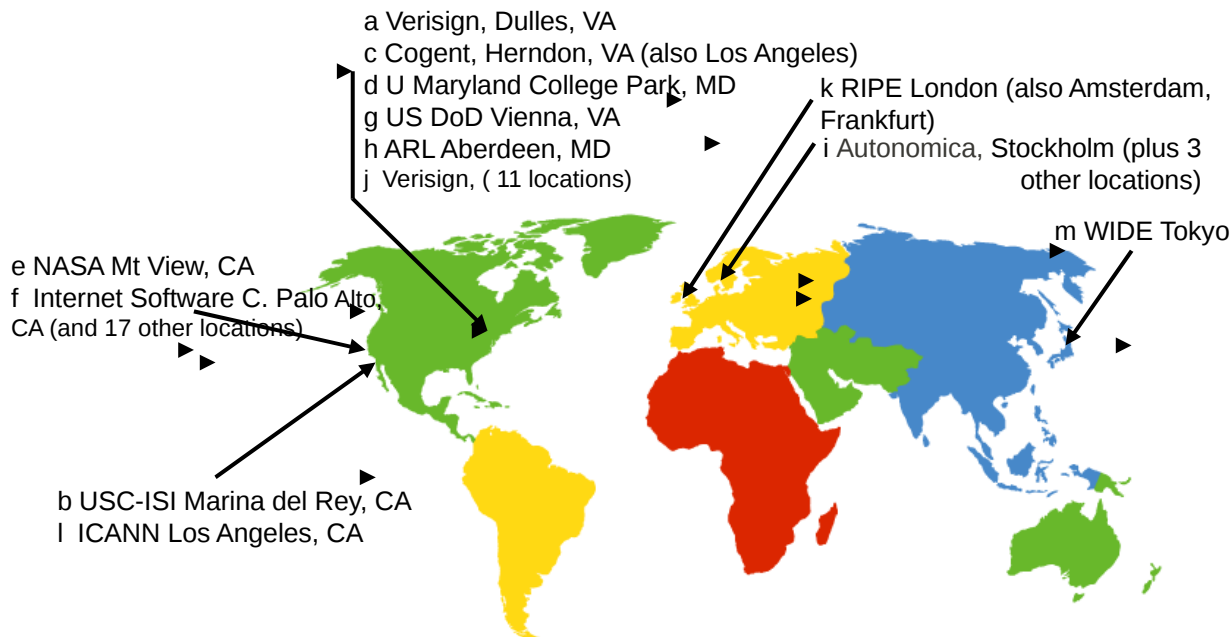


Client wants IP for www.amazon.com; 1st approx:

- Client queries a root server to find com DNS server
- Client queries com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: Root name servers

- contacted by local name server that can not resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server



13 root name servers worldwide



TLD and Authoritative Servers

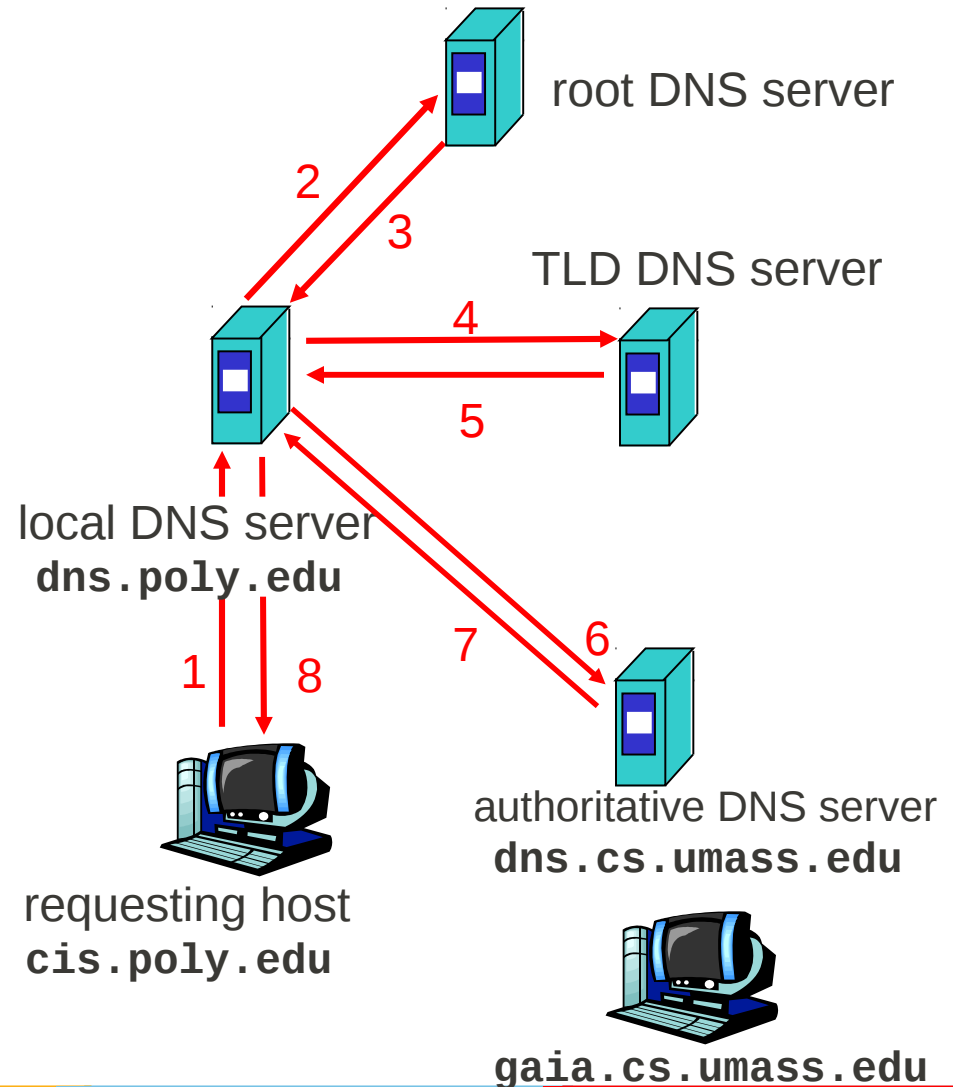
- **Top-level domain (TLD) servers:** responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
 - Network solutions maintains servers for com TLD
 - Educause for edu TLD
- **Authoritative DNS servers:** organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).
 - Can be maintained by organization or service provider

Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
 - Also called “default name server”
- When a host makes a DNS query, query is sent to its local DNS server
 - Acts as a proxy, forwards query into hierarchy.

Example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu



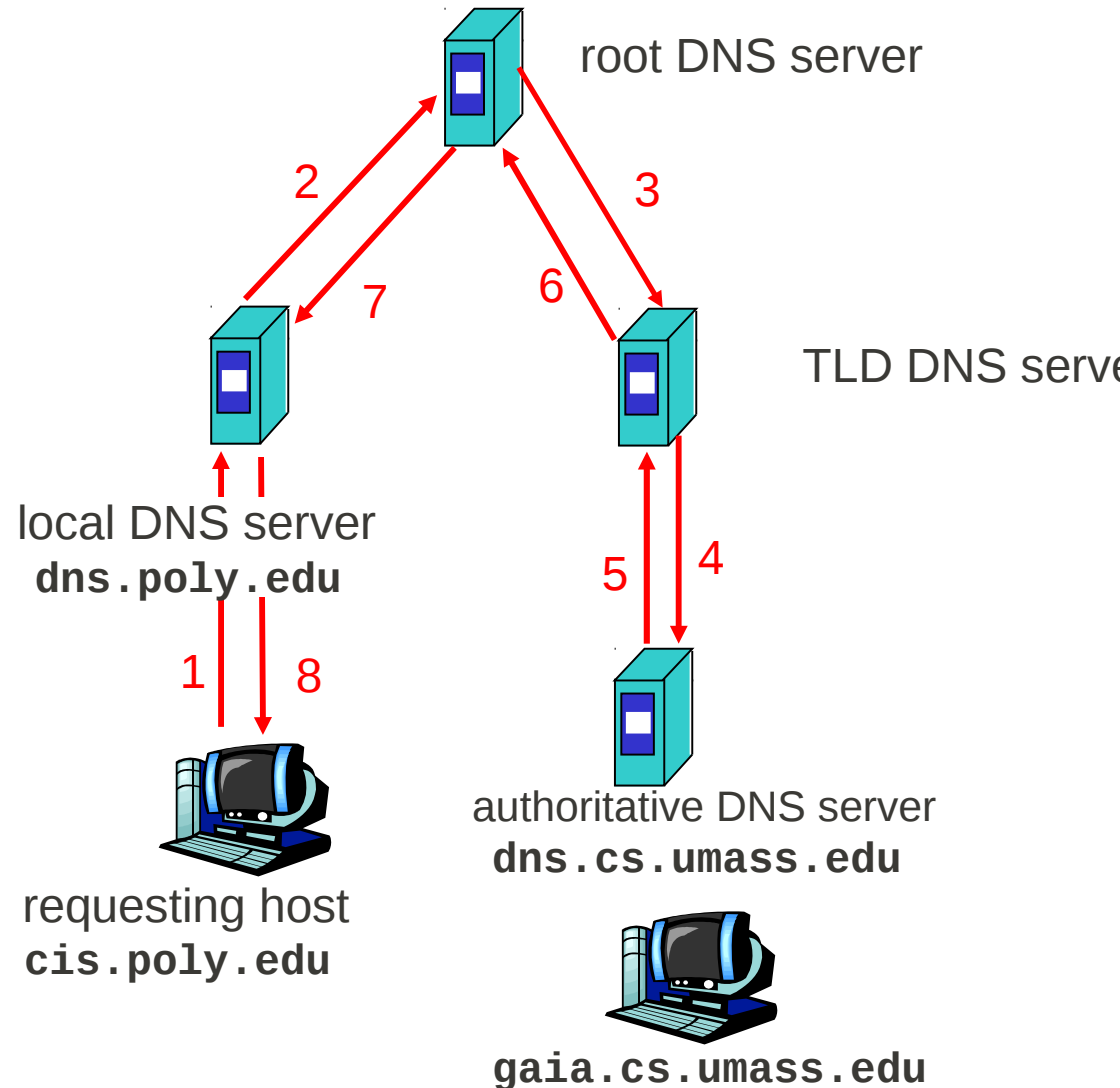
Recursive queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”





DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
 - cache entries timeout (disappear) after some time
 - TLD servers typically cached in local name servers
 - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
 - RFC 2136
 - <http://www.ietf.org/html.charters/dnsind-charter.html>

DNS records

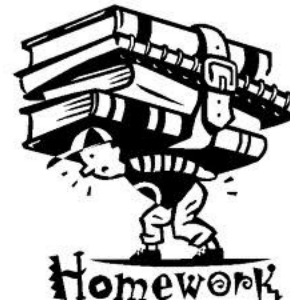
DNS: distributed db storing resource records (RR)

RR format: (**name**, **value**, **type**, **ttl**)

- Type=A
 - ❖ **name** is hostname
 - ❖ **value** is IP address
- Type=NS
 - **name** is domain (e.g. foo.com)
 - **value** is hostname of authoritative name server for this domain
- Type=CNAME
 - ❖ **name** is alias name for some “canonical” (the real) name
www.ibm.com is really servereast.backup2.ibm.com
 - ❖ **value** is canonical name
- Type=MX
 - ❖ **value** is name of mailserver associated with **name**

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*





Transport Layer

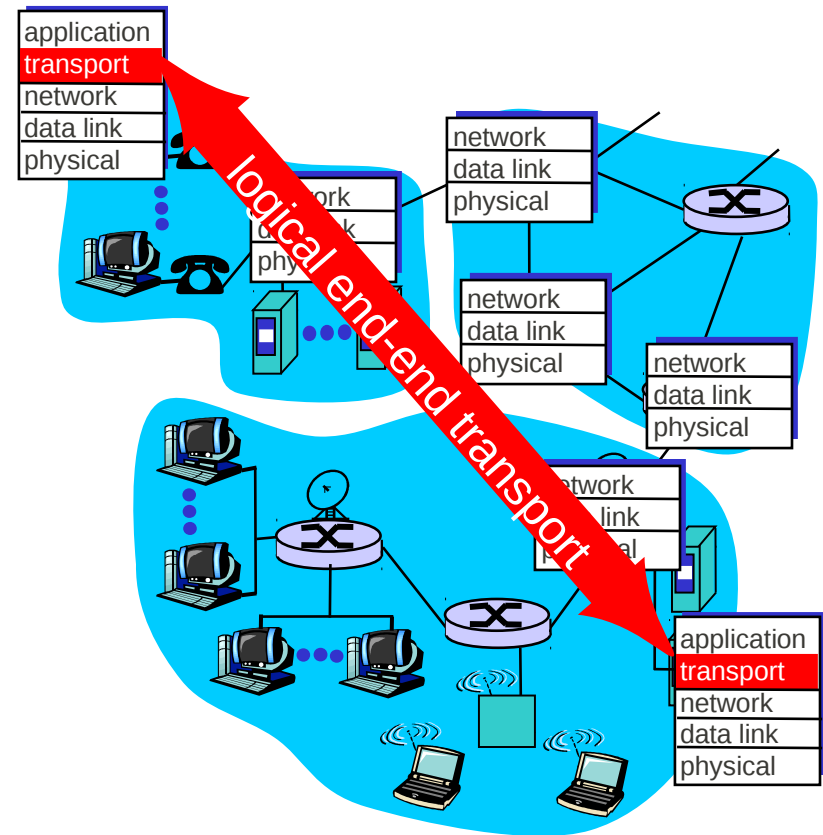
Transport Layer

Our goals:

- understand principles behind transport layer services:
 - multiplexing/demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- learn about transport layer protocols in the Internet:
 - UDP: connectionless transport
 - TCP: connection-oriented transport

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
 - send side: breaks app messages into **segments**, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
 - Internet: TCP and UDP



Transport vs. network layer

- *network layer*: logical communication between hosts
- *transport layer*: logical communication between processes
 - relies on, enhances, network layer services

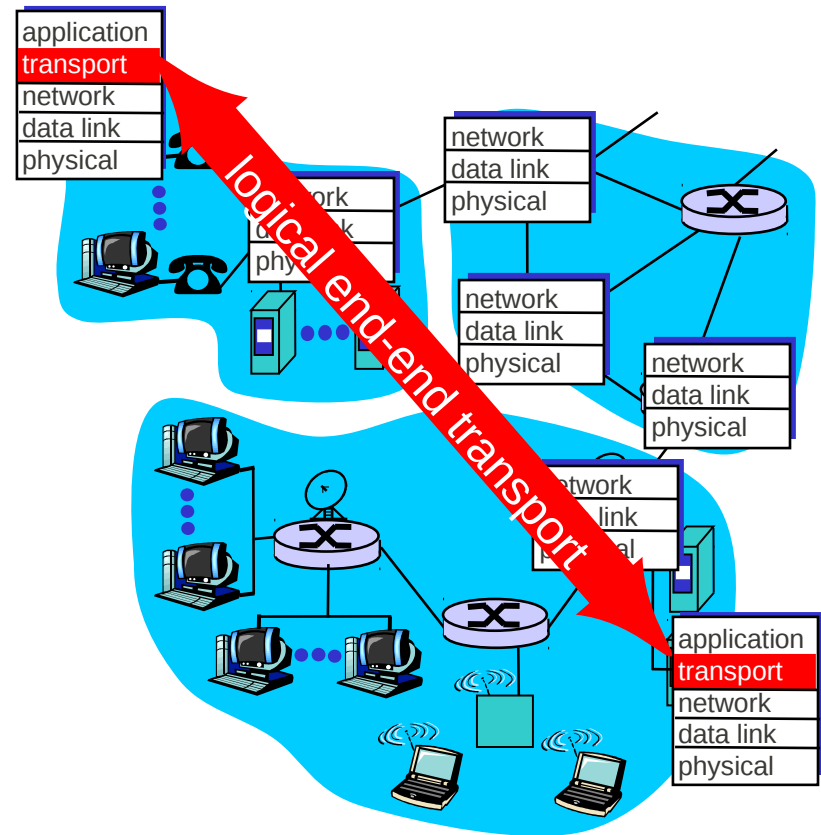
Household analogy:

12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- unreliable, unordered delivery: UDP
 - no-frills extension of “best-effort” IP
- services not available:
 - delay guarantees
 - bandwidth guarantees



Multiplexing and demultiplexing


Multiplexing/demultiplexing


Demultiplexing at rcv host:

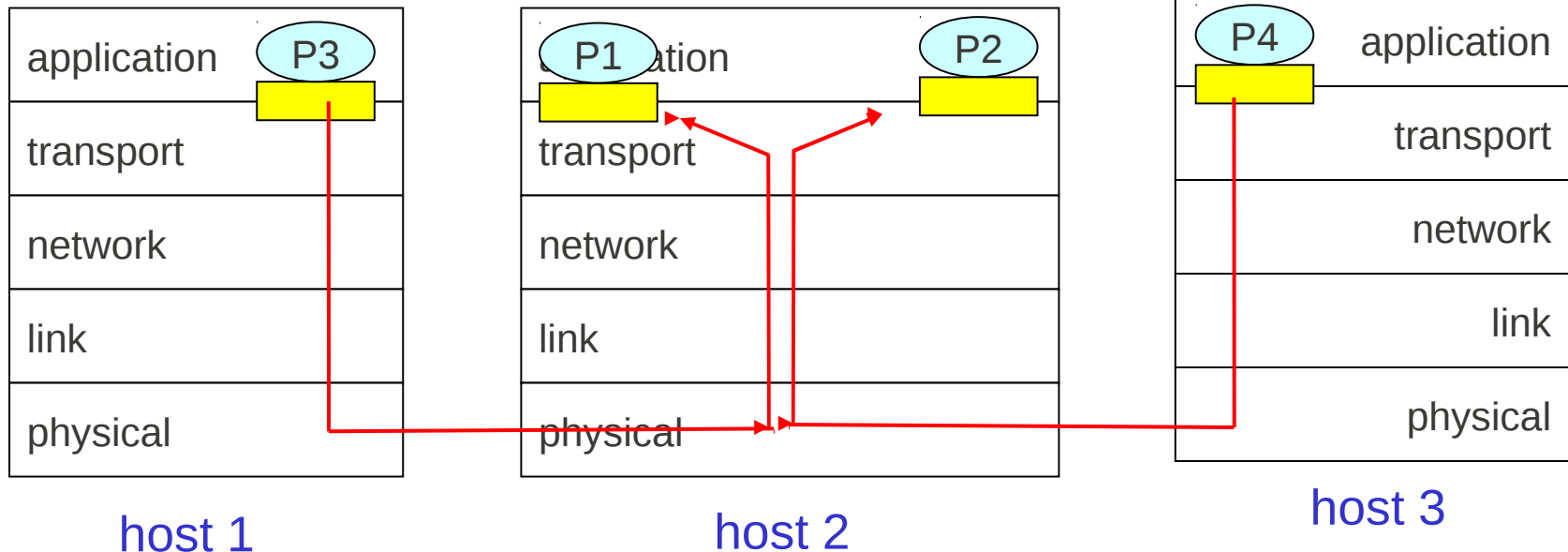
delivering received segments
to correct socket

Multiplexing at send host:

gathering data from multiple
sockets, enveloping data with
header (later used for
demultiplexing)

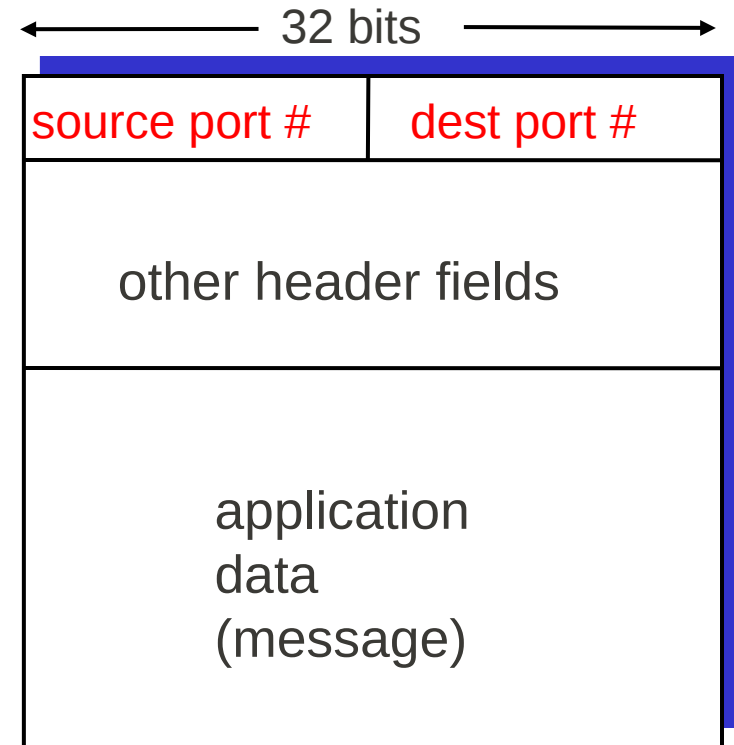
 = socket

 = process



How demultiplexing works

- host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries 1 transport-layer segment
 - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new  
    DatagramSocket(12534);
```

```
DatagramSocket mySocket2 = new  
    DatagramSocket(12535);
```

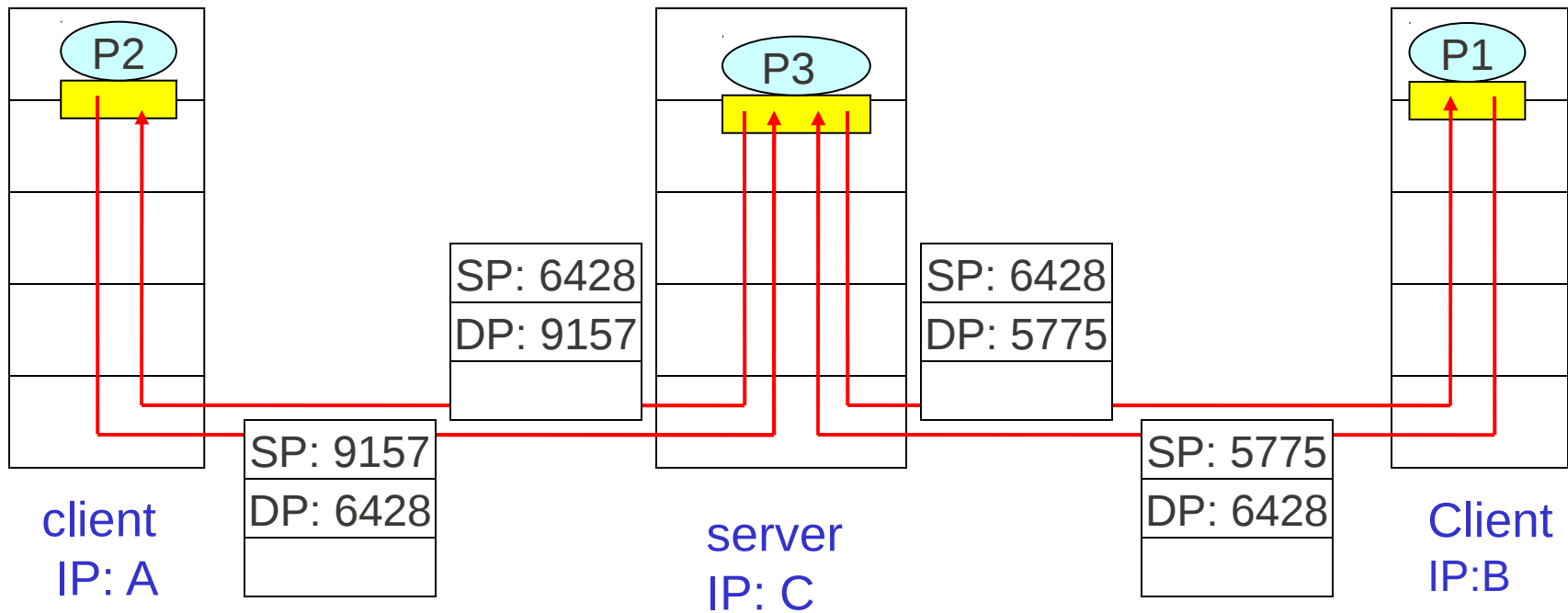
- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
 - checks destination port number in segment
 - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

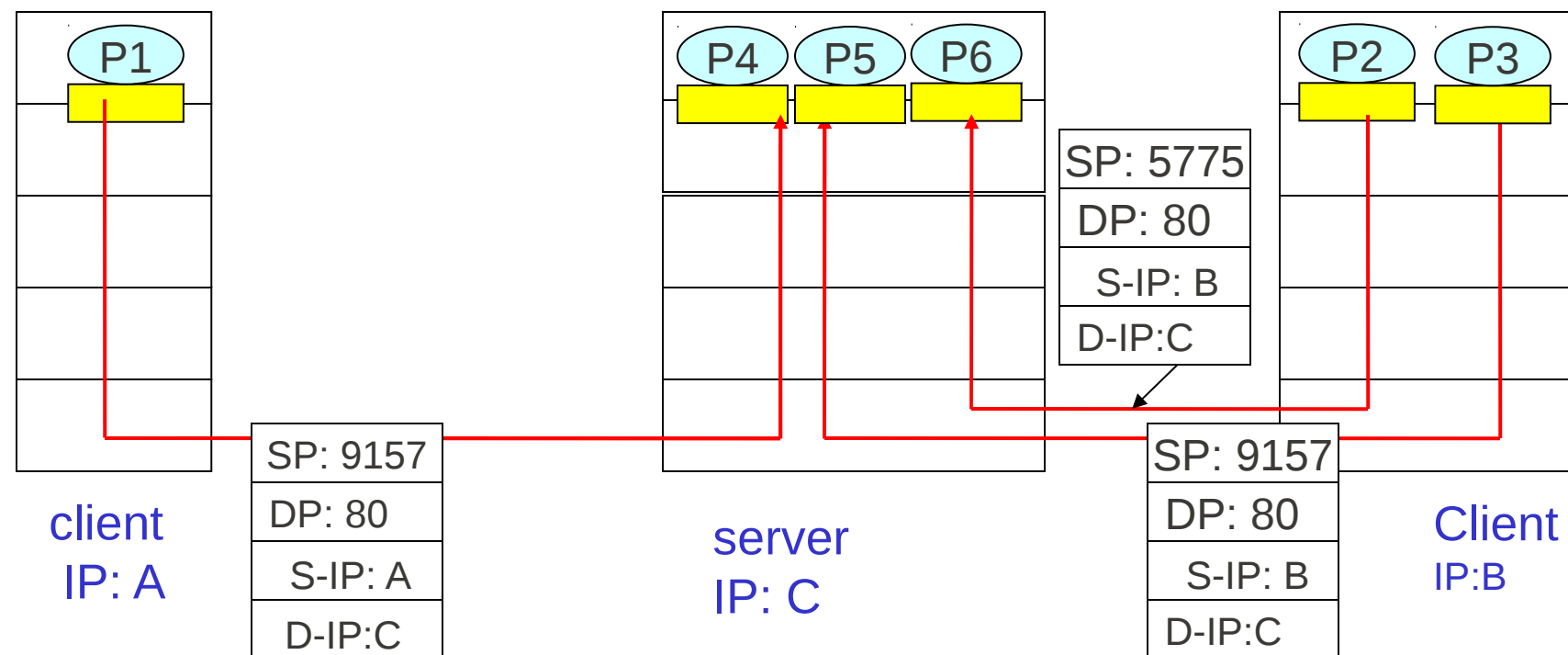


SP provides “return address”

Connection-oriented demux

- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client

Connection-oriented demux (cont)



Connectionless transport: UDP

UDP: User Datagram Protocol [RFC 768]

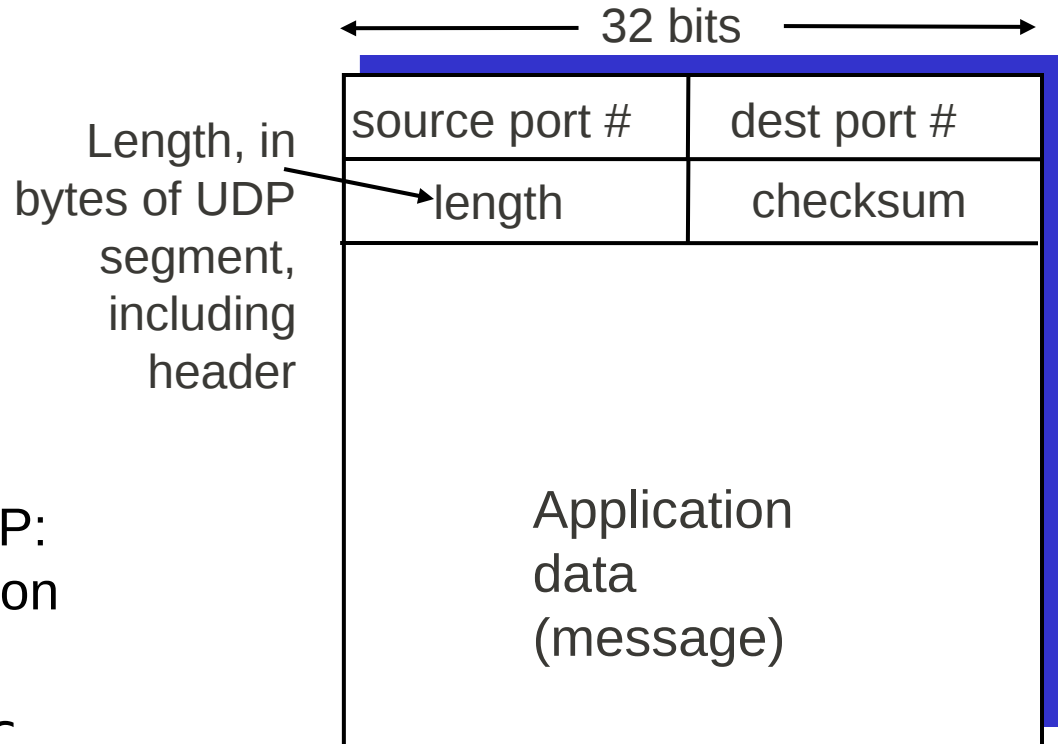
- “no frills,” “bare bones”
Internet transport protocol
- “best effort” service, UDP segments may be:
 - lost
 - delivered out of order to app
- *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: more

- often used for streaming multimedia apps
 - loss tolerant
 - rate sensitive
- other UDP uses
 - DNS
 - SNMP
- reliable transfer over UDP: add reliability at application layer
 - application-specific error recovery!



UDP segment format

UDP checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
 - check if computed checksum equals checksum field value:
 - NO - error detected
 - YES - no error detected.
But maybe errors nonetheless? More later
-

Internet Checksum Example

- Note
 - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
------------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1