EN.601.414/614 Computer Networks

Midterm Review

Xin Jin

Spring 2019 (MW 3:00-4:15pm in Shaffer 301)

JOHNS HOPKINS UNIVERSITY

https://github.com/xinjin/course-net

Midterm Exam

- **Time: 3-4:15pm March 13 (this Wednesday)**
- **Location: Shaffer 301**
- **Form: Closed-book**
  - Can bring one A4/letter paper with notes on both sides
  - Can bring a calculator
  - Anything else is prohibited
- **Senior Option**
  - Your final exam score will be the same as your midterm exam score

This review

- **Walk through what you're expected to know at this point: key topics, important aspects of each**
- **Not covered in review does NOT imply you don't need to know it**
  - ➢ But if it's covered today, you should know it
- **Summarize, not explain**
  - ➢ Stop me when you want to discuss something further!

Topics

- **Basics (lectures 1–3)**
- **Application layer (lectures 4, 5)**
  - HTTP, DNS, and CDN
- **Transport layer (lectures 6–9)**
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- **Network layer (lectures 10)**
  - Data plane

Basic concepts

- **You should know:**
  - ➢ Packet vs. circuit switching
  - ➢ Statistical multiplexing
  - ➢ Link characteristics
  - ➢ Packet delays

How are network resources shared?

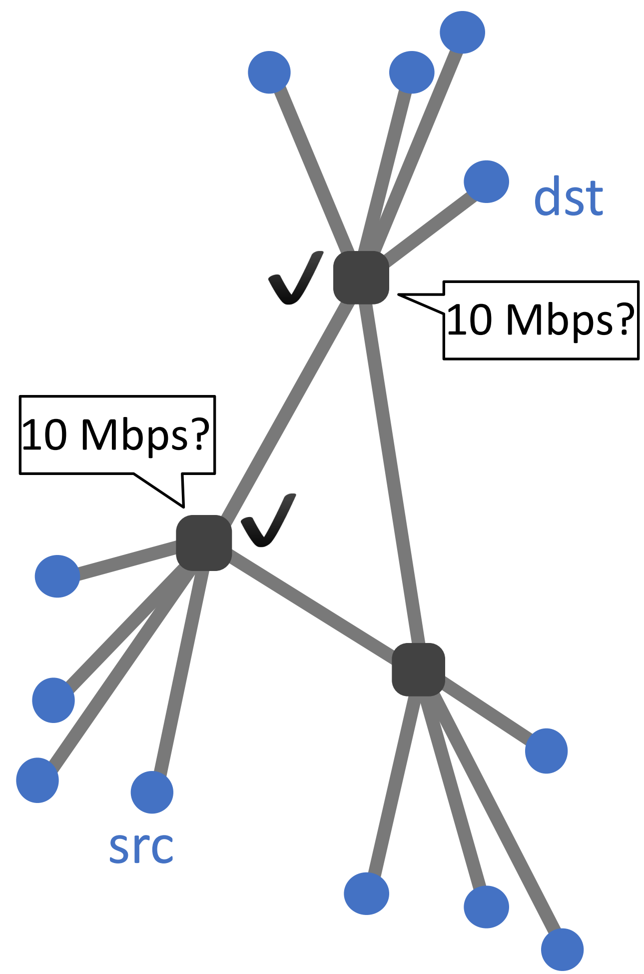
- **Two approaches**
  - ➢Reservations → circuit switching
  - ➢On-demand → packet switching

Two approaches to sharing

- **Packet switching**
  - Network resources consumed on demand per-packet
  - Admission control: per packet
- **Circuit switching**
  - Network resources reserved a priori at “connection” initiation
  - Admission control: per connection

Circuit switching

1. src sends reservation request to dst
2. Switches create circuit *after* admission control
3. src sends data
4. src sends teardown request



Packet switching

- **Data is sent as chunks of formatted bits (Packets)**
- **Packets consist of a “header” and “payload”**
- **Switches “forward” packets based on their headers**
- **Each packet travels independently**
- **No link resources are reserved in advance**

Statistical multiplexing

- **Allowing more demands than the network can handle**
  - Hoping that not all demands are required at the same time
  - Good for bursty traffic (average << peak demand)
  - Packet switching exploits statistical multiplexing better than circuit switching

Performance metrics

- **Delay**
- **Loss**
- **Throughput**

A network link

- **Link bandwidth**
  - Number of bits sent/received per unit time (bits/sec or bps)
- **Propagation delay**
  - Time for one bit to move through the link (seconds)
- **Bandwidth-Delay Product (BDP)**
  - Number of bits "in flight" at any time
- **BDP = bandwidth × propagation delay**

Delay

- **Consists of four components**
  - Transmission delay
  - Propagation delay
  - Queuing delay
  - Processing delay

due to link properties

due to traffic mix and switch internals

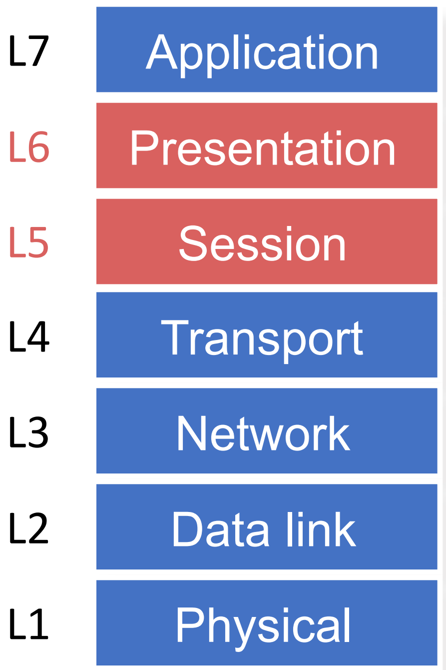
End-to-end delay

transmission
propagation
queueing
processing
transmission
propagation
queueing
processing
transmission
propagation

OSI layers

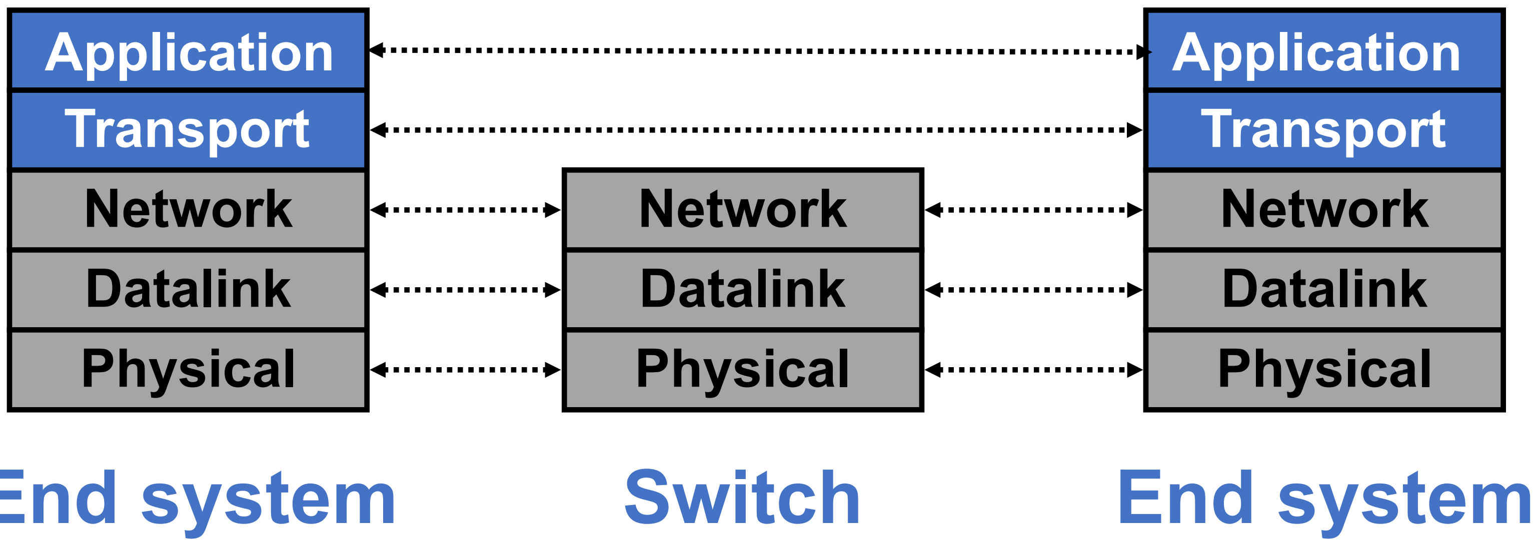
- **OSI stands for Open Systems Interconnection model**
  - ➢ Developed by the ISO

- **Session and presentation layers are often implemented as part of the application layer**

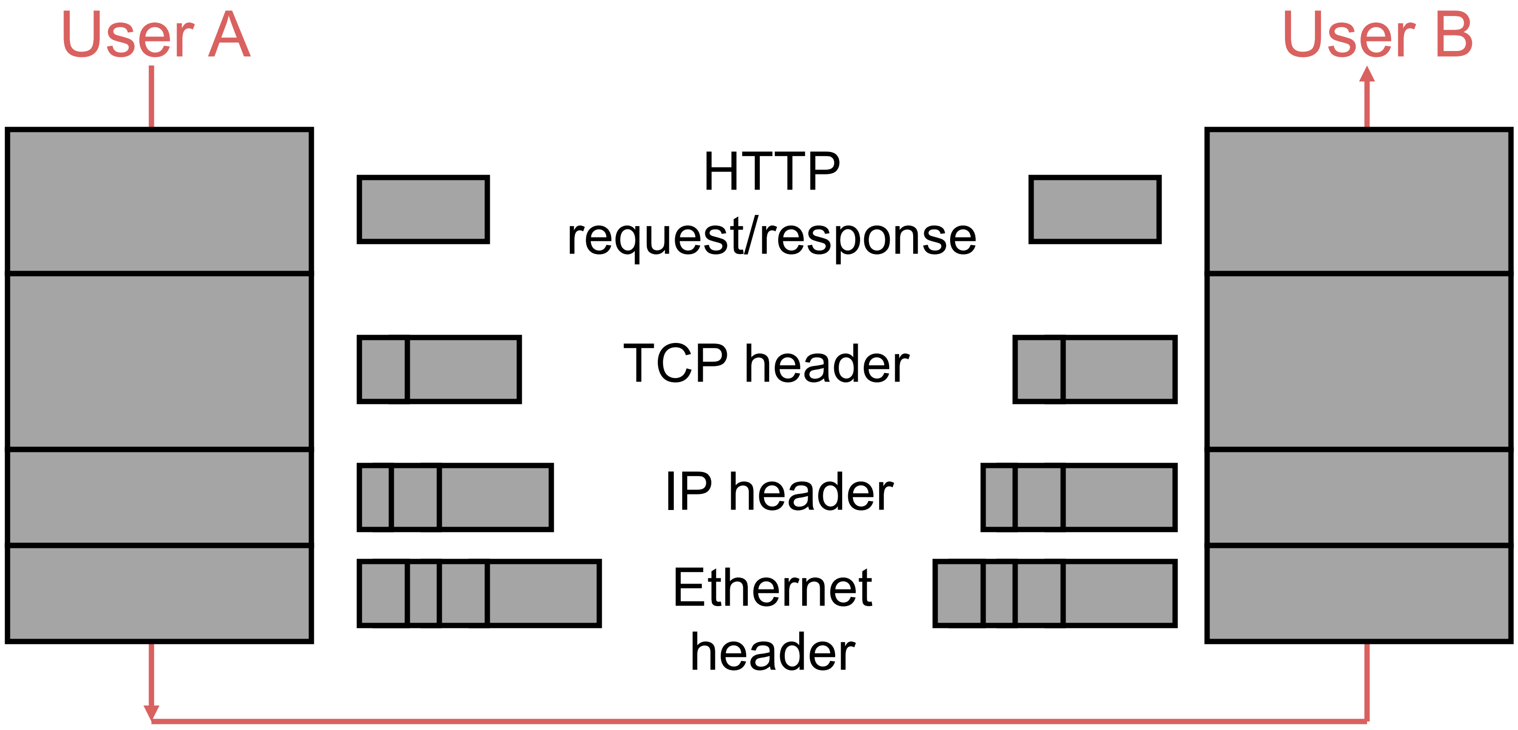


Layers in practice

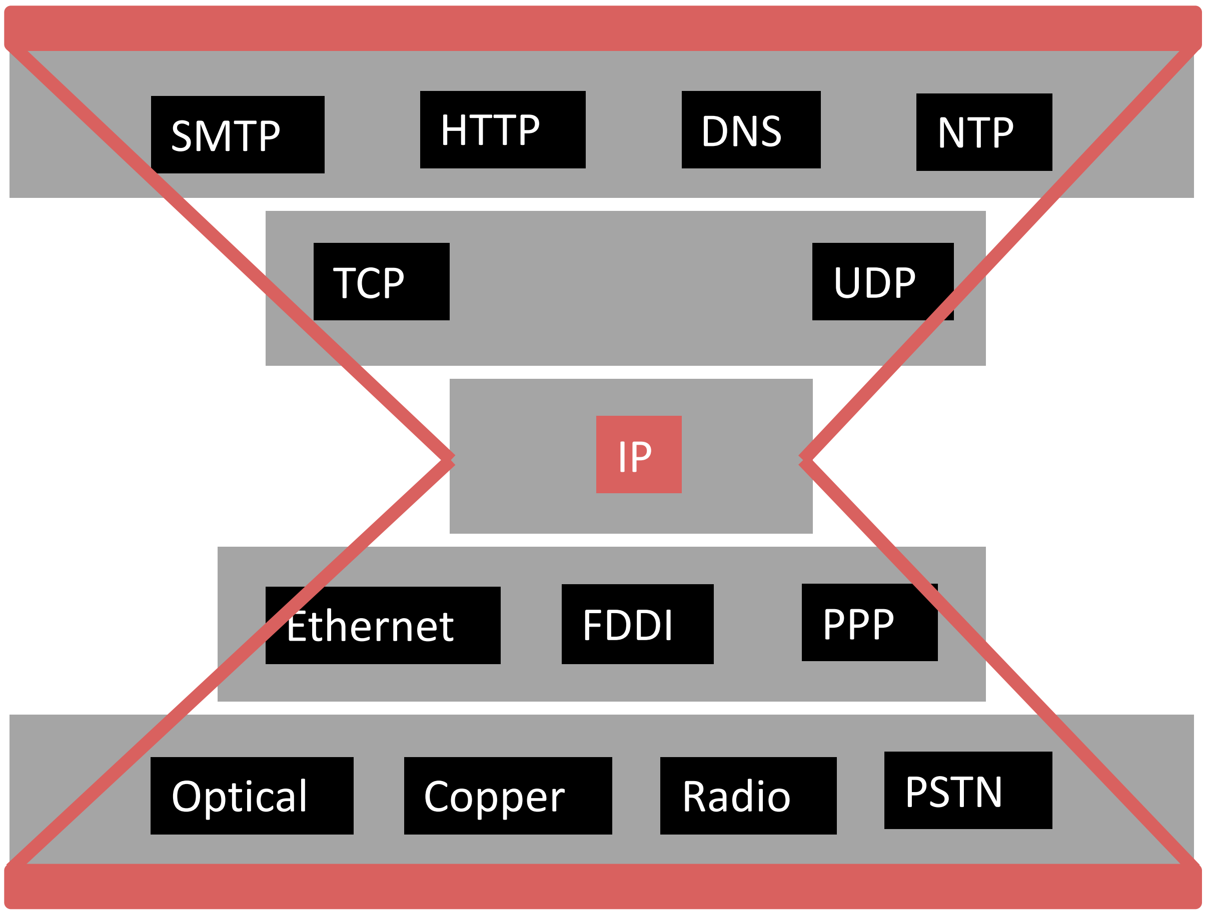
- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



Layer encapsulation: Protocol headers



IP is the narrow waist of the layering hourglass



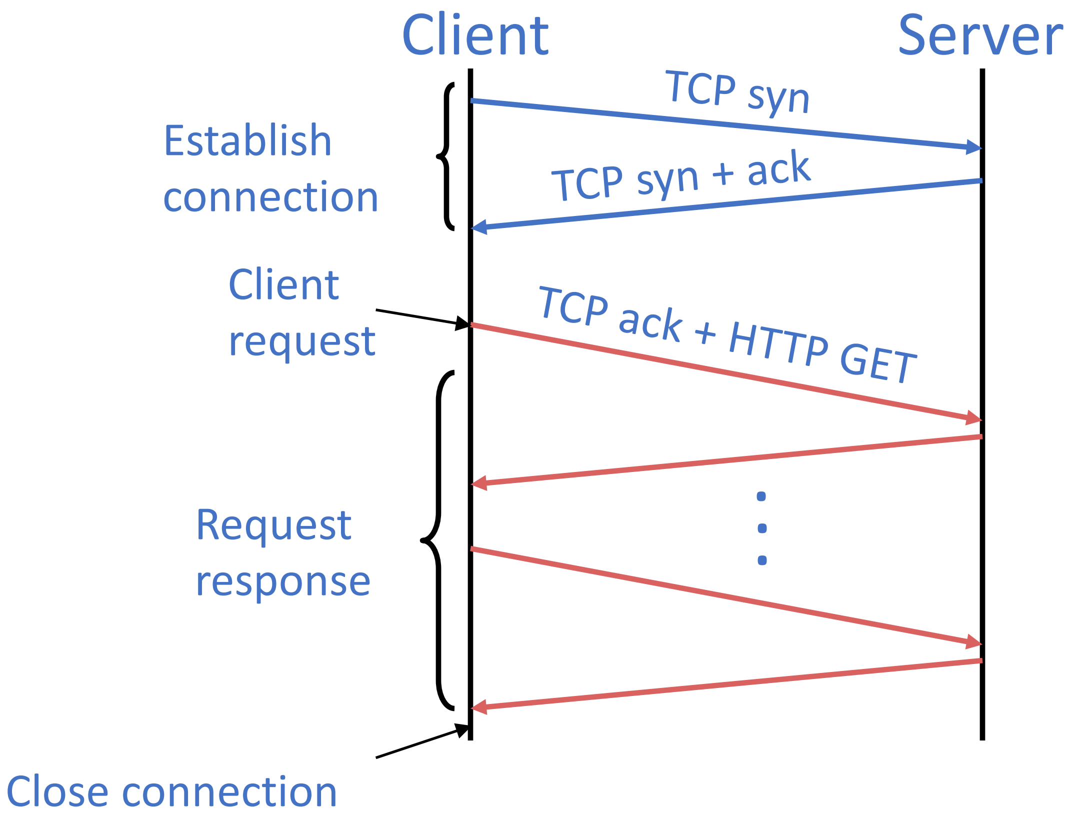
Topics

- **Basics (lectures 1–3)**
- **Application layer (lectures 4, 5)**
  - ➢HTTP, DNS, and CDN
- **Transport layer (lectures 6–9)**
  - ➢UDP vs. TCP
  - ➢TCP details: reliability and flow control
  - ➢TCP congestion control: general concepts only
- **Network layer (lectures 10)**
  - ➢Data plane

Hyper Text Transfer Protocol (HTTP)

- **Client-server architecture**
  - ➢Server is “always on” and “well known”
  - ➢Clients initiate contact to server
- **Synchronous request/reply protocol**
  - ➢Runs over TCP, Port 80
- **Stateless**
- **ASCII format**
  - ➢Before HTTP/2

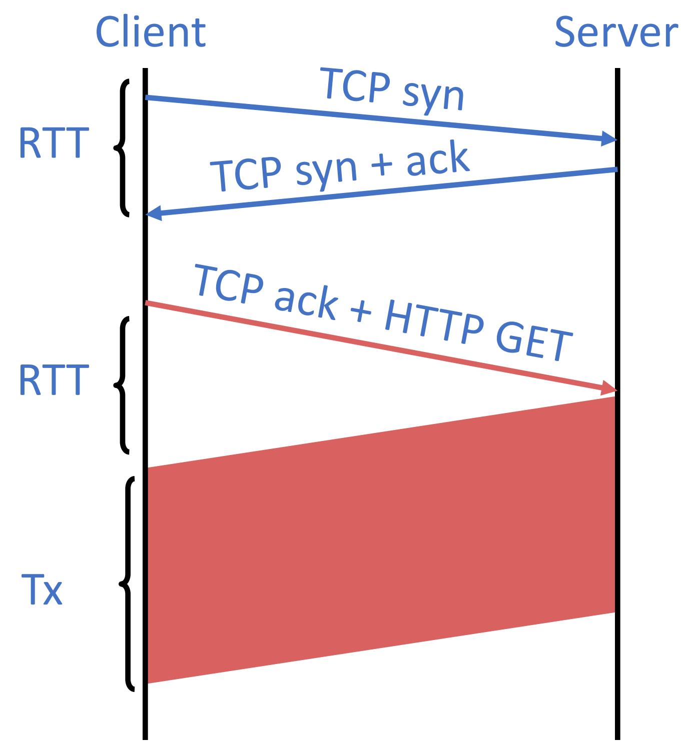
Steps in HTTP request/response



Object request response time

- **RTT (round-trip time)**
  - Time for a small packet to travel from client to server and back

- **Response time**
  - 1 RTT for TCP setup
  - 1 RTT for HTTP request and first few bytes
  - Transmission time
  - Total = 2RTT + Transmission Time



Improving HTTP performance

- **Optimizing connections using three "P"s**
  - Persistent connections
  - Parallel/concurrent connections
  - Pipelined transfers over the same connection
- **Caching**
  - Forward proxy: close to clients
  - Reverse proxy: close to servers
- **Replication**

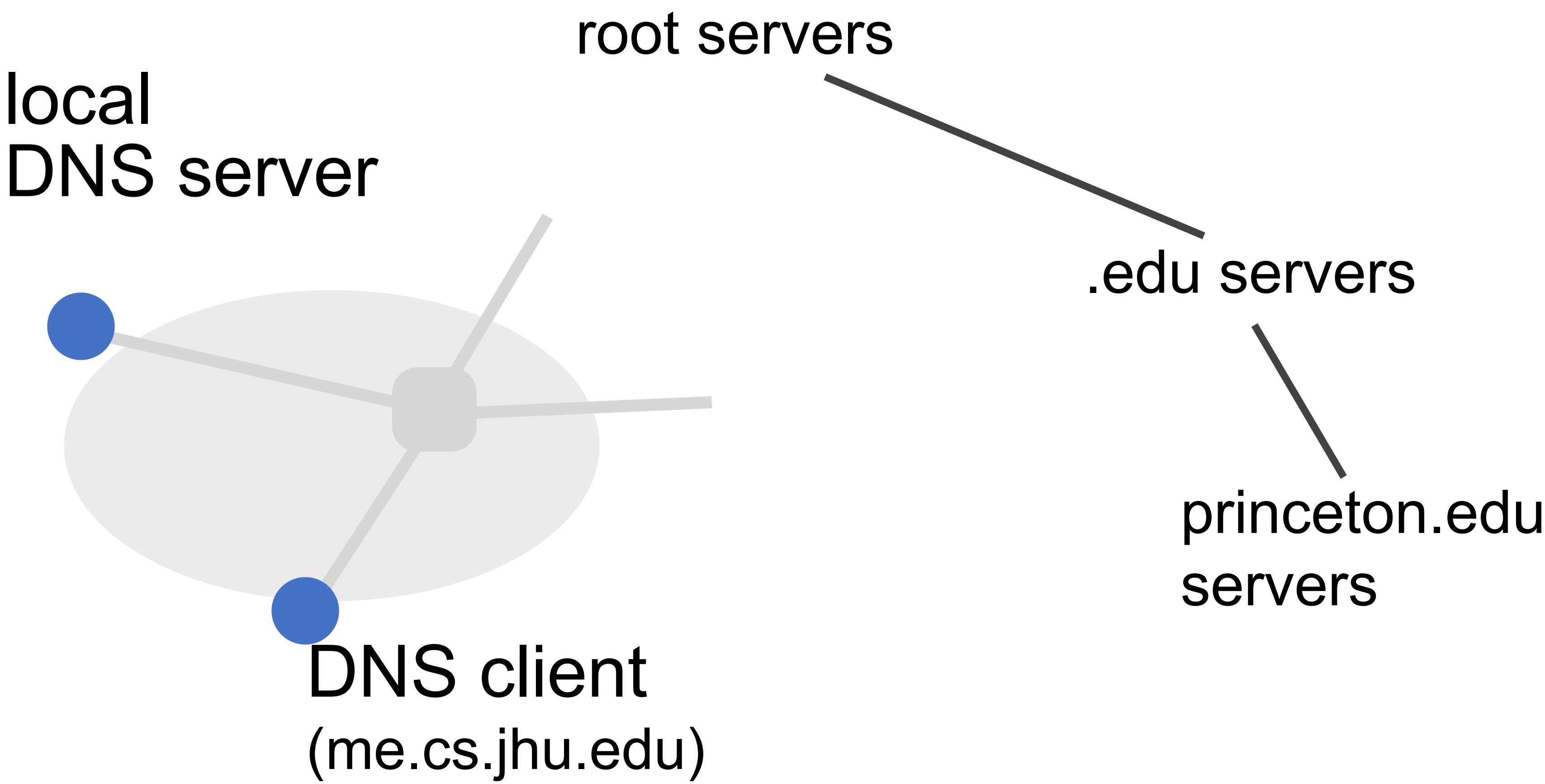
Content Distribution Networks (CDN)

- **Caching and replication as a service**
- **Combination of caching and replication**
  - Pull: Direct result of clients' requests (caching)
  - Push: Expectation of high access rate (replication)

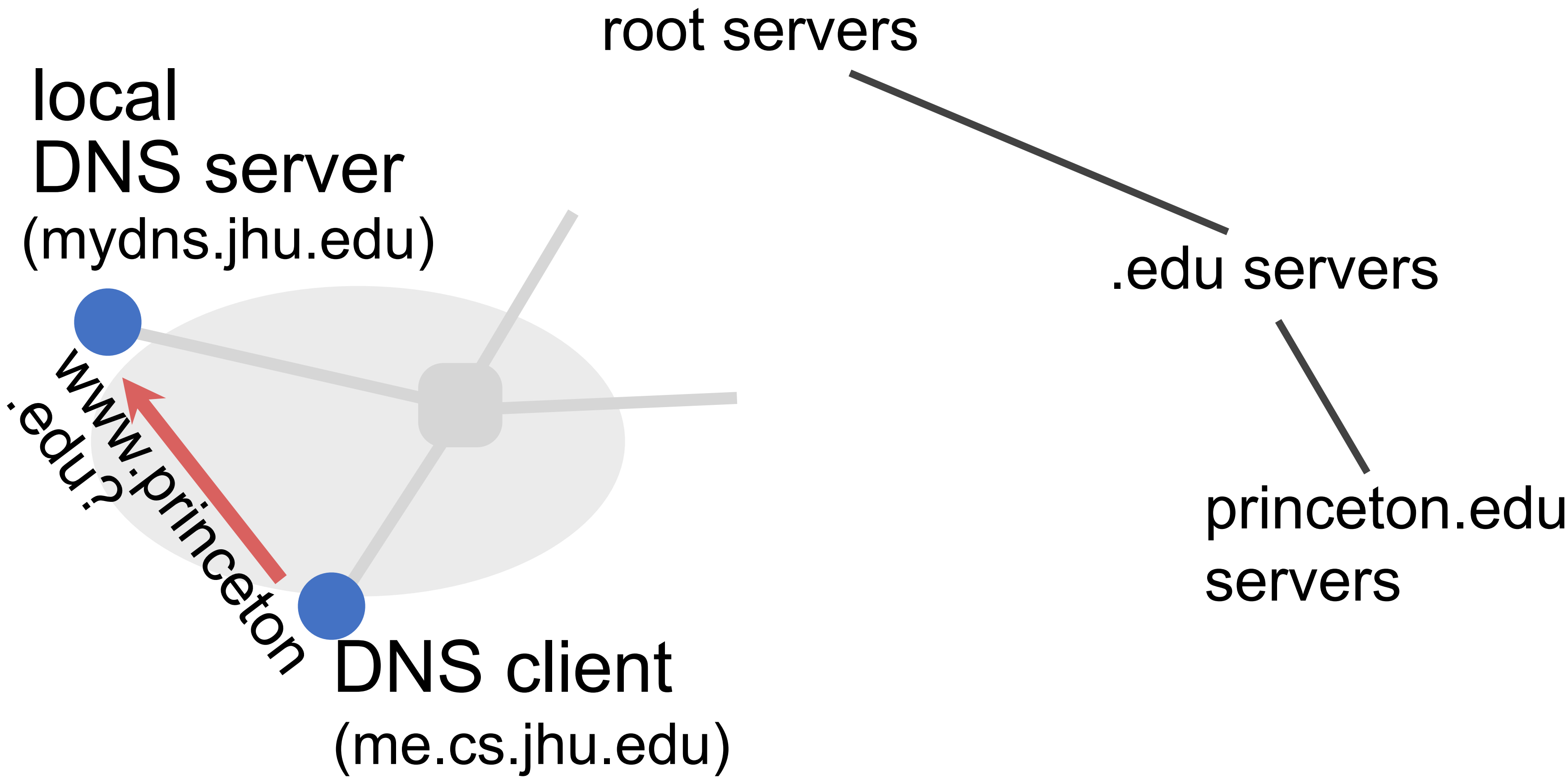
Hierarchies in the DNS

- **Three intertwined hierarchies**
  - ➢ Hierarchical namespace
    - As opposed to original flat namespace
  - ➢ Hierarchically administered
    - As opposed to centralized
  - ➢ (Distributed) hierarchy of servers
    - As opposed to centralized storage

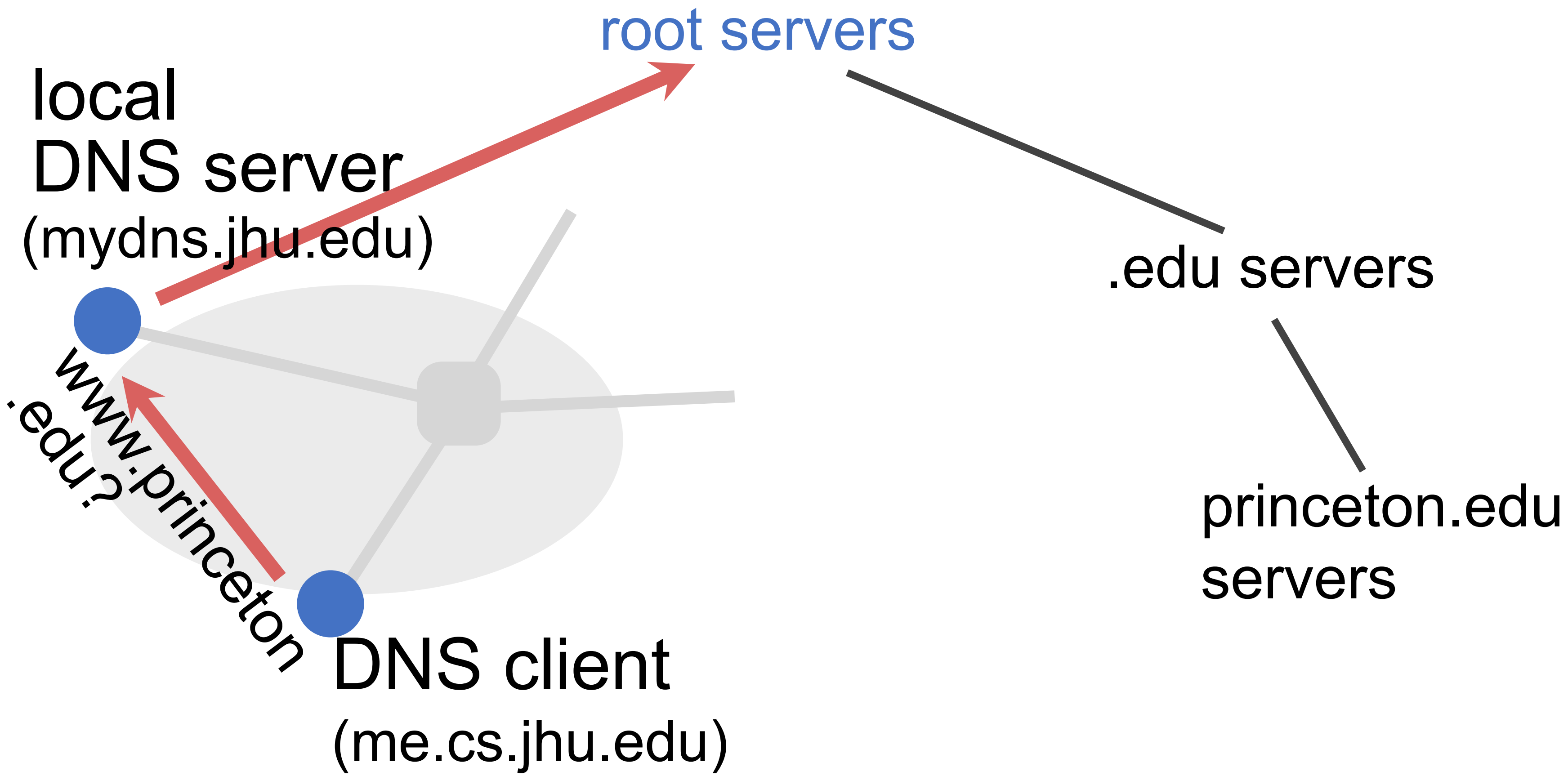
Name resolution



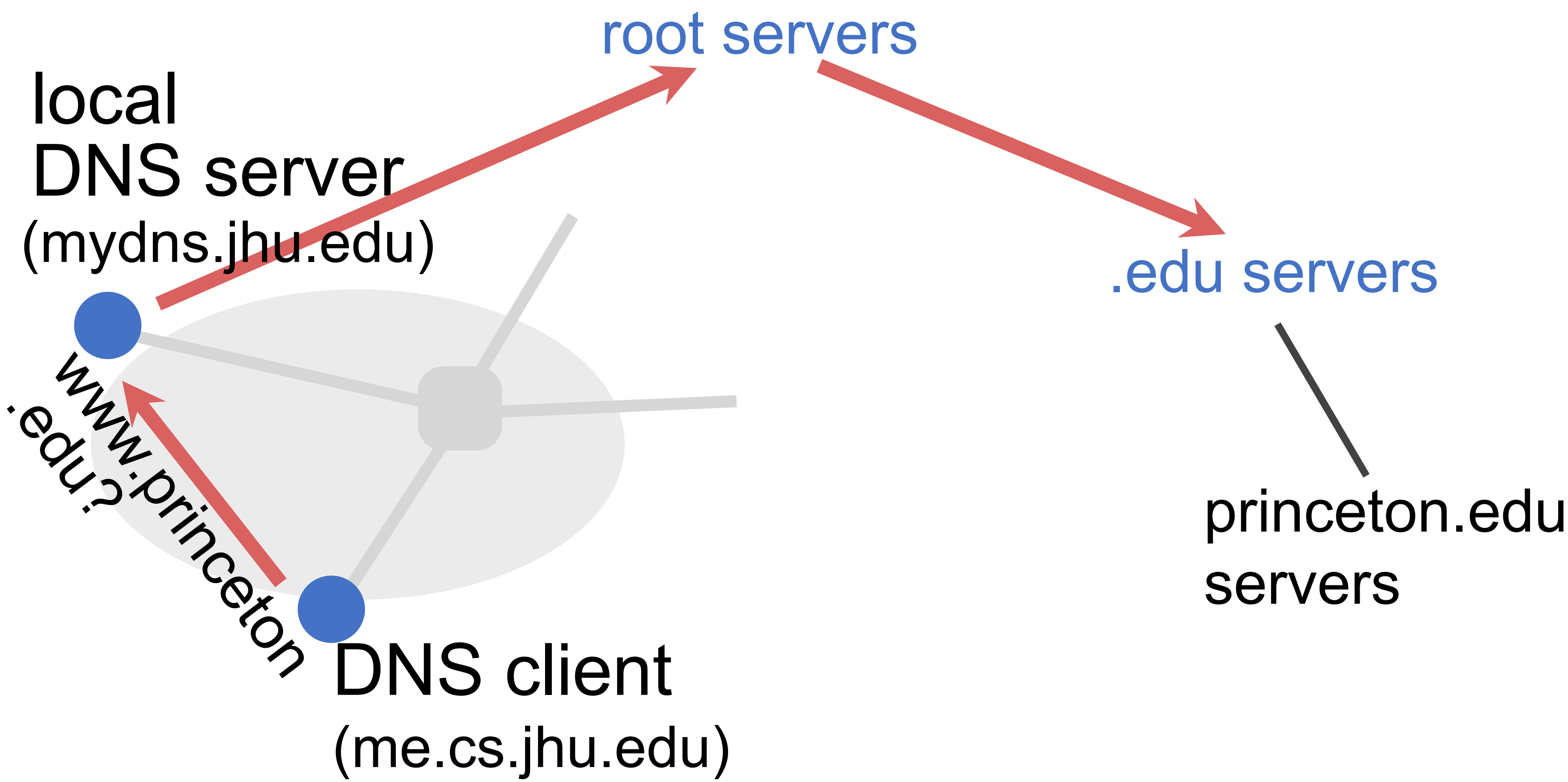
Name resolution



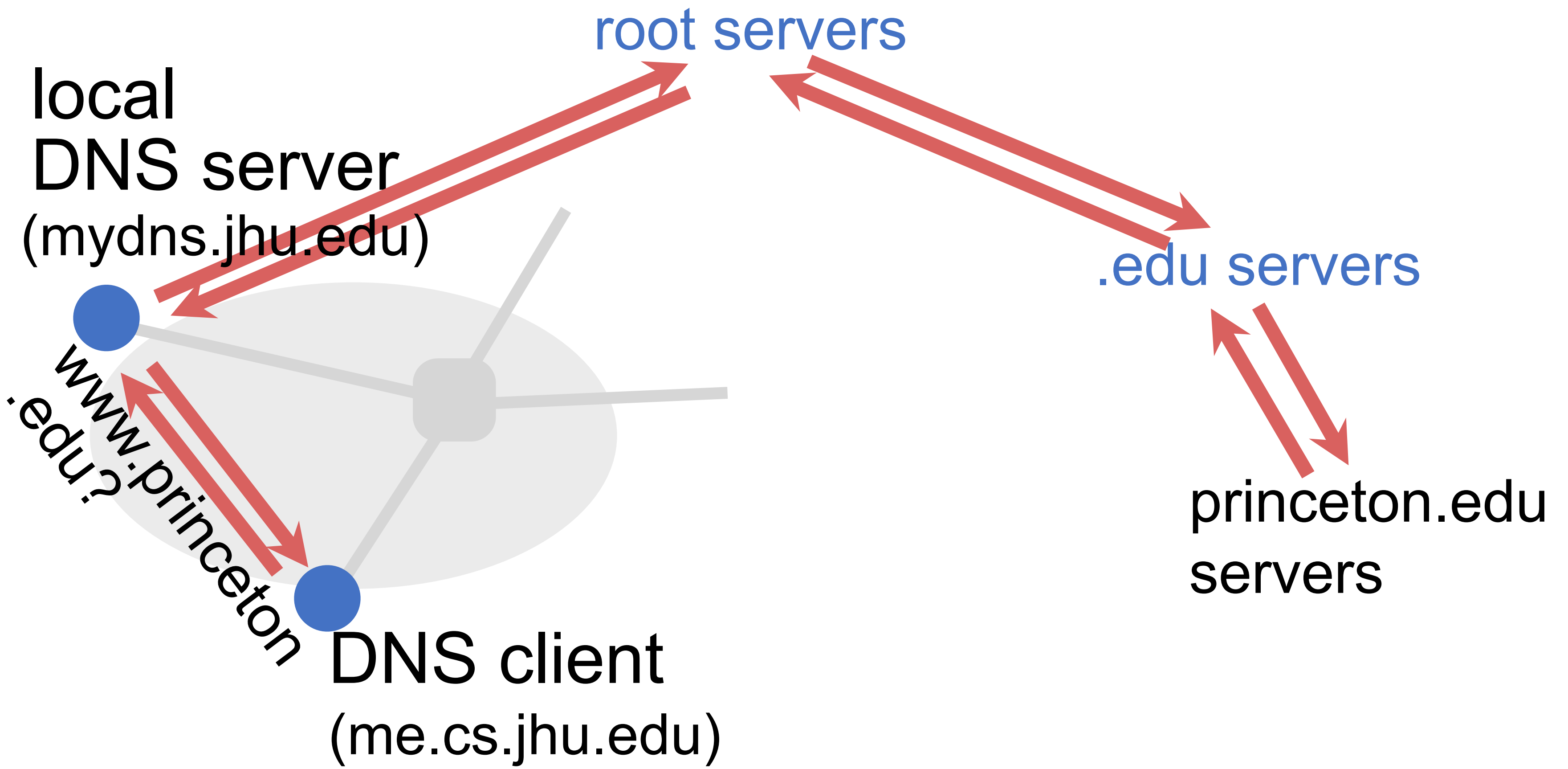
Name resolution



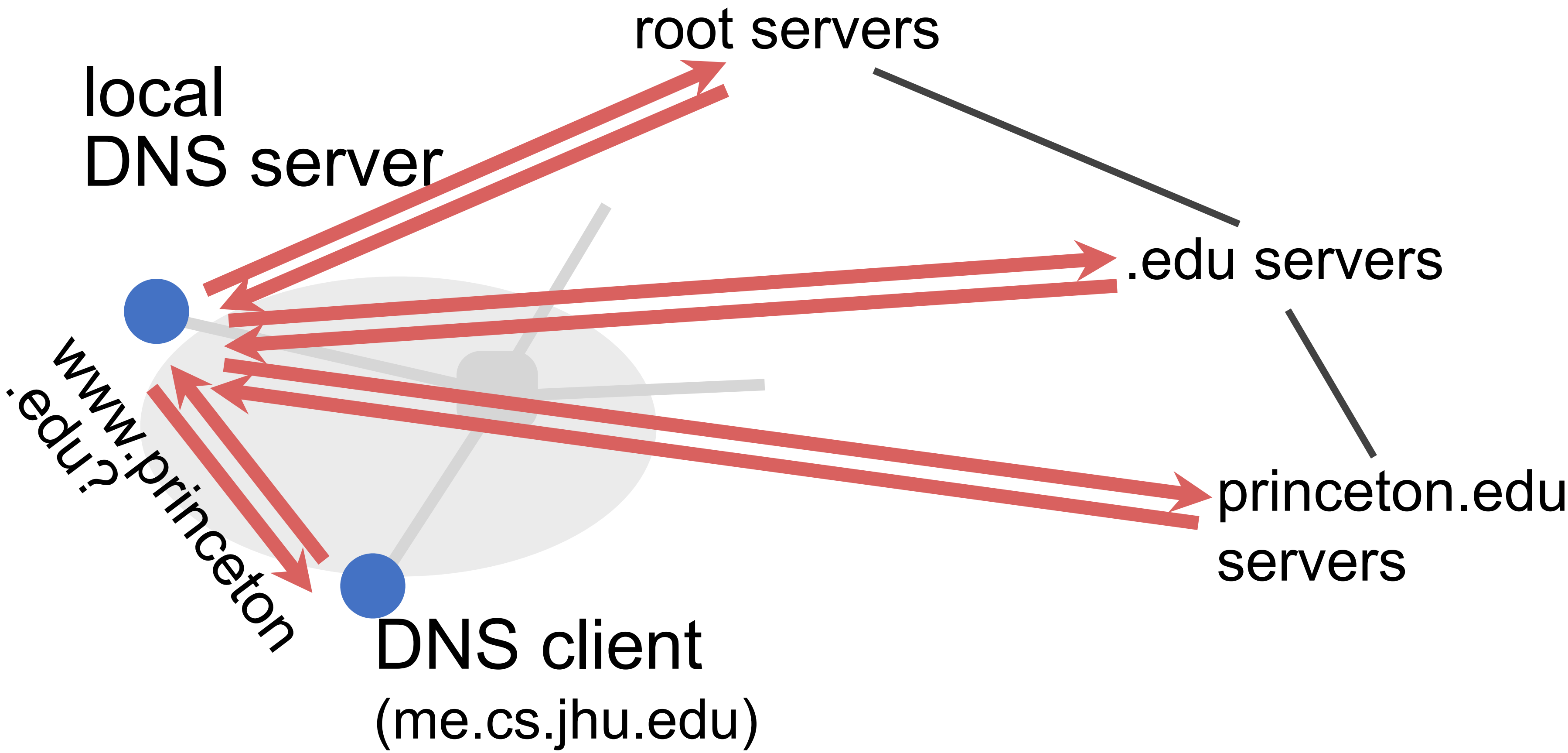
Name resolution



Name resolution: Recursive



Name resolution: Iterative



DNS caching

- **Performing all these queries takes time**
  - Up to 1-second latency before starting download
- **Caching can greatly reduce overhead**
  - The top-level servers very rarely change
  - Popular sites (e.g., www.cnn.com) visited often
  - Local DNS server often has the information cached
- **How DNS caching works**
  - DNS servers cache responses to queries
  - Responses include a “time to live” (TTL) field
  - Server deletes cached entry after TTL expires

Topics

- **Basics (lectures 1–3)**
- **Application layer (lectures 4, 5)**
  - HTTP, DNS, and CDN
- **Transport layer (lectures 6–9)**
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- **Network layer (lectures 10)**
  - Data plane

Role of the transport layer

- **(1) Communication between application processes**
  - Mux and demux from/to application processes
  - Implemented using ports
- **(2) Provide common end-to-end services for app layer**
  - Reliable, in-order data delivery
  - Well-paced data delivery

UDP vs. TCP

- Both UDP and TCP perform mux/demux via ports

	UDP	TCP
Data abstraction	Packets (datagrams)	Stream of bytes of arbitrary length
Service	Best-effort (same as IP)	•Reliability •In-order delivery •Congestion control •Flow control

Reliable transport: General concepts

- **Checksums (for error detection)**
- **Timers (for loss detection)**
- **Acknowledgments (feedback from receiver)**
  - Cumulative: “received everything up to X”
  - Selective: “received X”
- **Sequence no (detect duplicates, accounting)**
- **Sliding windows (for efficiency)**

You should know:

- what these concepts are
- why they exist
- how TCP uses them

Designing a reliable transport protocol

- **Stop and wait is correct but inefficient**
  - Works packet by packet (of size DATA)
  - Throughput is (DATA/ RTT)
- **Sliding window: use pipelining to increase throughput**
  - n packets at a time results in higher throughput
  - MIN(n*DATA/RTT, Link Bandwidth)

The TCP abstraction

- **TCP delivers a reliable, in-order, byte stream**
- **Reliable: TCP resends lost packets (recursively)**
    - Until it gives up and shuts down connection
- **In-order: TCP only hands consecutive chunks of data to application**
- **Byte stream: TCP assumes there is an incoming stream of data, and attempts to deliver it to app**

Things to know about TCP

- **How TCP achieves reliability**
- **RTT estimation**
- **Connection establishment/teardown**
- **Flow Control**
- **Congestion Control**

- **For each, know how the functionality is implemented and why it is needed**

Reliability

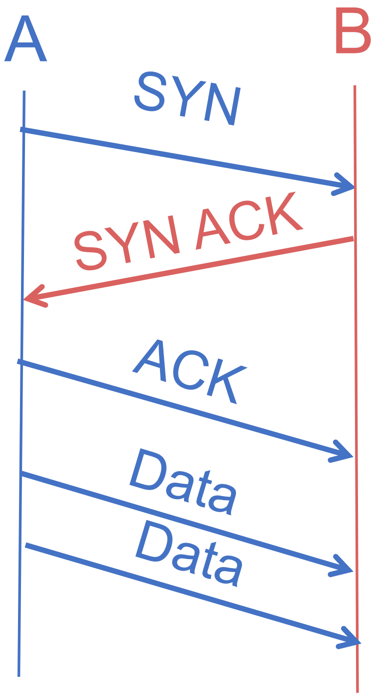
- **Having TCP take care of it simplifies application development**
- **How**
    - ➢Checksums and timers (for error and loss detection)
    - ➢Fast retransmit (to detect faster-than-timeout loss)
    - ➢Cumulative ACKs (receiver feedback: what's lost?)
    - ➢Sliding windows (for efficiency)
    - ➢Buffers at sender (hold packets until ACKs arrive)
    - ➢Buffers at receiver (to reorder packets before delivery to application)

RTT estimation

- **TCP uses timeouts to retransmit packets**
  - ➢But RTT may vary (significantly!) for different reasons and on different timescales
    - due to temporary congestion
    - due to long-lived congestion
    - due to a change in routing paths
- **An incorrect RTT estimate might introduce spurious retransmissions or overly long delays**
- **Proposed solutions use EWMA, incorporate deviations**

Establishing a TCP connection

- **Three-way handshake to establish connection**
  - ➢ Host A sends a SYN (open; "synchronize sequence numbers") to host B
  - ➢ Host B returns a SYN acknowledgment (SYN ACK)
  - ➢ Host A sends an ACK to acknowledge the SYN ACK
- **Three-way handshake to terminate (normal operation)**



Flow control

- **Why?**
  - TCP at the receiver must buffer a packet until all packets before it (in byte-order) have arrived and the receiving application has consumed available bytes
  - Hence, receiver advances its window when the receiving application consumes data
  - Sender advances its window when new data ACK’d
  - Risk of sender over-runing the receiver’s buffers
- **How?**
  - “Advertised Window” field in TCP header

Congestion control

- **Why?**
  - ➢ Because the network itself can be the bottleneck
  - ➢ Should make efficient use of available network capacity
    - While sharing available capacity fairly with other flows
    - And adapting to changes in available capacity
- **How?**
  - ➢ Dynamically adapts the size of the sending window

Put together

- **Flow Control**
  - Restrict window to RWND to make sure that the receiver isn't overwhelmed
- **Congestion Control**
  - Restrict window to CWND to make sure that the network isn't overwhelmed
- **Together**
  - Restrict window to min{RWND, CWND} to make sure that neither the receiver nor the network are overwhelmed

CC implementation

- **States at sender**
  - CWND (initialized to a small constant)
  - ssthresh (initialized to a large constant)
  - dupACKcount and timer
- **Events**
  - ACK (new data)
  - dupACK (duplicate ACK for old data)
  - Timeout

Event: ACK (new data)

- **If CWND < ssthresh**
  - ➢CWND += 1

- *CWND packets per RTT*
- *Hence, after one RTT with no drops:*
  CWND = 2xCWND

Event: ACK (new data)

- **If CWND < ssthresh**
  - CWND += 1

Slow start phase

- **Else**
  - CWND = CWND + 1/CWND

Congestion avoidance phase

> - *CWND packets per RTT*
> - *Hence, after one RTT with no drops:*
>   *CWND = CWND + 1*

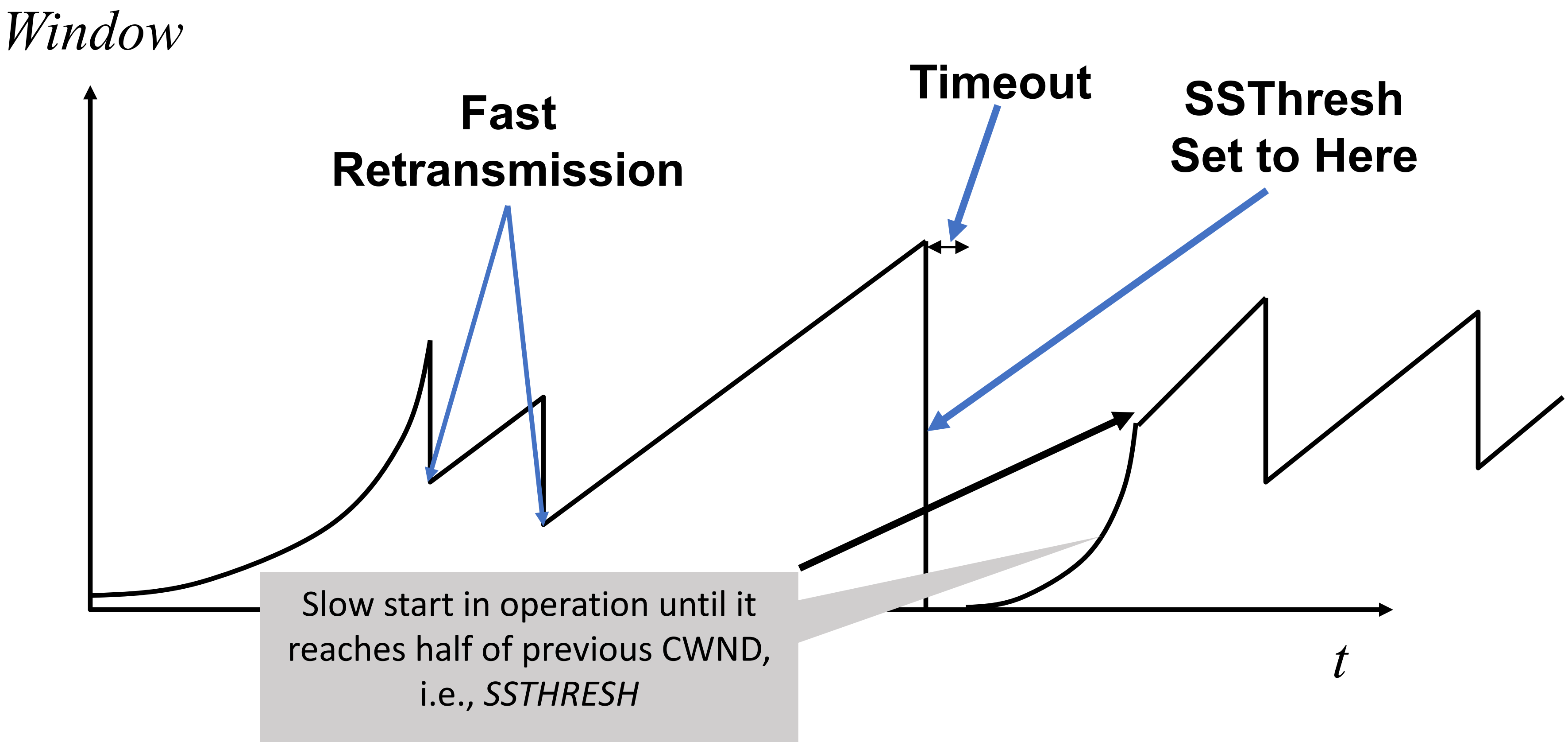
Event: TimeOut

- **On Timeout**
  - ssthresh ← CWND/2
  - CWND ← 1

Event: dupACK

- **dupACKcount ++**
- **If dupACKcount = 3 /* fast retransmit */**
  - ssthresh = CWND/2
  - CWND = CWND/2

Example



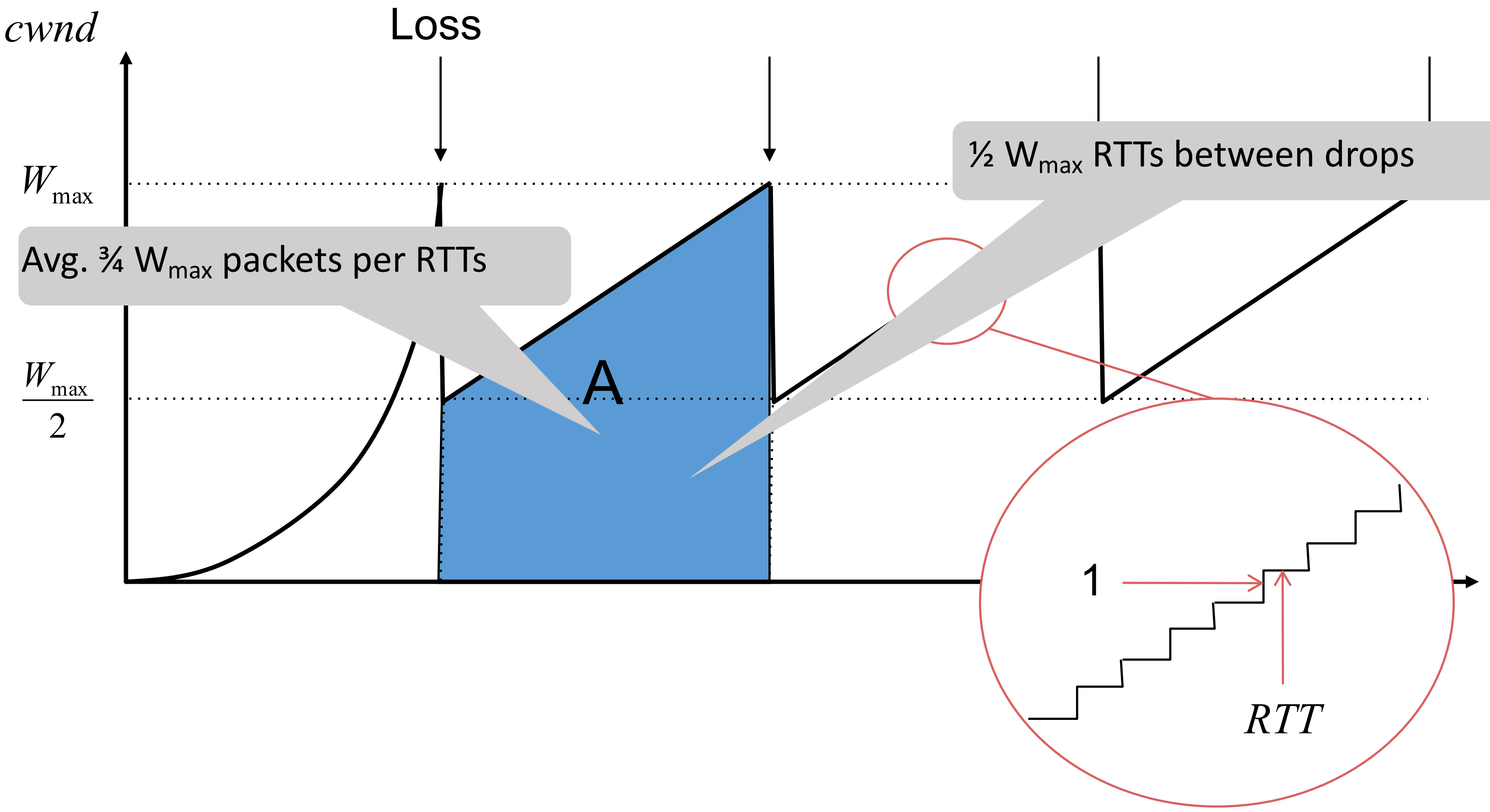
Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

TCP flavors

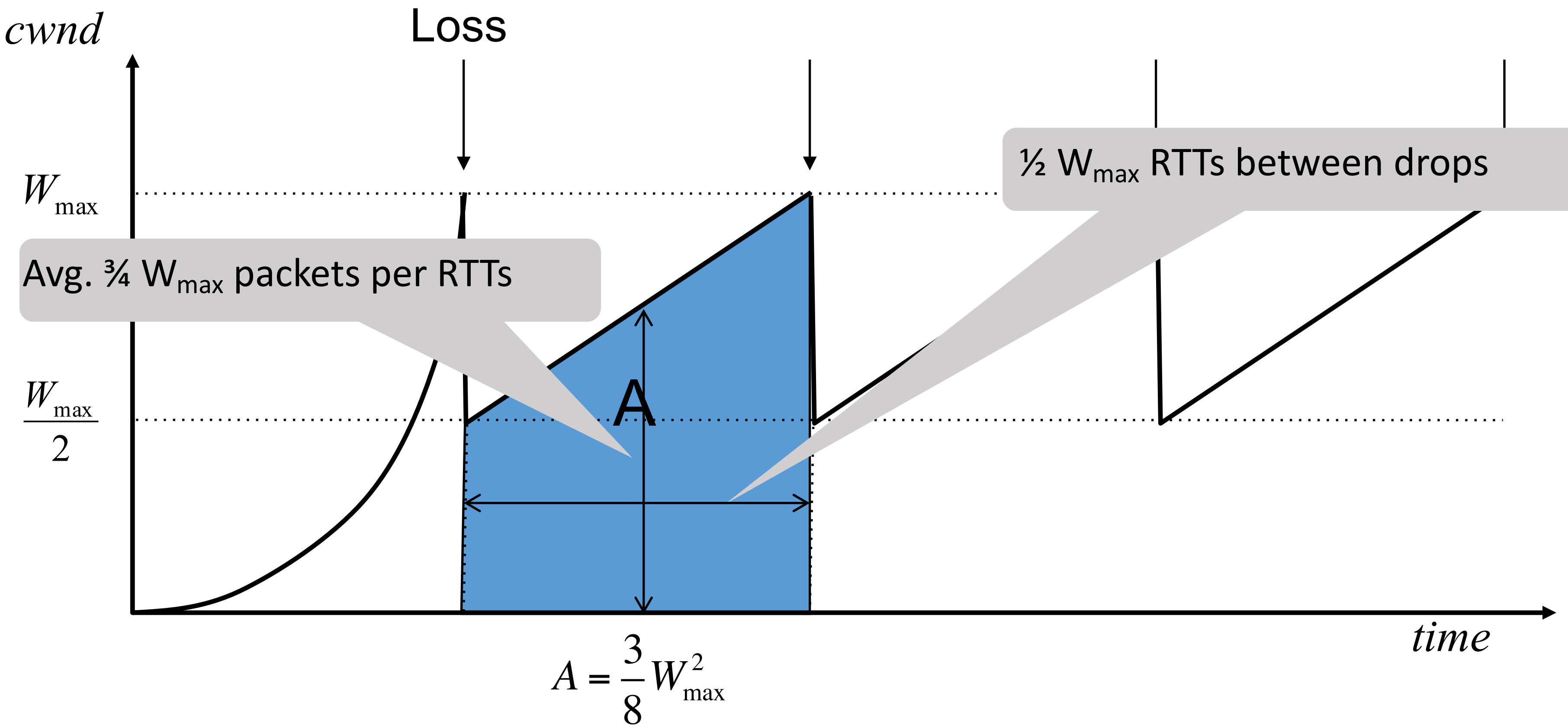
- **TCP-Tahoe**
  - ➢CWND =1 on 3 dupACKs
- **TCP-Reno**
  - ➢CWND =1 on timeout
  - ➢CWND = CWND/2 on 3 dupACKs
- **TCP-newReno**
  - ➢TCP-Reno + improved fast recovery
- **TCP-SACK**
  - ➢Incorporates selective acknowledgements

Our default assumption

A simple model for TCP throughput



A simple model for TCP throughput



Implications on High-speed TCP

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

- **Assume RTT = 100ms, MSS=1500bytes, BW=100Gbps**
- **What value of p is required to reach 100Gbps throughput?**
  - ~ 2×10^{-12}
- **How long between drops?**
  - ~ 16.6 hours
- **How much data has been sent in this time?**
  - ~ 6 petabits

Topics

- **Basics (lectures 1–3)**
- **Application layer (lectures 4, 5)**
  - HTTP, DNS, and CDN
- **Transport layer (lectures 6–9)**
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- **Network layer (lectures 10)**
  - Data plane

Forwarding vs. routing

- **Forwarding: "data plane"**
  - Directing one data packet
  - Each router using local routing state
- **Routing: "control plane"**
  - Computing the forwarding tables that guide packets
  - Jointly computed by routers using a distributed algorithm

- **Very different timescales!**

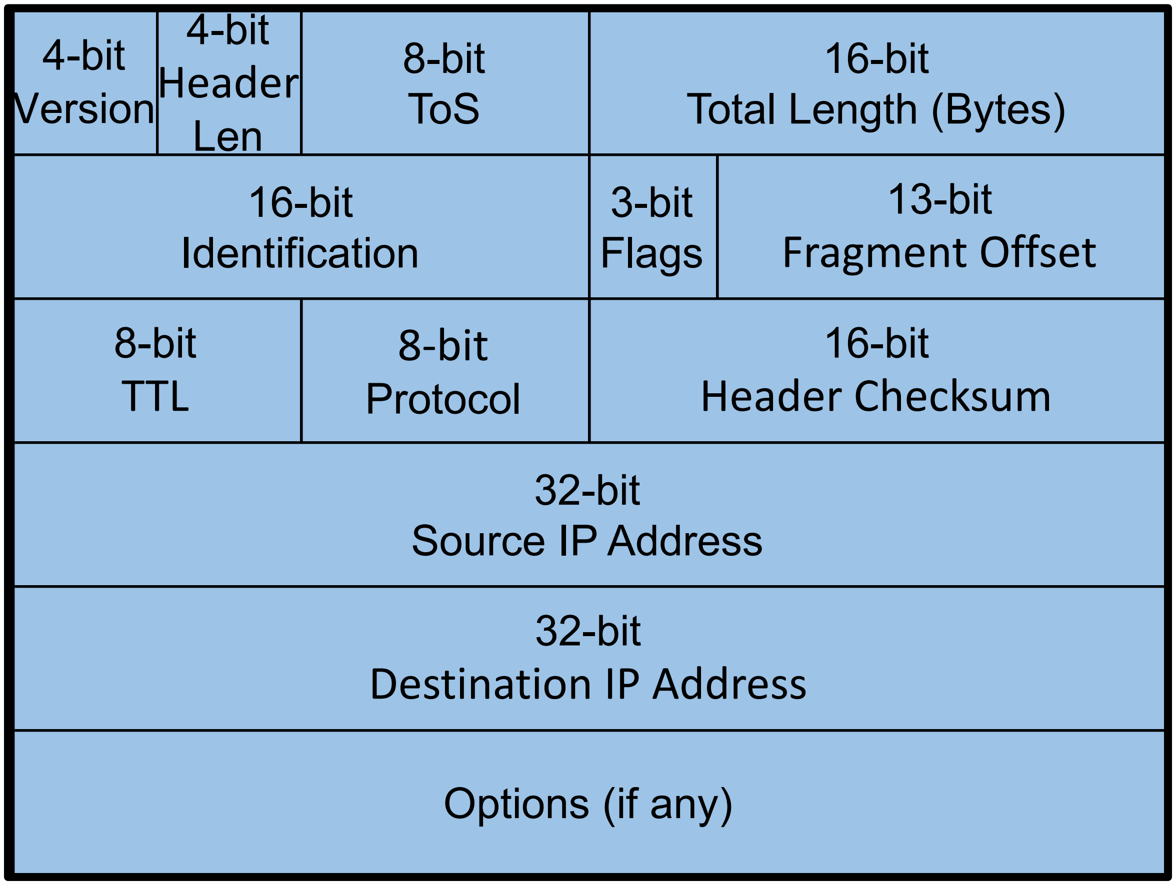
Designing the IP header

- **Think of the IP header as an interface**
  - Between the source and destination end-systems
  - Between the source and network (routers)
- **Designing an interface**
  - What task(s) are we trying to accomplish?
  - What information is needed to do it?
- **Header reflects information needed for basic tasks**

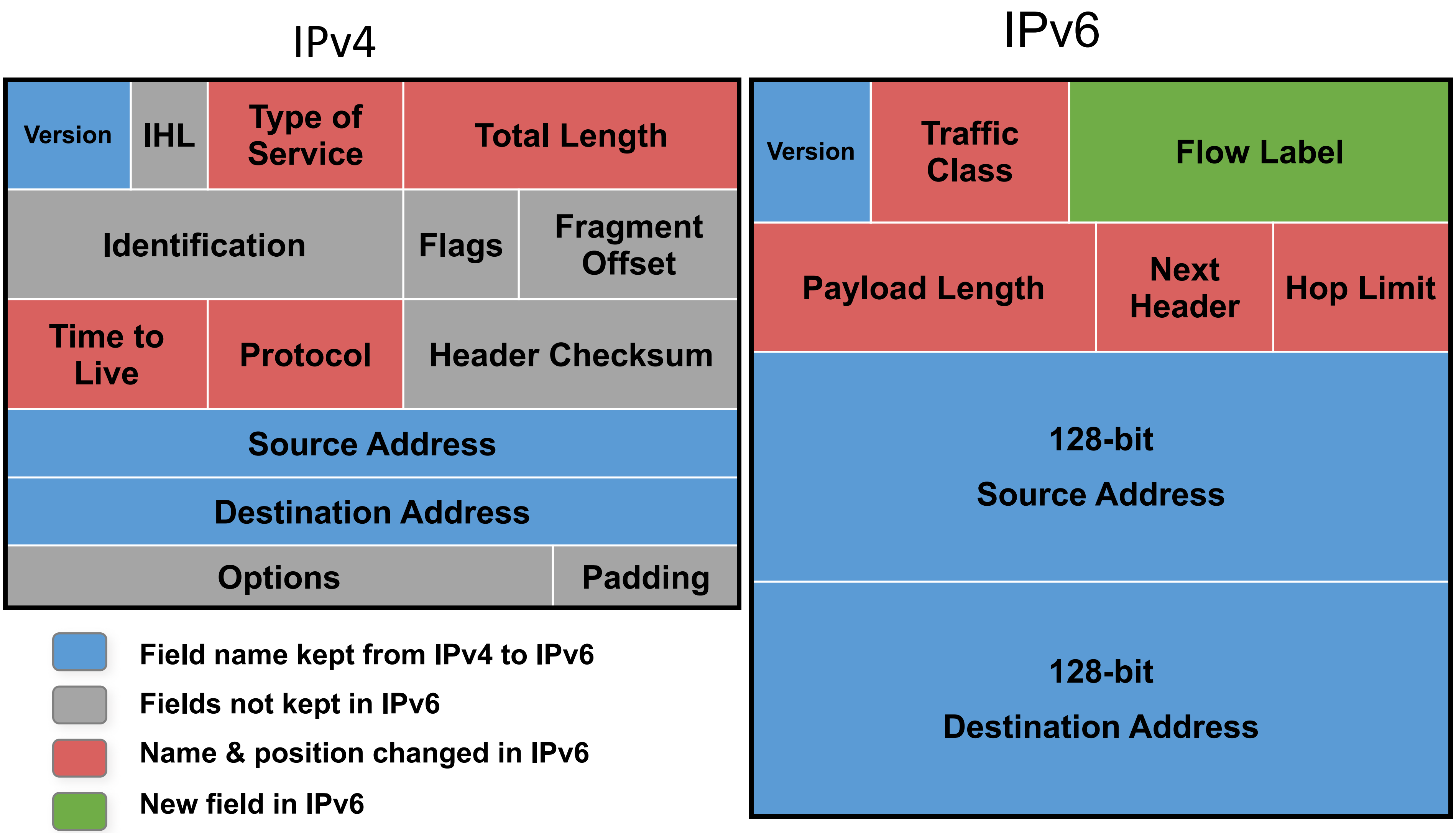
What information do we need?

- **Parse packet**
  - IP version number (4 bits), packet length (16 bits)
- **Carry packet to the destination**
  - Destination's IP address (32 bits)
- **Deal with problems along the way**
  - Loops: TTL (8 bits)
  - Corruption: checksum (16 bits)
  - Packet too large: fragmentation fields (32 bits)

IP packet structure



IPv4 and IPv6 header comparison



Philosophy of changes

- **Don’t deal with problems: leave to ends**
  - Eliminated fragmentation and checksum
  - Why retain TTL?
- **Simplify handling:**
  - New options mechanism (uses next header)
  - Eliminated header length
    - Why couldn’t IPv4 do this?
- **Provide general flow label for packet**
  - Not tied to semantics
  - Provides great flexibility

Thanks!
Q&A