EECS 489
Computer Networks

Fall 2020

Mosharaf Chowdhury

Material with thanks to Aditya Akella, Sugih Jamin, Philip Levis, Sylvia Ratnasamy, Peter Steenkiste, and many other colleagues.

Logistics

- Open book/text/notes, but OFFLINE
  - Except for taking the exam over the Internet
- You’re NOT allowed to write/run any programs
- You’re NOT allowed to collaborate with anyone

General guidelines (1)

- Test only assumes material covered in lecture, discussion sections, quizzes, and assignments
  - Text: only to clarify details and context for the above
- The test doesn't require you to do complicated calculations
  - Use this as a hint to determine if you're on right track
- You don't need to memorize anything
- You do need to understand how things work

General guidelines (2)

- Be prepared to:
  - Weigh design options outside of the context we studied them in
  - Contemplate new designs we haven't covered in detail but can be put together
    - e.g., I introduce a new IP address format; how does this affect..."
  - Reason from what you know about the pros/cons of solutions we did study

General guidelines (3)

- Exam format
  - Q1: True-False questions
    - Wrong answer results in negative marks
  - Q2: MCQ questions
  - Q3-QN networking use cases
    - Questions not ordered in terms of complexity
- ~80 minutes
  - About 10 minutes more than a typical EECS489 midterm *without* increasing complexity

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 Covered in Review 1

- Basics (lectures 1–2)
- Application layer (lectures 3–5)
  - HTTP, DNS, and CDN
  - Video Streaming

Topics

- Transport layer (lectures 6–9)
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- Network layer (lecture 10–11)
  - Overview
  - 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 (concepts only)

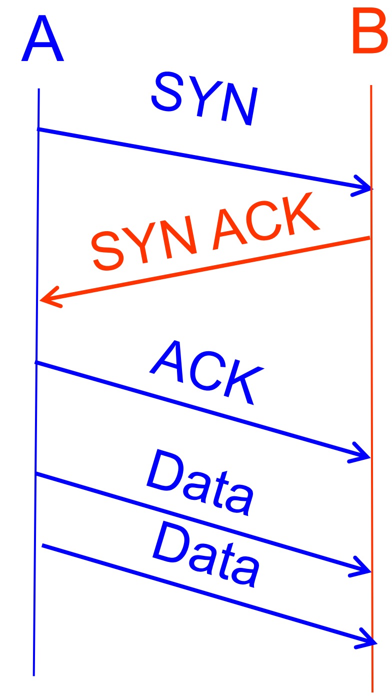
- For each, know how the functionality is implemented and why it is needed

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)

Establishing/terminating 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 overrunning 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

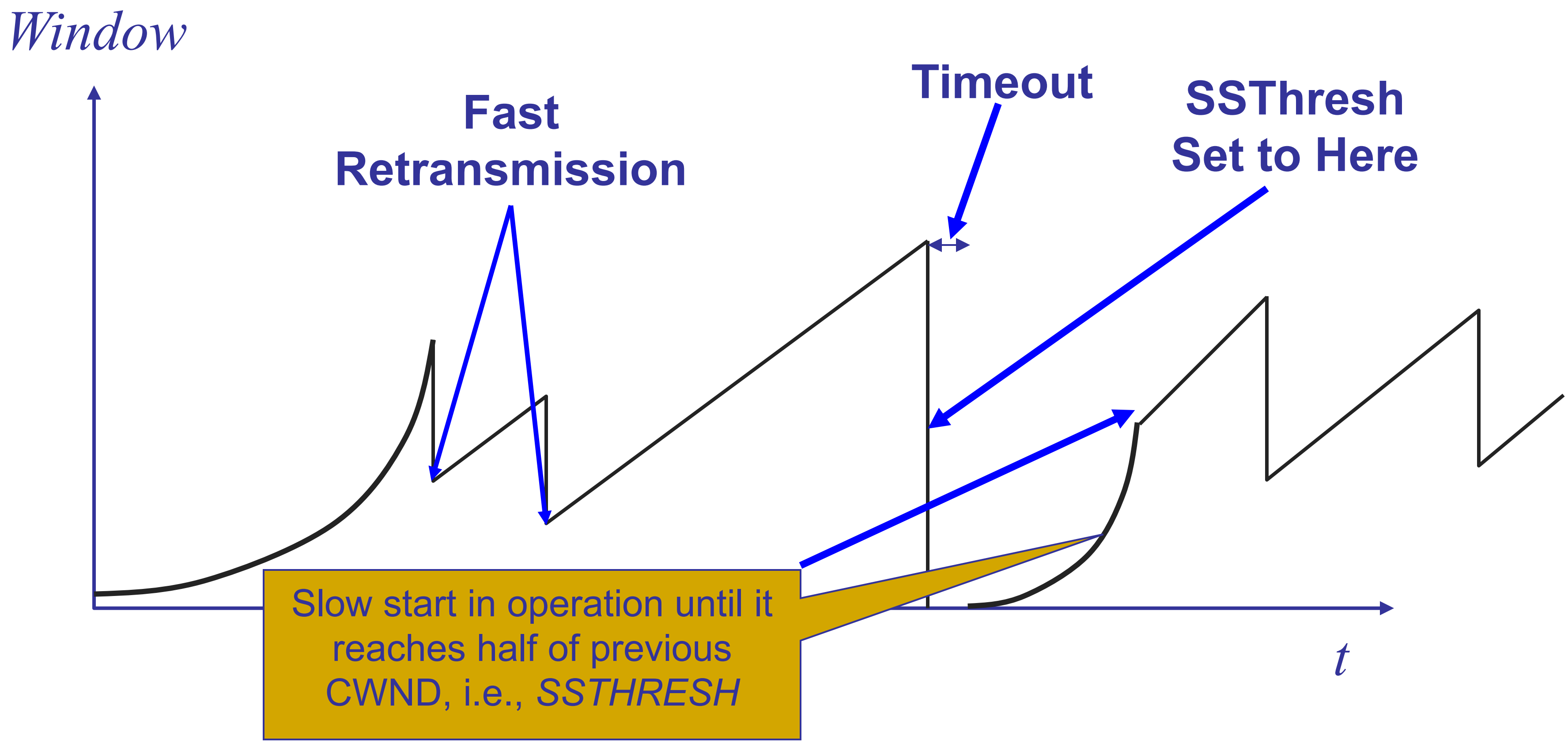
Event: TimeOut

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

Event: dupACK

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

Example

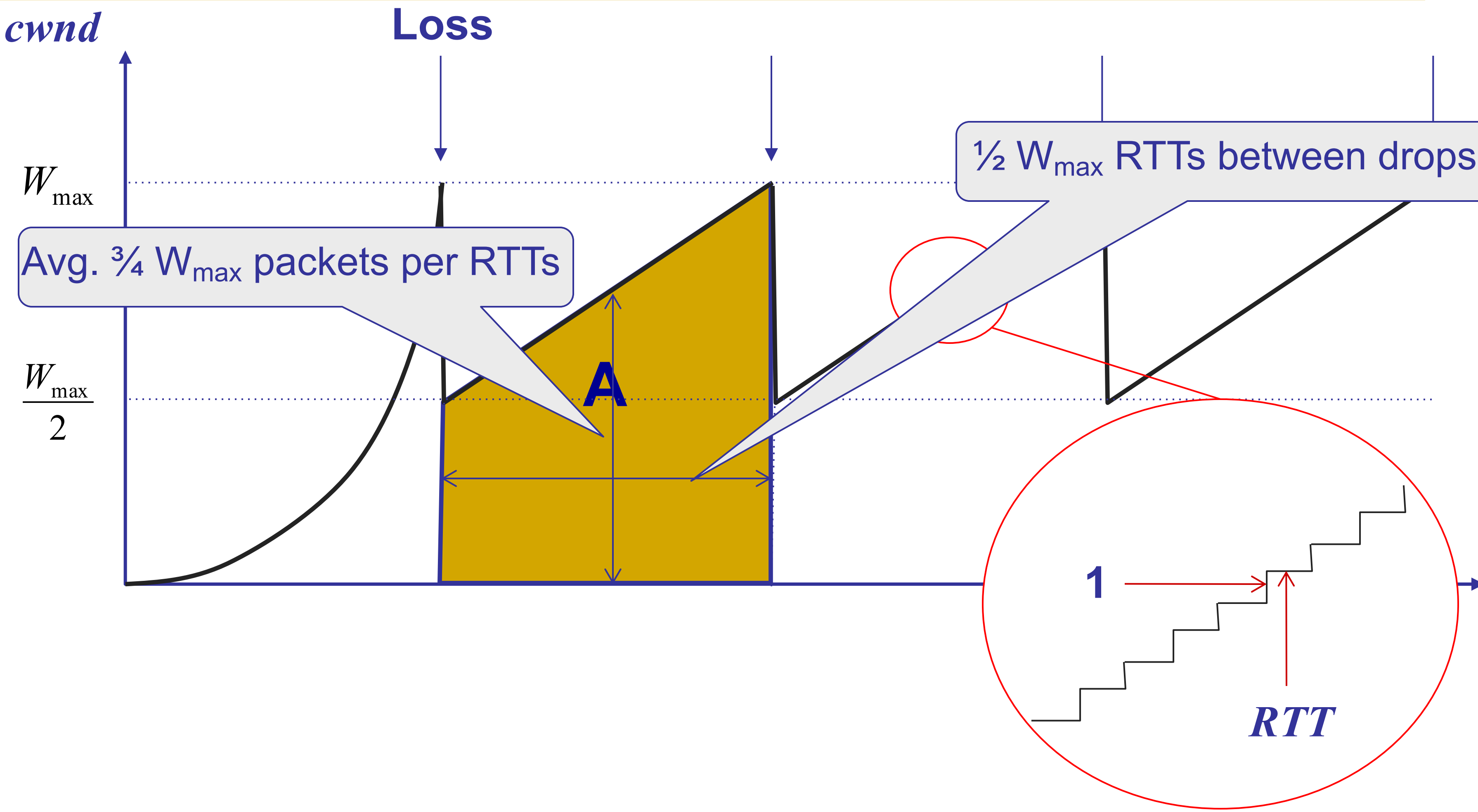


Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

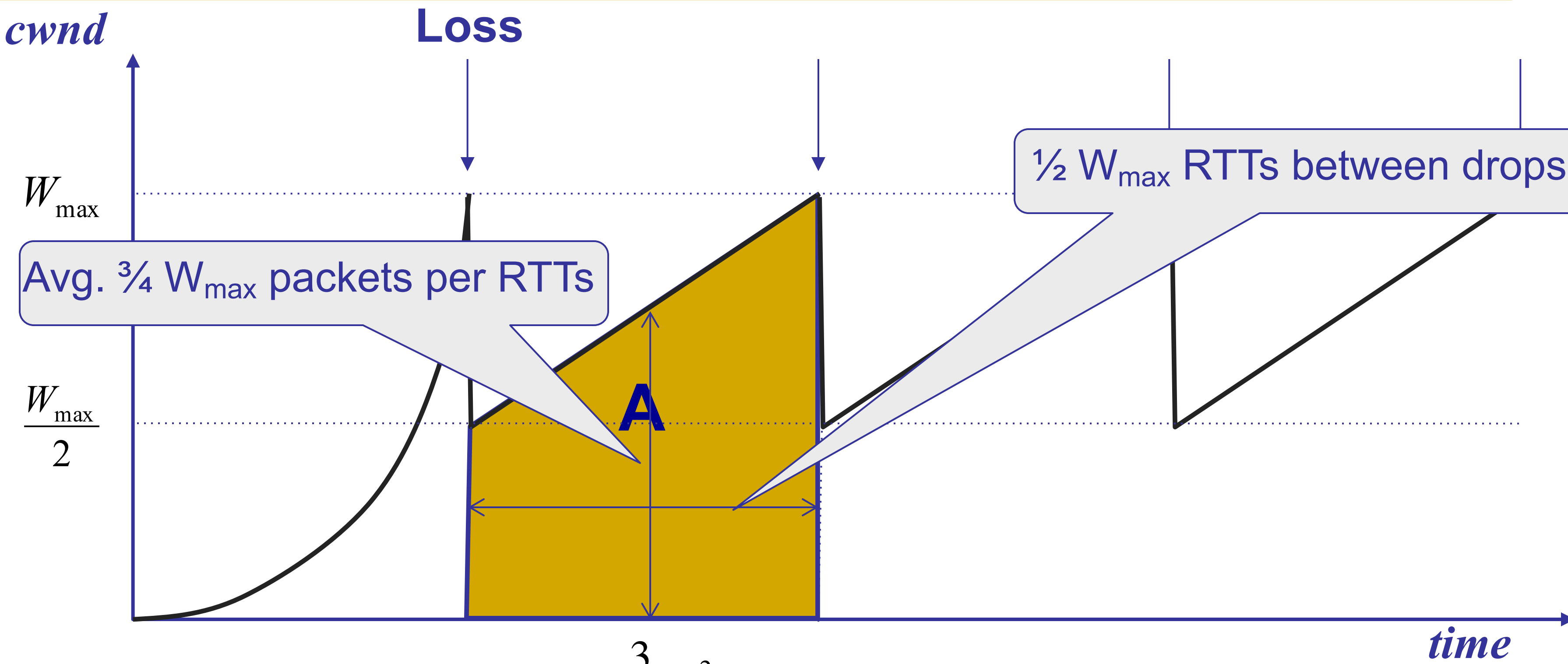
TCP flavors

- 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

A simple model for TCP throughput



A simple model for TCP throughput



$$A = \frac{3}{8} W_{max}^2$$

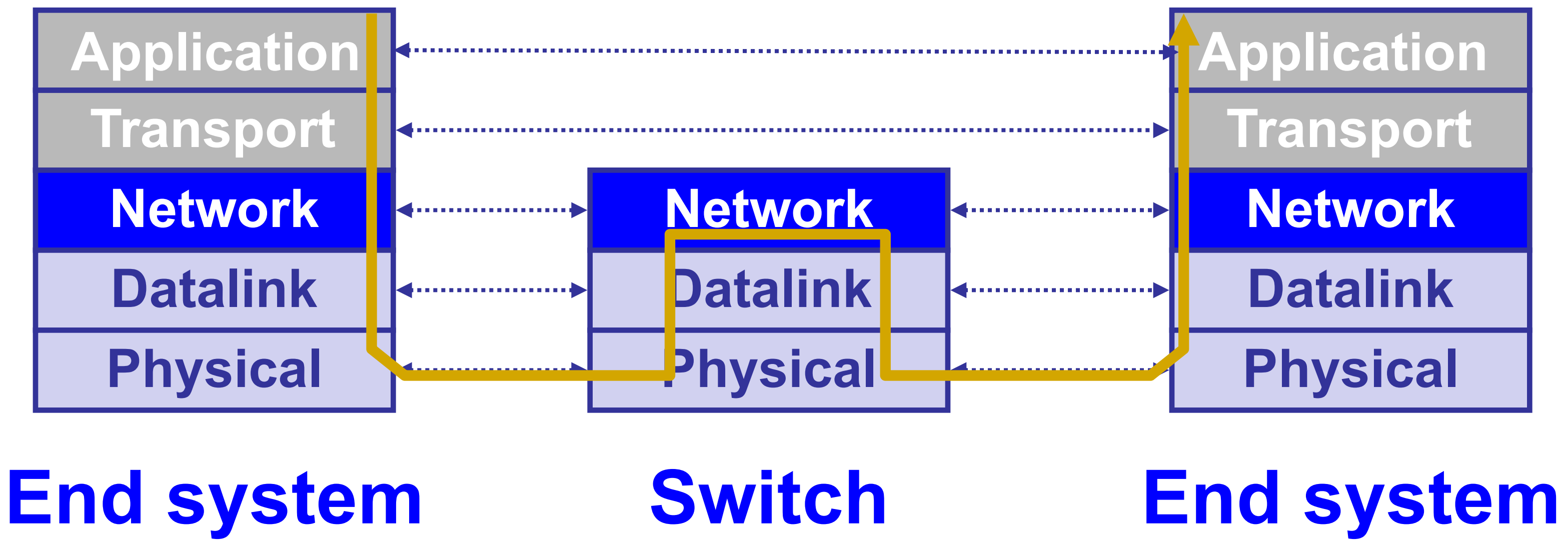
5-MINUTE BREAK!

Topics

- Transport layer (lectures 6–9)
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- Network layer (lecture 10–11)
  - Overview
  - Data plane

Network layer

- Present everywhere
- Performs addressing, forwarding, and routing, among other tasks



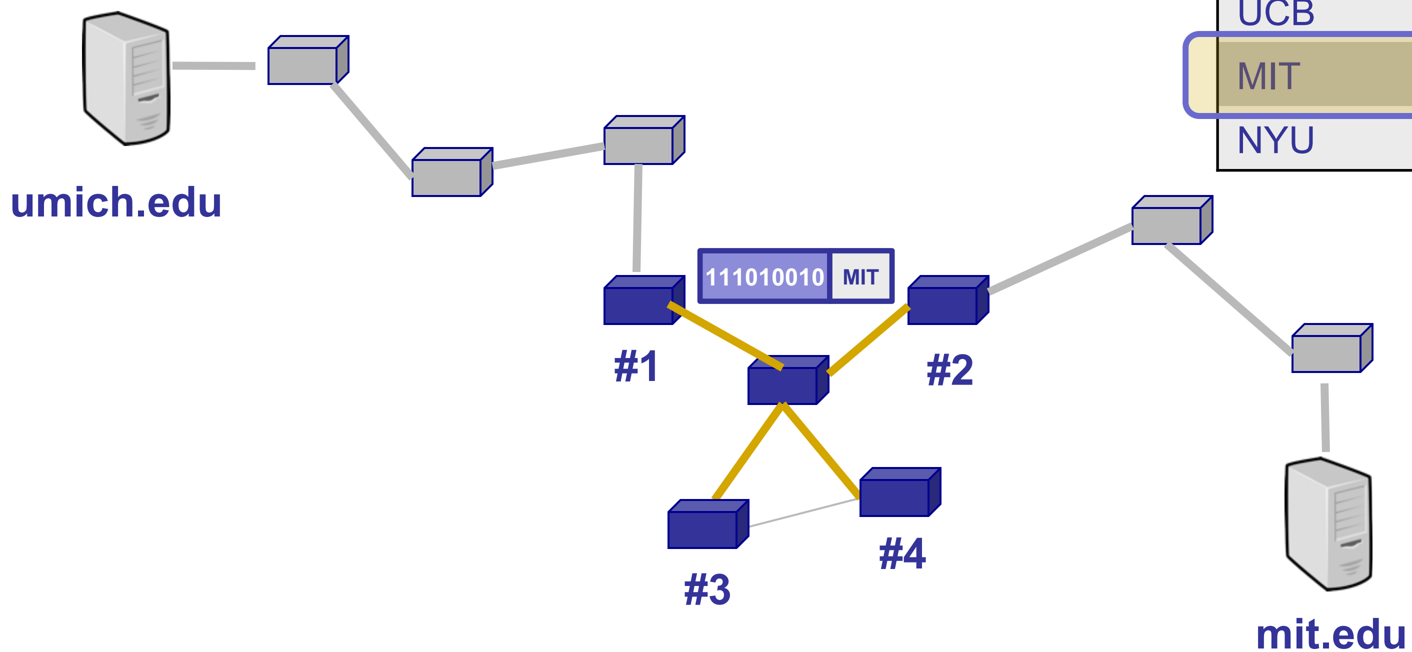
Forwarding vs. routing

- 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

Forwarding

Forwarding Table

Destination	Next Hop
UMICH	1
UCB	3
MIT	2
NYU	4



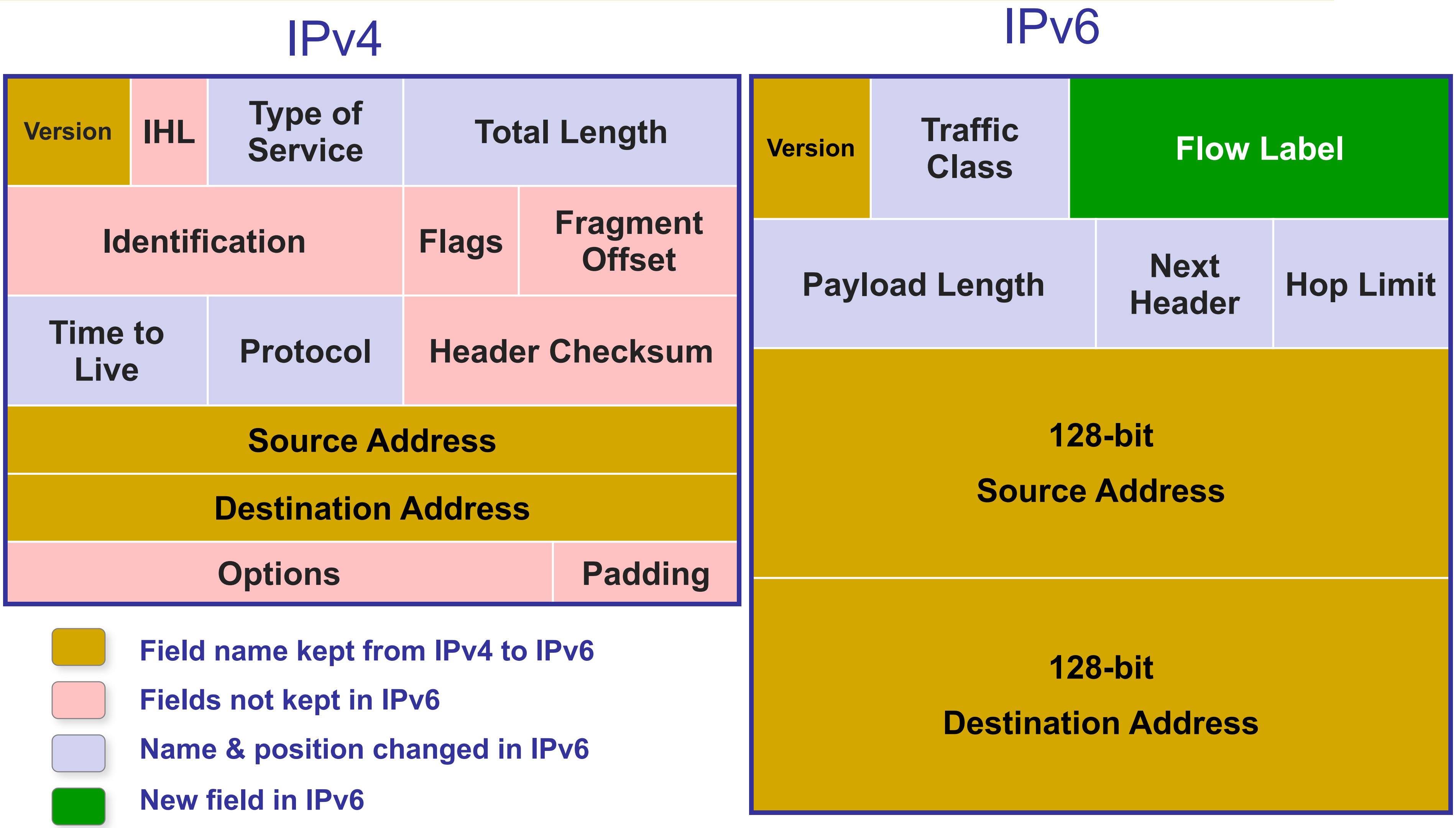
Designing the IP header

- 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)

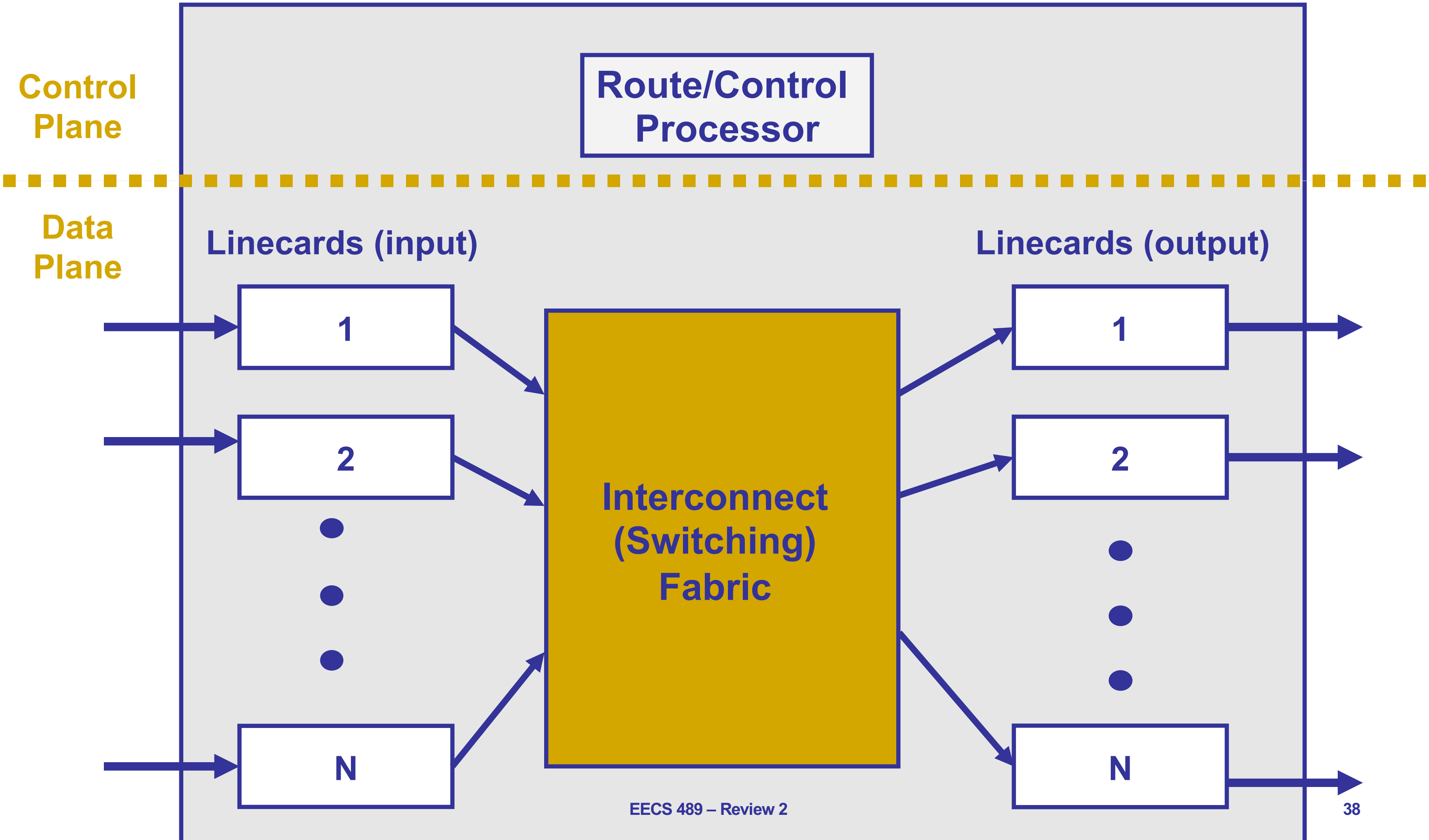
IPv4 and IPv6 header comparison



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

What's inside a router?



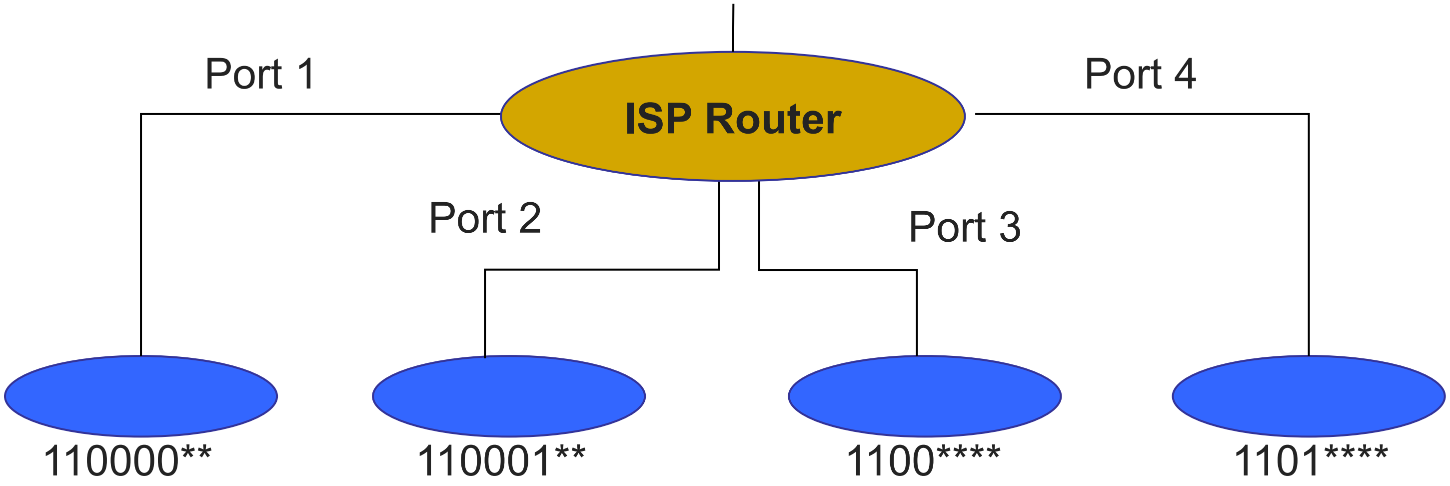
Input linecards

- Main challenge is processing speeds
- Tasks involved:
  - Update packet header (easy)
  - LPM lookup on destination address (harder)
- Mostly implemented with specialized hardware

Looking up the output port

- One entry for each address → 4 billion entries!
- For scalability, addresses are aggregated

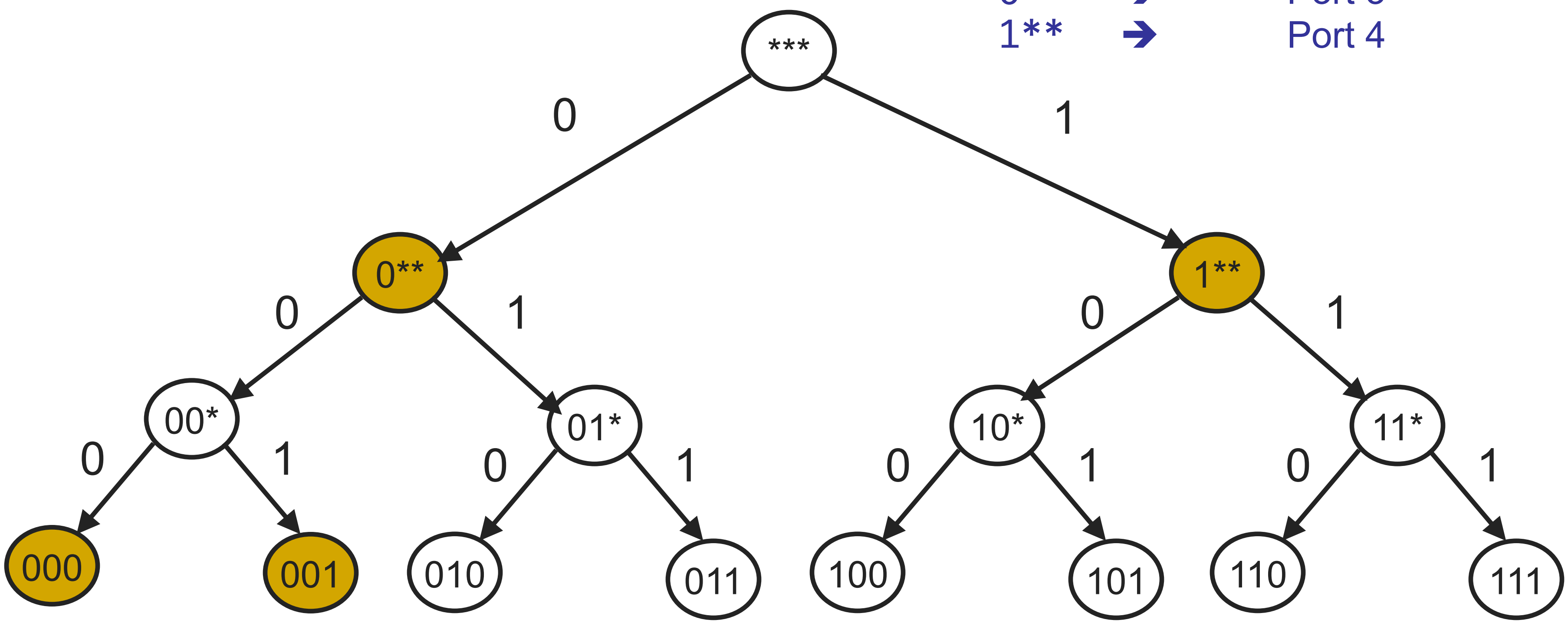
Longest prefix matching



Send to the port with the longest prefix match

Tree structure

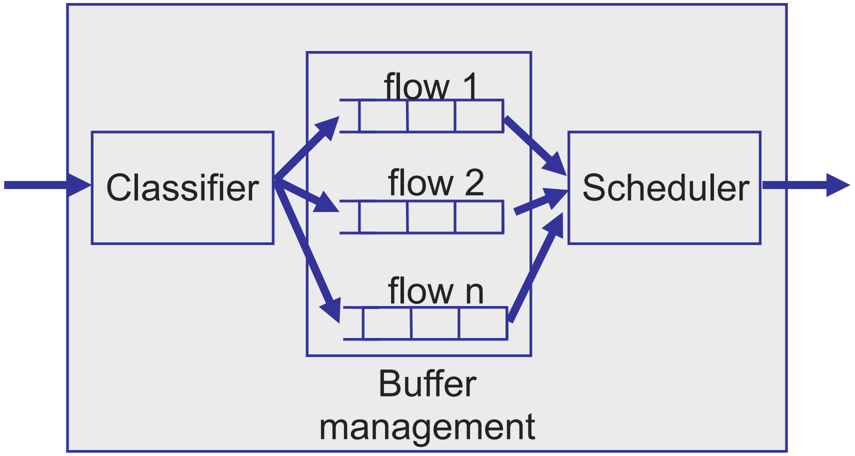
000	→	Port 1
001	→	Port 2
0**	→	Port 3
1**	→	Port 4



Record port associated with latest match, and only override when it matches another prefix during walk down tree

Output linecards

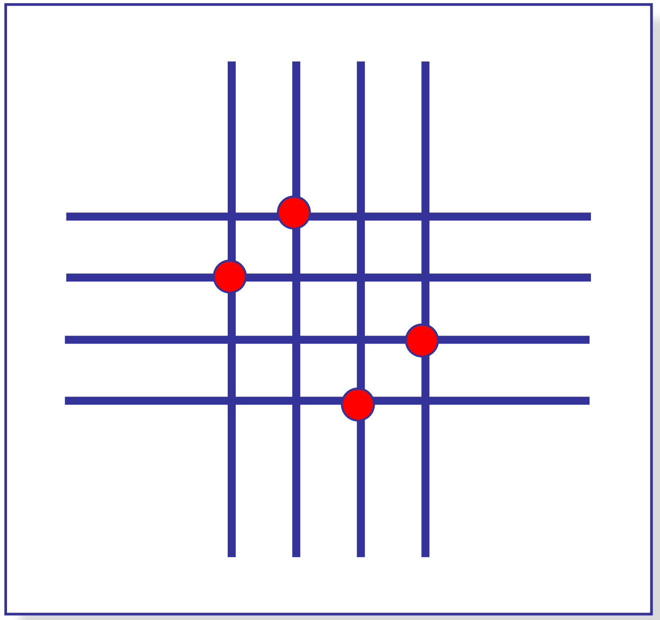
- Packet classification: map packets to flows
- Buffer management: decide when and which packet to drop
- Scheduler: decide when and which packet to transmit



Crossbar interconnect

- 2N buses intersecting with each other:
  - N input
  - N output
- Non-blocking

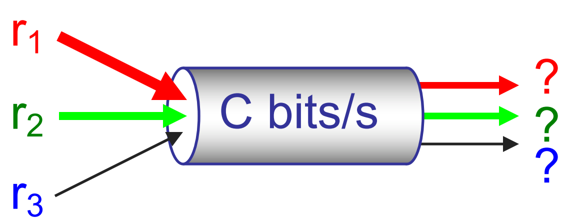
Input ports



Output ports

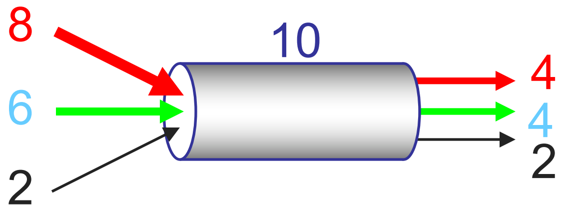
Max-Min fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where f is the unique value such that $\mathrm{Sum}(a_i) = C$



Example

- $C = 10$; $r_1 = 8$, $r_2 = 6$, $r_3 = 2$; $N = 3$
- $C/3 = 3.33 \rightarrow$
  - r_3 needs only 2
    - » Can service all of r_3
  - Remove r_3 from the accounting: $C = C - r_3 = 8$; $N = 2$
- $C/2 = 4 \rightarrow$
  - Can't service all of r_1 or r_2
  - So hold them to the remaining fair share: $f = 4$



$f = 4$:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Max-Min fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where f is the unique value such that $\mathrm{Sum}(a_i) = C$
- If you don't get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size

Summary

- Demo Exam on Canvas