EECS 489

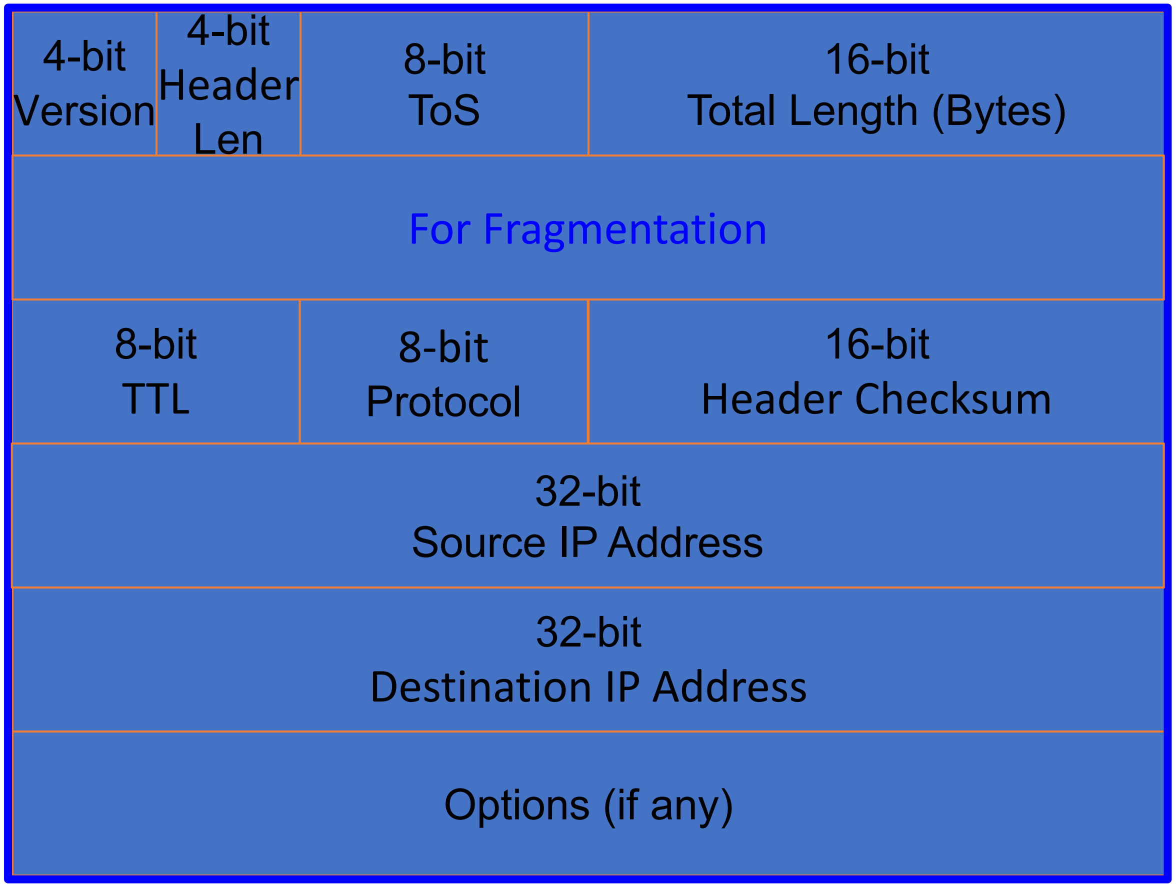
Computer Networks

IP Routers

Agenda

- Finish up network layer
- IP routers
- Router-assisted congestion control

IP packet structure



4-bit Version

4-bit Header Len

8-bit ToS

16-bit Total Length (Bytes)

For Fragmentation

8-bit TTL

8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

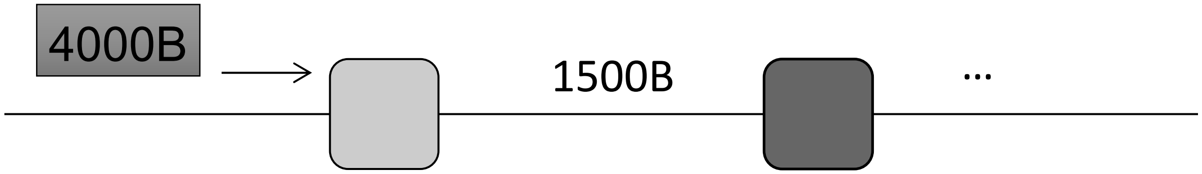
Dealing with fragmentation

A closer look at fragmentation

- Every link has a “Maximum Transmission Unit” (MTU)
  - Largest number of bits it can carry as one unit
- A router can split a packet into multiple “fragments” if the packet size exceeds the link’s MTU
- Must reassemble to recover original packet

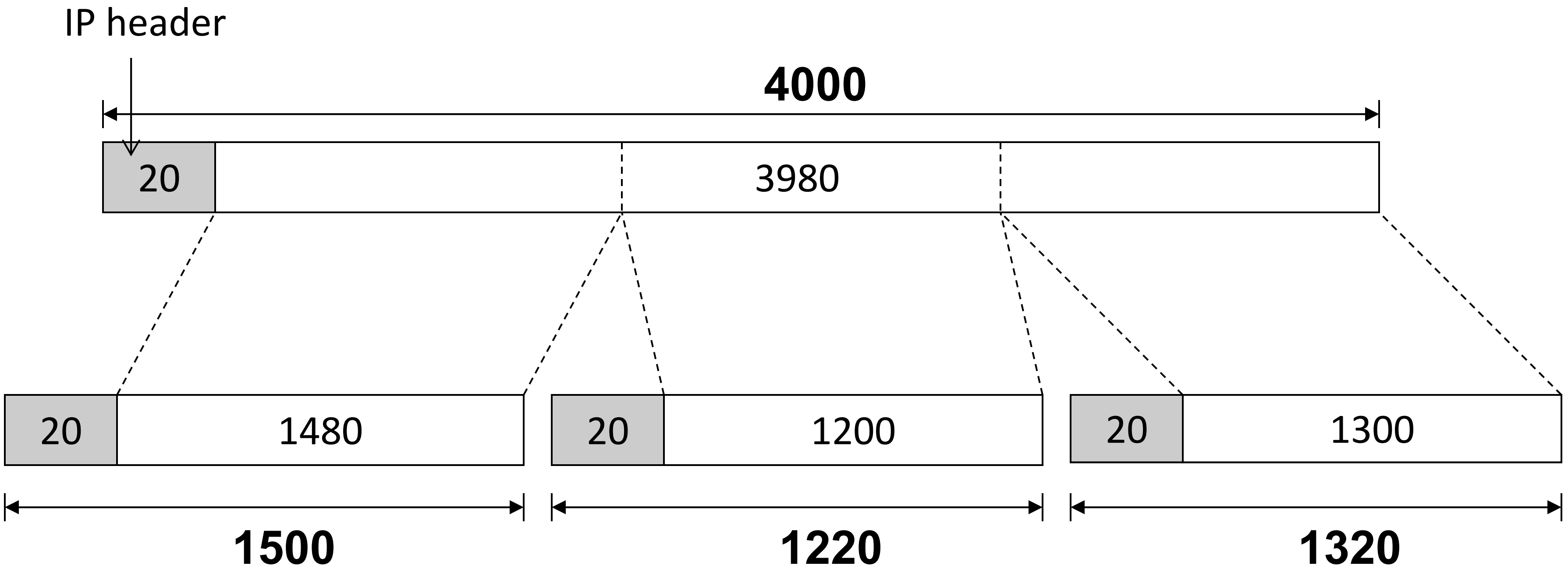
Example of fragmentation

- A 4000 byte packet crosses a link w/ MTU=1500B

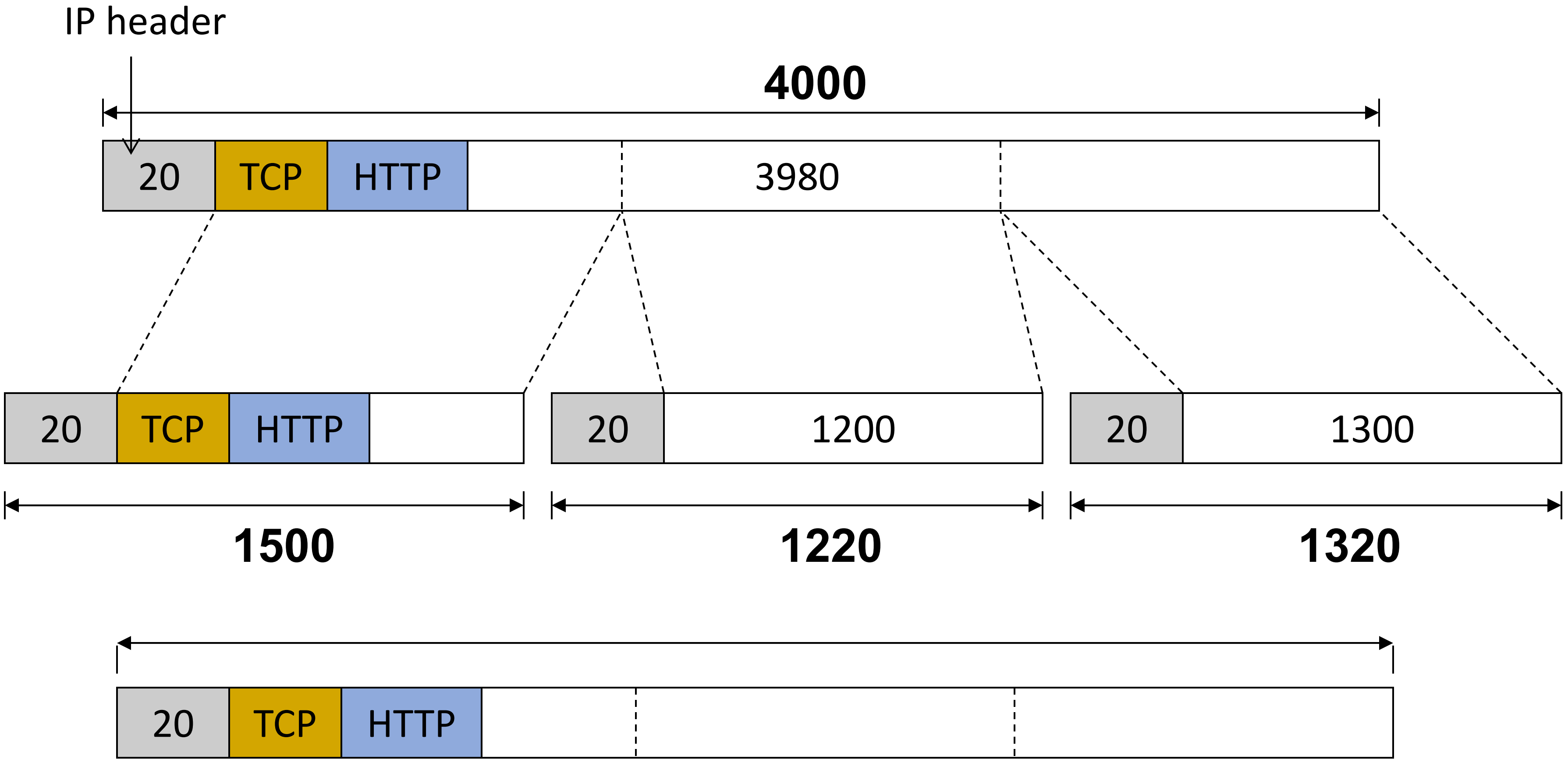


Example of fragmentation

- A 4000 byte packet crosses a link w/ MTU=1500B



Why reassemble?



Must reassemble before sending packet to higher layers!

A few considerations

- Where to reassemble?
- Fragments can get lost
- Fragments can follow different paths
- Fragments can get fragmented again

Where should reassembly occur?

- Classic case of E2E principle
- At next-hop router imposes burden on network
  - Complicated reassembly algorithm
  - Must hold onto fragments/state
- Any other router may not work
  - Fragments may take different paths
- Little benefit, large cost for network reassembly
- Hence, reassembly is done at the destination

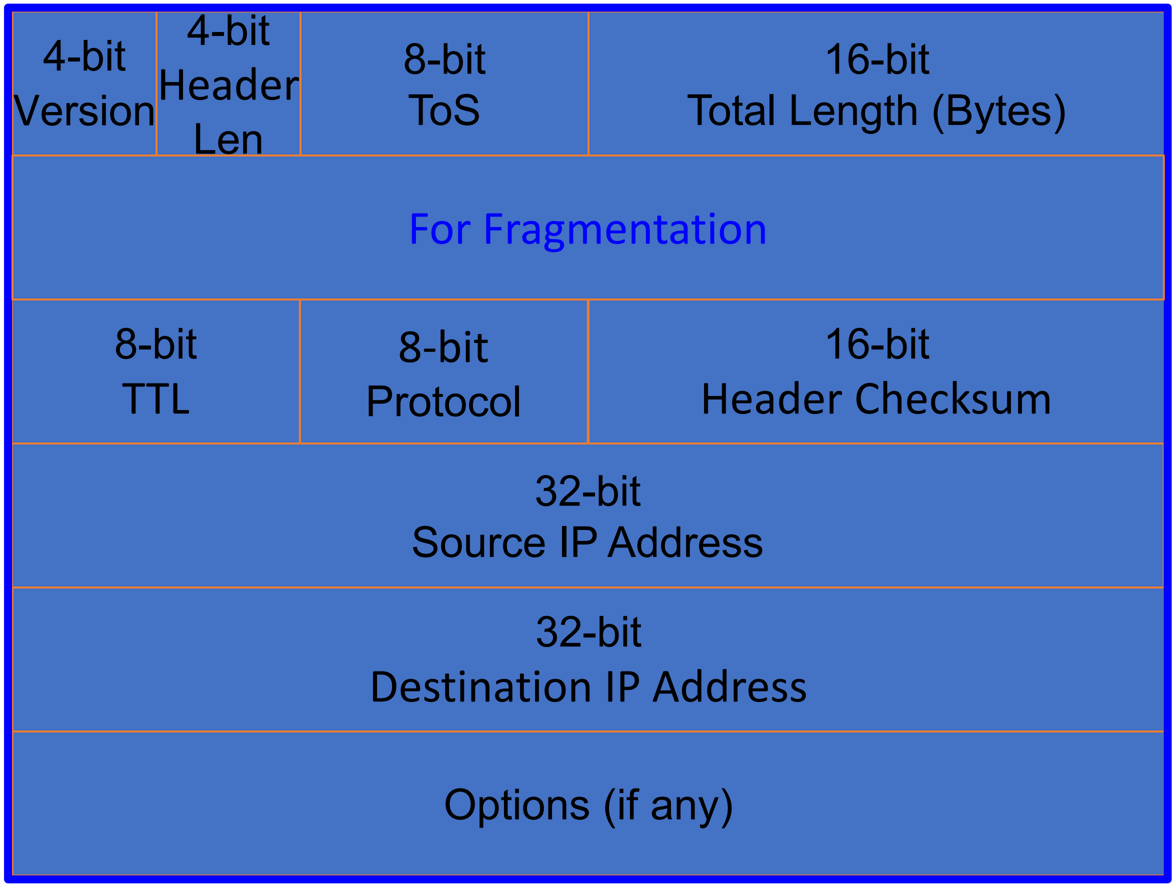
Reassembly: What fields?

- Need a way to identify fragments of the packet
  - Introduce an identifier
- Fragments can get lost
  - Need some form of sequence number or offset
- Sequence numbers / offset
  - How do I know when I have them all? (need max seq# / flag)
  - What if a fragment gets re-fragmented?

IPv4’s fragmentation fields

- Identifier: which fragments belong together
- Flags:
  - Reserved: ignore
  - DF: don’t fragment
    - May trigger error message back to sender
  - MF: more fragments coming
- Offset: portion of original payload this fragment contains
  - In 8-byte units

IP packet structure



4-bit Version | 4-bit Header Len | 8-bit ToS | 16-bit Total Length (Bytes)

For Fragmentation

8-bit TTL | 8-bit Protocol | 16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

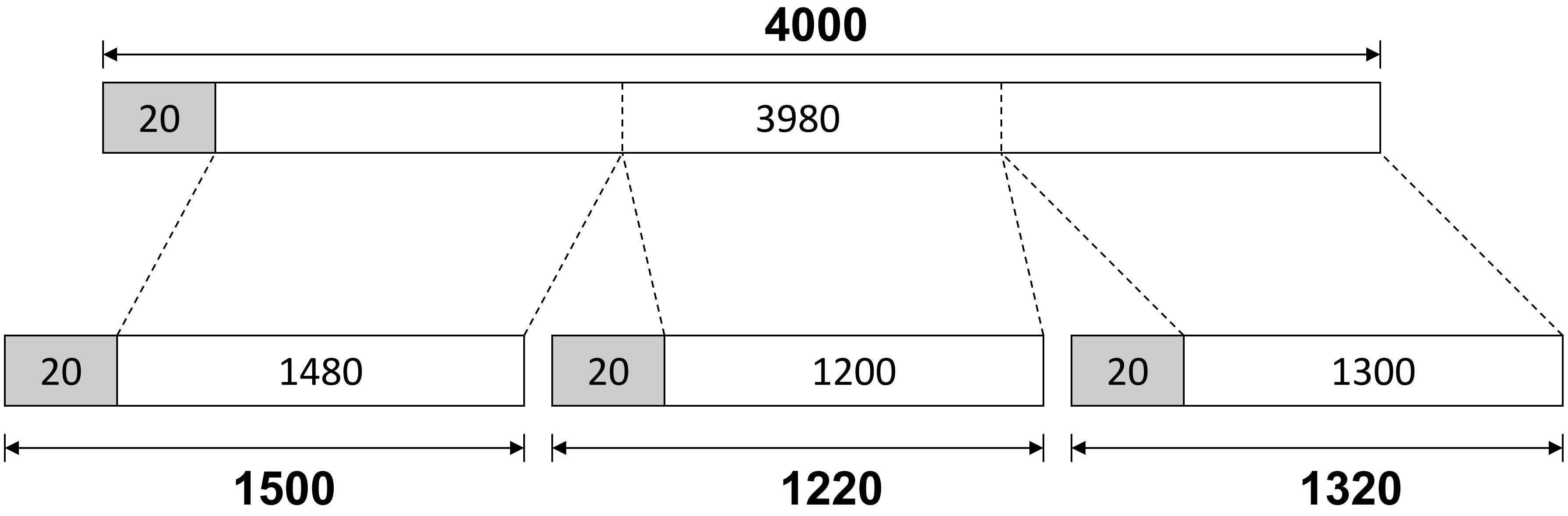
Options (if any)

Why this works

- Fragment without MF set (last fragment)
  - Tells host which are the last bits in original payload
- All other fragments fill in holes
- Can tell when holes are filled, regardless of order
  - Use offset field
- Q: why use a byte-offset for fragments rather than numbering each fragment?
  - Allows further fragmentation of fragments

Example of fragmentation (contd.)

- Packet split into 3 pieces
- Example:



Example of fragmentation, contd.

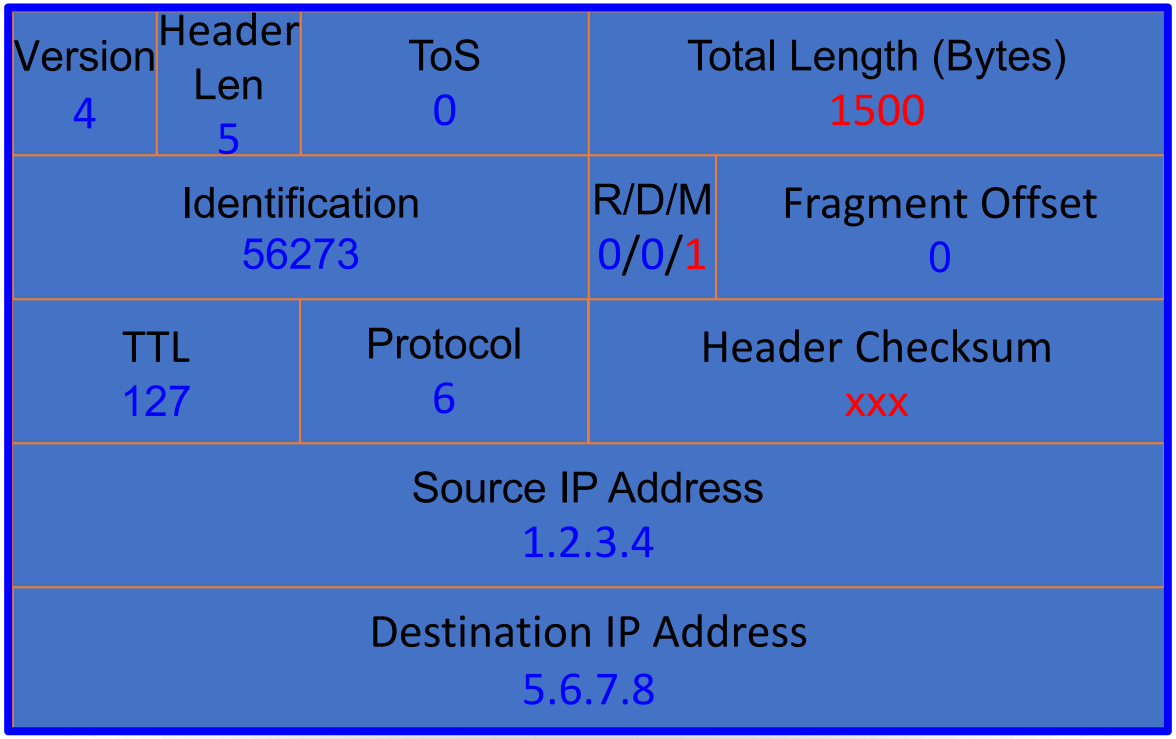
- 4000 byte packet from host 1.2.3.4 to 5.6.7.8 traverses a link with MTU 1,500 bytes

<table>
<tr><td>Version
4</td><td>Header Len
5</td><td>ToS
0</td><td colspan="2">Total Length (Bytes)
4000</td></tr>
<tr><td colspan="3">Identification
56273</td><td>R/D/M
0/0/0</td><td>Fragment Offset
0</td></tr>
<tr><td colspan="2">TTL
127</td><td>Protocol
6</td><td colspan="2">Header Checksum
44019</td></tr>
<tr><td colspan="5">Source IP Address
1.2.3.4</td></tr>
<tr><td colspan="5">Destination IP Address
5.6.7.8</td></tr>
</table>

(3980 more bytes of payload here)

Example of fragmentation, contd.

- Datagram split into 3 pieces. Possible first piece:



<table>
<tr><td>Version
4</td><td>Header Len
5</td><td>ToS
0</td><td colspan="2">Total Length (Bytes)
1500</td></tr>
<tr><td colspan="3">Identification
56273</td><td>R/D/M
0/0/1</td><td>Fragment Offset
0</td></tr>
<tr><td colspan="2">TTL
127</td><td>Protocol
6</td><td colspan="2">Header Checksum
xxx</td></tr>
<tr><td colspan="5">Source IP Address
1.2.3.4</td></tr>
<tr><td colspan="5">Destination IP Address
5.6.7.8</td></tr>
</table>

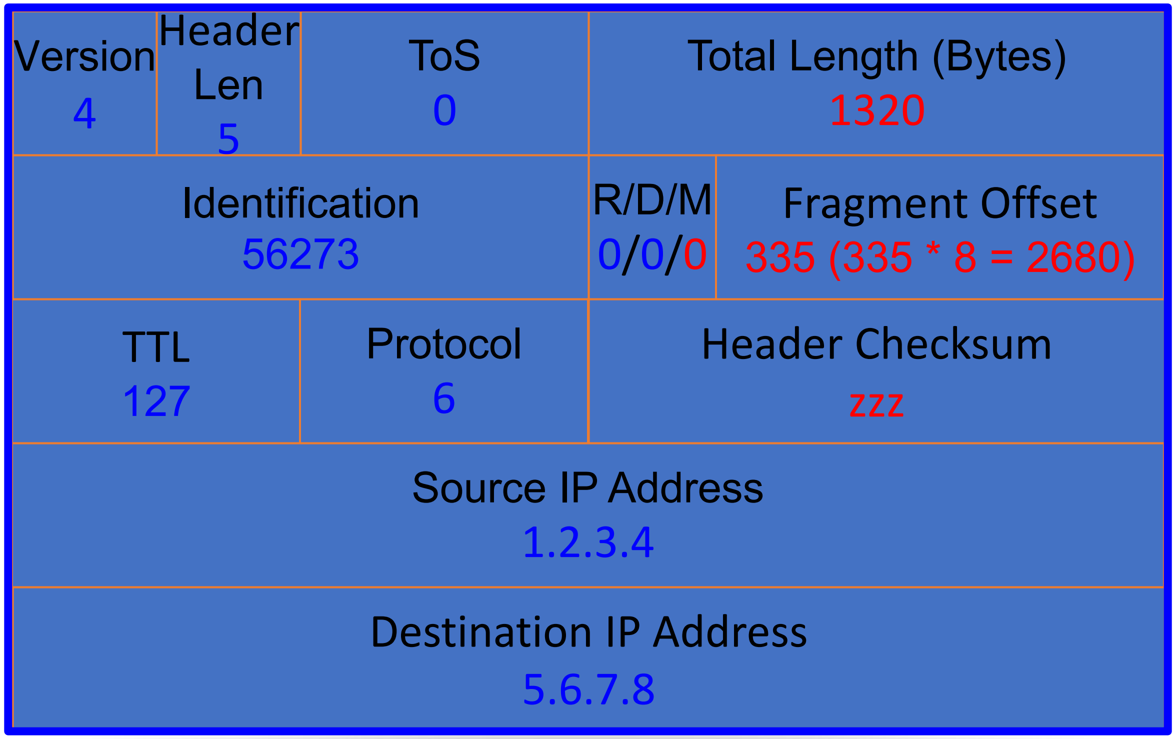
Example of fragmentation, contd.

- Possible second piece: Frag#1 covered 1480bytes

<table>
<tr><td>Version
4</td><td>Header Len
5</td><td>ToS
0</td><td colspan="2">Total Length (Bytes)
1220</td></tr>
<tr><td colspan="3">Identification
56273</td><td>R/D/M
0/0/1</td><td>Fragment Offset
185 (185 * 8 = 1480)</td></tr>
<tr><td colspan="2">TTL
127</td><td>Protocol
6</td><td colspan="2">Header Checksum
yyy</td></tr>
<tr><td colspan="5">Source IP Address
1.2.3.4</td></tr>
<tr><td colspan="5">Destination IP Address
5.6.7.8</td></tr>
</table>

Example of fragmentation, contd.

- Possible third piece: 1480+1200 = 2680



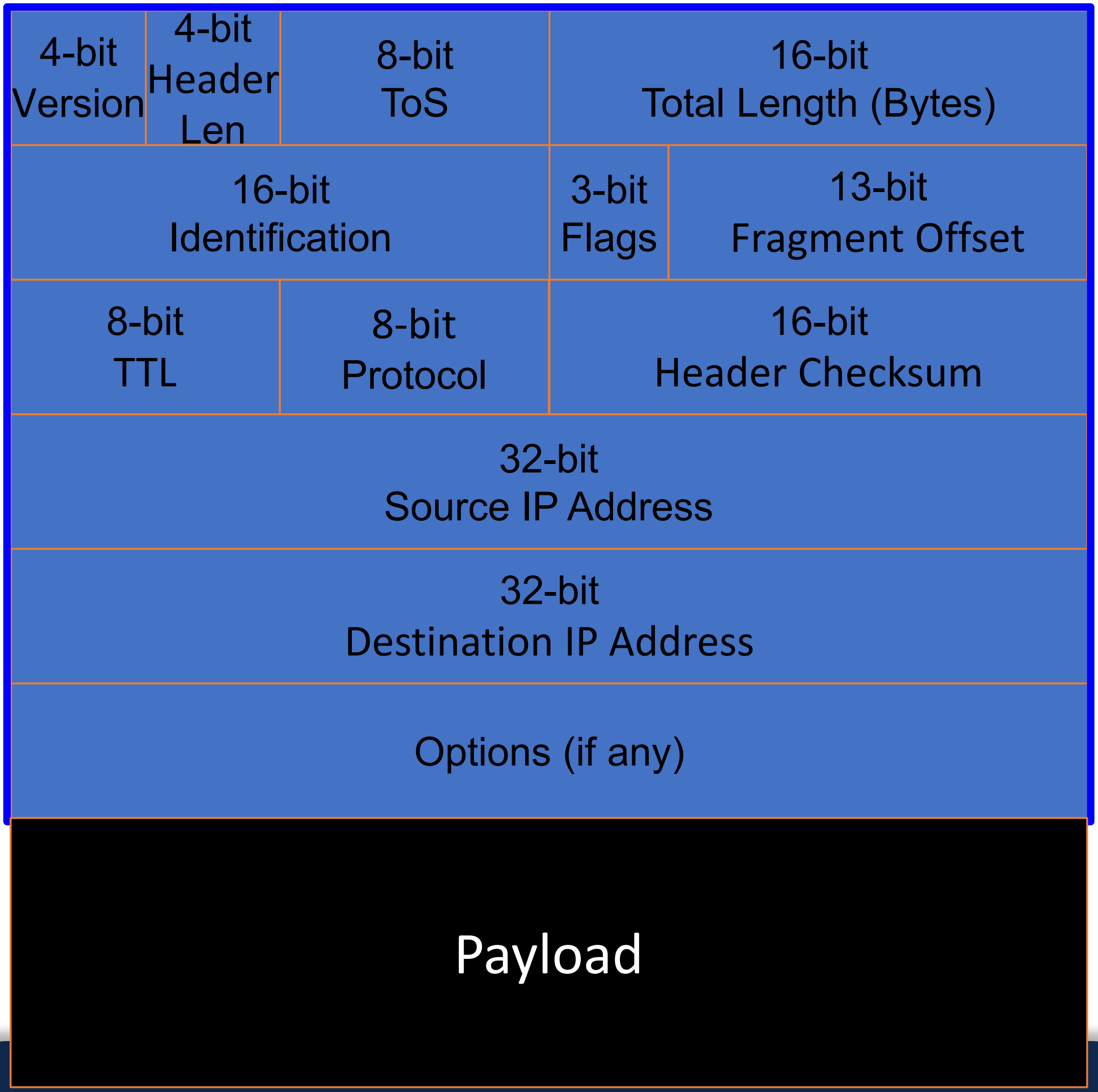
<table>
<tr><td>Version
4</td><td>Header Len
5</td><td colspan="2">ToS
0</td><td colspan="4">Total Length (Bytes)
1320</td></tr>
<tr><td colspan="4">Identification
56273</td><td>R/D/M
0/0/0</td><td colspan="3">Fragment Offset
335 (335 * 8 = 2680)</td></tr>
<tr><td colspan="2">TTL
127</td><td colspan="2">Protocol
6</td><td colspan="4">Header Checksum
zzz</td></tr>
<tr><td colspan="8">Source IP Address
1.2.3.4</td></tr>
<tr><td colspan="8">Destination IP Address
5.6.7.8</td></tr>
</table>

A quick look into IPv6

IPv6

- Motivated (prematurely) by address exhaustion
  - Addresses four times as big (128-bit)
- Focused on simplifying IP
  - Got rid of all fields that were not absolutely necessary
- Result is an elegant, if unambitious, protocol

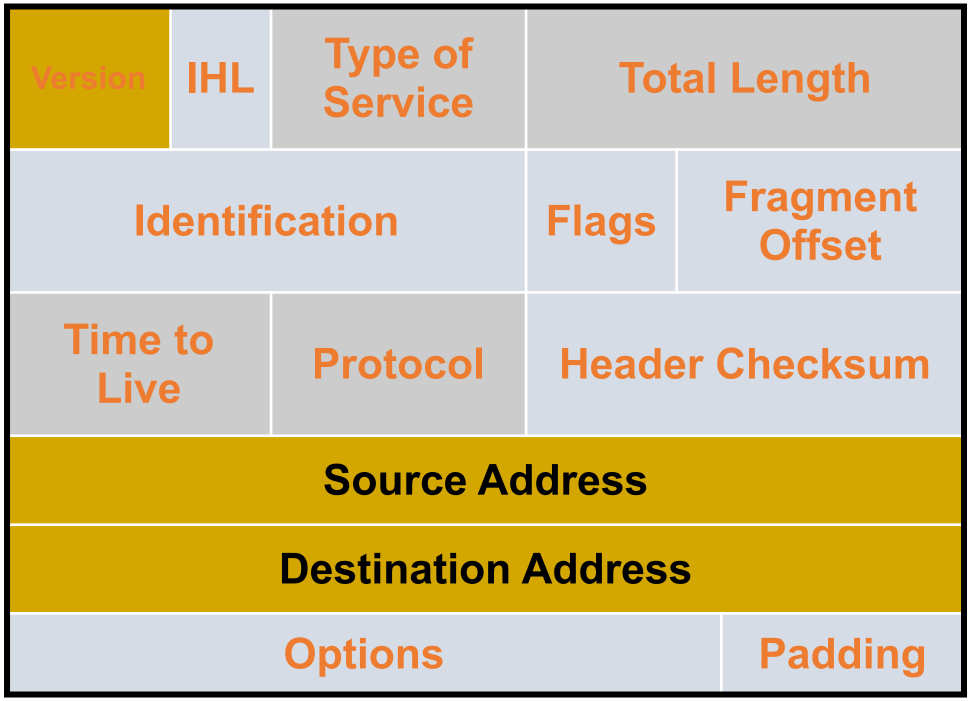
What “clean up” would you do?



<table>
<tr><td>4-bit
Version</td><td>4-bit
Header
Len</td><td>8-bit
ToS</td><td colspan="2">16-bit
Total Length (Bytes)</td></tr>
<tr><td colspan="3">16-bit
Identification</td><td>3-bit
Flags</td><td>13-bit
Fragment Offset</td></tr>
<tr><td colspan="2">8-bit
TTL</td><td>8-bit
Protocol</td><td colspan="2">16-bit
Header Checksum</td></tr>
<tr><td colspan="5">32-bit
Source IP Address</td></tr>
<tr><td colspan="5">32-bit
Destination IP Address</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
<tr><td colspan="5">Payload</td></tr>
</table>

IPv4 and IPv6 header comparison

IPv4



Version | IHL | Type of Service | Total Length

Identification | Flags | Fragment Offset

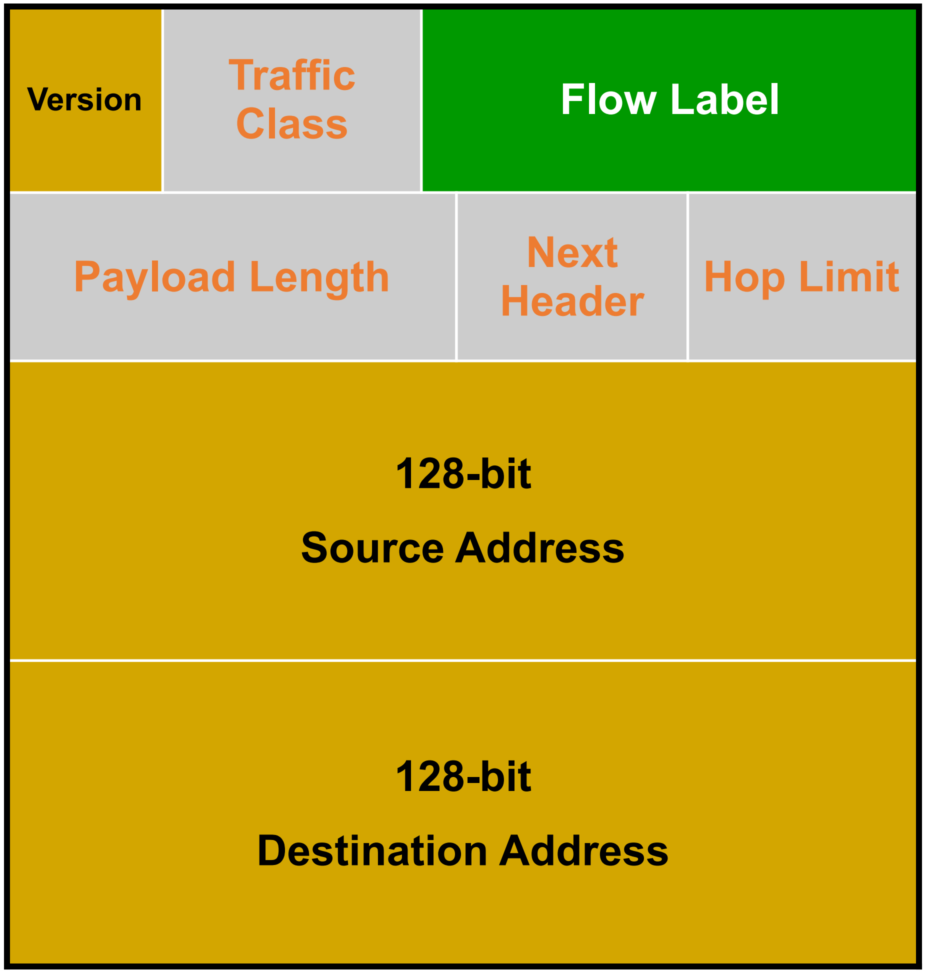
Time to Live | Protocol | Header Checksum

Source Address

Destination Address

Options | Padding

IPv6



Version | Traffic Class | Flow Label

Payload Length | Next Header | Hop Limit

128-bit Source Address

128-bit Destination Address

- 🟨 Field name kept from IPv4 to IPv6
- ▢ Fields not kept in IPv6
- ▨ Name & position changed in IPv6
- 🟩 New field in IPv6

Summary of changes

- Eliminated fragmentation (why?)
- Eliminated checksum (why?)
- New options mechanism (why?)
- Eliminated header length (why?)
- Expanded addresses
- Added Flow Label

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

Summary

- Network layer can be divided into data plane and control plane
  - Data plane deals with “how?”
  - Control plane deals with “what?”
- IP is simple yet nuanced

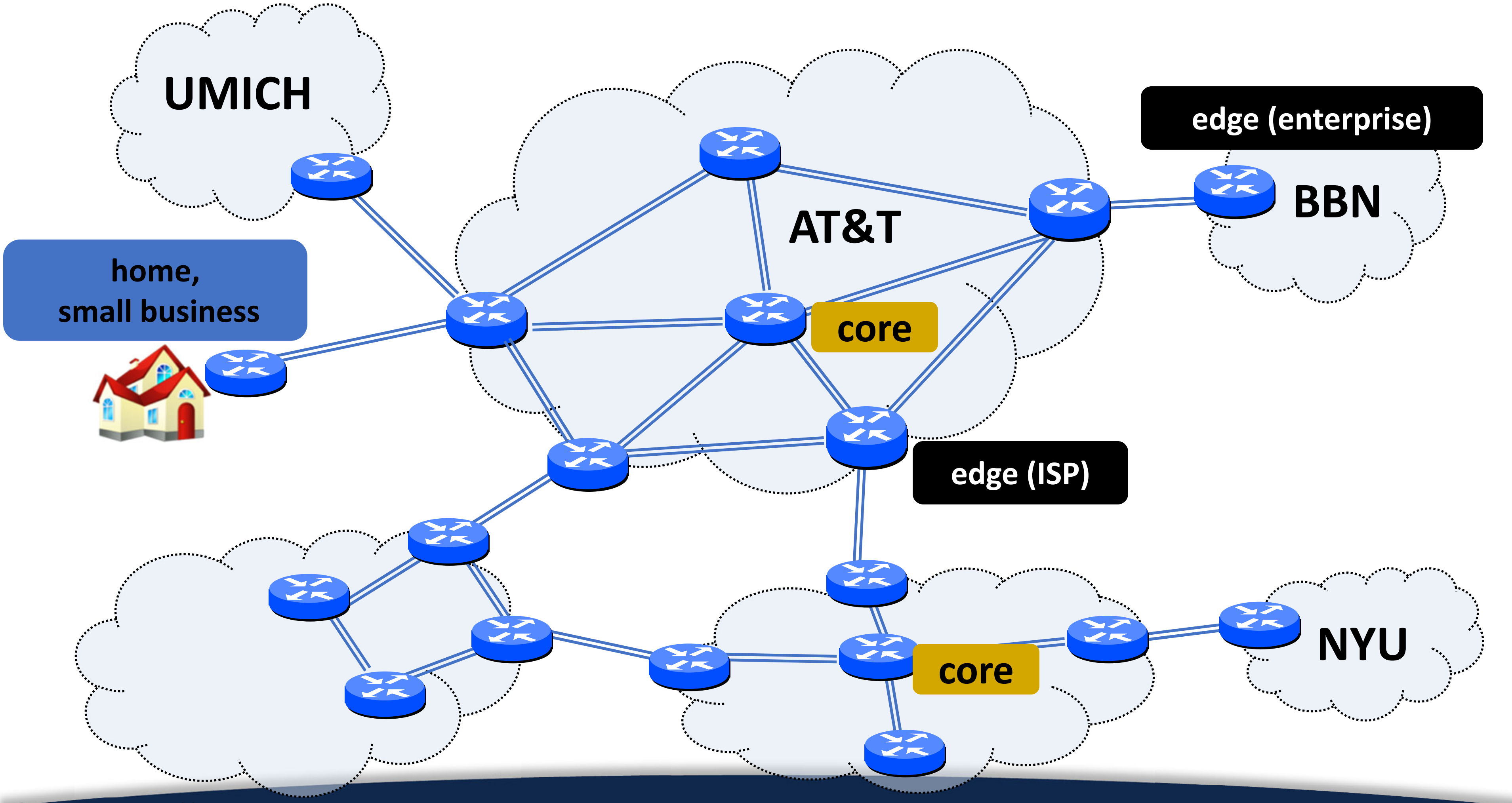
IP routers

- Core building block of the Internet infrastructure
- $120B+ industry
- Vendors: Cisco, Huawei, Juniper, Alcatel-Lucent (account for >90%)

Router definitions

- Router capacity = N x R
- N = Number of external router “ports”
- R = Speed (“line rate”) of a port

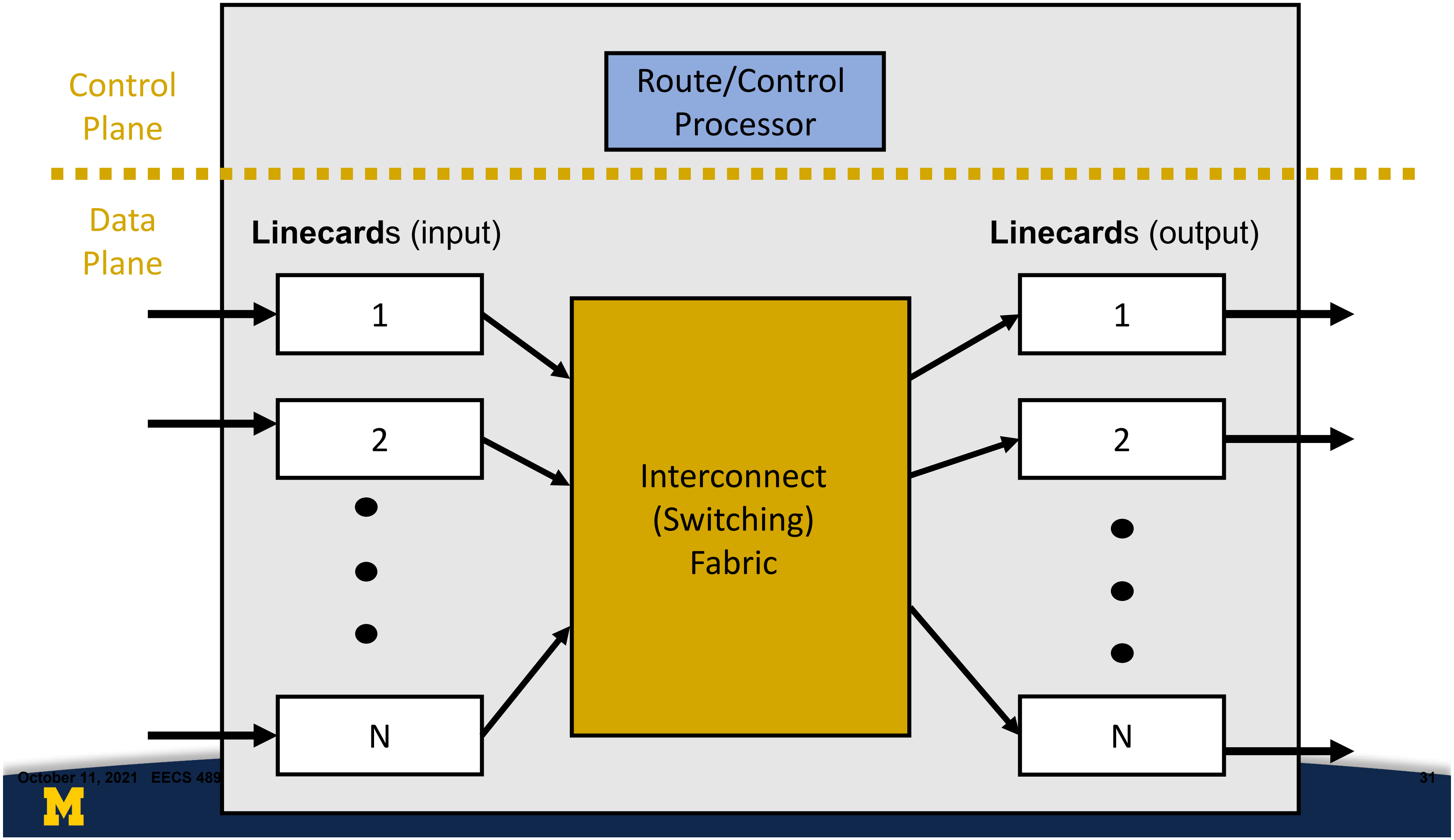
Networks and routers



Many types of routers

- Core
  - R = 10/40/100 Gbps
  - NR = O(100) Tbps (Aggregated)
- Edge
  - R = 1/10/40
  - NR = O(100) Gbps
- Small business
  - R = 10/100/1000 Mbps
  - NR < 10 Gbps

What’s inside a router?



What’s inside a router?

- Linecards
  - Input linecards process packets on their way in
  - Output linecards process packets on way out
  - Input and output for the same port are on the same physical linecard
- Interconnect/switching fabric
  - Transfers packets from input to output ports

Input linecards

- Tasks
  - Receive incoming packets (physical layer stuff)
  - Update the IP header
    - TTL, Checksum, Options and Fragment (maybe)
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- Challenge: speed!
  - 100B packets @ 40Gbps → new packet every 20 nano secs!
  - Typically implemented with specialized ASICs (network processors)

Looking up the output port

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

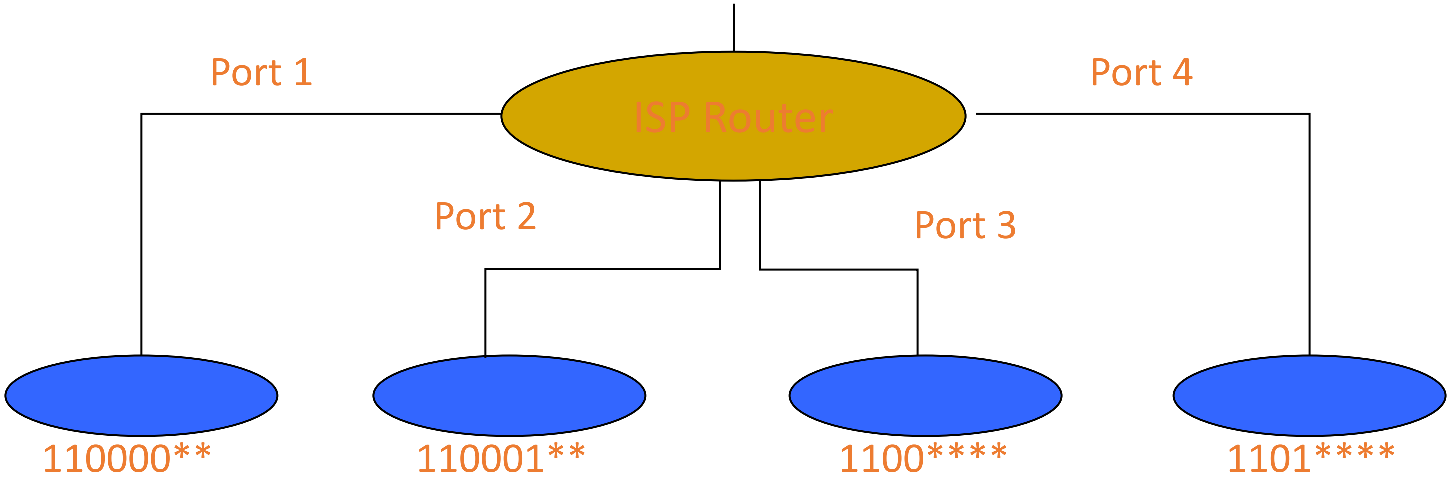
Example

- Router with 4 ports
- Destination address range mapping
  - 11 00 00 00 to 11 00 00 11: Port 1
  - 11 00 01 00 to 11 00 01 11: Port 2
  - 11 00 10 00 to 11 00 11 11: Port 3
  - 11 01 00 00 to 11 01 11 11: Port 4

Example

- Router with 4 ports
- Destination address range mapping
  - 11 00 00 00 to 11 00 00 11: Port 1
  - 11 00 01 00 to 11 00 01 11: Port 2
  - 11 00 10 00 to 11 00 11 11: Port 3
  - 11 01 00 00 to 11 01 11 11: Port 4

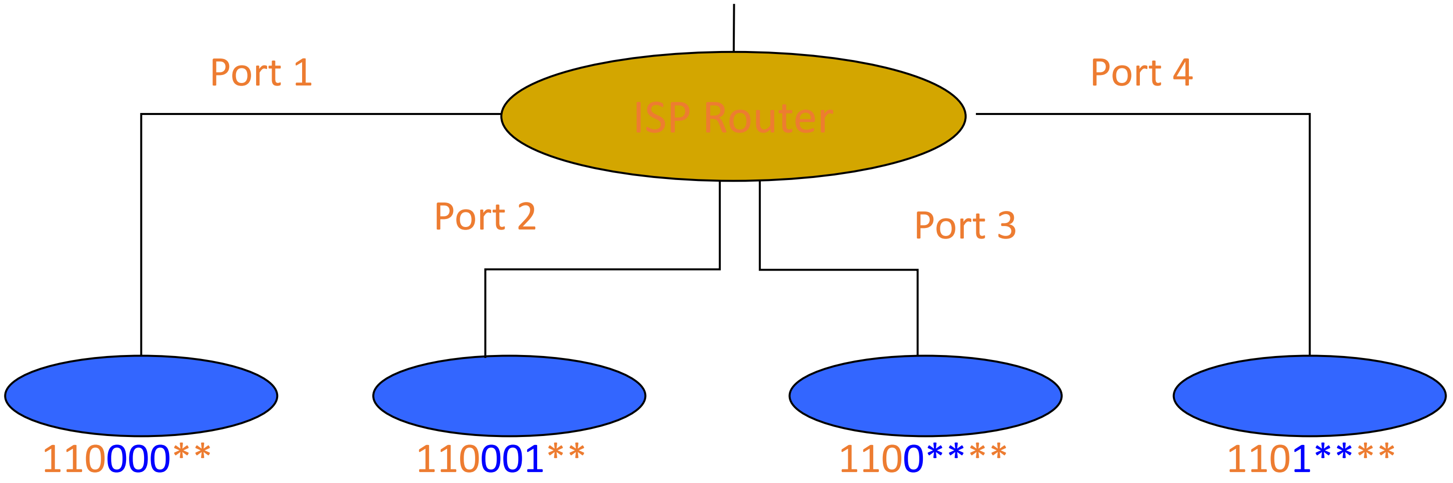
Longest prefix matching



Finding match efficiently

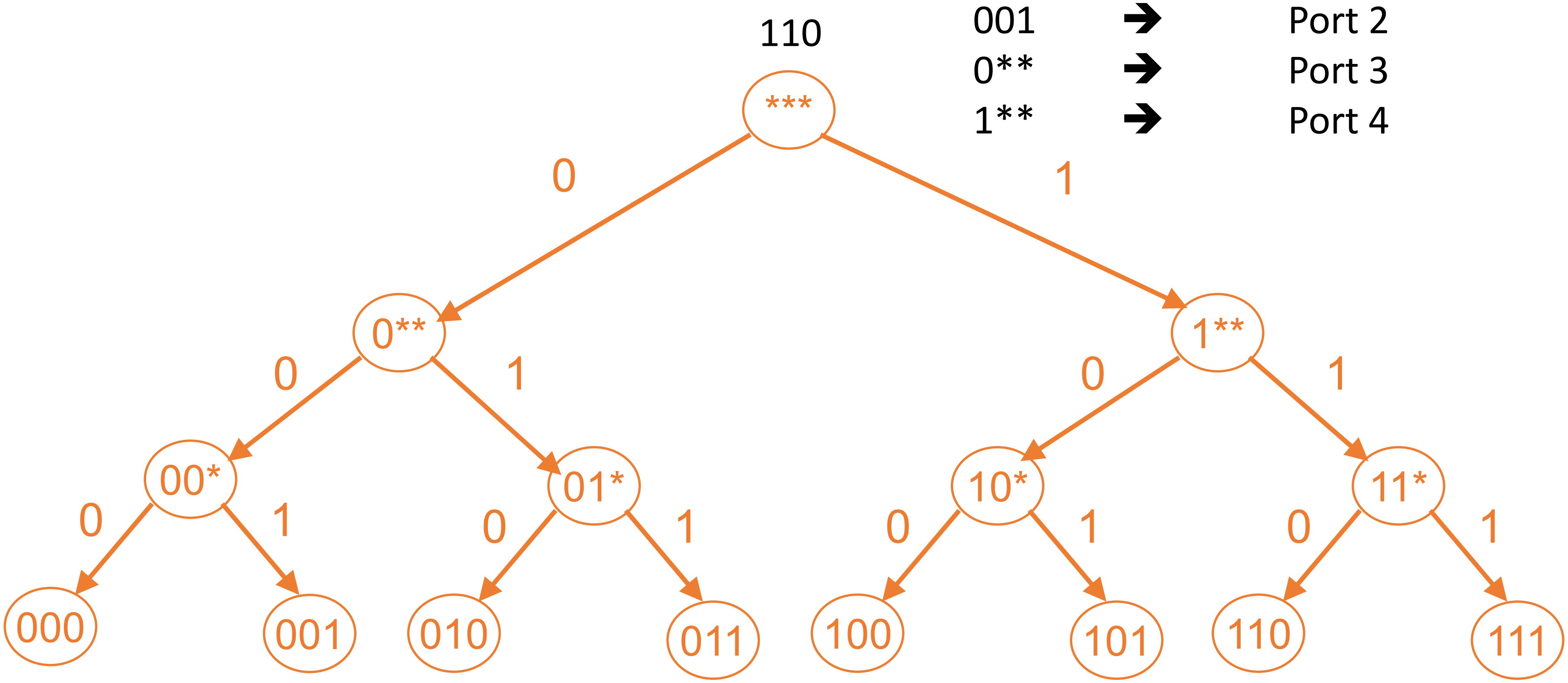
- Testing each entry to find a match scales poorly
  - On average: O(number of entries)
- Leverage tree structure of binary strings
  - Set up tree-like data structure

Longest prefix matching (LPM)



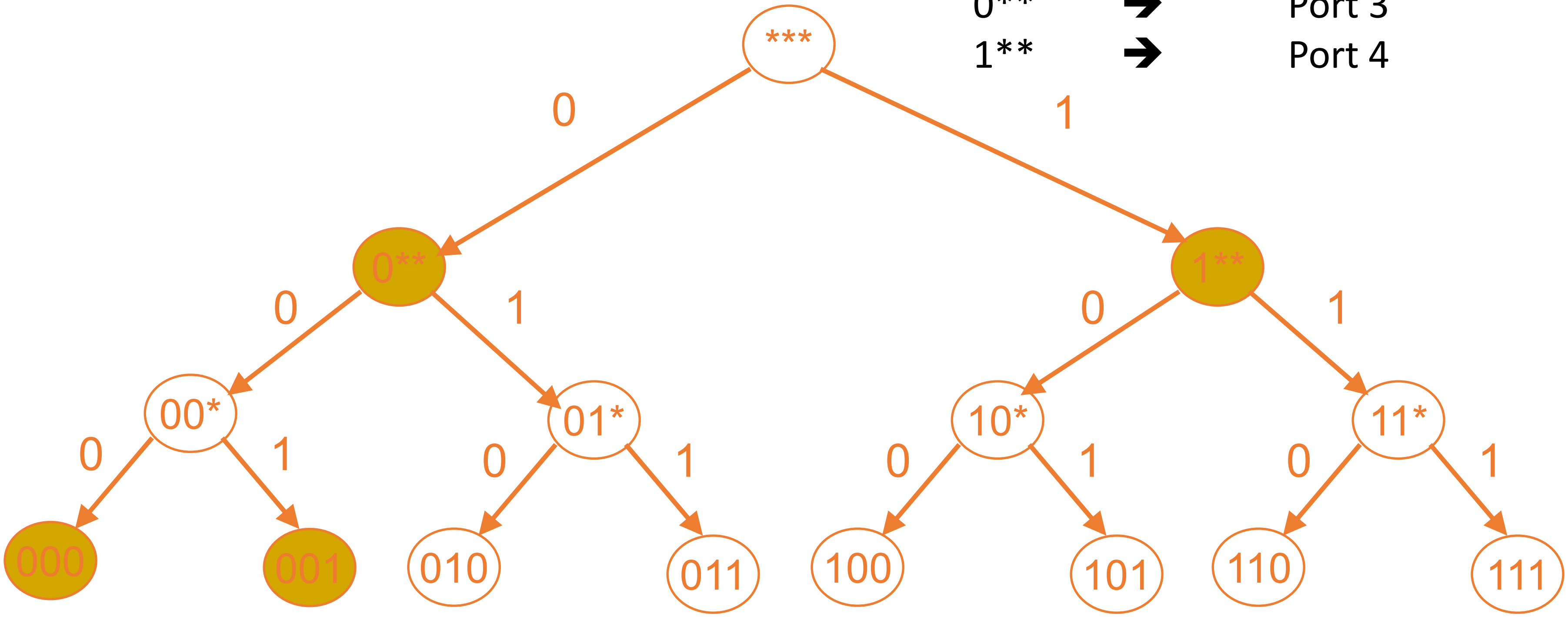
Tree structure

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



Tree structure

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



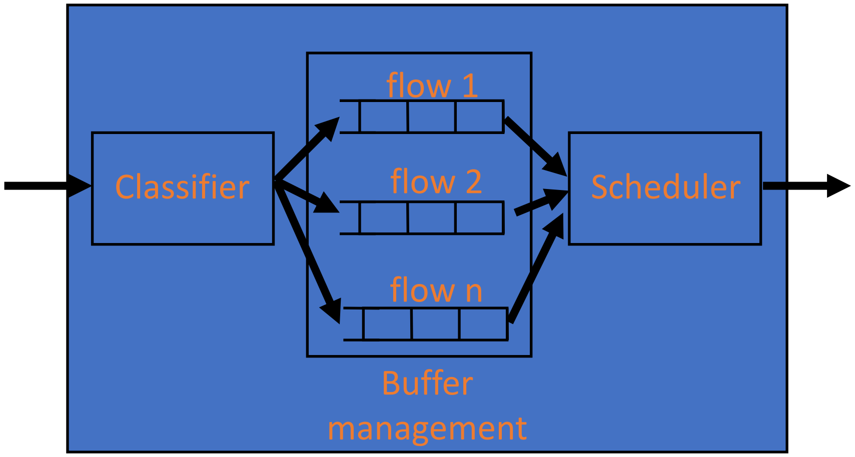
Record port associated with latest match, and only override when it matches another prefix during walk down tree

Input linecards

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

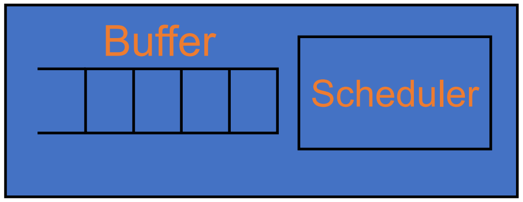
Output linecards

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



Simplest: FIFO router

- No classification
- Drop-tail buffer management: when buffer is full drop the incoming packet
- First-In-First-Out (FIFO) Scheduling: schedule packets in the same order they arrive



Packet classification

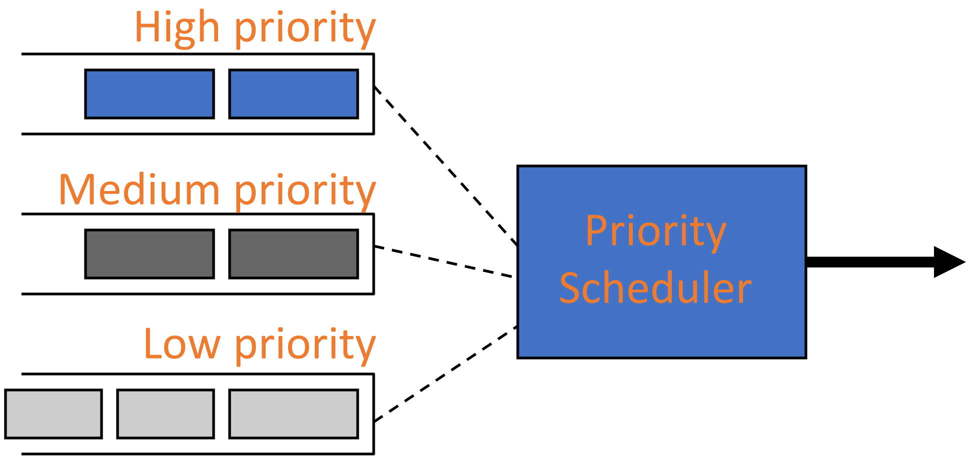
- Classify an IP packet based on a number of fields in the packet header, e.g.,
  - Source/destination IP address (32 bits)
  - Source/destination TCP port number (16 bits)
  - Type of service (TOS) byte (8 bits)
  - Type of protocol (8 bits)
- In general fields are specified by range
  - Classification requires a multi-dimensional range search!

Scheduler

- One queue per “flow”
- Scheduler decides when and from which queue to send a packet
- Goals of a scheduling algorithm
  - Fast!
  - Depends on the policy being implemented (fairness, priority, etc.)

Priority scheduler

- Priority scheduler: packets in the highest priority queue are always served before the packets in lower priority queues



Round-robin scheduler

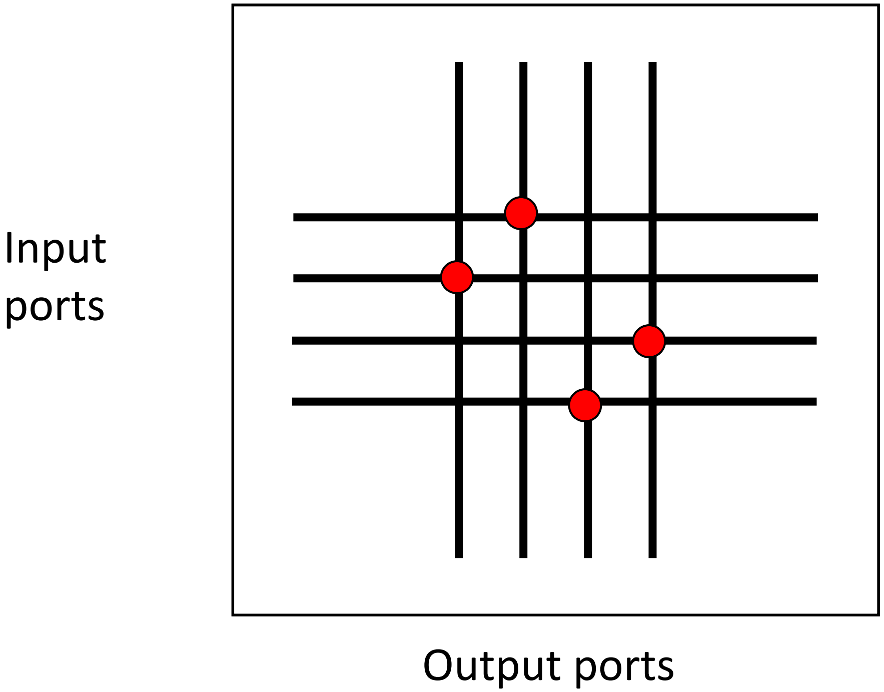
- Round robin: packets are served from each queue in turn
- Fair queuing (FQ): round-robin for packets of different size
- Weighted fair queueing (WFQ): serve proportional to weight
  - FQ gives equal weight to each flow

Connecting inputs to outputs: Switching fabric

- Mini-network
- Three primary ways to switch
  - Switching via shared memory
  - Switching via a bus
  - Switching via an inter-connection network
    - For example, cross-bar

Crossbar fabric

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



Router-assisted Congestion control

Recap: TCP problems

- Misled by non-congestion losses
- Fills up queues leading to high delays

Routers tell endpoints if they're congested

- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Saw tooth discovery too choppy for some apps

Routers tell endpoints what rate to send at

- Unfair under heterogeneous RTTs
- Tight coupling with reliability mechanisms
- End hosts can cheat

Routers enforce fair sharing

Could fix many of these with some help from routers!

Router-assisted congestion control

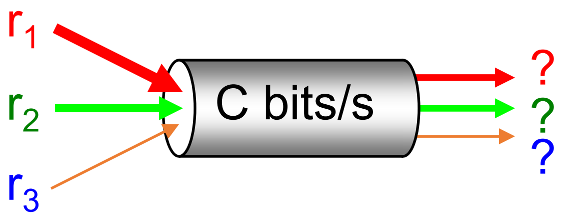
- Three tasks for congestion control
    - Isolation/fairness
    - Adjustment
    - Detecting congestion

Fairness: General approach

- Routers classify packets into “flows”
  - Let’s assume flows are TCP connections
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
  - When line becomes free, take packet from next flow in a fair order
- What does “fair” mean exactly?

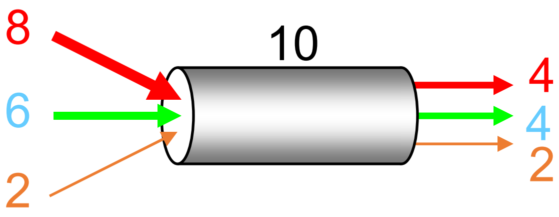
Max-Min fairness

- 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 →
  - 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 →
  - 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

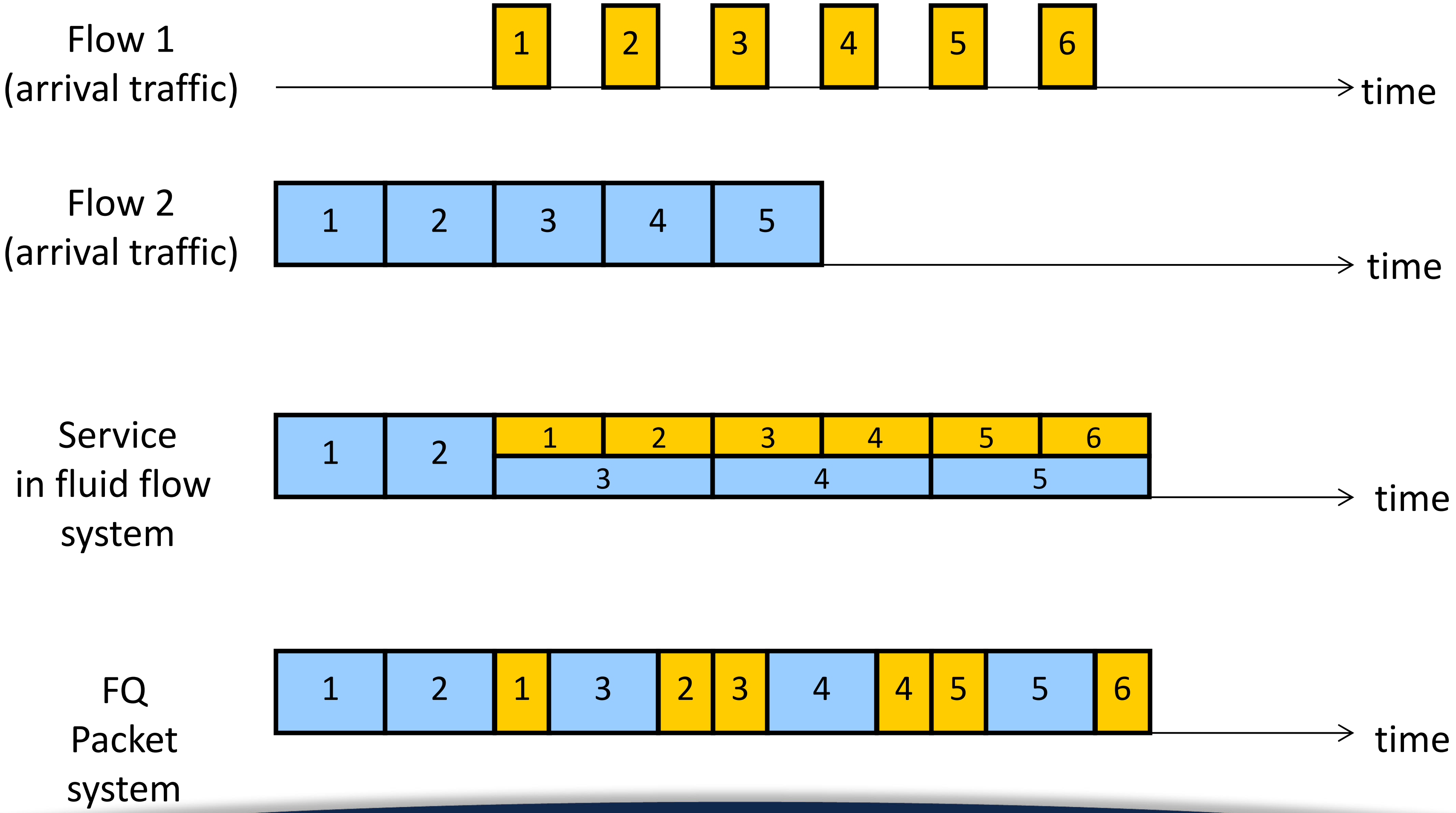
How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin (“fluid flow”)
- Can you do this in practice?
  - No, packets cannot be preempted
- But we can approximate it
  - This is what “fair queuing” routers do

Fair Queuing (FQ)

- For each packet, compute the time at which the last bit of a packet would have left the router if flows are served bit-by-bit
- Then serve packets in the increasing order of their deadlines

Example


Flow 1
(arrival traffic)
1
2
3
4
5
6
time
Flow 2
(arrival traffic)
1
2
3
4
5
time
Service
in fluid flow
system
1
2
1
2
3
4
5
6
3
4
5
time
FQ
Packet
system
1
2
1
3
2
3
4
4
5
5
6
time

Fair Queuing (FQ)

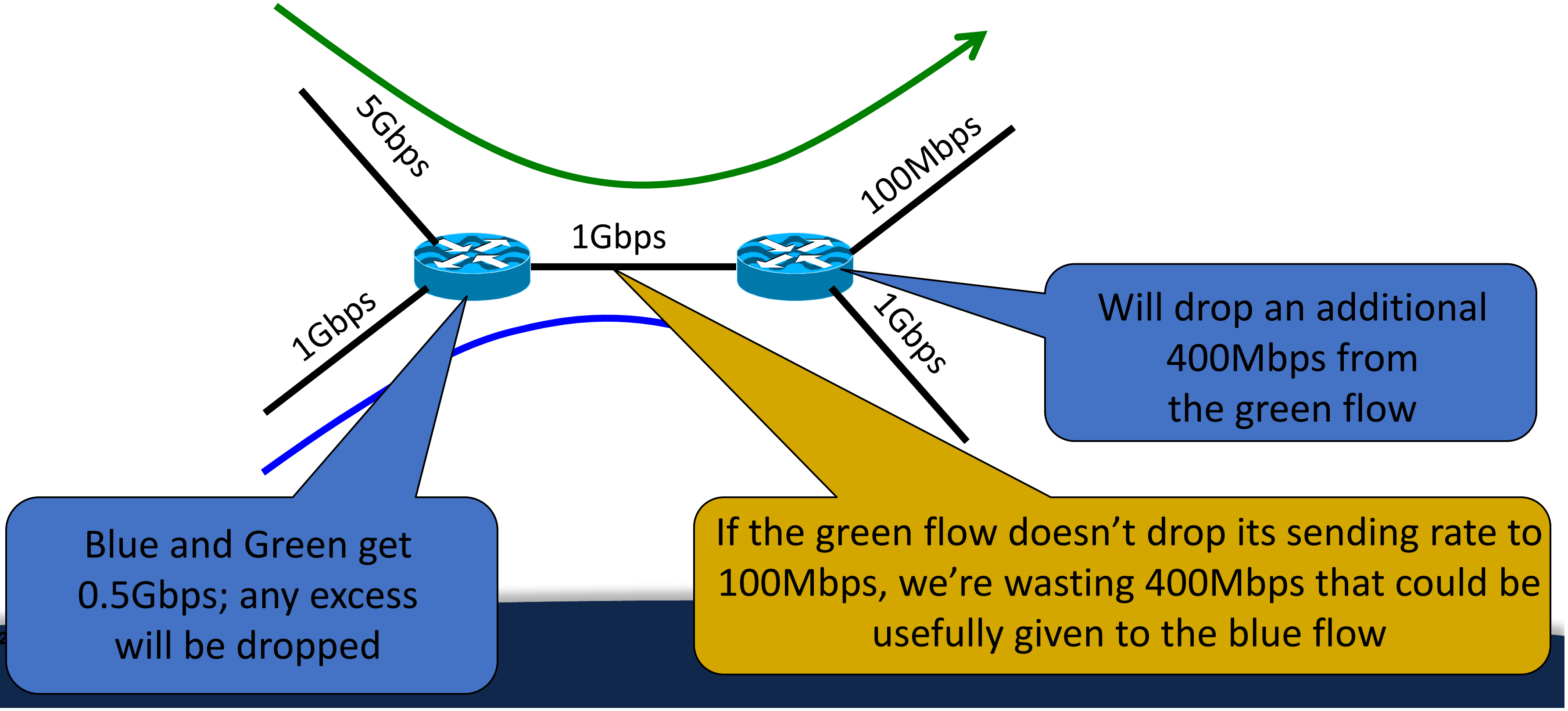
- Implementation of round-robin generalized to the case where not all packets are equal sized
- Weighted fair queuing (WFQ): assign different flows different shares
- Today, some form of WFQ implemented in almost all routers
  - Not the case in the 1980-90s, when CC was being developed
  - Mostly used to isolate traffic at larger granularities (e.g., per-prefix)

FQ vs. FIFO

- FQ advantages:
  - Isolation: cheating flows don’t benefit
  - Bandwidth share does not depend on RTT
  - Flows can pick any rate adjustment scheme they want
- Disadvantages:
  - More complex than FIFO: per flow queue/state, additional per-packet book-keeping

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion



Blue and Green get 0.5Gbps; any excess will be dropped

If the green flow doesn't drop its sending rate to 100Mbps, we're wasting 400Mbps that could be usefully given to the blue flow

Will drop an additional 400Mbps from the green flow

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion
  - Robust to cheating, variations in RTT, details of delay, reordering, retransmission, etc.
- But congestion (and packet drops) still occurs
- We still want end-hosts to discover/adapt to their fair share!
- What would the end-to-end argument say w.r.t. congestion control?

Fairness is a controversial goal

- What if you have 8 flows, and I have 4?
  - Why should you get twice the bandwidth?
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Why shouldn't you be penalized for using more scarce bandwidth?
- What is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?

Router-Assisted Congestion Control

- CC has three different tasks:
  - Isolation/fairness
  - Rate adjustment
  - Detecting congestion

Why not let routers tell what rate end hosts should use?

- Packets carry “rate field”
- Routers insert “fair share” f in packet header
- End-hosts set sending rate (or window size) to f
  - Hopefully (still need some policing of end hosts!)
- This is the basic idea behind the “Rate Control Protocol” (RCP) from Dukkipati et al. ’07
  - Flows react faster

Router-Assisted Congestion Control

- CC has three different tasks:
  - Isolation/fairness
  - Rate adjustment
  - Detecting congestion

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
  - Tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
  - i.e., end-host reacts as though it saw a drop

ECN

- Advantages:
  - Don’t confuse corruption with congestion; recovery w/ rate adjustment
  - Can serve as an early indicator of congestion to avoid delays
  - Easy (easier) to incrementally deploy
    - Today: defined in RFC 3168 using ToS/DSCP bits in the IP header
    - Common in datacenters

Summary

- IP routers form the backbone of the Internet
- Aims for speed while providing fairness
- Routers can assist in addressing/mitigating many of TCP’s shortcomings

Bonus Quiz 10 – IP Routers
Due Wednesday at midnight

https://forms.gle/8fKCHXMDESngfkxD6