

# CS168: Midterm Review

...

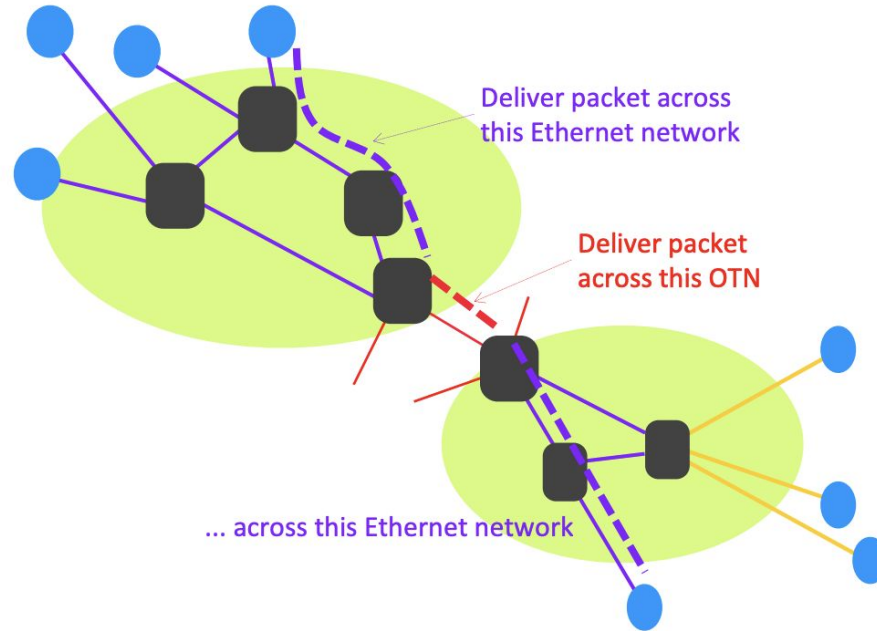
Intro to the Internet  
Spring 2024

# Agenda

- Topical Review [~1 hour]
- Review Worksheet [~50 minutes]
- Disclaimer: this review session is **not comprehensive**
  - Summarizes all the topics we've studied so far this semester, but may gloss over some of the finer details

# Internet Architecture

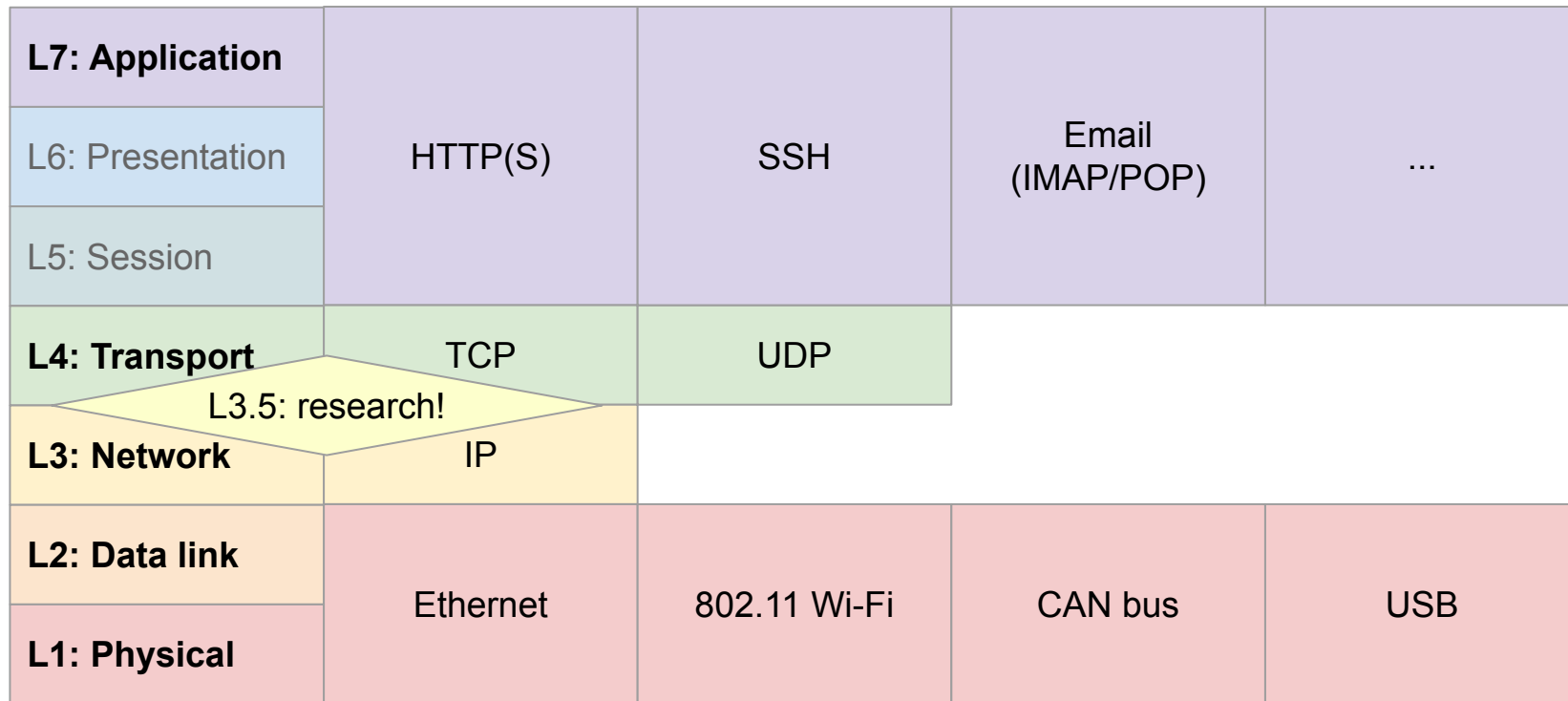
# Local vs. Global Delivery



# Layering

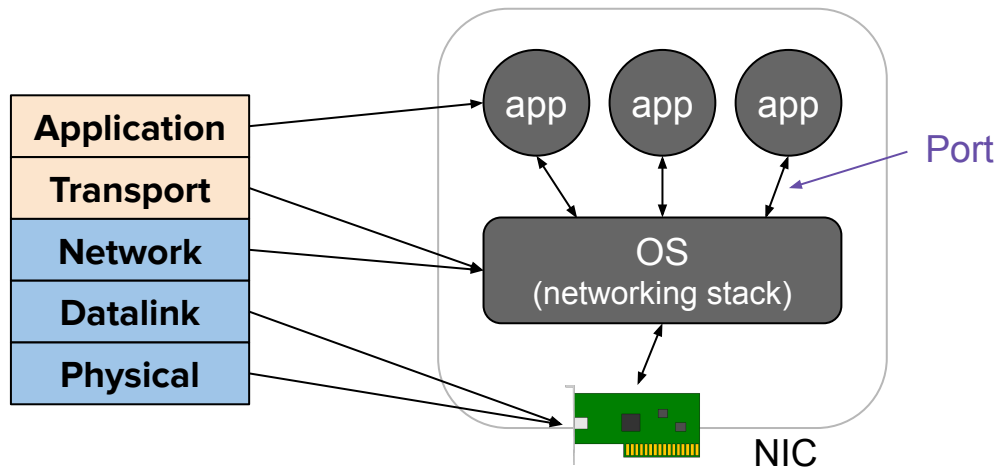
<b>L7: Application</b>	do the thing
L6: Presentation	(ignored here)
L5: Session	(ignored here)
<b>L4: Transport</b>	<b>beyond</b> delivery: (un)reliability, packet assembly, congestion control, ...
<b>L3: Network</b>	<b>global</b> delivery, best-effort
<b>L2: Data link</b>	<b>local</b> delivery, best-effort
<b>L1: Physical</b>	physical transfer of bits

# Layering in practice



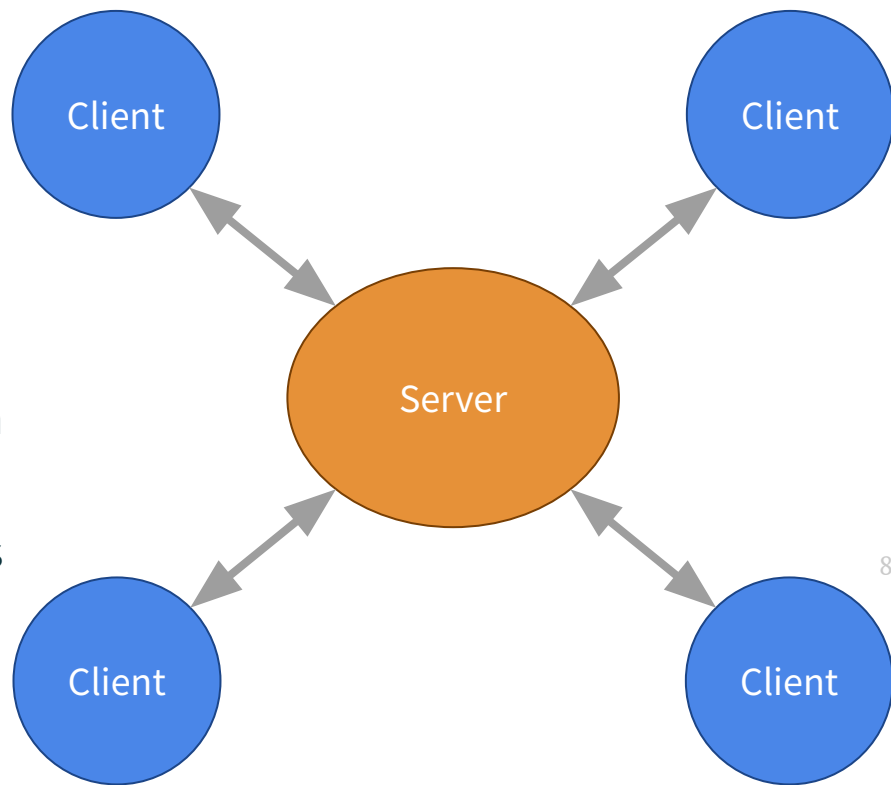
# Sockets

- Developed here, at UC Berkeley!
- OS abstraction for **connections**
- Allow L7 applications to operate on data streams (not packets)
  - Connect, listen, accept, send, receive
- Open a socket between:
  - Source IP address : *port*
  - Destination IP address : *port*



# Connections

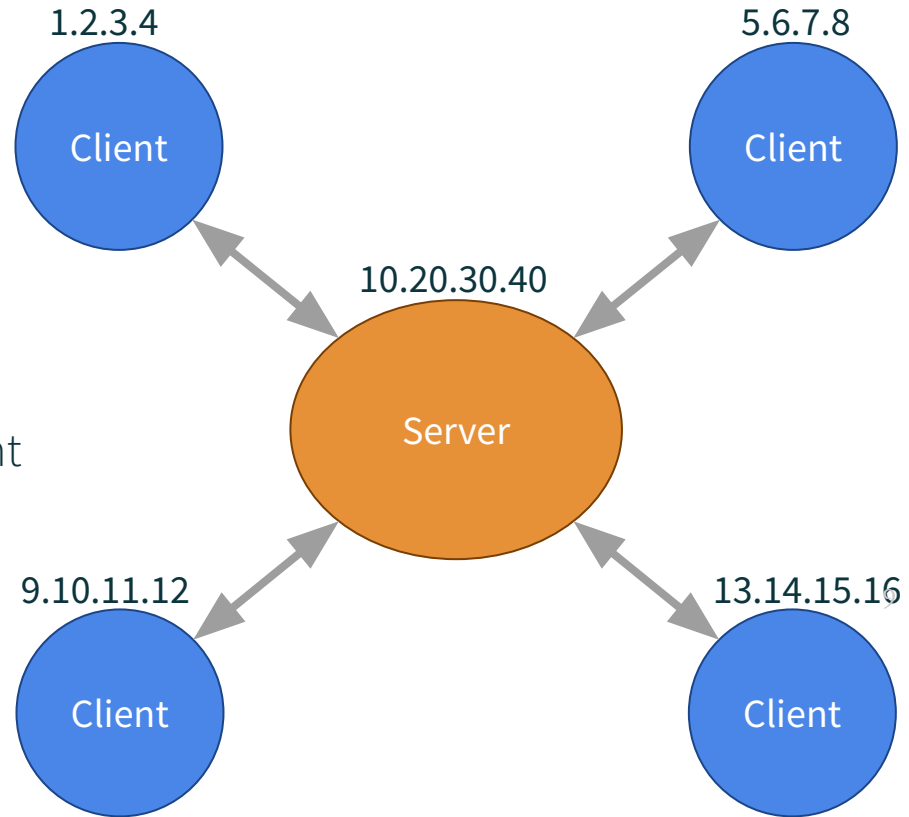
- Two types of sockets
  - Server and Client
- Servers *listen* for clients to connect to them
  - Wait until a connection is attempted
    - Accept and dispatch connection
  - Usually serving many clients at once
- Clients *initiate* new connections to servers
- Example
  - Server: berkeley.edu
  - Client: Your internet browser





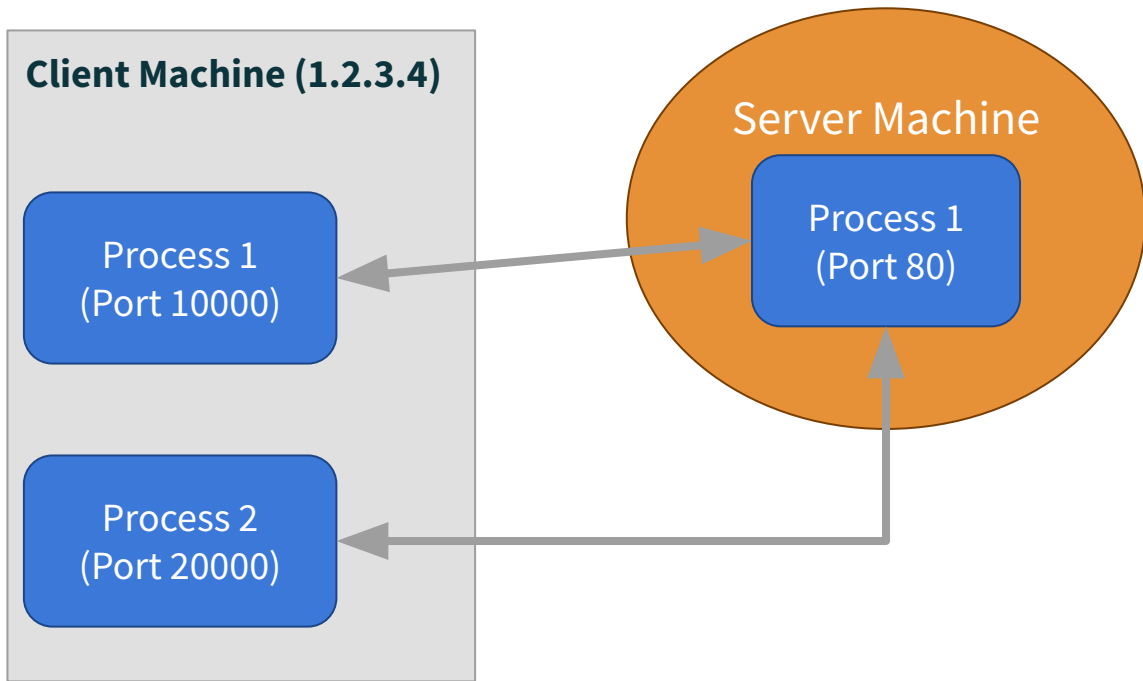
# Connections

- Hosts have addresses
  - Unique identifier (just like a street address)
- Clients (different users) find servers with their addresses
  - Servers send data back with the client address



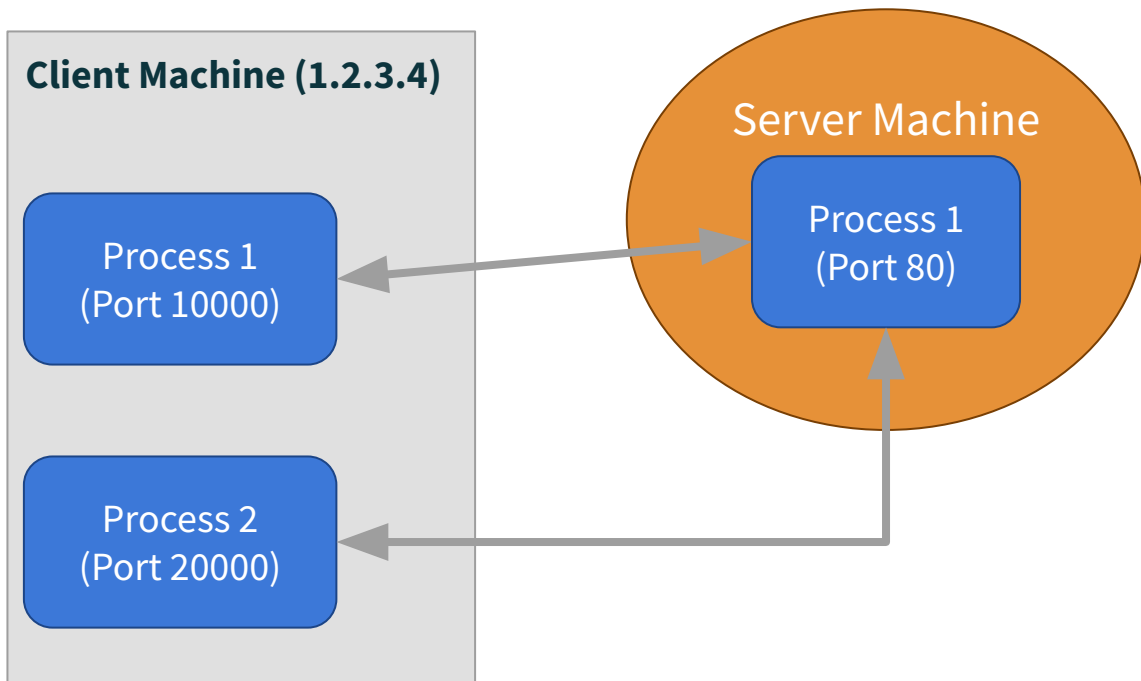
# Ports

- Sockets are identified by unique IP:port pairs
- A port is a number that the OS associates with a socket when it is created
  - i.e. sending to address “1.2.3.4:10000” would send data to the socket owned by Process 1



# Ports

- Packets carry port number
- Servers listen on a port
  - Which one depends on application
  - HTTP: 80
  - SSH: 22
- Client process connects to well known port
- Client also has a port
  - Randomly assigned by OS
  - Used by OS to send data to correct process



# Delays

# Delays

- How long does it take for your packet to travel through the network?
- It depends on...
  - how much data you're sending and the link speed  
→ transmission delay
  - your distance from the destination  
→ propagation delay
  - the traffic pattern  
→ queuing delay

# Delays

## Transmission Delay:

- How long it takes for the all bits in the packet to get on the wire
  - The time between when the first and last bits enter the link
- Limited by the **bandwidth**
  - **Bandwidth:** Number of bits you can send through a wire per unit of time

## Propagation Delay:

- End-to-end transmission time of one bit
- Depends on the **length of the link**
- Does NOT depend on the size of the packet

$$\text{Transmission Delay} = \frac{\text{packet size (bytes)}}{\text{bandwidth}}$$

$$\text{Propagation Delay} = \frac{\text{length of link (meters)}}{\text{speed of light (meters/second)}}$$

# Bandwidth Delay Product (BDP)

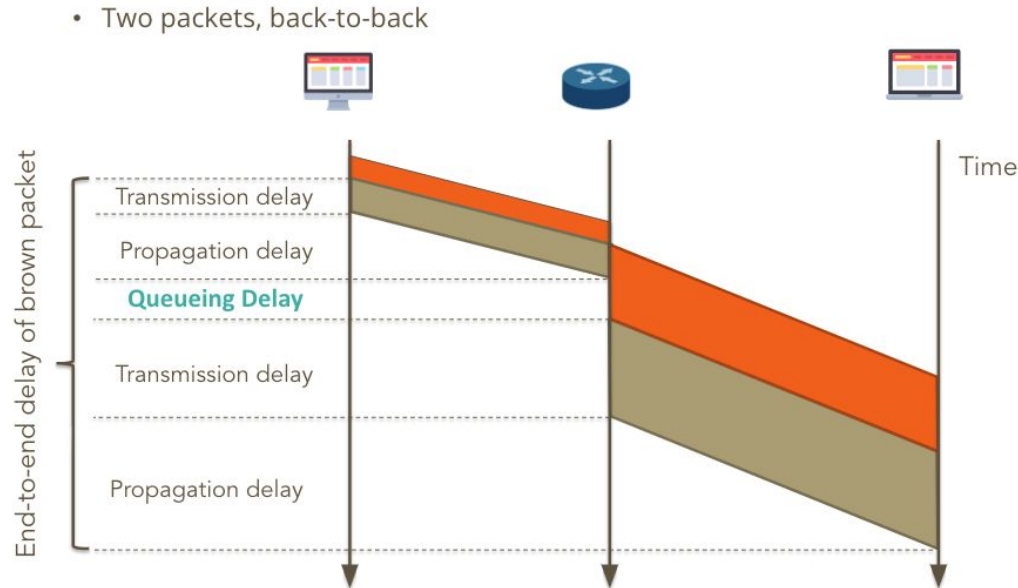
- Now that we know the propagation delay, we can tell how many bits are “in flight” (on the link) at any time



$$\text{Bandwidth Delay Product (BDP)} = \text{Bandwidth} \cdot \text{Propagation Delay}$$

# Queuing Delay

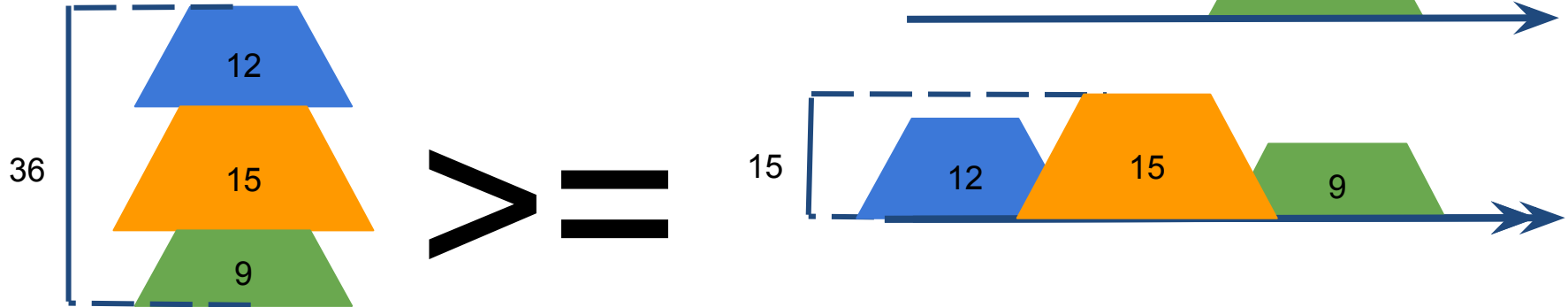
- How long the packet waits to get transmitted on the wire
- Happens only when arrival rate is greater than transmission rate





# Statistical Multiplexing

- Sum of the peaks is always greater than the peak of the sums (peak of the aggregate). Usually much greater.



# Routing Concepts

# Distance-Vector Routing

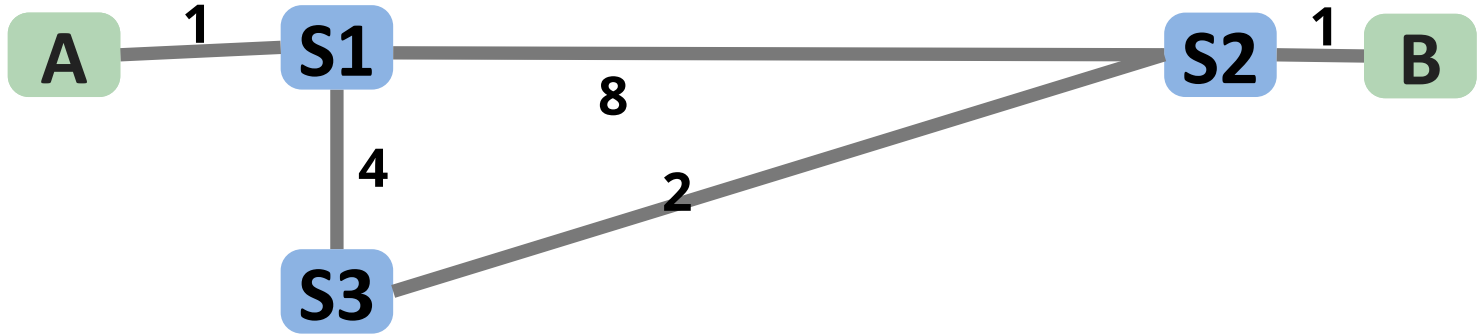
- Each router:
  - Has a map of **destinations** to **next hop/distance**
  - Connected hosts are **statically programmed** into the table

Destination	NextHop, Dist.
X	Direct, 1
Y	S1, 8
Z	S2, 10

# D-V Review

Destination	NextHop, Dist.
A	Direct, 1

Destination	NextHop, Dist.
B	Direct, 1

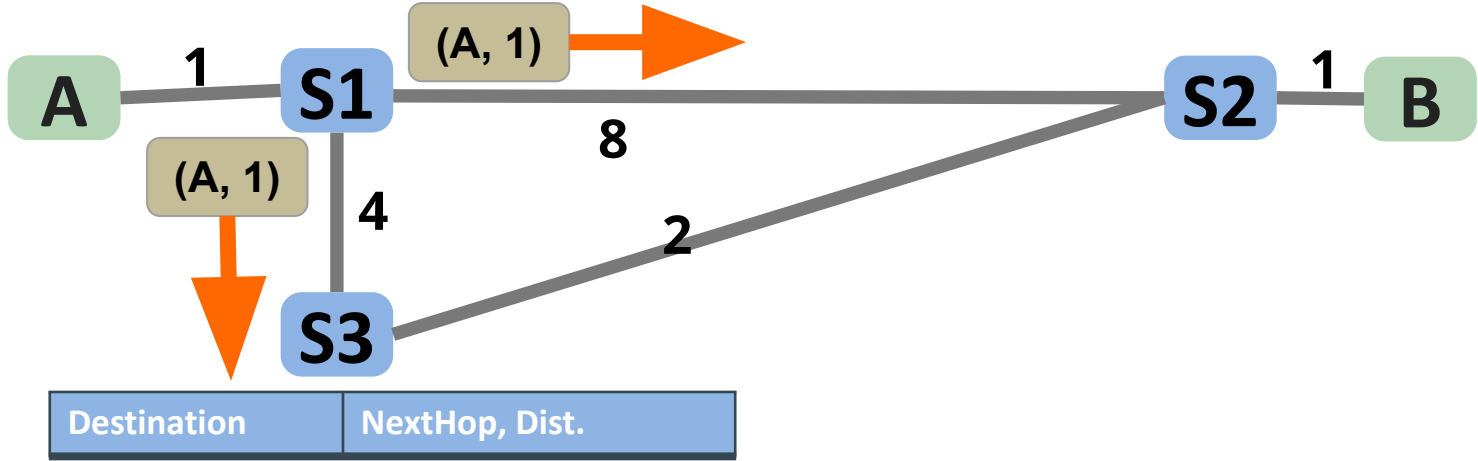


Destination	NextHop, Dist.
-------------	----------------

# D-V Review

Destination	NextHop, Dist.
A	Direct, 1

Destination	NextHop, Dist.
B	Direct, 1

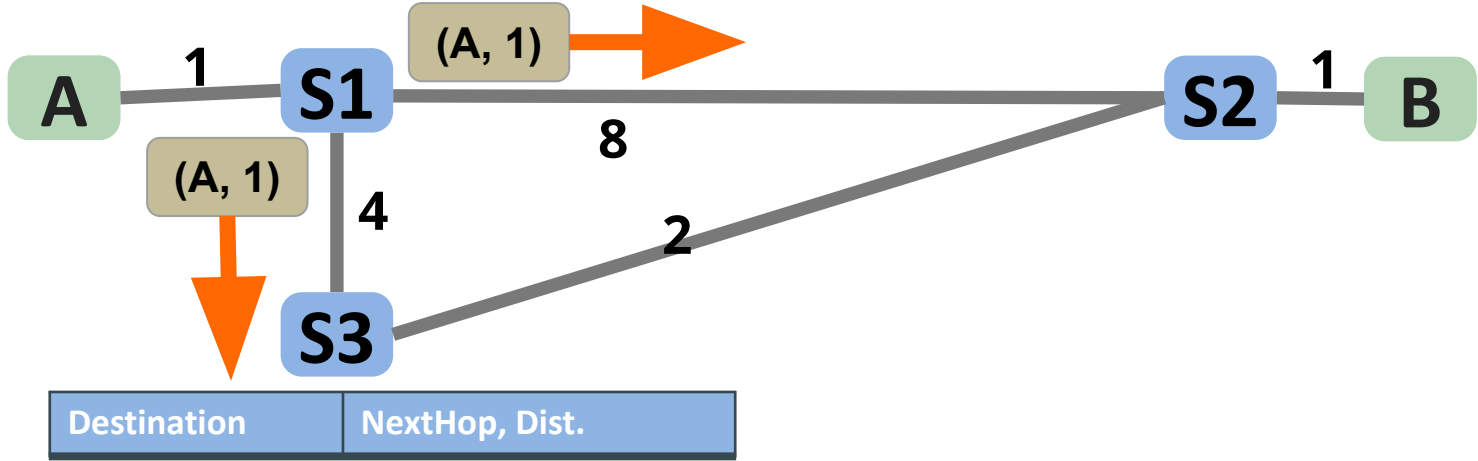


# D-V Review

Destination	NextHop, Dist.
A	Direct, 1



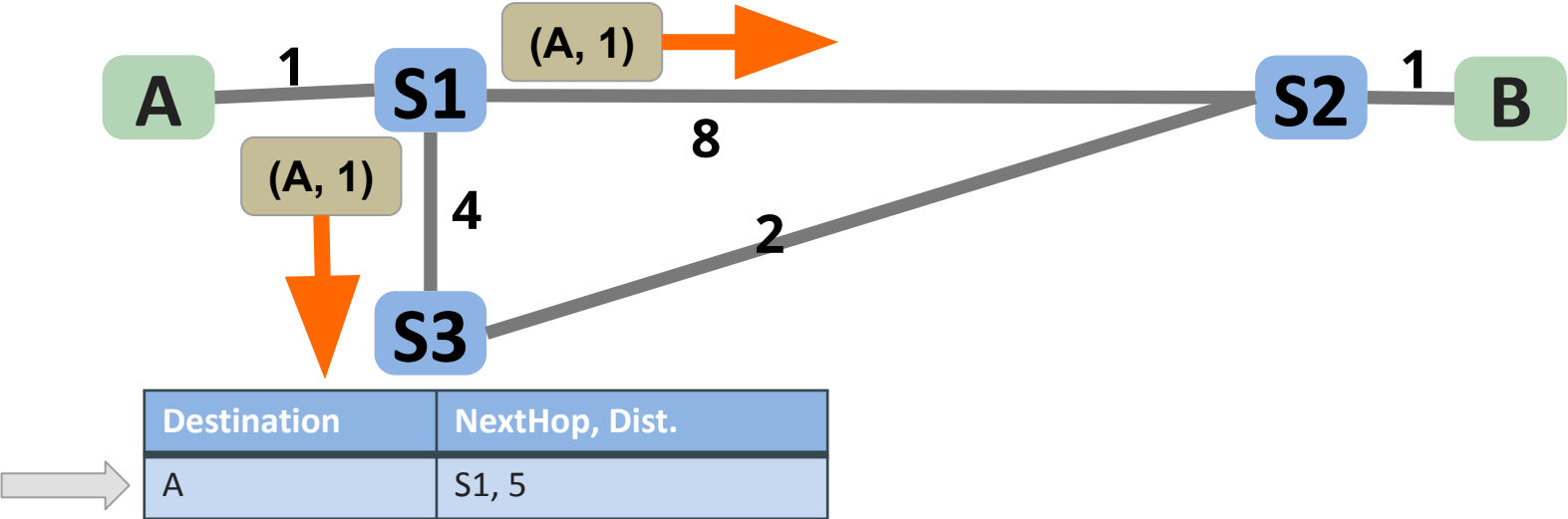
Destination	NextHop, Dist.
B	Direct, 1
A	S1, 9



# D-V Review

Destination	NextHop, Dist.
A	Direct, 1

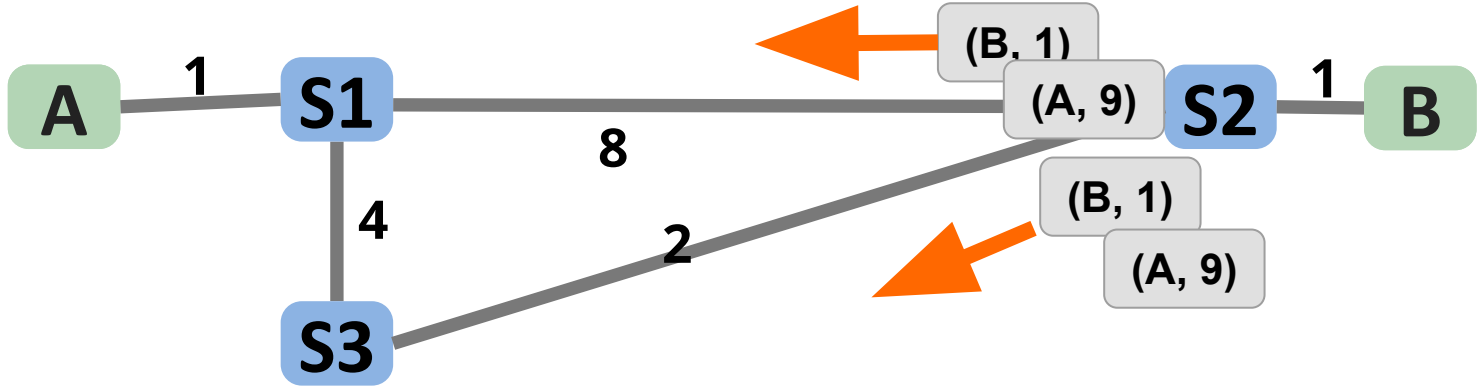
Destination	NextHop, Dist.
B	Direct, 1
A	S1, 9



# D-V Review

Destination	NextHop, Dist.
A	Direct, 1

Destination	NextHop, Dist.
B	Direct, 1
A	S1, 9



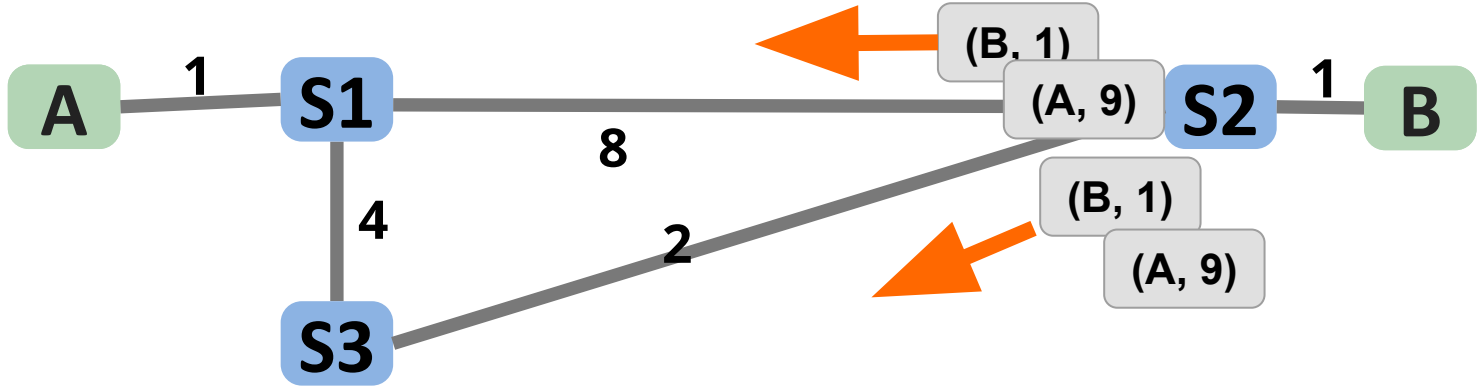
Destination	NextHop, Dist.
A	S1, 5



# D-V Review

Destination	NextHop, Dist.
A	Direct, 1
B	S2, 9

Destination	NextHop, Dist.
B	Direct, 1
A	S1, 9

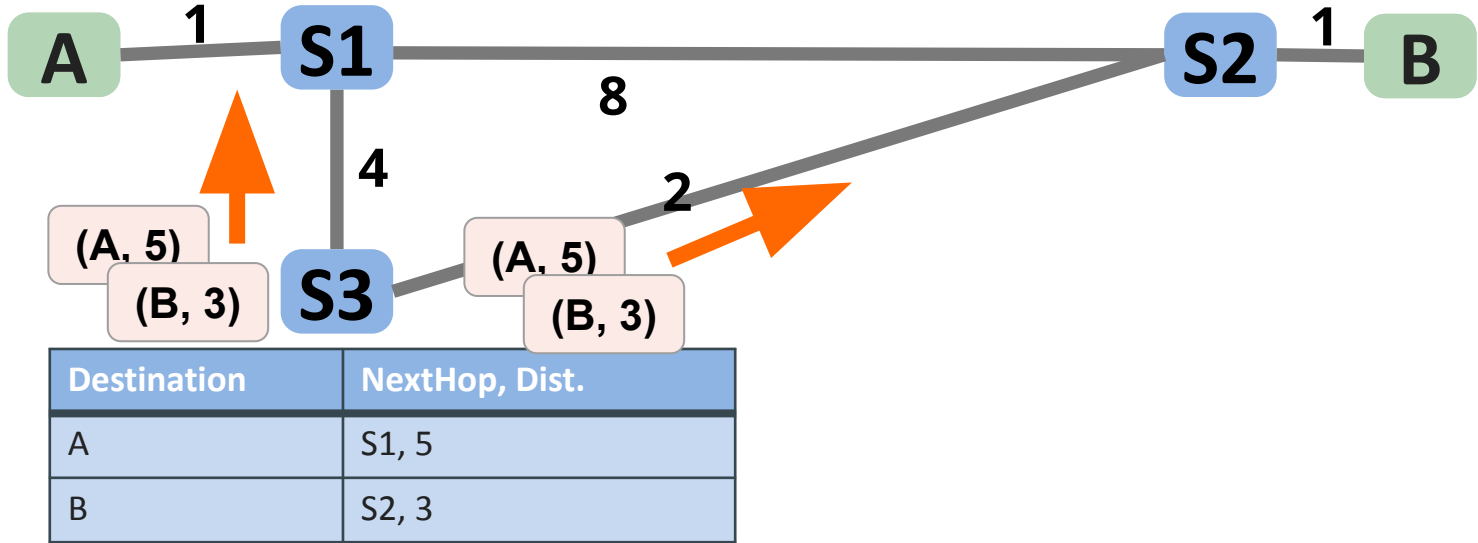


Destination	NextHop, Dist.
A	S1, 5
B	S2, 3

# D-V Review

Destination	NextHop, Dist.
A	Direct, 1
B	S2, 9

Destination	NextHop, Dist.
B	Direct, 1
A	S1, 9

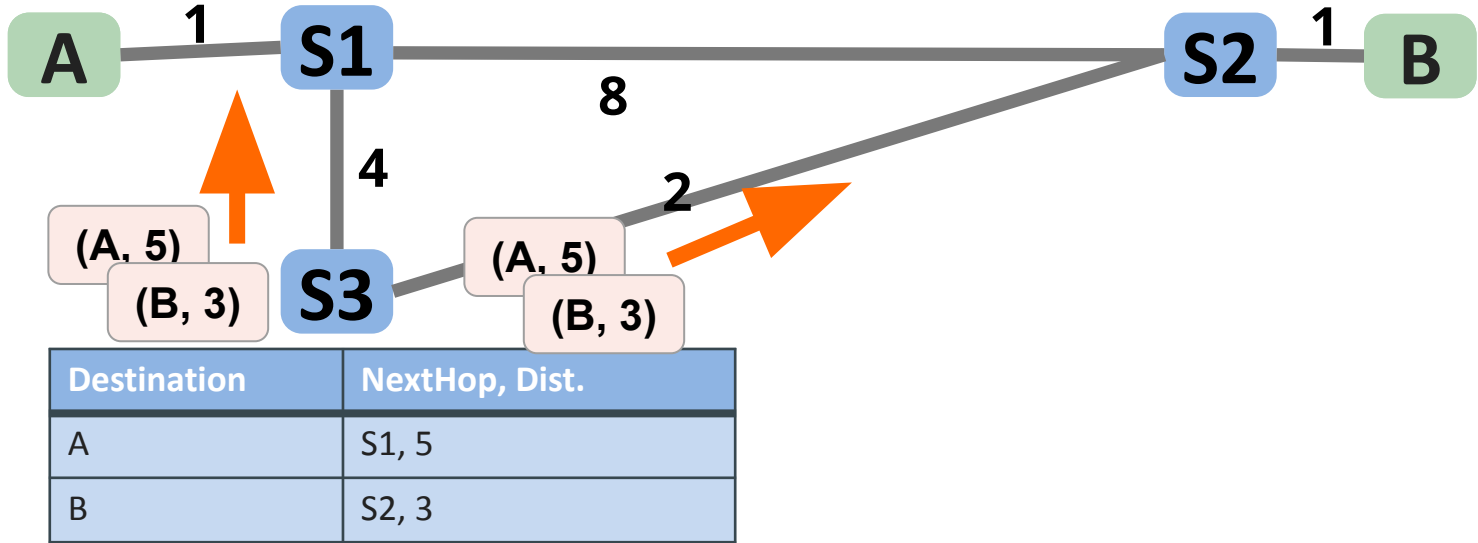


# D-V Review

Destination	NextHop, Dist.
A	Direct, 1
B	S3, 7



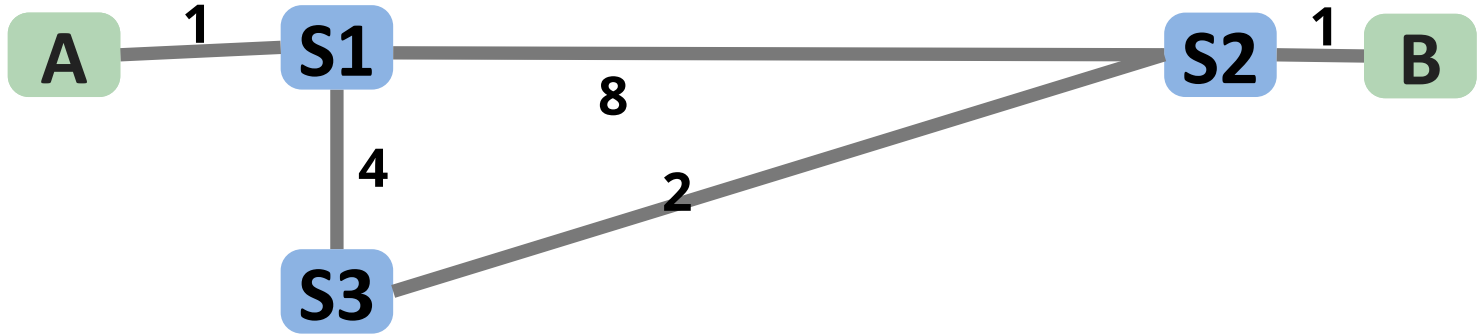
Destination	NextHop, Dist.
B	Direct, 1
A	S3, 7



# D-V Review (Converged)

Destination	NextHop, Dist.
A	Direct, 1
B	S3, 7

Destination	NextHop, Dist.
B	Direct, 1
A	S3, 7



Destination	NextHop, Dist.
A	S1, 5
B	S2, 3

# Distance-Vector Key Points

- Scalable
  - (unlike link-state) routers don't need global network topology
- Distributed
  - Routers communicate with neighbors to compute routes
- Minimizes Distance
  - Avoids loops and minimizes “distance,” not necessarily physical distance (really can minimize any value: price, latency, etc.)

# Protocol

## Each router advertises its routes to all of its neighbors...

- Periodically
  - Every so many seconds
- Whenever its table changes
  - Due to: new advertisements from neighbors, local link failure, new local link, route timeouts, ...

In *theory*, you only need the first.

In *practice*, the first alone is sufficient to eventually converge, but the second *isn't* if advertisements are dropped. Thus, the second acts like an *optimization*.

# Algorithm

Upon receiving advertisement from its neighbor, a router updates its table if one of the following is true:

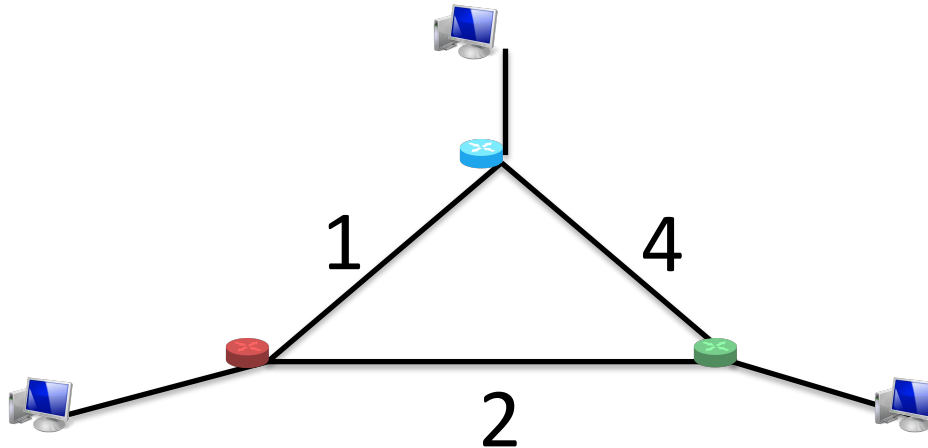
1. The destination isn't already in its table at all
2. The route in the table is worse than the one advertised
3. The advertiser is the current next hop

```
on new_advertisement A from neighbor N
    if ( A.dst not in table
        OR (A.dist + dist_to_neighbor(N)) < table[A.dst].dist
        OR table[A.dst].next_hop == N ):
        table[A.dst].dist = A.dist + dist_to_neighbor(N)
        table[A.dst].next_hop = N
```

# Link State Routing

Each router knows its own local “link state”:

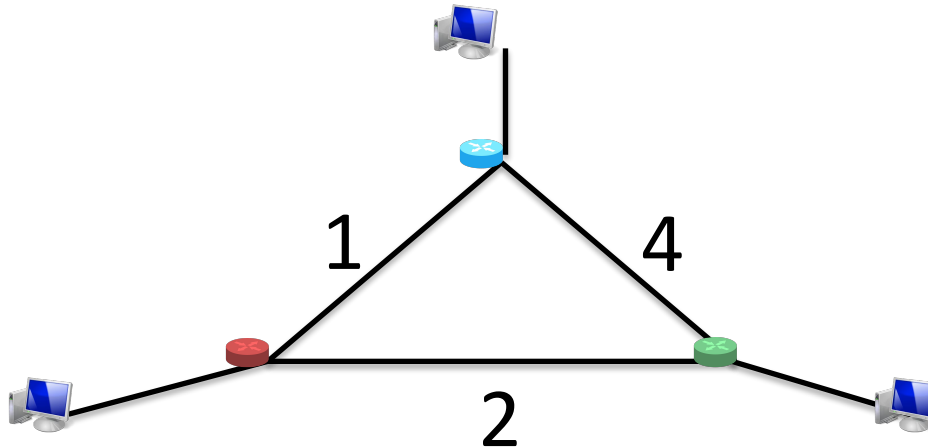
- State of each link to its neighbor (up/down)
- Associated costs





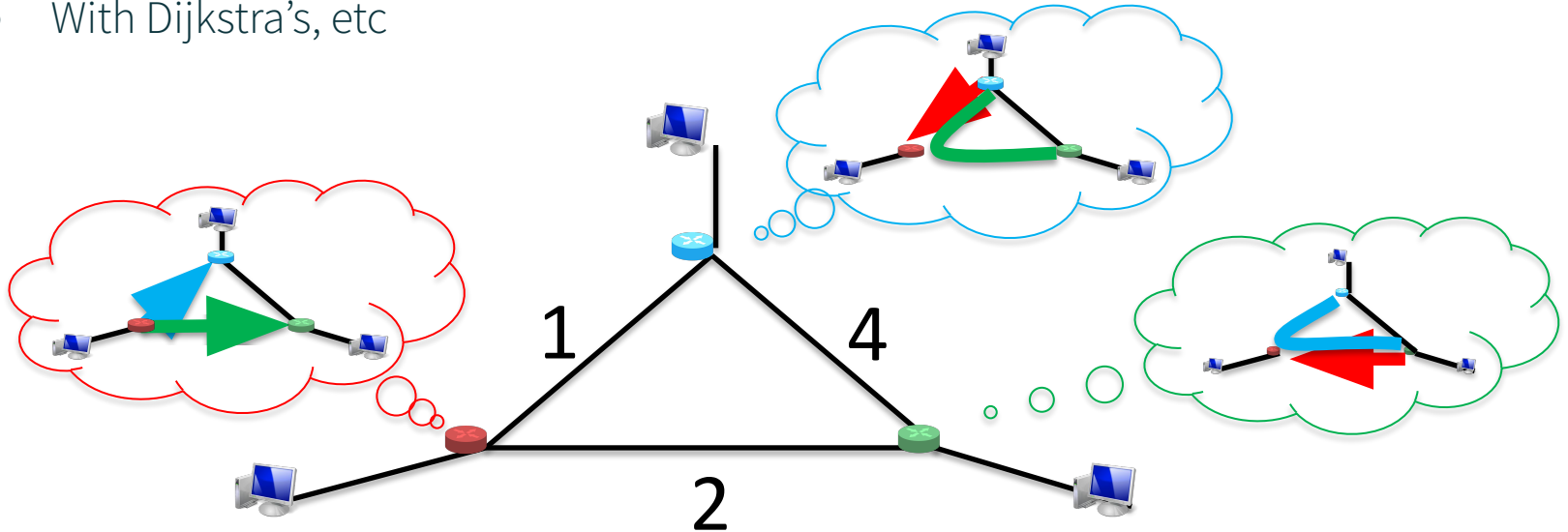
# Link State Routing

1. Router floods its link state to all other routers.
2. Each router learns global network topology
3. Then, computes shortest path themselves!
  - With Dijkstra's, etc



# Link State Routing

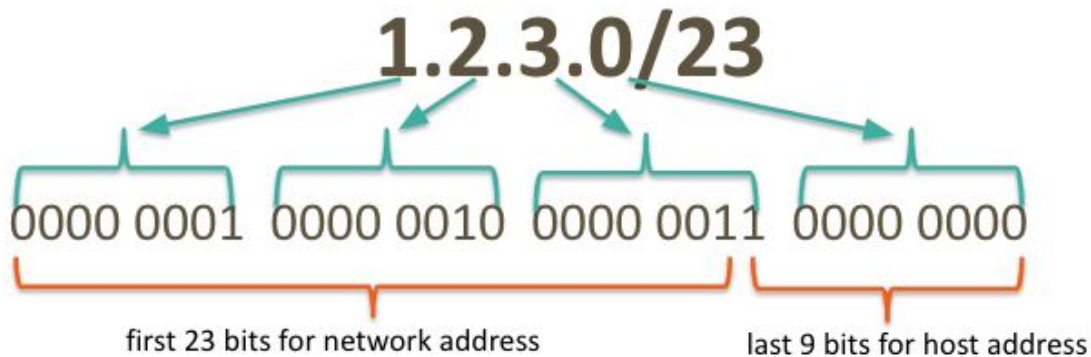
1. Router floods its link state to all other routers.
2. Each router learns global network topology
3. Then, computes shortest path themselves!
  - With Dijkstra's, etc



# IP Addressing

# IP Address

- 32 bits (for IPv4), split into 4 bytes, written in decimal
  - Network prefix: /<bits>
- Size of network address, counting from the leftmost bit
  - Example: 1.2.3.0/23



# Network prefixes (netmasks)

- Prefix dedicated to network address
  - How can we tell if a host is in a network?
- Check if the prefix matches!
  - (bitwise AND:  $\text{addr} \& \text{mask} == \text{mask}$ )

**Mask: 123.96.0.0/12**

**01111011 . 0110**0000 . 00000000 . 00000000

Addr: 123.100.42.6

**01111011 . 0110**0100 . 00101010 . 00000110

# Classful Addressing

- Network classes:
  - A (/8): first 8 bits devoted to network
  - First bit is fixed to 0.
  - first byte from 1 to 126 (0 and 127 are reserved)
  - Can have ~16M hosts, only  $2^7 - 2 = 126$  nets.



- B (/16): first 16 bits devoted to network (first byte from 128 to 191)
  - First two bits are fixed to 10
  - Can have ~65K hosts, ~16K nets



- C (/24): first 24 bits devoted to network (first byte from 192 to 223)
  - First three bits are fixed to 110
  - Can have only 254 hosts (255 is reserved for last byte) ~2M nets



Network Bits

Host Bits

# Classless Inter-Domain Routing (CIDR)

- Use two 32-bit numbers to represent a network
  - Network address = IP Address bitwise AND Subnet Mask
    - IP Address is 192.138.12.2
    - Subnet Mask is 255.248.0.0

network address 192.136.0.0/13

IP Address    1100 0000 . 1000 1010 . 0000 1100 . 0000 0010

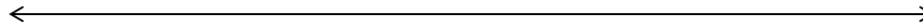
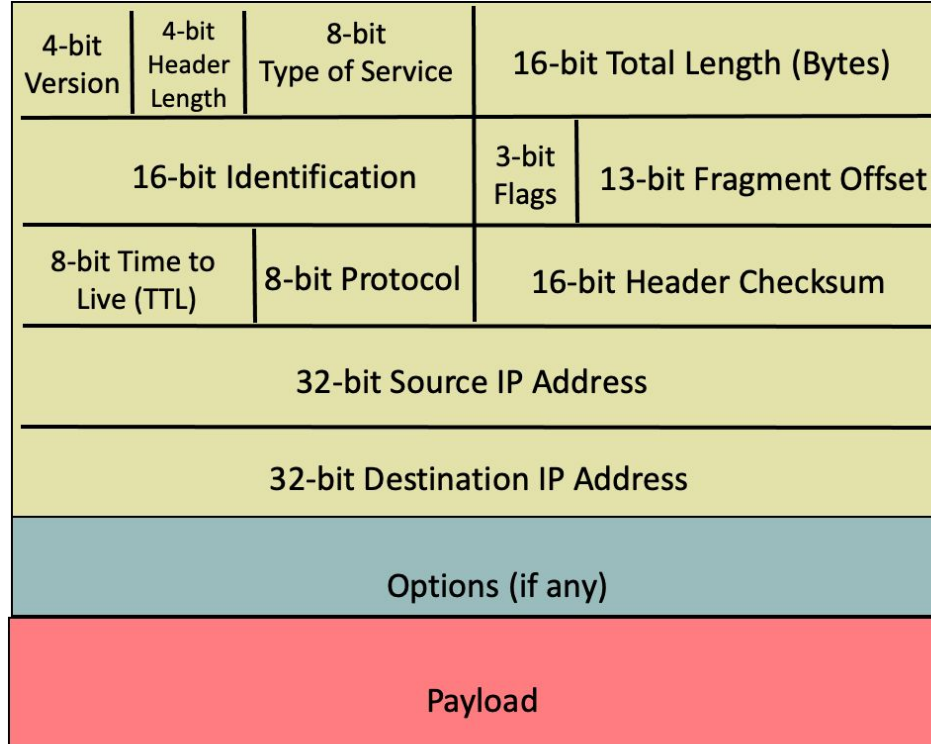
Subnet Mask   1111 1111 . 1111 1000 . 0000 0000 . 0000 0000

- Flexible division of bits:
  - More choices for the size of the network and hosts
- Offers better size routing table and efficient IP address space

# IP Packets and Forwarding



# IPv4 Packet Structure



# IP Header Design Remarks

- Header corrupted:
  - Check using Checksum field
    - Recomputed at each hop because of changed TTL
- Potential loops:
  - Track how long the packet has traveled: TTL field
    - Decrement TTL by 1 at each hop; discard packet if TTL reaches 0
- Packet too large:
  - Use fragmentation and reassemble packets at the end host: Total Length, Identification, Flags (MF), Fragment Offset

# IPv6

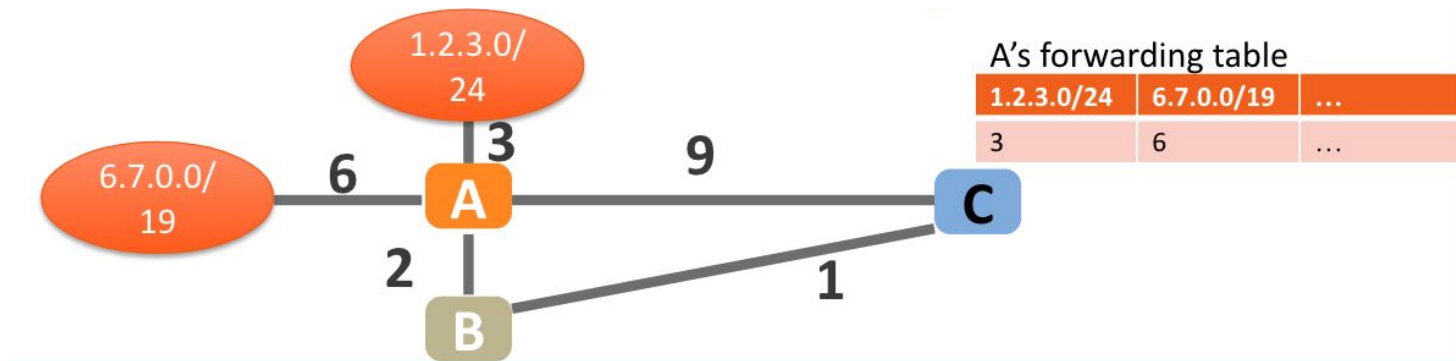
- Motivated by address exhaustion
  - Addresses four times as big
- Took the opportunity to do some “spring cleaning”
  - Got rid of all fields that were not absolutely necessary
- Result is an elegant, if unambitious, protocol

# IPv6 Changes

- Expanded addresses
- Eliminated checksum
- Eliminated fragmentation
- New options mechanism
  - Treated as a separate header instead of variable-length field
- Eliminated header length
- Added Flow Label
  - *Explicit* mechanism to denote related streams of packets

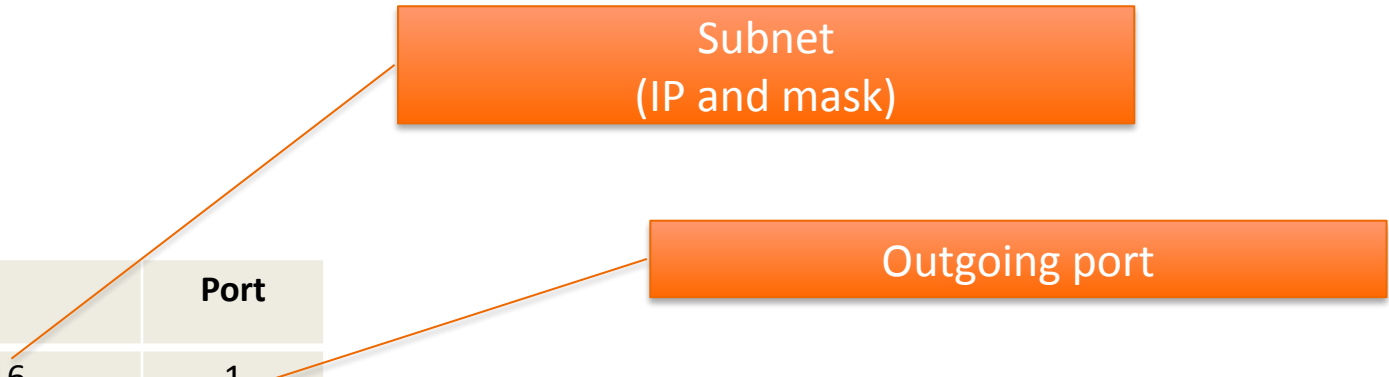
# Prefixes

- How do routers generate subnet masks?
  - Not from the data packets
  - From the routing algorithm
- A will advertise to B and C paths to prefixes
  - B and C will add these entries to their routing tables



# Example

Subnet (IP and mask)		Outgoing port	
Rule	Port		
191.168.0.0/16	1		
191.168.12.16/28	3		
191.168.12.18/31	2		
191.168.8.0/21	4		



**Row 1:** Packets destined for address in the 191.168.0.0/16 subnet should go out on port 1.

# Forwarding Tables

- Potential problem:
  - What if a packet matches multiple entries in our table?

IP address:

191.168.12.23 = 1011 1111 1010 1000 0000 1100 0001 0111

Routing table entries:

191.168.0.0/16 = 1011 1111 1010 1000 0000 0000 0000 0000

191.168.12.16/28 = 1011 1111 1010 1000 0000 1100 0001 0000

191.168.8.0/21 = 1011 1111 1010 1000 0000 1000 0000 0000

# Longest Prefix Matching (LPM)

- Each entry in a forwarding table may specify a sub network
- An address may match more than one forwarding table entry
- If an address matches multiple entries, the most specific entry is used
- The most specific matching is the entry with the longest subnet mask



# LPM Example

IP address: 191.168.12.23

Binary notation: 1011 1111 1010 1000 0000 1100 0001 0111

matching 1<sup>st</sup> row's rule...

1011 1111 1010 1000 0000 0000 0000 0000

Rule	Port
191.168.0.0/16 <b>MATCHED</b>	1
191.168.12.16/28	3
191.168.12.18/31	2
191.168.8.0/21	4

# LPM Example

IP address: 191.168.12.23

Binary notation: 1011 1111 1010 1000 0000 1100 0001 0111

Matching 2<sup>nd</sup> row's rule...

Rule	Port
191.168.0.0/16	1
191.168.12.16/28	3
191.168.12.18/31	2
191.168.8.0/21	4

1011 1111 1010 1000 0000 0000 0000 0000  
1011 1111 1010 1000 0000 1100 0001 0000

# LPM Example

IP address: 191.168.12.23

Binary notation: 1011 1111 1010 1000 0000 1100 0001 0111

Matching 3<sup>rd</sup> row's rule...

Rule	Port
191.168.0.0/16	1
191.168.12.16/28	3
191.168.12.18/31	2
191.168.8.0/21	4

1011 1111 1010 1000 0000 0000 0000 0000  
1011 1111 1010 1000 0000 1100 0001 0000  
1011 1111 1010 1000 0000 1100 0001 0010

# LPM Example

IP address: 191.168.12.23

Binary notation: 1011 1111 1010 1000 0000 1100 0001 0111

Matching 4<sup>th</sup> row's rule...

Rule	Port
191.168.0.0/16	1
191.168.12.16/28	3
191.168.12.18/31	2
191.168.8.0/21	4

1011 1111 1010 1000 0000 0000 0000 0000  
1011 1111 1010 1000 0000 1100 0001 0000  
1011 1111 1010 1000 0000 1100 0001 0010  
1011 1111 1010 1000 0000 1000 0000 0000

# LPM Example

IP address: 191.168.12.23


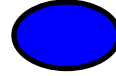
Binary notation: 1011 1111 1010 1000 0000 1100 0001 0111

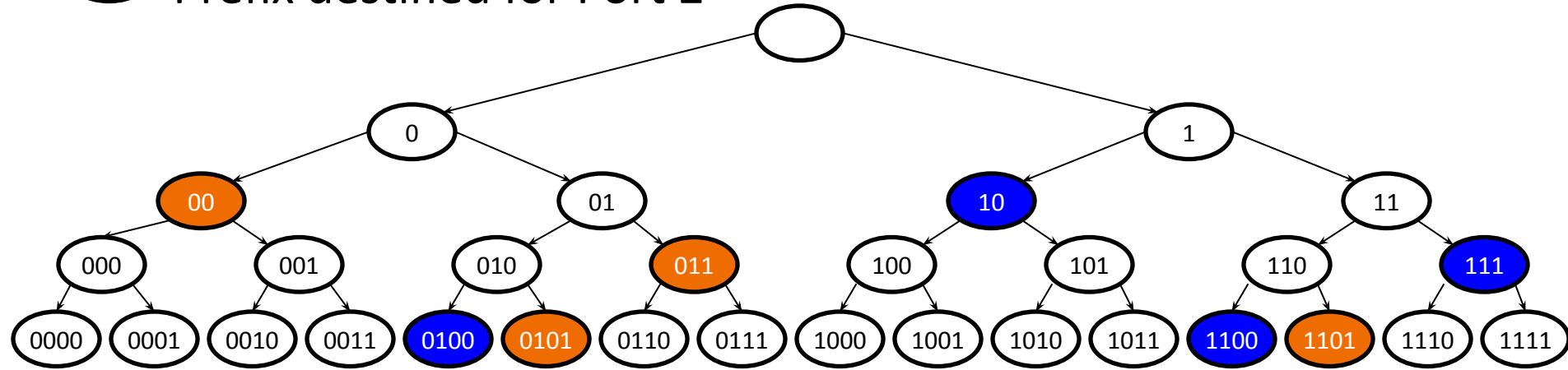
Rule	Port	
191.168.0.0/16	1	1011 1111 1010 1000 0000 0000 0000 0000
191.168.12.16/28	3	1011 1111 1010 1000 0000 1100 0001 0000
191.168.12.18/31	2	1011 1111 1010 1000 0000 1100 0001 0010
191.168.8.0/21	4	1011 1111 1010 1000 0000 1000 0000 0000

1<sup>st</sup>, 2<sup>nd</sup>, and 4<sup>th</sup> row all matched the address with their prefixes.

Since 2<sup>nd</sup> row has the **longest** prefix matched, router is going to forward the packet to port 3 according to 2<sup>nd</sup> row.



# LPM: Tree Representation

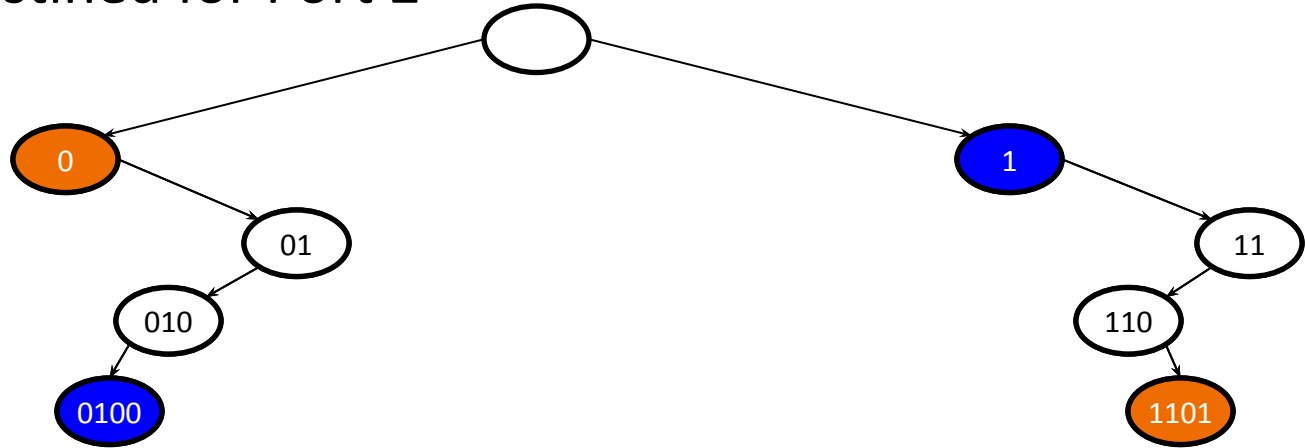
-  Prefix destined for Port 1
-  Prefix destined for Port 2



No packet will match more than one prefix  
All paths reach a unique prefix

# LPM: Tree Representation

-  Prefix destined for Port 1
-  Prefix destined for Port 2



Packets now may match many prefixes

Forward the packet based on the deepest matching node (longest prefix)

**BGP**



# Interdomain Routing

- **Interdomain** routing is between autonomous systems (AS)
  - Similar goals as intradomain routing with scalability + policy compliance
  - Autonomous systems want privacy and autonomy
- Border gateway protocol (BGP) is current design
  - Extends on top of DV (with some crucial differences)

# Export & Selection

- If you are an AS:
  - Route Selection
    - **Where you send your packets**
    - Determine how to choose a valid route to a given IP prefix, when multiple paths through ASes
  - Route Export
    - **Which ASes will receive your route**
    - Other ASes will *select* your route and **send traffic to you**

# Types of ASes (domains)

- **Stub:** only sends/receives traffic for its users
  - companies, universities, etc.
- **Transit:** carries traffic for other ASes
  - Global ISPs (Tier 1): fully connected mesh
  - Regional ISPs (Tier 2)
  - Local ISPs (Tier 3)
- Lower tiers buy service from higher tiers
- What's the relationship between AS and ISP?
  - All ISPs are ASes, but not all ASes are ISPs
  - E.g. UC Berkeley is not an ISP but it is an AS

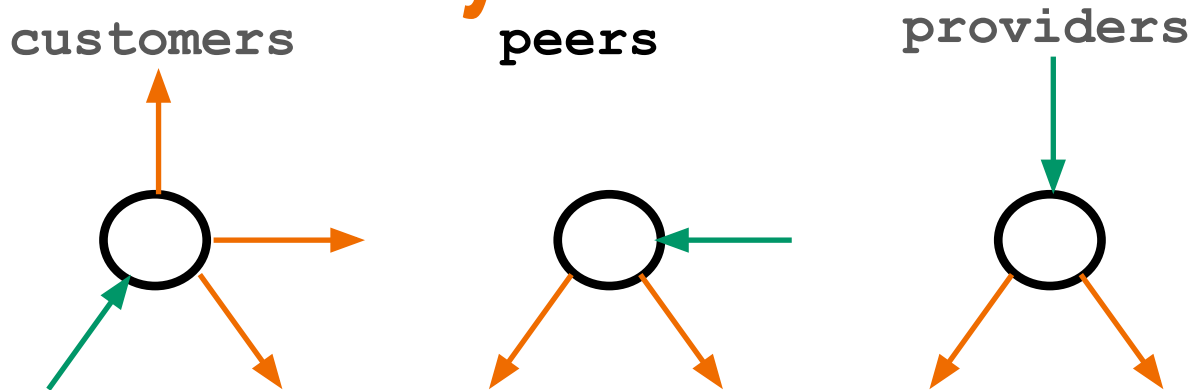
# Business Relationship among ASes

- Two ASes will **connect** only if they have business relationship:
  - **Customer-Provider**
    - *Provider B carries customer A's traffic for a fee*
  - **Peers**
    - *Peers A, B carry each other's traffic for free*
- What roles can a global ISP (Tier 1) have?
  - Provider to Tier 2 or Tier 3
  - Peer to other global ISP (tier 1)
  - Not a customer!

# Gao-Rexford Policy

Destination prefix advertised by...	Export route to...
Customer	Everyone (providers, peers, other customers)
Peer	Customers
Provider	Customers

# Gao-Rexford Policy Continued

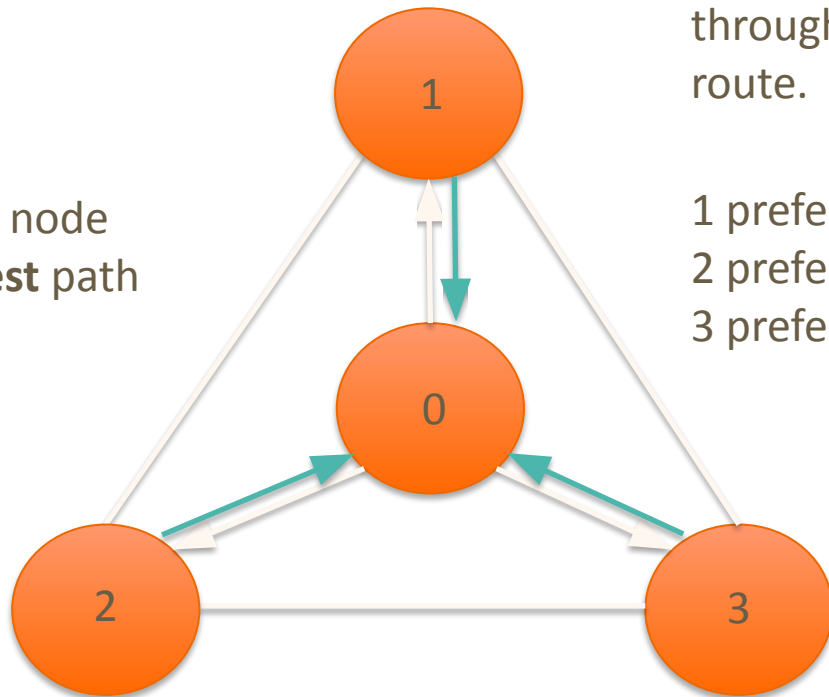


- **Green** arrow is where you learn the route
- **Orange** arrows are where you export the route
- With Gao-Rexford
  - The AS policy graph is a DAG
  - Routes are “valley free”/”single-peaked”

# Policy Oscillation

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



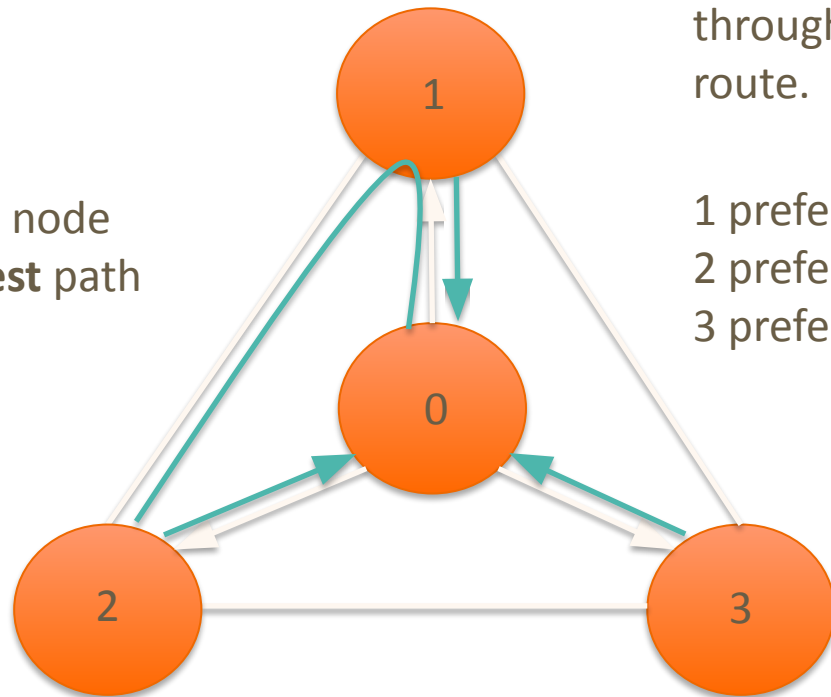
Each node **prefers** route through neighbor over direct route.

1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2

# Policy Oscillation 1

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



**1 advertises  $1 \rightarrow 0$  to 2**

Each node **prefers** route through neighbor over direct route.

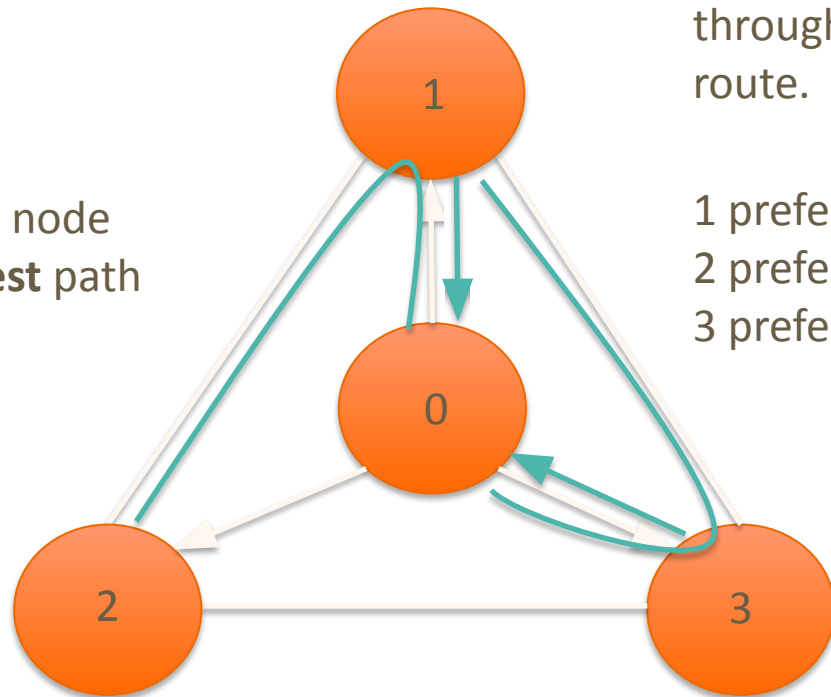
1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2



# Policy Oscillation 2

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



Each node **prefers** route through neighbor over direct route.

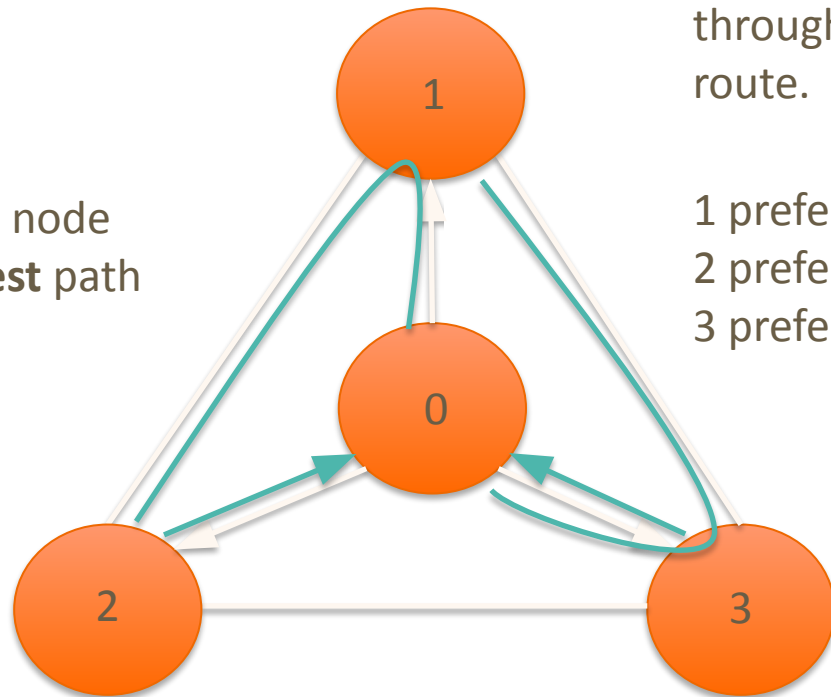
1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2

**3 advertises  $3 \rightarrow 0$  to 1**

# Policy Oscillation 3

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



Each node **prefers** route through neighbor over direct route.

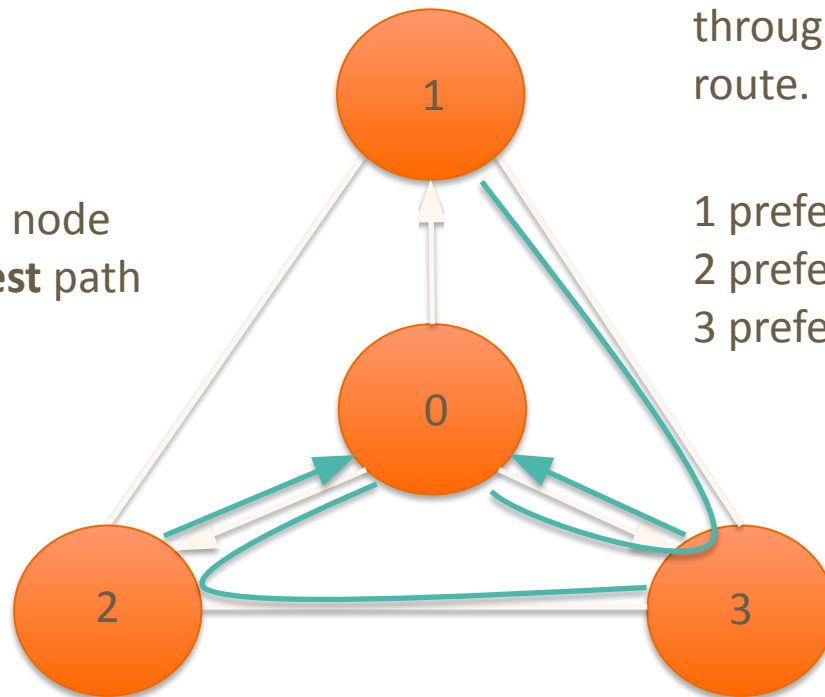
1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2

**1 withdraws its path of  $1 \rightarrow 0$  from 2  
(because 1 now takes  $1 \rightarrow 3 \rightarrow 0$ )**

# Policy Oscillation 4

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



Each node **prefers** route through neighbor over direct route.

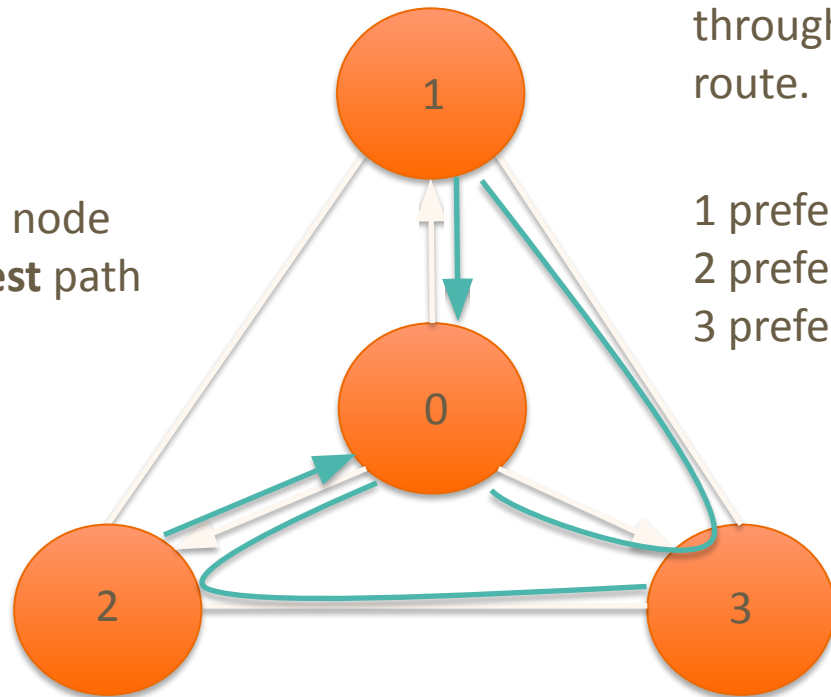
1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2

**2 now advertises  $2 \rightarrow 0$  to 3**  
**(3 would take it as it favors its neighbor)**

# Policy Oscillation 5

Suppose **initially** each node only knows the **shortest** path to 0 (green arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



Each node **prefers** route through neighbor over direct route.

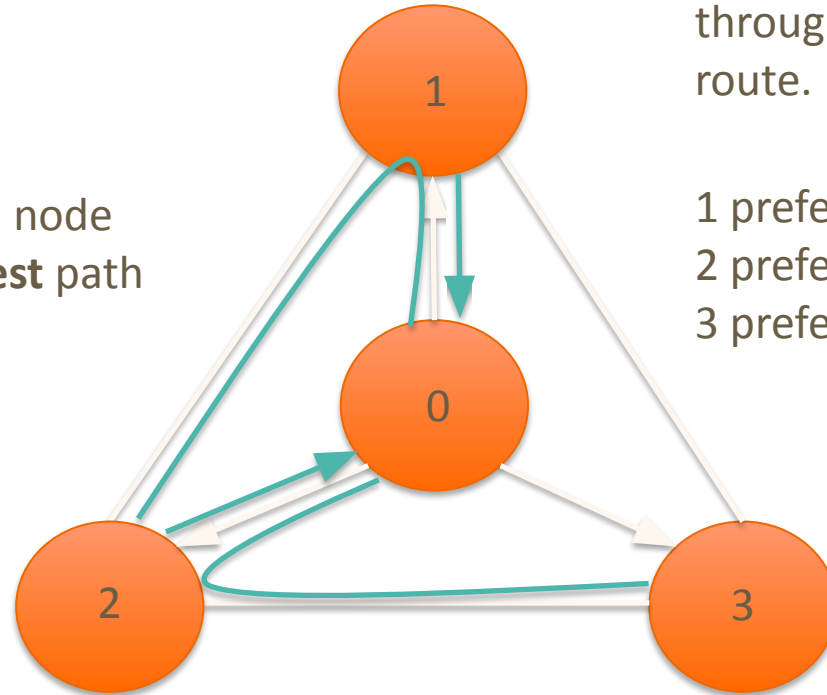
1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2

**3 now withdraws  $3 \rightarrow 0$  from 1**

# Policy Oscillation 6

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$   
2 knows  $2 \rightarrow 0$   
3 knows  $3 \rightarrow 0$



Each node **prefers** route through neighbor over direct route.

1 prefers reaching 0 through 2 or 3  
2 prefers reaching 0 through 1 or 3  
3 prefers reaching 0 through 1 or 2

**1 again advertises its path  $1 \rightarrow 0$**

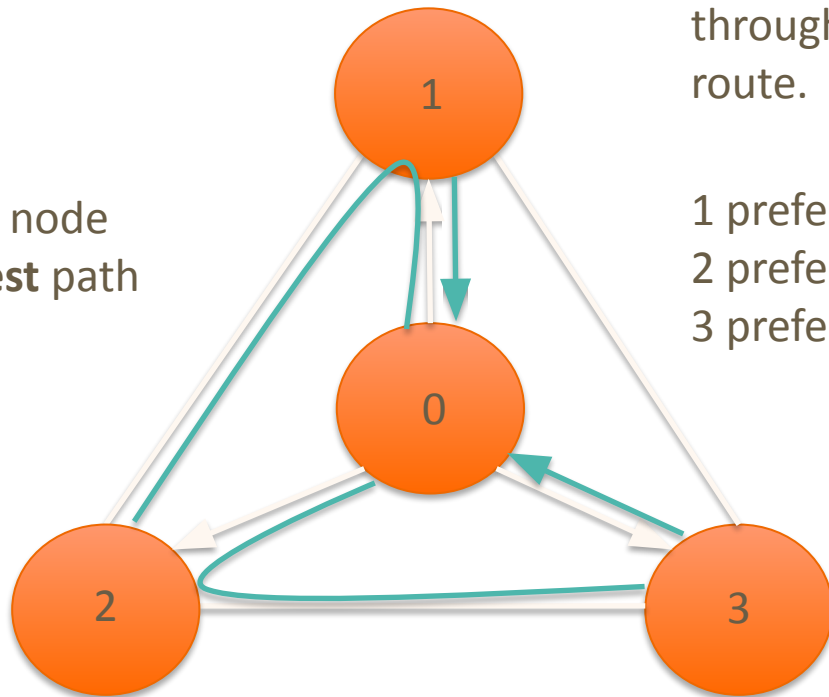
# Policy Oscillation 7

Suppose **initially** each node only knows the **shortest** path to 0 (**green** arrow).

1 knows  $1 \rightarrow 0$

2 knows  $2 \rightarrow 0$

3 knows  $3 \rightarrow 0$



Each node **prefers** route through neighbor over direct route.

1 prefers reaching 0 through 2 or 3

2 prefers reaching 0 through 1 or 3

3 prefers reaching 0 through 1 or 2

**2 withdraws its path  $2 \rightarrow 0$  from 3**

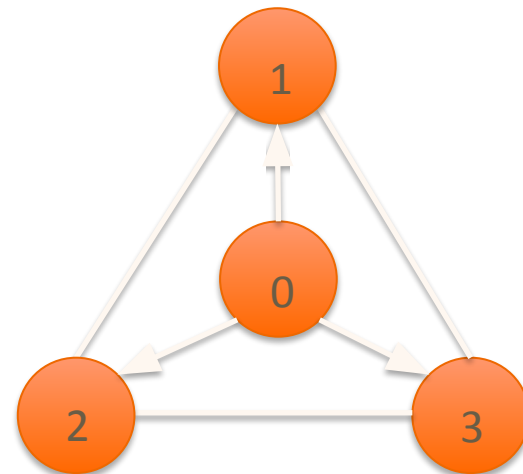
*Back to where we started!*

**Why doesn't this happen in reality?**

**Gao-Rexford**

# Gao-Rexford avoids Policy Oscillation

- Example shown before did not use Gao-Rexford (why?)
  - 1, 2, and 3 are **peers**
  - 0 is the **provider** to 1, 2, and 3
  - Peers ***don't*** advertise route learned from providers to each other
    - i.e. 1 would never advertise 1->0 (learned from 1's provider 0) to 2 (1's peer)

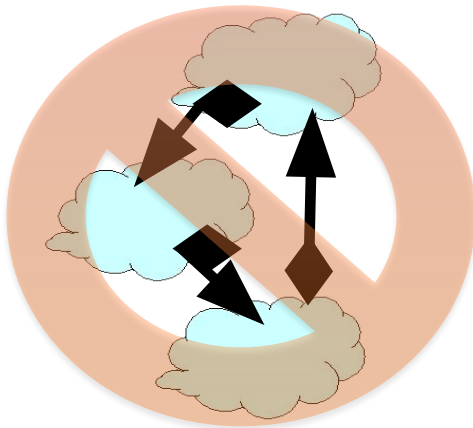


Destination prefix advertised by...	Export route to...
Peer	Customers



# Business Relationship Restrictions

- The graph of **peering** relations can be *cyclic*
  - The peer of my peer can also be my peer
  - For example, global ISPs all peer with each other
- The graph of **customer-provider** relations must be *acyclic*



# Three parts of Gateway Protocols

- **eBGP**
  - Between border routers in **different ASes**
  - Learn about **external routes**
- **iBGP**
  - Between border routers and other routers **within a single AS**
  - Learn **which border router** to use to reach external destinations
- **IGP**
  - The **protocol** used for **intradomain** routing (e.g. OSPF).
    - Shortest path to **subnet in the same AS**
    - Shortest path to **border router** for given external network
  - Just a different name for L3 routing as we've talked about earlier

# BGP Route Attributes

Attributes: Parameters used in route selection

- **Local** attributes
  - ASes keep them private
  - Not included in eBGP route announcements
  - E.g. LOCAL\_PREF, IGP path
- **Nonlocal** attributes:
  - propagated with eBGP route announcements
  - E.g. AS\_PATH, MED

# Route Selection in Priority Order

Priority	Rule	Remarks
1	LOCAL PREF	Pick <b>highest</b> LOCAL PREF
2	ASPATH	Pick <b>shortest</b> ASPATH length
3	IGP path	Lowest IGP cost to next hop (egress router)
4	MED	<b>Lowest</b> MED preferred
5	Router ID	Smallest next-hop router's IP address as tie-breaker

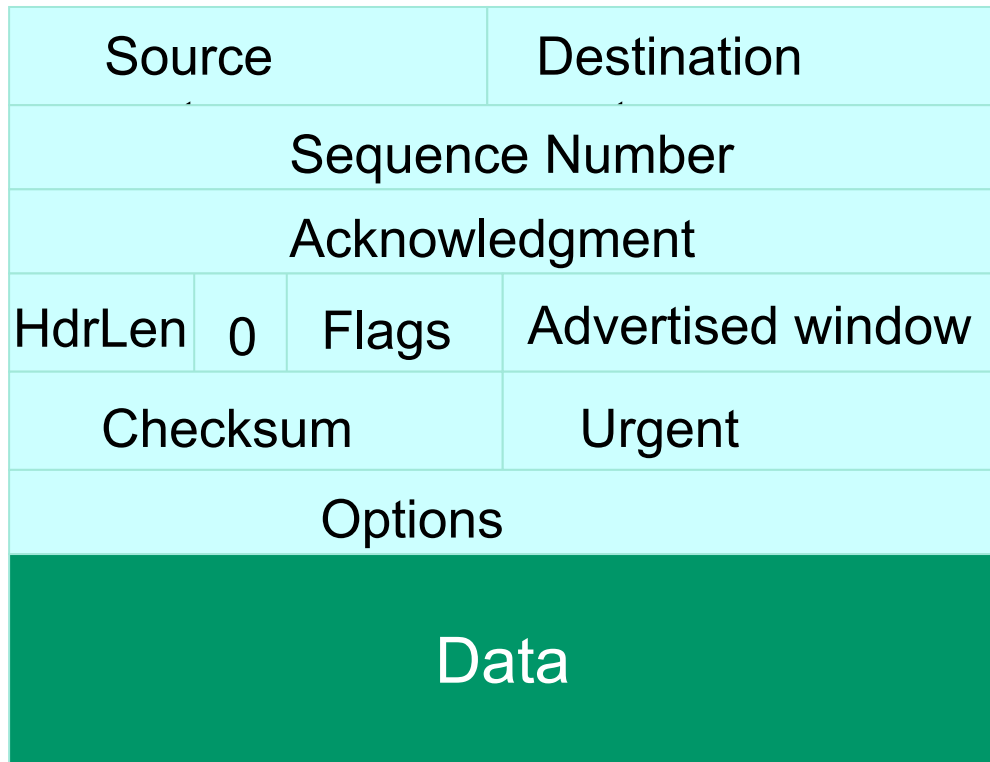
# Reliability / TCP

# Reliability

- Best-effort network
  - Need to handle packet loss, corruption, reordering, delays, duplications, etc.
- Building blocks
  - Checksums: detect corruption
  - Feedback: positive/negative feedback from receiver
  - Retransmissions: sender re-sends packets
  - Timeouts: when to resend a packet
  - Sequence numbers: indicate which packets have been received
- Design considerations
  - Window size, nature of feedback, detection of loss, response to loss

# TCP

- L4 Protocol (Transport)
- Byte Stream
- Bi-directional
- Reliable
- In-order

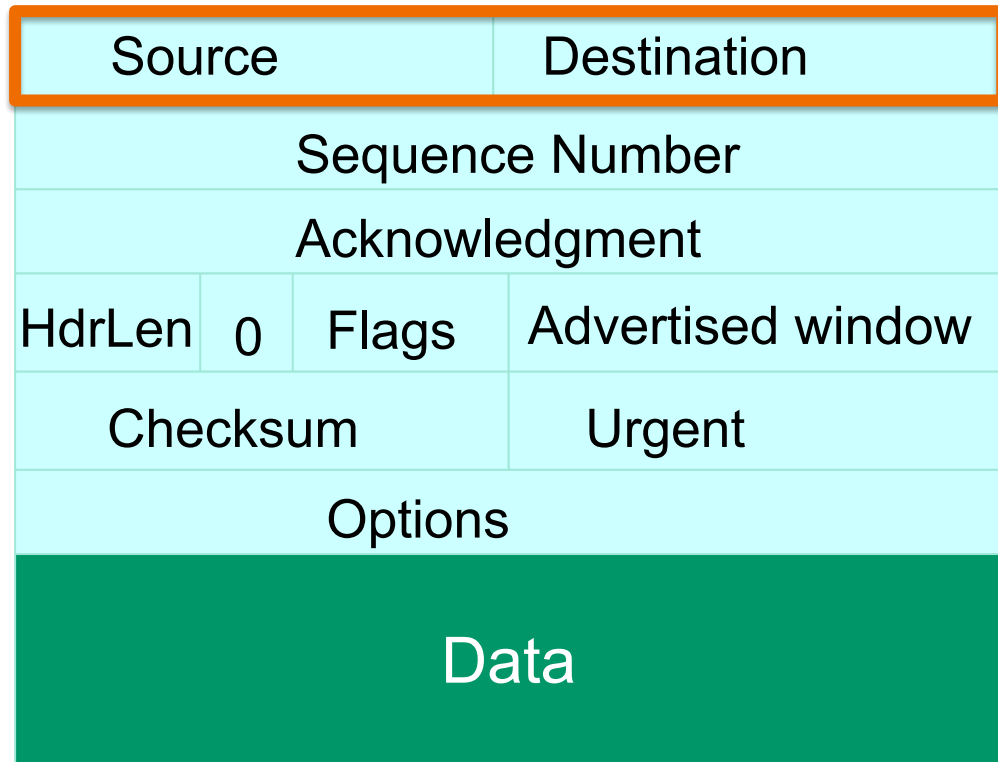


# TCP Header

- Host port numbers
  - Multiplexing and demultiplexing
  - 16 bits

But wait - why no addr?!

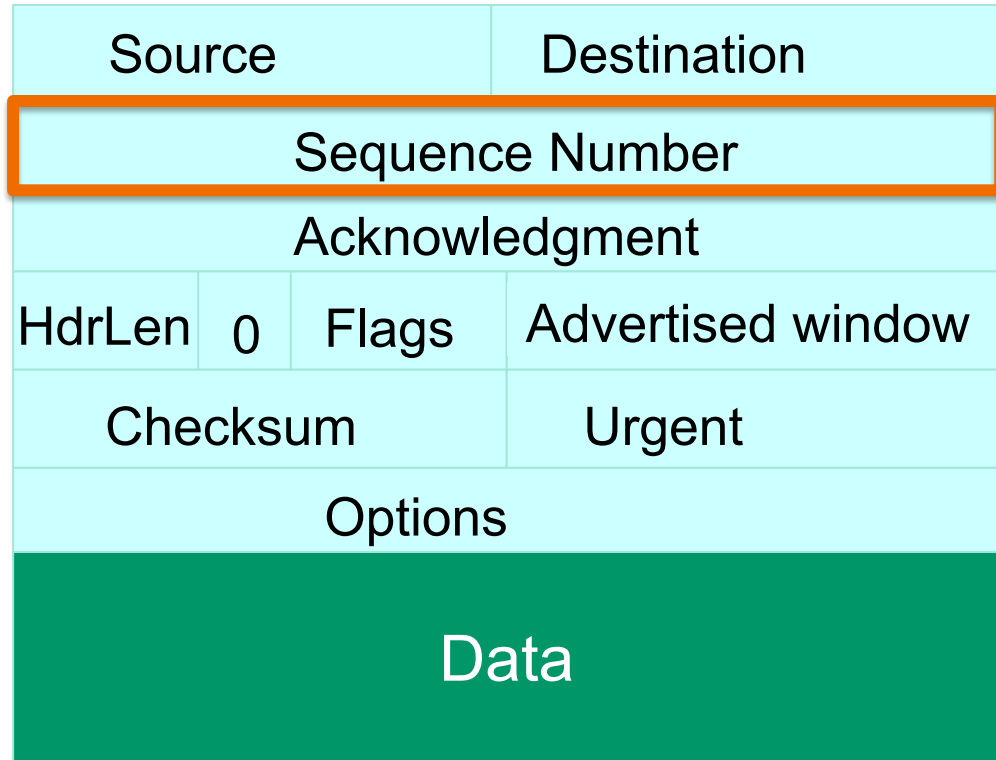
- IP header has addr.





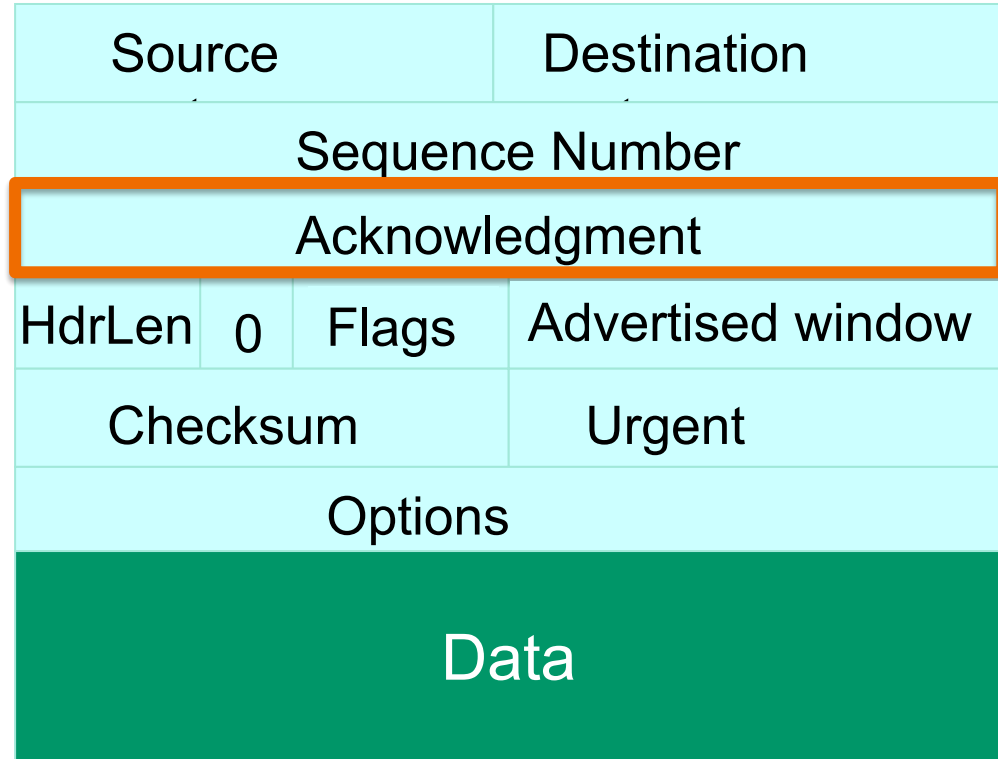
# TCP Header

- Byte offset
  - Of first payload byte
  - Initialized randomly
- **Byte Stream Protocol**
  - Seq # refers to **bytes**



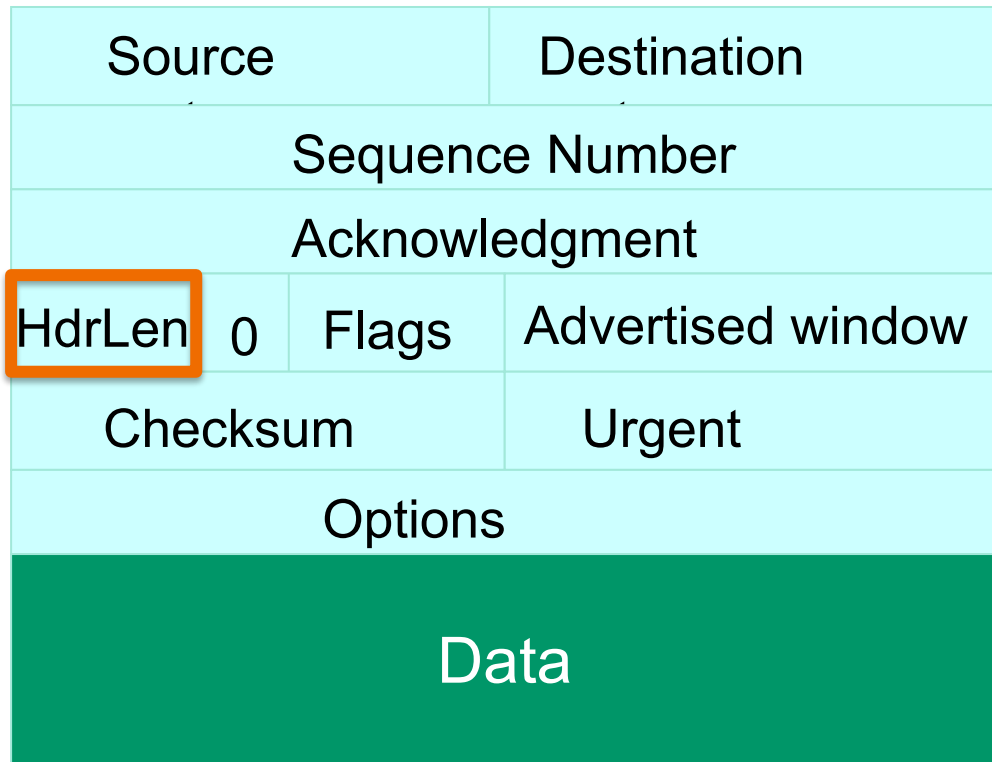
# TCP Header

- Cumulative ACKs
- Seq # of **next byte**
- Data packets carry ACKs



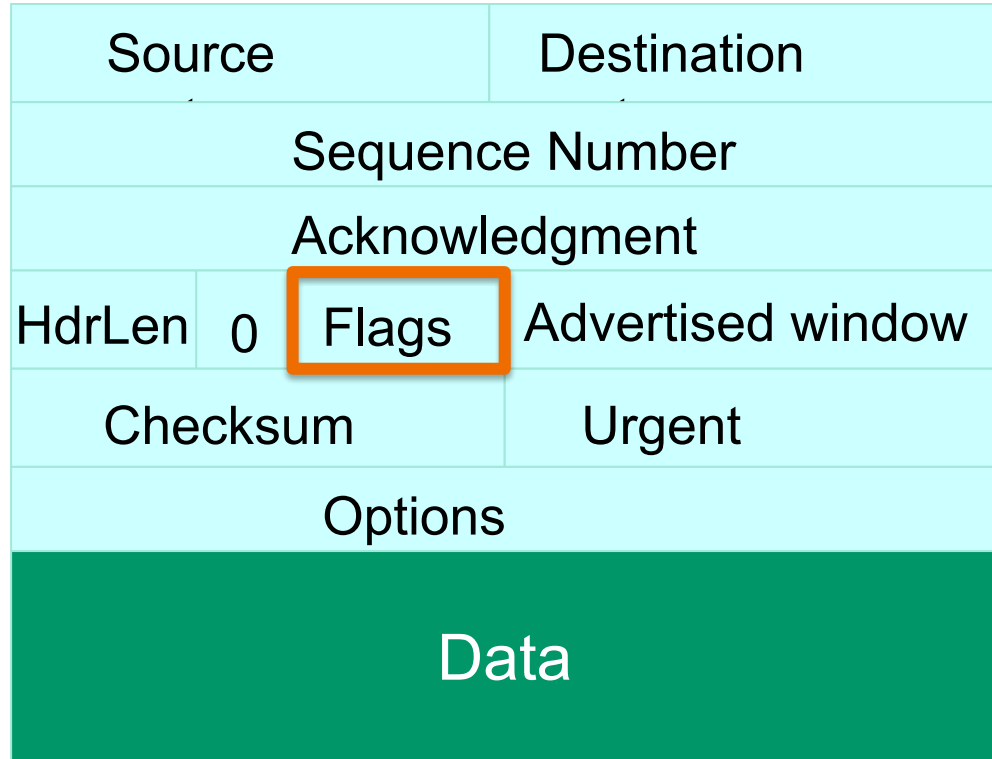
# TCP Header

- Header length
  - 4 bits
  - In 4-byte words
- Minimum 5 words (20B)
- Maximum 15 words
  - Why?



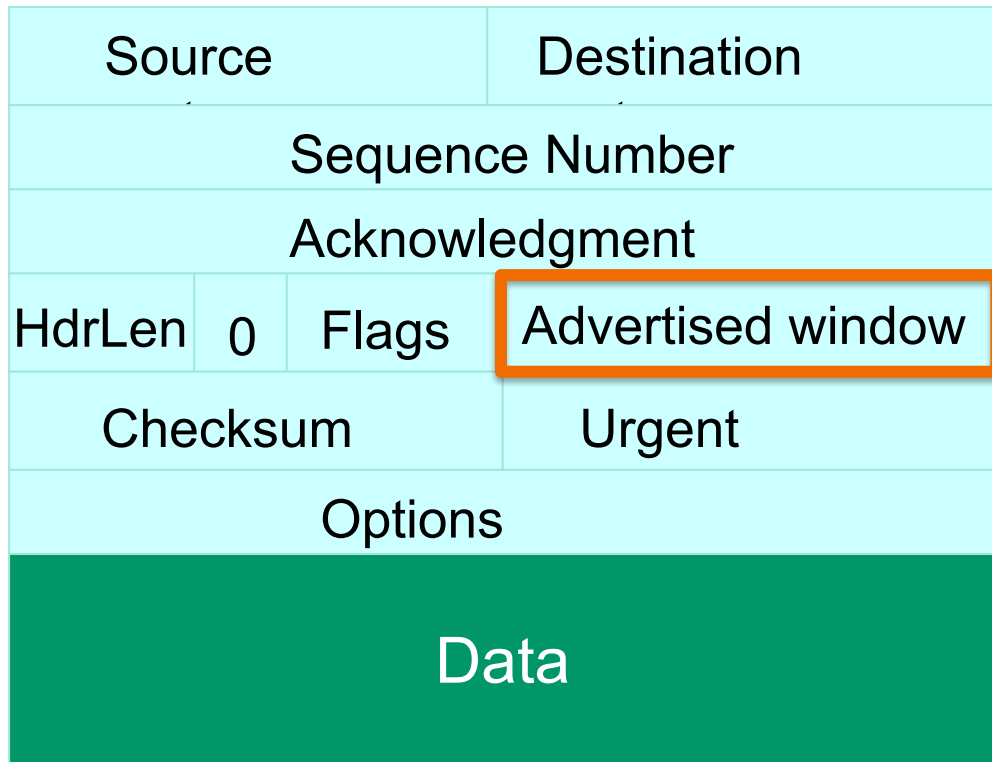
# TCP Header

- SYN
  - SYNchronize initial state
- ACK
  - ACKnowledgement
- FIN
  - No more data
- RST
  - Connection ReSeT



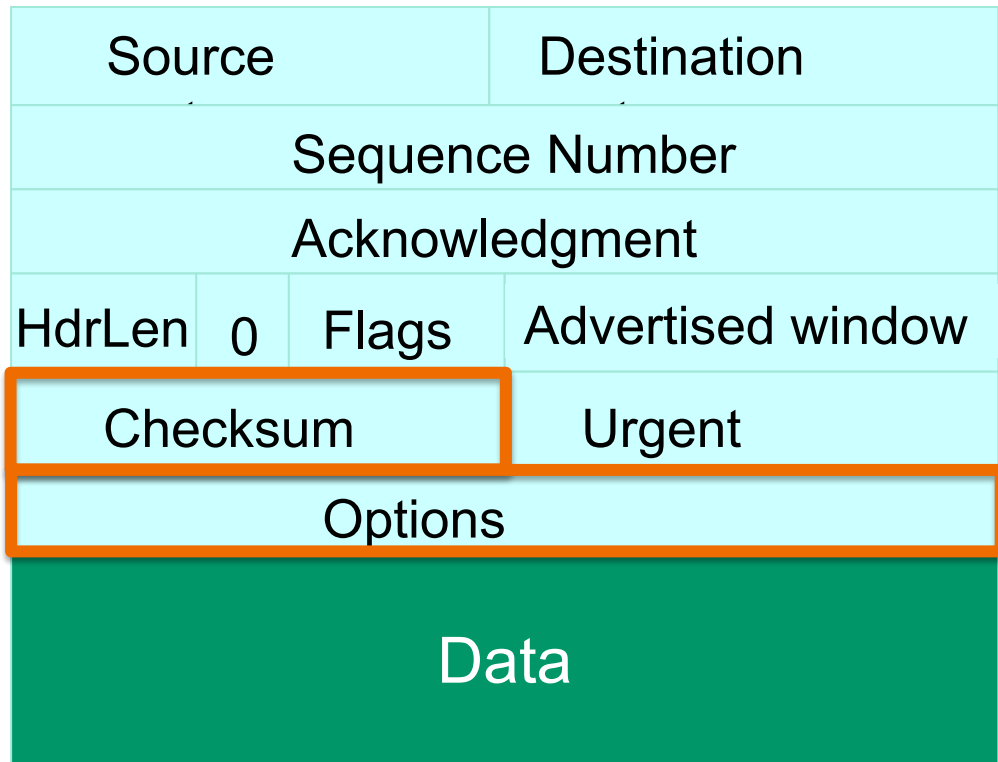
# TCP Header

- Receive Window Size
  - Maximum receiver buffer
- Limits sending rate
  - Don't send faster than receiver can process



# TCP Header

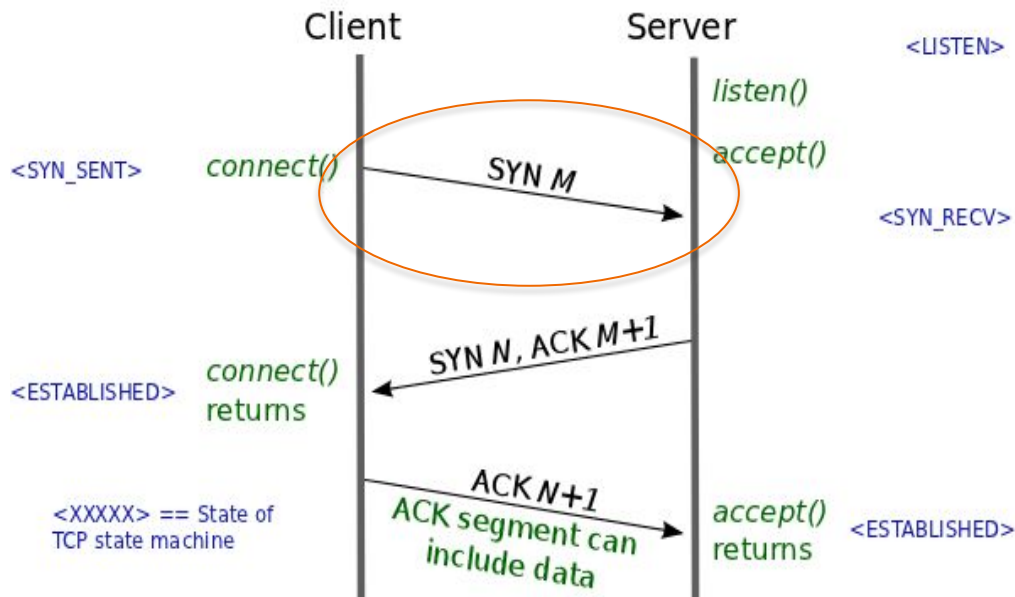
- Checksum
  - Includes header and payload
- Options common
  - Unlike IP
  - Not covered
  - Assume no options unless specified



# Connection Establishment

# Three Way Handshake

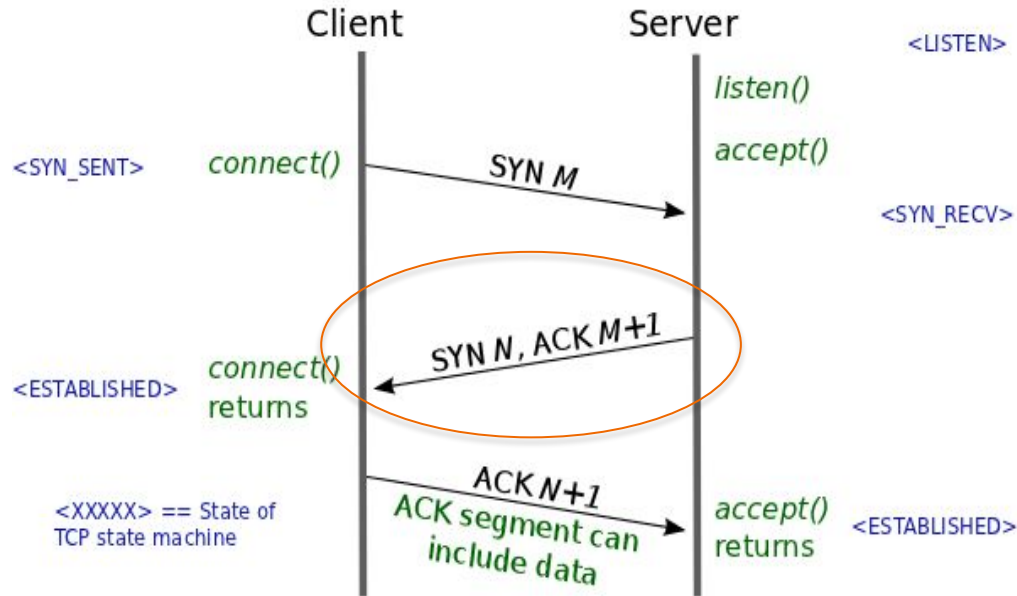
- Client sends server a SYN
  - A TCP packet with the SYN flag set
- SYN packet carries initial state
  - Receive window
  - Source port number
  - Initial sequence number





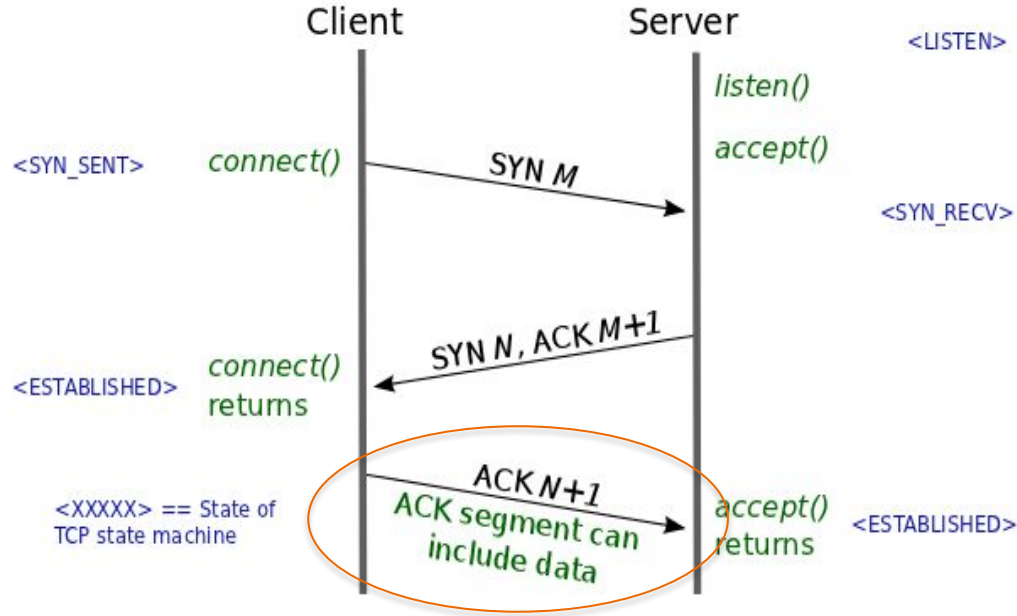
# Three Way Handshake

- Server responds with a SYN-ACK
- Acknowledge the client's SYN
- Respond with the server's own SYN
- Carries server's initial state
  - Receive window
  - Source port number
  - Initial sequence number



# Three Way Handshake

- Client responds with an ACK
- Now the connection is established!
  - Client and server can freely communicate



# Connection Establishment

- Sequence number and acknowledgement number don't start from zero
- Instead client and server choose a **random** initial seq #
- Must be communicated between client and server
- Client/Server exchange state
  - Initial Sequence numbers
  - Port number
  - TCP Options

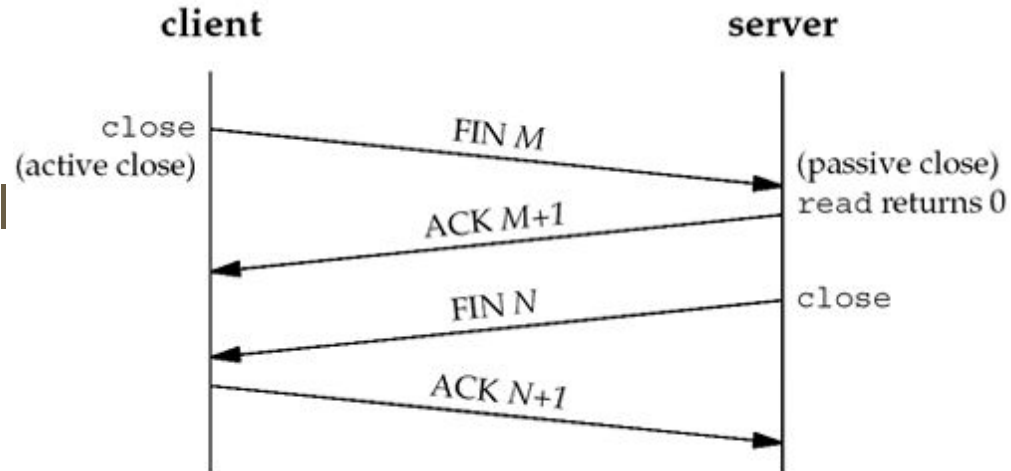
# Teardown

# Two Teardown Methods

- Four way handshake
  - Graceful/normal teardown
- Reset
  - Exceptional Teardown

# Normal Teardown

- One side closes their connection
  - Indicates they will send no more data
  - Sends a FIN
  - Receives an ACK
- Other side may continue transmitting
- Eventually it closes as well
  - Sends a FIN
  - Receives an ACK
- Connection is closed



# TCP Reset

- Reset the connection
  - Most commonly, attempt to SYN on a closed port
- Send RST packet(s) only (no ACK)

# Reliability



# Reliability

- Cumulative ACKs
  - Allow detection of dropped packets
- How do we know a packet was lost?
  - Timeout
  - Duplicate ACKs

# Timeout

- Retransmission Time Out (RTO)
  - Timeout after which packets are retransmitted
  - Based on a constantly updated RTT estimate (and variance)
- Single timer (not per-packet)
  - Each received ACK of new data resets RTO
  - If RTO times out
    - Retransmit packet containing “next byte”

# Timeout

- What if RTO is too large?
- Packet is dropped . . .
  - Wait
  - Keep waiting
  - ... Keep waiting
  - Timer goes off
    - Finally retransmit
- Can we do better?

# Duplicate ACKs

- Transmit
  - Seq 1000
  - Seq 2000
  - **Seq 3000** *Dropped in flight*
  - Seq 4000
  - Seq 5000
  - Seq 6000
- What ACKs do we receive?
  - ACK 2000
  - ACK 3000
  - ACK 3000
  - ACK 3000
  - ACK 3000
- Duplicate ACKs indicate packet loss/reorder

# Congestion Control

# Goal of Congestion Control

- Limit the # of packets in flight
  - Utilize our fair share of bandwidth...
  - But don't overload the network
- Adapt to the right bandwidth
- Be fair
  - Links are shared among many hosts

# Congestion Control: Windows

- Receive Window (RWND)
  - What rate at which the **receiver** can process packets
- Congestion Window (CWND)
  - What rate at which the **network** can process packets
- Sending rate
  - Smaller of the two
- In this class, we assume  $CWND \ll RWND$ 
  - **Network** will determine our sending rate

# TCP: loss-based feedback

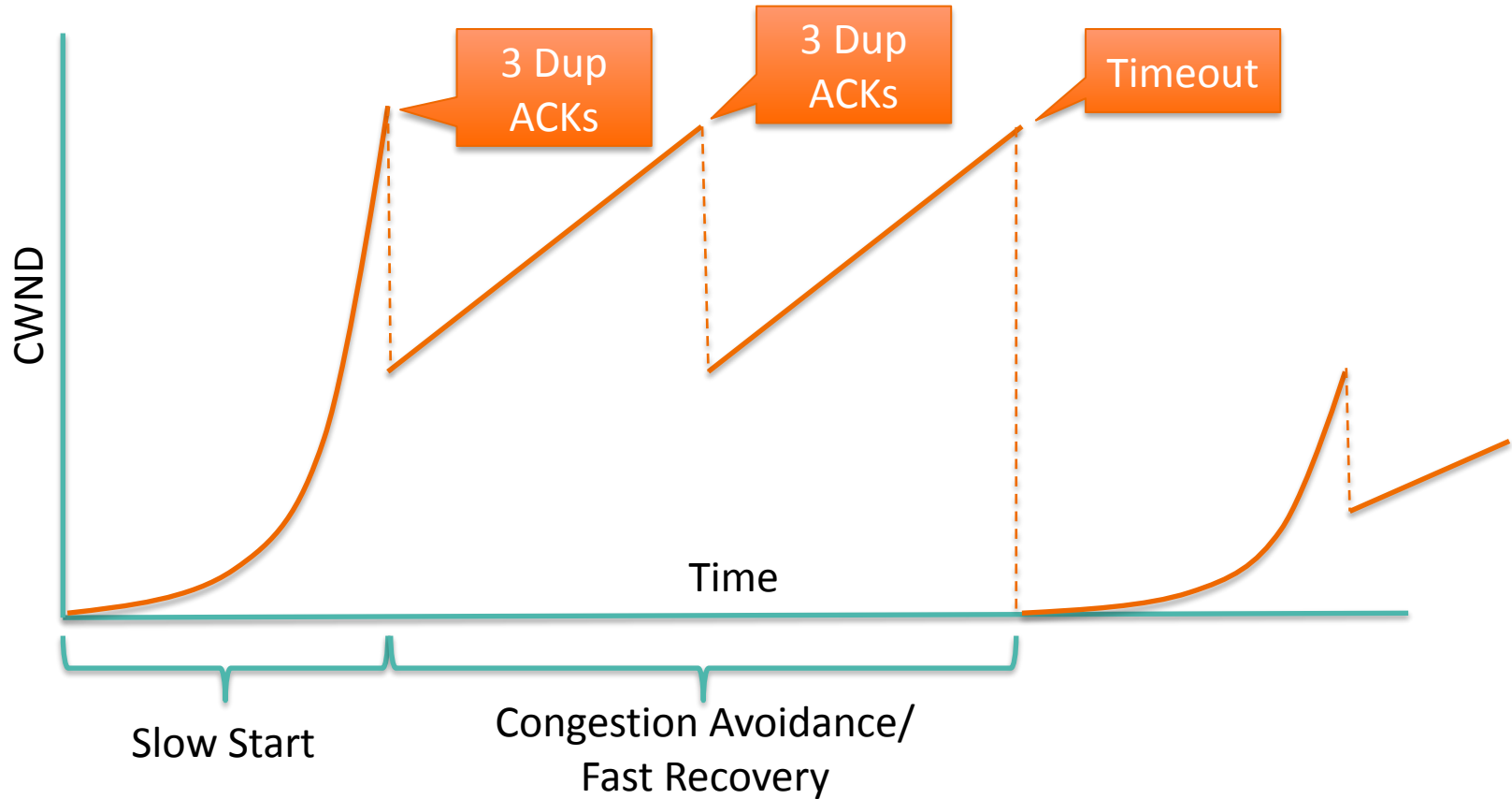
Idea: drop implies congestion

- 3 dupACK: minor congestion (ACKs get through)
- Timeout: major congestion (*nothing* gets through)

TCP's response depends on the *kind* of loss.



# TCP Sawtooth

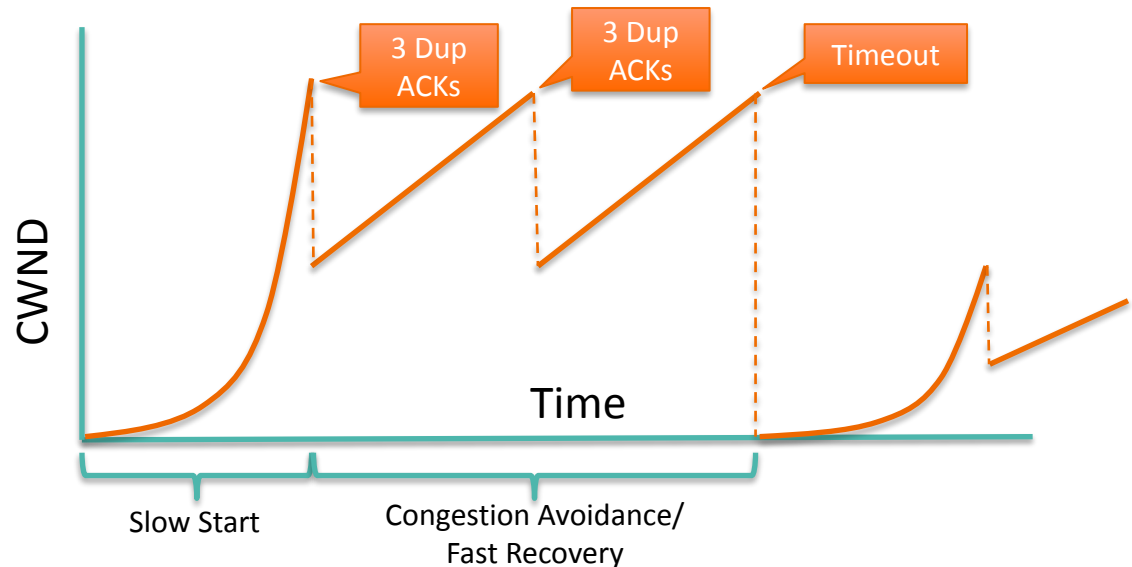


# Utilization

- The TCP sawtooth alternates between:
  - *Over-utilizing* bandwidth (causing drops)
  - *Under-utilizing* bandwidth
- Smart choices around buffering can result in higher utilization by absorbing the increase in window size

# Three States

1. Slow Start
2. Congestion Avoidance
3. Fast Recovery



# Implementation

- State at sender
  - CWND
    - Max sending rate without congesting network (assuming  $CWND \ll RWND$ )
  - ssthresh
    - Threshold CWND for exiting slow start
  - dupACKcount
    - Count of contiguous duplicate ACKs received
  - timer

# Congestion Control Mechanics

1. Slow Start
  - *Rapidly* increase our initial sending rate until we hit bottleneck
2. Congestion Avoidance
  - Adapting our sending rate to current network conditions
  - AIMD (**A**dditive **I**ncrease, **M**ultiplicative **D**ecrease)
3. Fast Recovery
  - Optimizing recovery from isolated loss
  - Detected through Duplicate ACKs

# Implementation

- Events at sender
  - ACK (new data)
  - dupACK (duplicate ACK for old data)
  - Timeout
- ... receiver just receives packets and sends ACKs

# Slow Start

- Value of CWND starts at (small constant) \* MSS
- For each packet that is acknowledged, increase the CWND by 1
  - Effectively **doubles** CWND every **RTT**!
- Window goes from  $1 \rightarrow 2 \rightarrow 4 \rightarrow \dots$

# Slow Start -- Intuition

- Instead of blasting packets based on the receive window
- Build up initial transmission rate *slowly*
- Back off when we've exceeded the capacity



# Slow Start – When Does It End?

- 2 Ways
  - 1) If  $CWND > ssthresh$ 
    - Enter congestion avoidance
  - 2) If we get 3 duplicate ACKs
    - Enter Fast Recovery
- If timeout:
  - Restart slow start,  $ssthresh = cwnd/2$ ,  $CWND = 1$

# Congestion Avoidance -- Intuition

- In the steady state
- Constantly probe for more bandwidth
- When we've exceeded – back off aggressively

# Congestion Avoidance

- Growth is more conservative than slow start
- After each **new** ACK, increase CWND by  $1 / \text{CWND}$ 
  - After one **RTT**, CWND will have **increased by ~1**
- When does it stop?
  - 1) Timeout → back to slow start
  - 2) 3 duplicate ACKS → Fast recovery

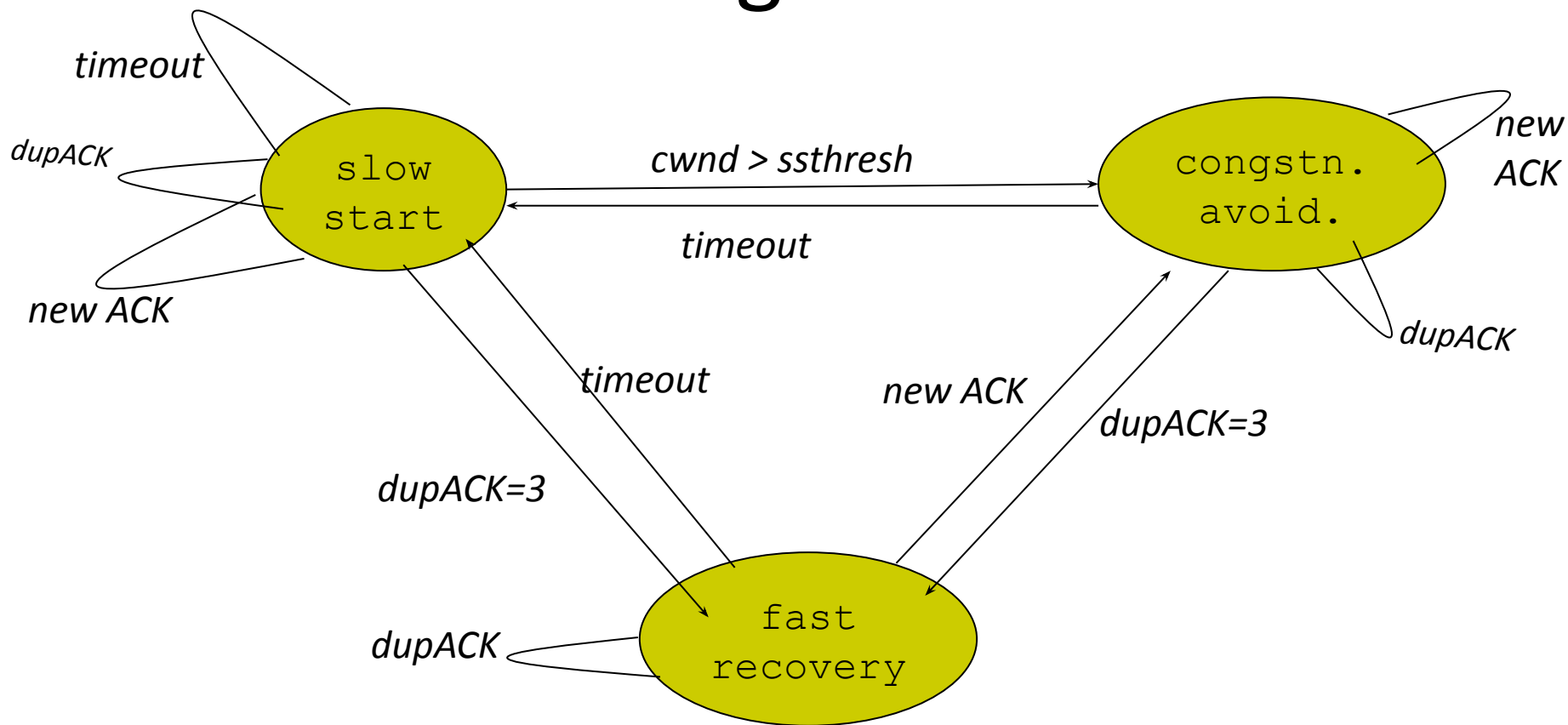
# Fast Recovery – Intuition

- A single lost packet
  - May just be a fluke
- Resetting CWND may be too aggressive
- Instead just retransmit that single packet
  - And continue as if nothing happened

# Fast Recovery

- Every **duplicate** ACK increases the window by 1
- When does it stop?
  - 1) Timeout → back to Slow Start
  - 2) New ACK → back to Congestion Avoidance

# The Big Picture



# CC Throughput

- Average Throughput:

Average # Bytes in Flight / RTT

$$\sqrt{\frac{3}{2}} \frac{\text{MSS}}{\text{RTT} \sqrt{p}}$$

- MSS - Maximum Segment Size
- RTT - Round Trip Time
- p - Probability of Packet Loss

# Study Tips

- Review quizzes, past exams, and discussion questions
  - Best source of practice material
  - Ask questions on our midterm exam threads!
- Focus on understanding concepts versus memorizing solutions
  - Enables you to overcome any curveballs we throw at you
  - Quality > Quantity
- You got this!



# Good Luck!

