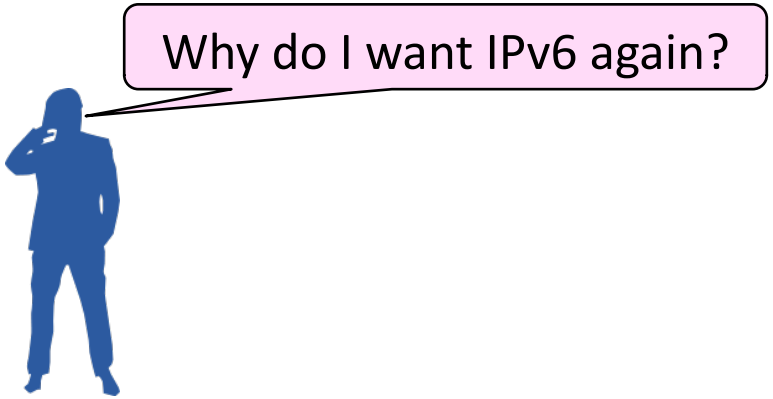
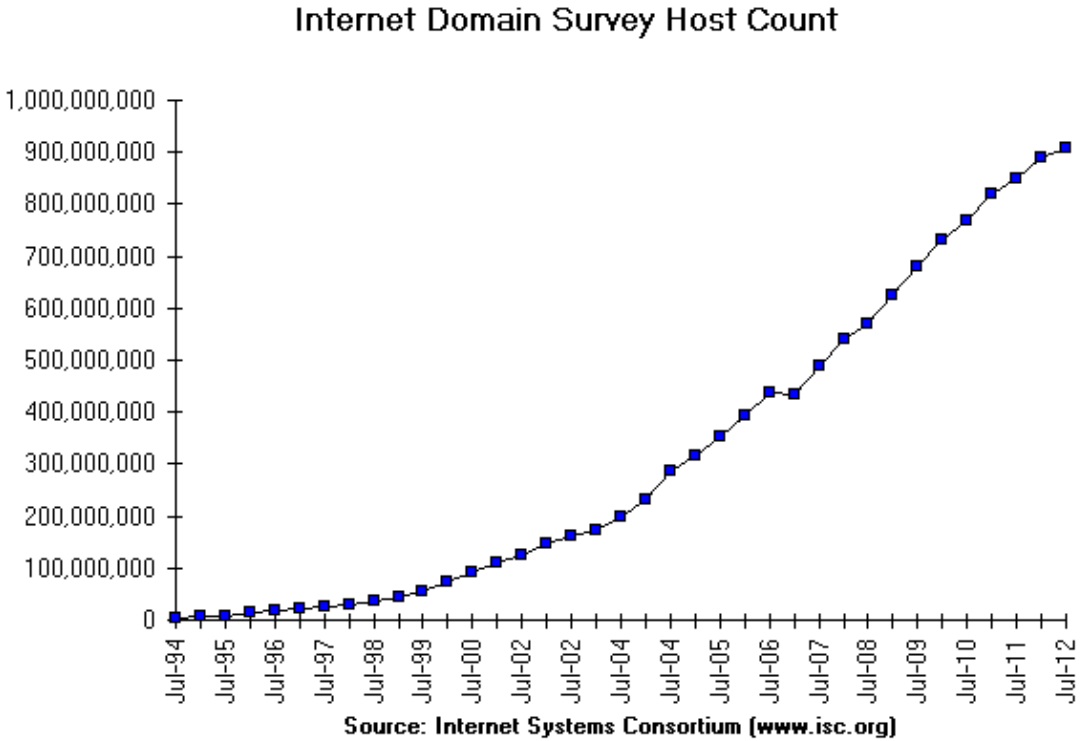
Topic

- IP version 6, the future of IPv4 that is now (still) being deployed



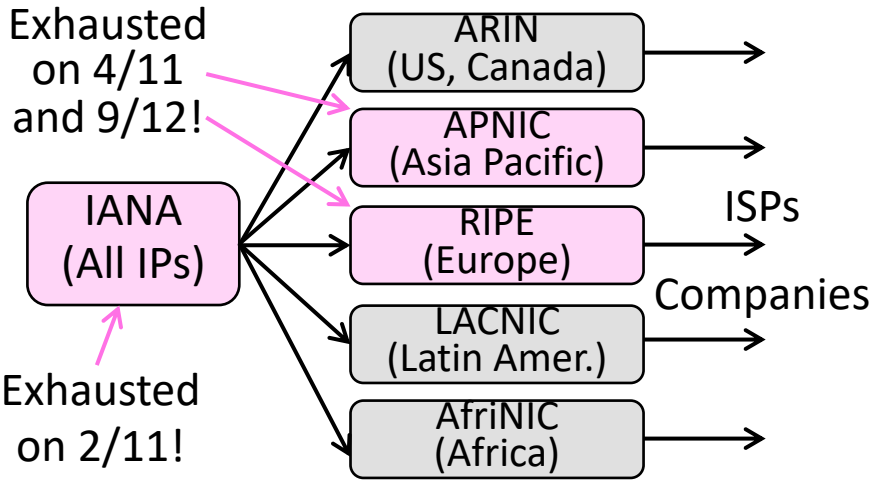
Internet Growth

- At least a billion Internet hosts and growing …
- And we're using 32-bit addresses!



The End of New IPv4 Addresses

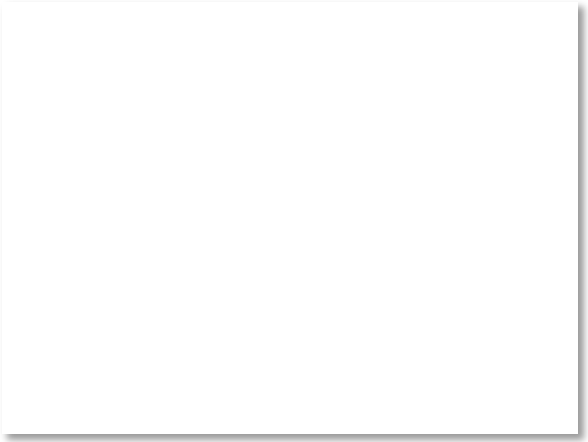
- Now running on leftover blocks held by the regional registries; much tighter allocation policies



End of the world ? 12/21/12?

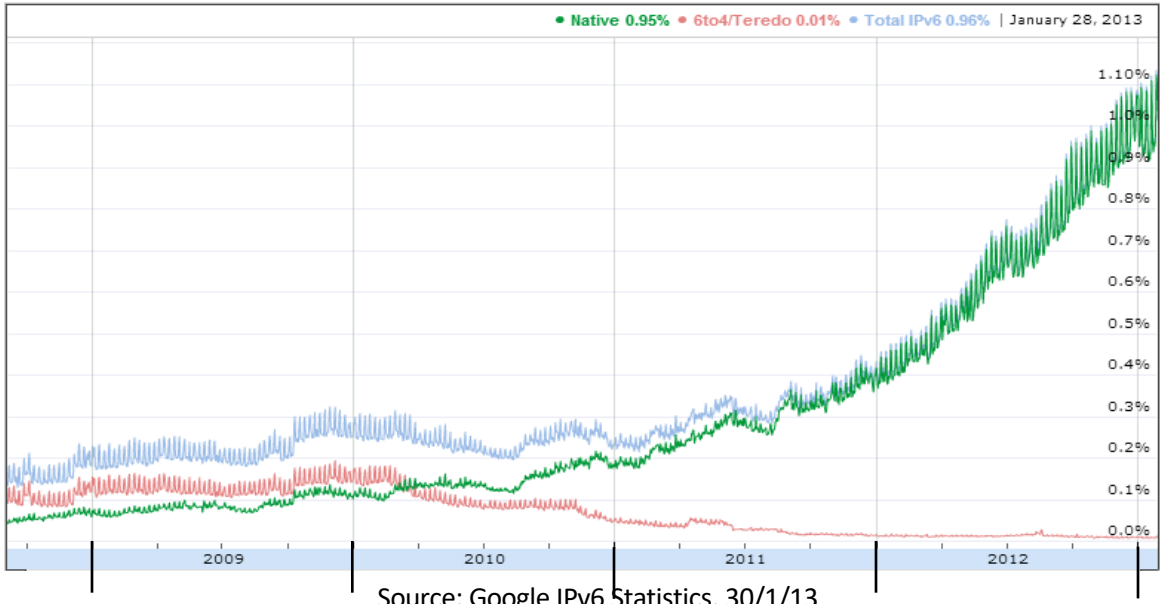
IP Version 6 to the Rescue

- Effort started by the IETF in 1994
  - Much larger addresses (128 bits)
  - Many sundry improvements
- Became an IETF standard in 1998
  - Nothing much happened for a decade
  - Hampered by deployment issues, and a lack of adoption incentives
  - Big push ~2011 as exhaustion looms



IPv6 Deployment

Percentage of users accessing Google via IPv6

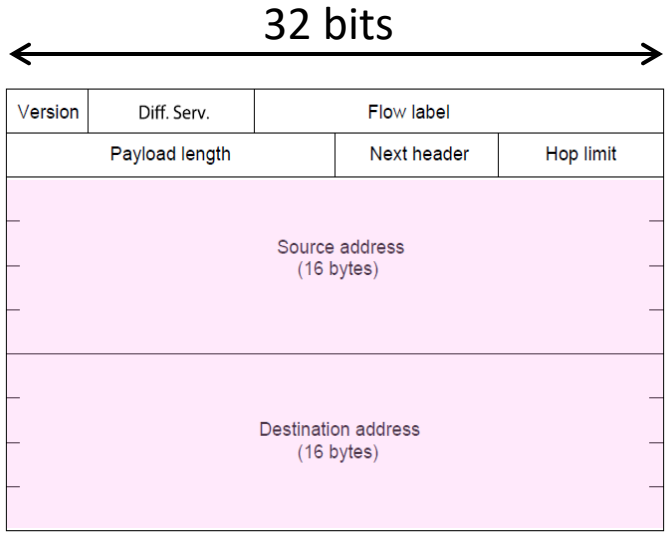


Source: Google IPv6 Statistics, 30/1/13

Time for growth!

IPv6

- Features large addresses
  - 128 bits, most of header
- New notation
  - 8 groups of 4 hex digits (16 bits)
  - Omit leading zeros, groups of zeros

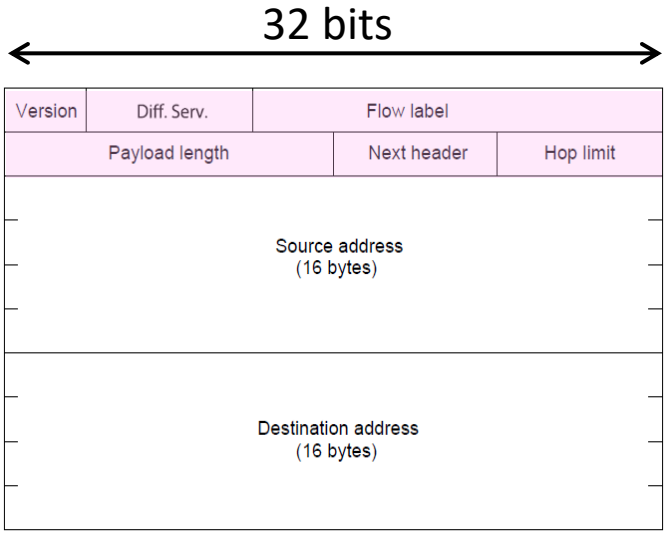


Ex: 2001:0db8:0000:0000:0000:ff00:0042:8329

→

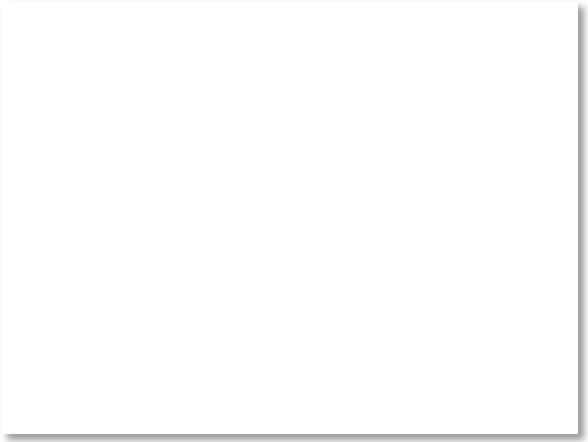
IPv6 (2)

- Lots of other, smaller changes
  - Streamlined header processing
  - Flow label to group of packets
  - Better fit with “advanced” features (mobility, multicasting, security)



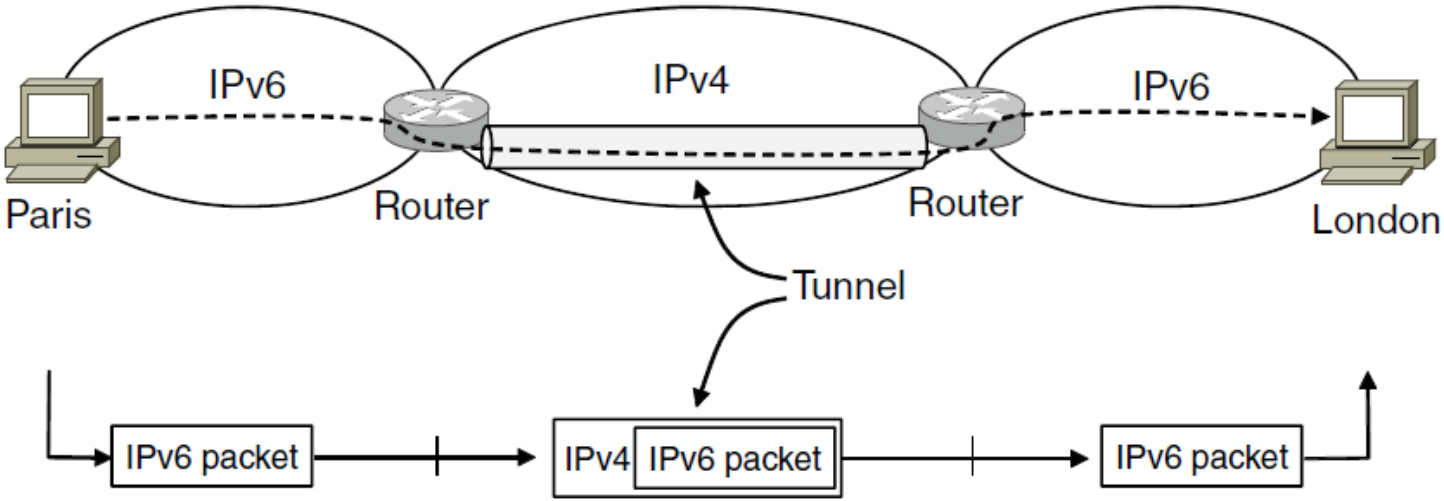
IPv6 Transition

- The Big Problem:
  - How to deploy IPv6?
  - Fundamentally incompatible with IPv4
- Dozens of approaches proposed
  - Dual stack (speak IPv4 and IPv6)
  - Translators (convert packets)
  - Tunnels (carry IPv6 over IPv4) »



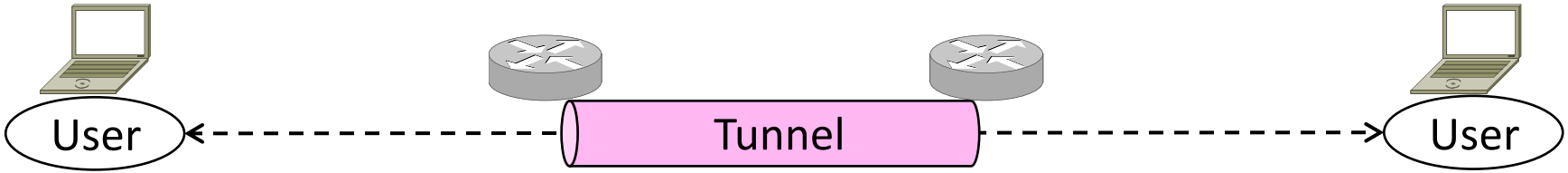
Tunneling

- Native IPv6 islands connected via IPv4
  - Tunnel carries IPv6 packets across IPv4 network



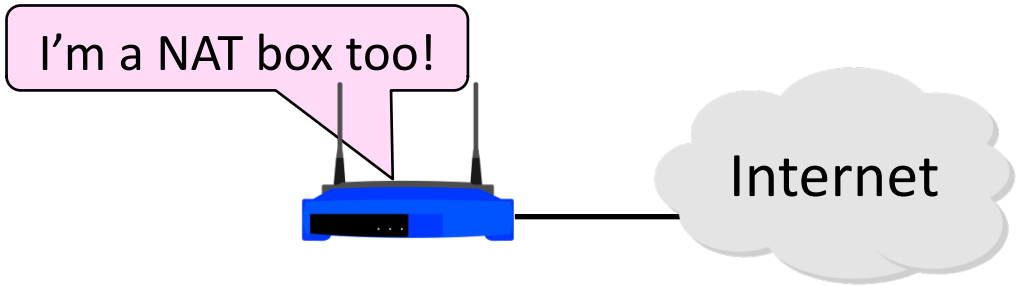
Tunneling (2)

- Tunnel acts as a single link across IPv4 network



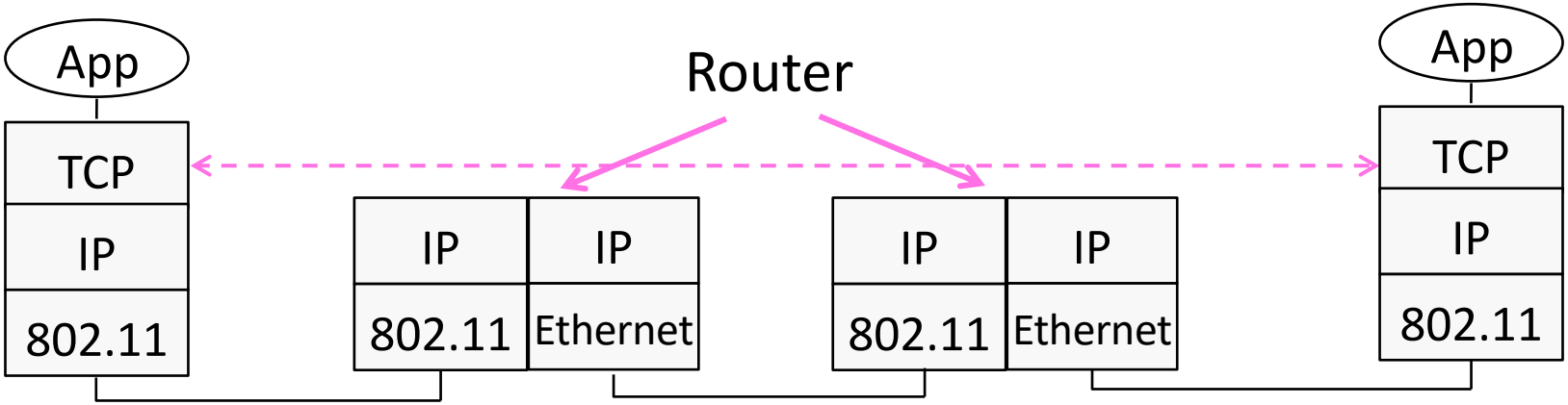
Topic

- What is NAT (Network Address Translation)? How does it work?
  - NAT is widely used at the edges of the network, e.g., homes



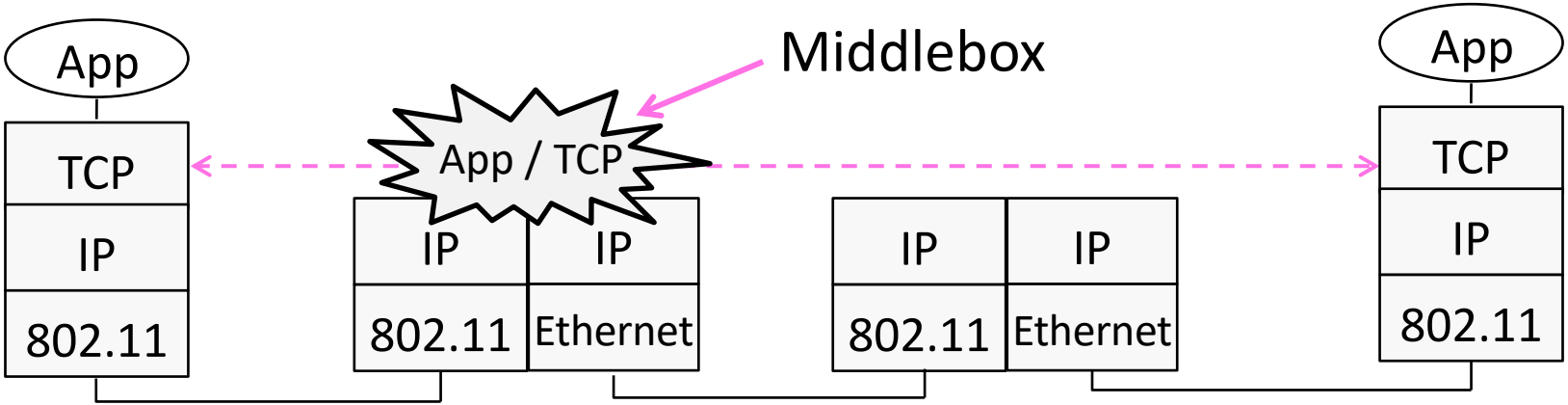
Layering Review

- Remember how layering is meant to work?
  - "Routers don't look beyond the IP header." Well …



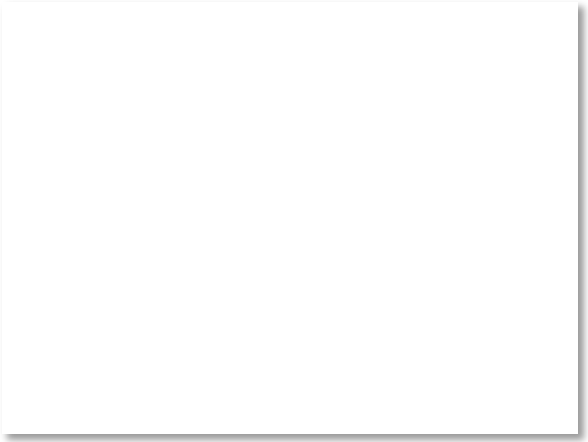
Middleboxes

- Sit “inside the network” but perform “more than IP” processing on packets to add new functionality
  - NAT box, Firewall / Intrusion Detection System



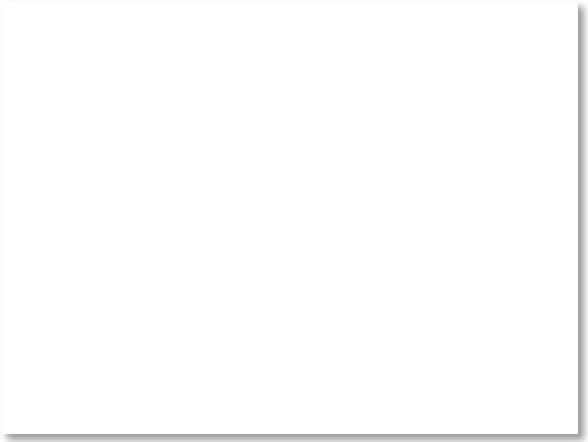
Middleboxes (2)

- Advantages
  - A possible rapid deployment path when there is no other option
  - Control over many hosts (IT)
- Disadvantages
  - Breaking layering interferes with connectivity; strange side effects
  - Poor vantage point for many tasks



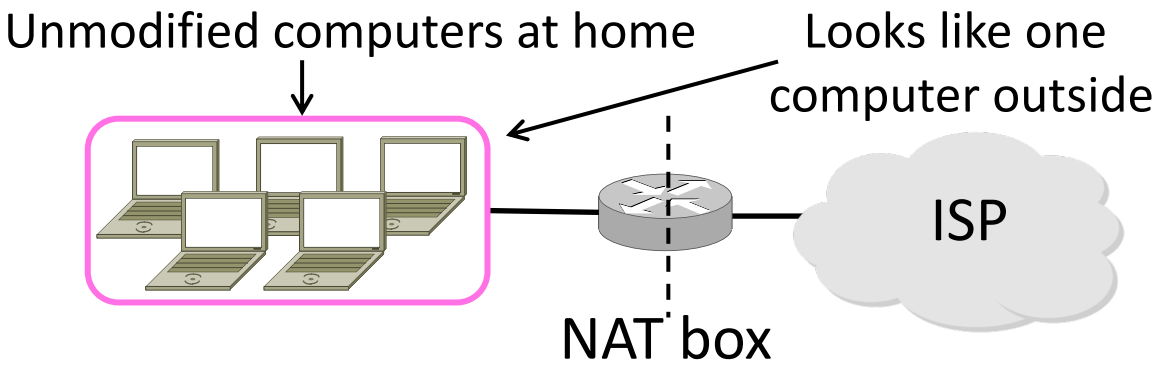
NAT (Network Address Translation) Box

- NAT box connects an internal network to an external network
  - Many internal hosts are connected using few external addresses
  - Middlebox that “translates addresses”
- Motivated by IP address scarcity
  - Controversial at first, now accepted



NAT (2)

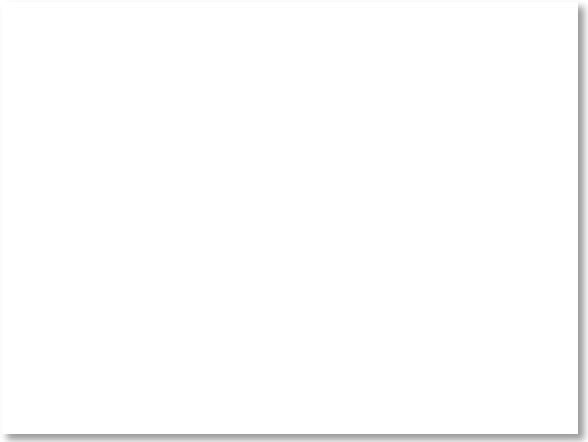
- Common scenario:
  - Home computers use “private” IP addresses
  - NAT (in AP/firewall) connects home to ISP using a single external IP address



How NAT Works

- Keeps an internal/external table
  - Typically uses IP address + TCP port
  - This is address and port translation

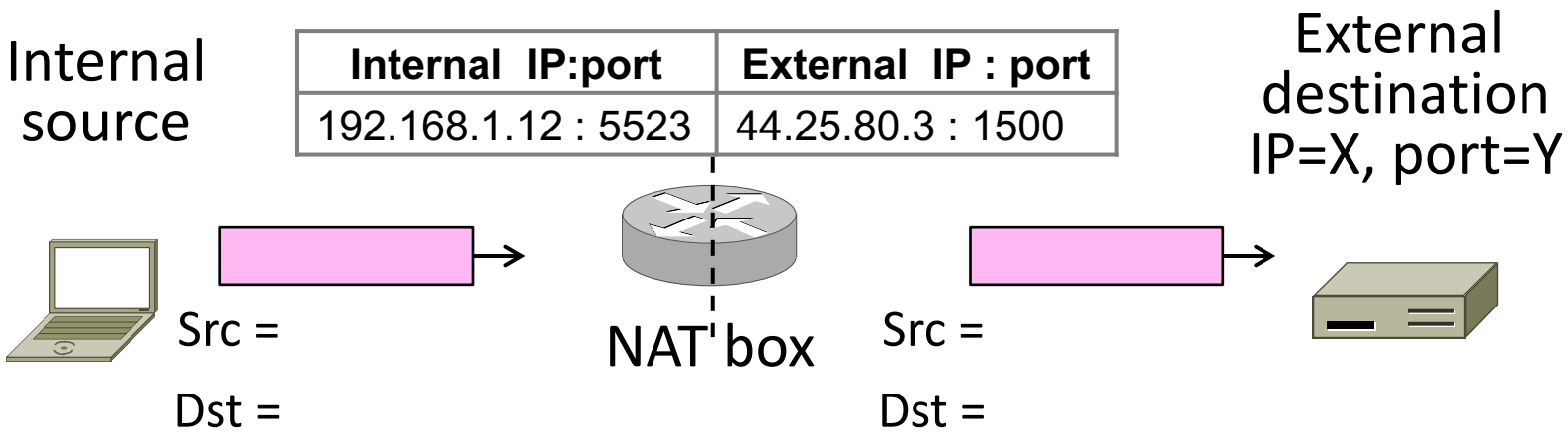
What host thinks	What ISP thinks
Internal IP:port	**External IP : port**
192.168.1.12 : 5523	44.25.80.3 : 1500
192.168.1.13 : 1234	44.25.80.3 : 1501
192.168.2.20 : 1234	44.25.80.3 : 1502



- Need ports to make mapping 1-1 since there are fewer external IPs

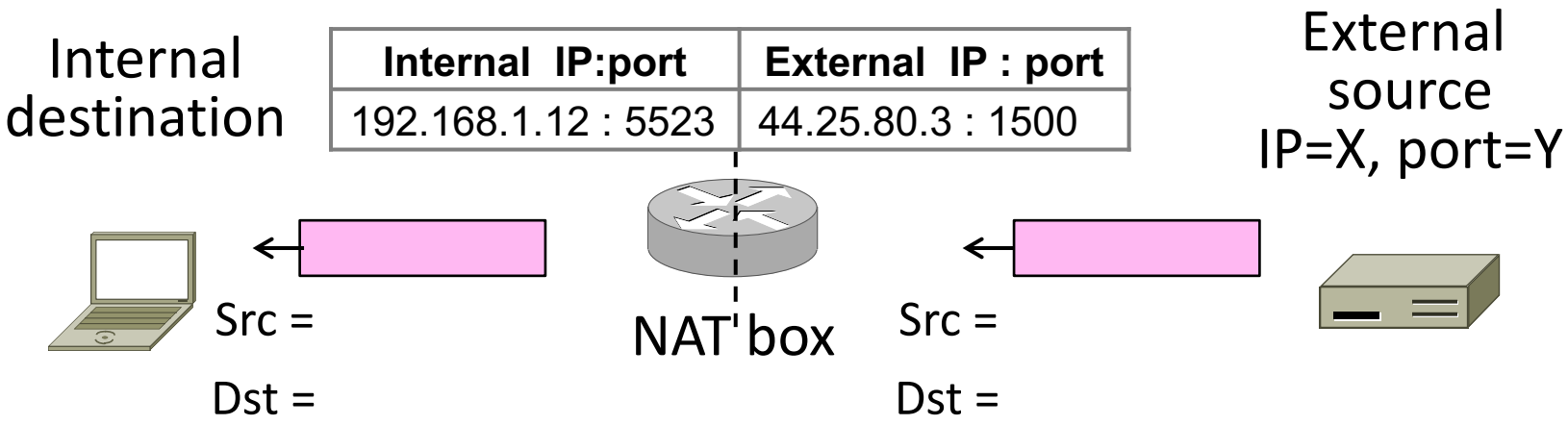
How NAT Works (2)

- Internal → External:
  - Look up and rewrite Source IP/port



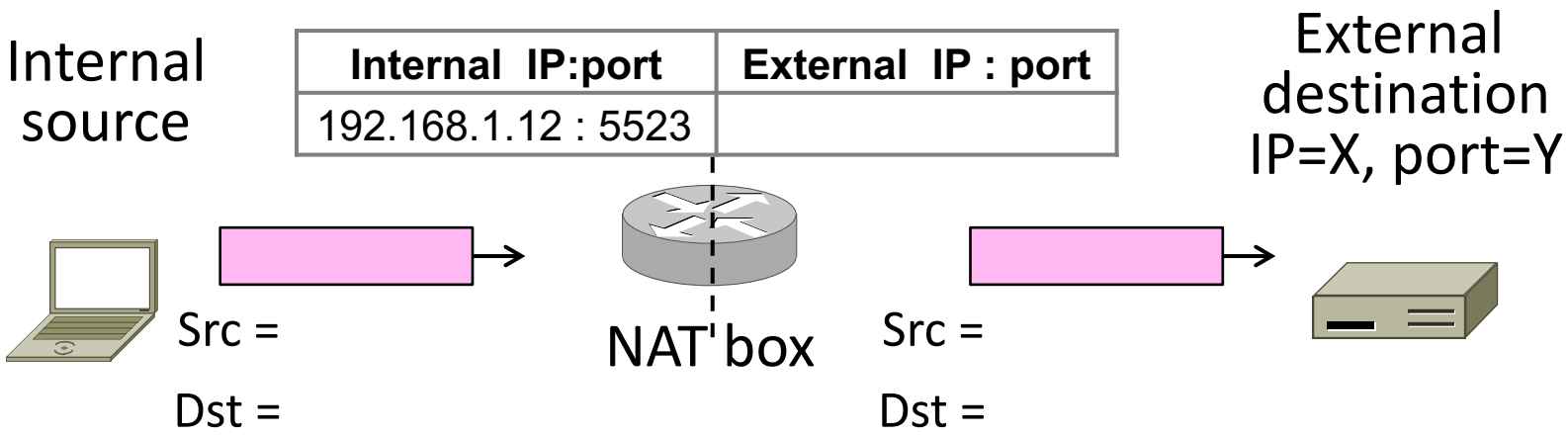
How NAT Works (3)

- External → Internal
  - Look up and rewrite Destination IP/port



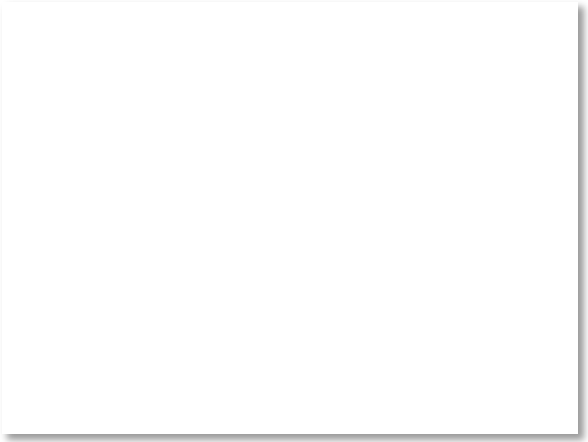
How NAT Works (4)

- Need to enter translations in the table for it to work
  - Create external name when host makes a TCP connection



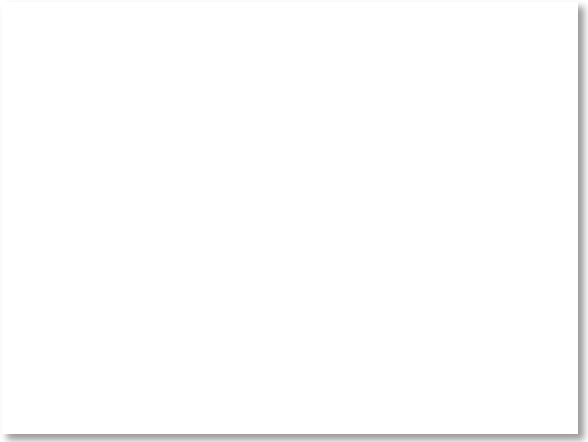
NAT Downsides

- Connectivity has been broken!
  - Can only send incoming packets after an outgoing connection is set up
  - Difficult to run servers or peer-to-peer apps (Skype) at home
- Doesn't work so well when there are no connections (UDP apps)
- Breaks apps that unwisely expose their IP addresses (FTP)



NAT Upsides

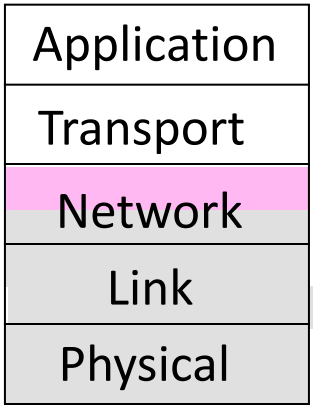
- Relieves much IP address pressure
  - Many home hosts behind NATs
- Easy to deploy
  - Rapidly, and by you alone
- Useful functionality
  - Firewall, helps with privacy



- Kinks will get worked out eventually
  - "NAT Traversal" for incoming traffic

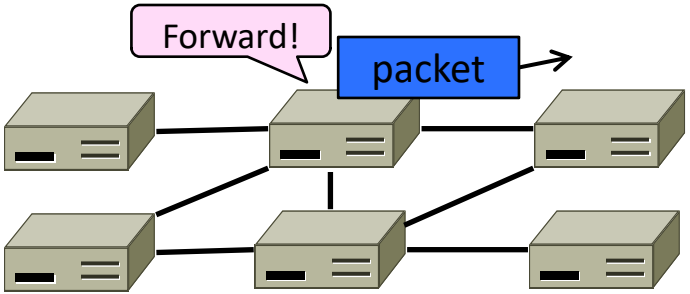
Where we are in the Course

- More fun in the Network Layer!
  - We've covered packet forwarding
  - Now we'll learn about routing

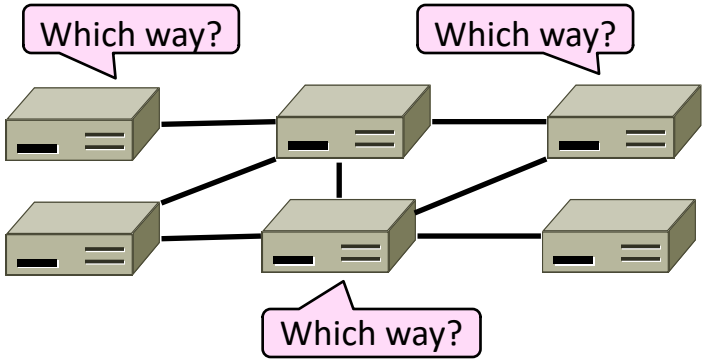


Routing versus Forwarding

- Forwarding is the process of sending a packet on its way

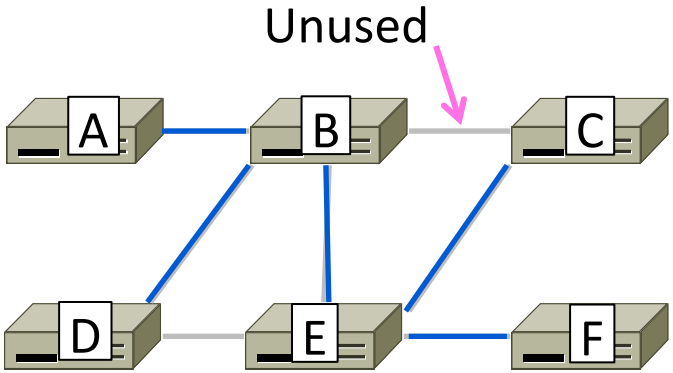


- Routing is the process of deciding in which direction to send traffic

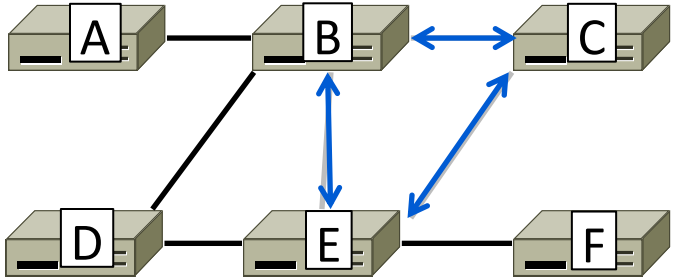


Improving on the Spanning Tree

- Spanning tree provides basic connectivity
  - e.g., some path B→C



- Routing uses all links to find "best" paths
  - e.g., use BC, BE, and CE



Perspective on Bandwidth Allocation

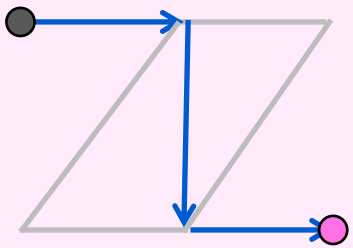
- Routing allocates network bandwidth adapting to failures; other mechanisms used at other timescales

Mechanism	Timescale / Adaptation
Load-sensitive routing	Seconds / Traffic hotspots
Routing	Minutes / Equipment failures
Traffic Engineering	Hours / Network load
Provisioning	Months / Network customers

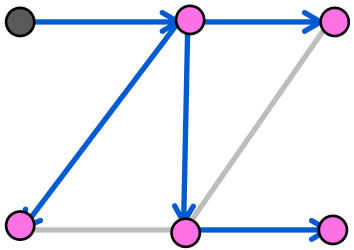
Delivery Models

- Different routing used for different delivery models

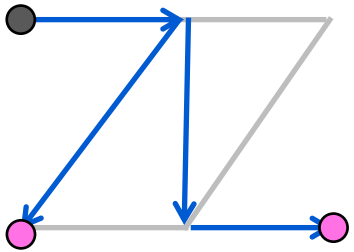
Unicast (§5.2)



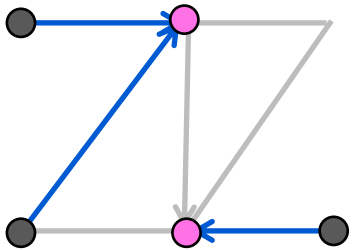
Broadcast (§5.2.7)



Multicast (§5.2.8)



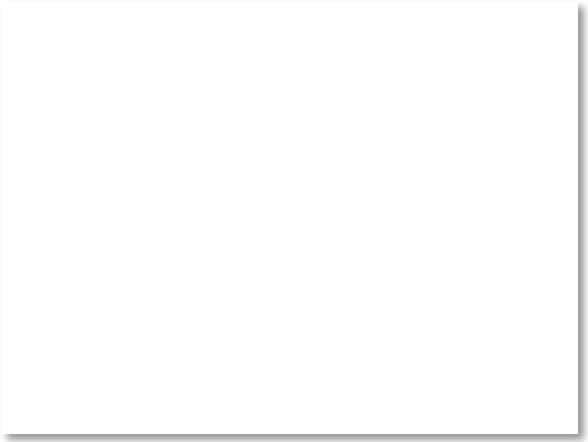
Anycast (§5.2.9)



Goals of Routing Algorithms

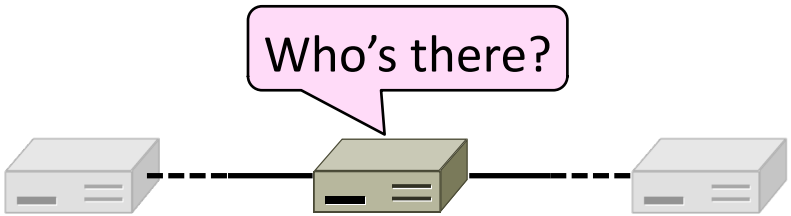
- We want several properties of any routing scheme:

Property	Meaning
Correctness	Finds paths that work
Efficient paths	Uses network bandwidth well
Fair paths	Doesn't starve any nodes
Fast convergence	Recovers quickly after changes
Scalability	Works well as network grows large



Rules of Routing Algorithms

- Decentralized, distributed setting
  - All nodes are alike; no controller
  - Nodes only know what they learn by exchanging messages with neighbors
  - Nodes operate concurrently
  - May be node/link/message failures

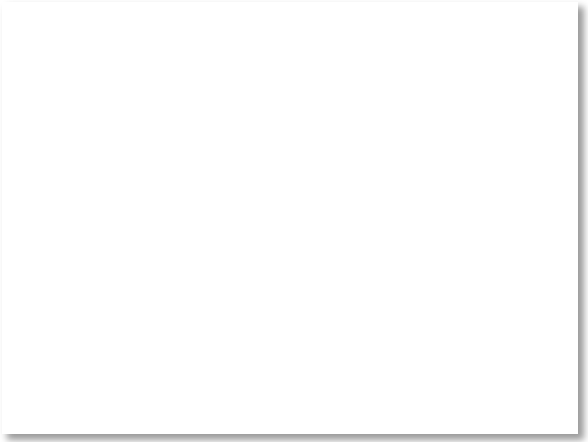


Topics

- IPv4, IPv6, NATs and all that } Last time

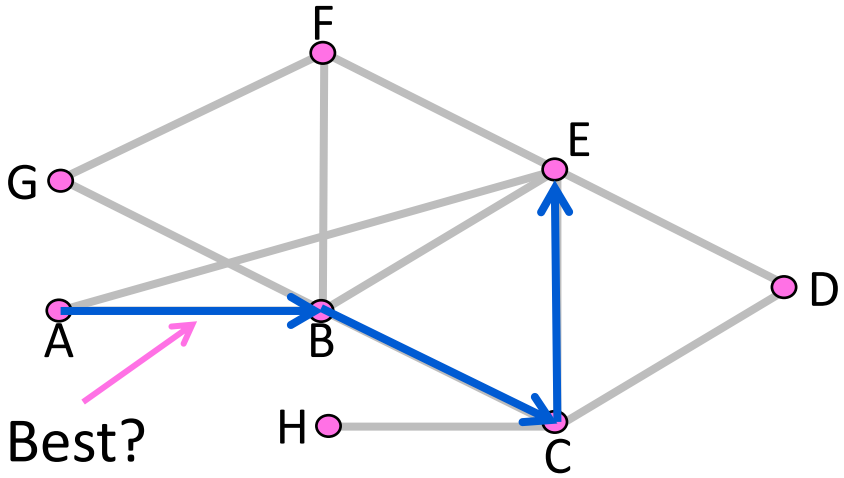
- Shortest path routing
- Distance Vector routing
- Flooding
- Link-state routing
- Equal-cost multi-path
- Inter-domain routing (BGP)

} This time



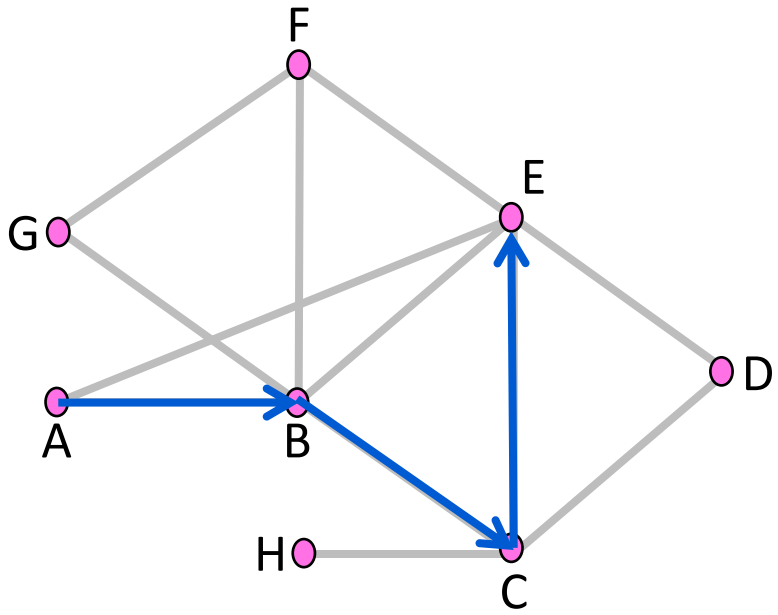
Topic

- Defining “best” paths with link costs
  - These are shortest path routes



What are “Best” paths anyhow?

- Many possibilities:
  - Latency, avoid circuitous paths
  - Bandwidth, avoid slow links
  - Money, avoid expensive links
  - Hops, to reduce switching
- But only consider topology
  - Ignore workload, e.g., hotspots

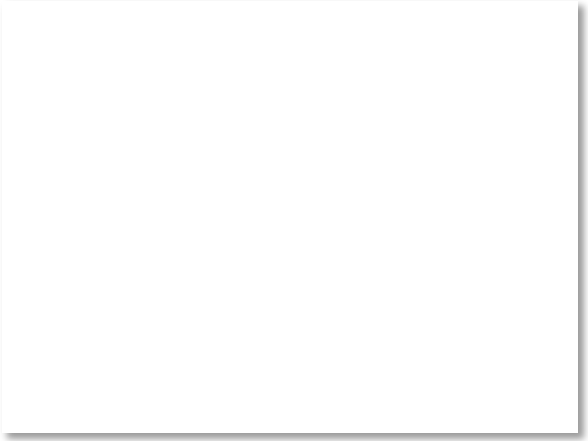


Shortest Paths

We'll approximate "best" by a cost function that captures the factors

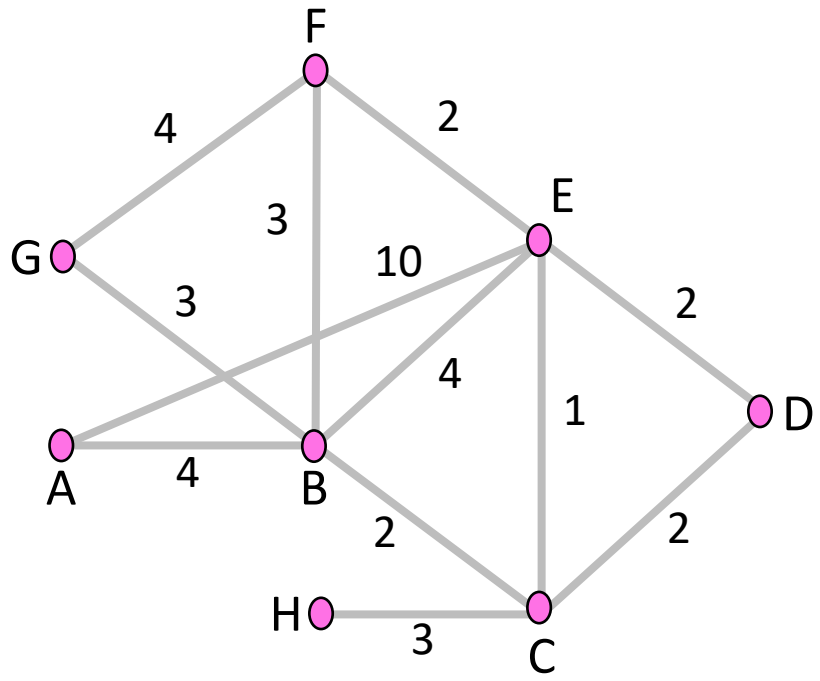
- Often call lowest "shortest"

1. Assign each link a cost (distance)
2. Define best path between each pair of nodes as the path that has the lowest total cost (or is shortest)
3. Pick randomly to any break ties



Shortest Paths (2)

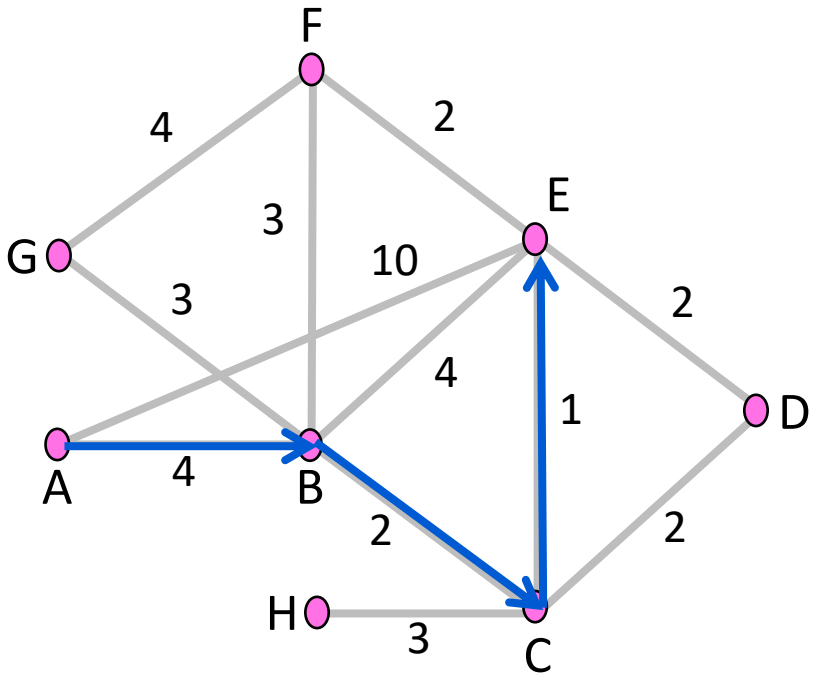
- Find the shortest path A → E
- All links are bidirectional, with equal costs in each direction
  - Can extend model to unequal costs if needed



Shortest Paths (3)

- ABCE is a shortest path
- dist(ABCE) = 4 + 2 + 1 = 7

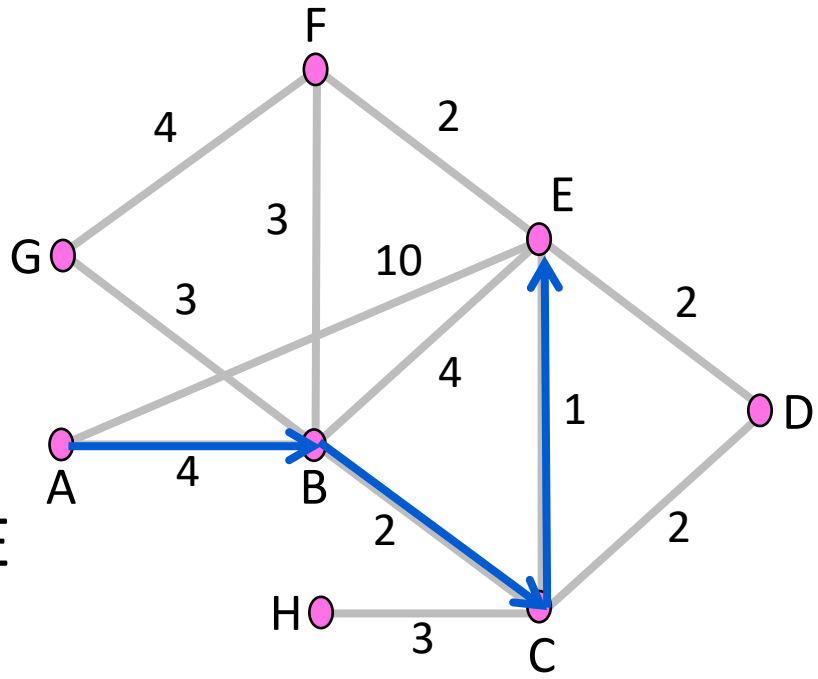
- This is less than:
  - dist(ABE) = 8
  - dist(ABFE) = 9
  - dist(AE) = 10
  - dist(ABCDE) = 10



Shortest Paths (4)

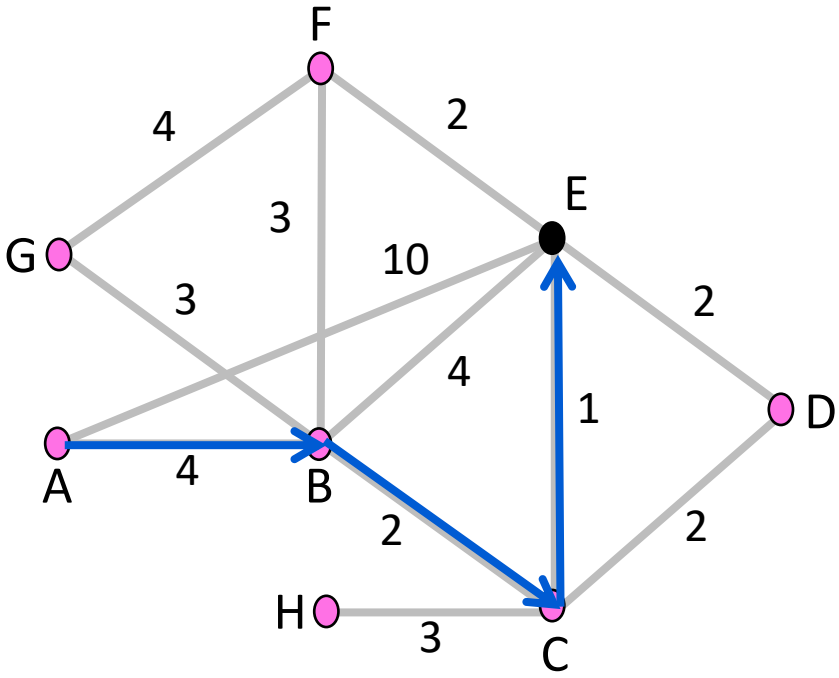
- Optimality property:
  - Subpaths of shortest paths are also shortest paths
- ABCE is a shortest path

→So are ABC, AB, BCE, BC, CE



Sink Trees

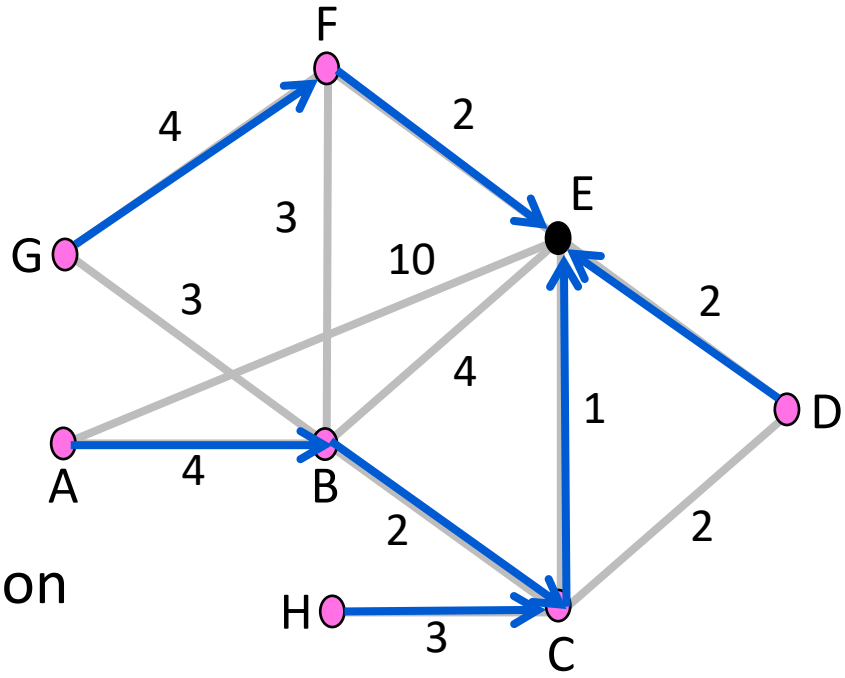
- Sink tree for a destination is the union of all shortest paths towards the destination
  - Similarly source tree
- Find the sink tree for E



Sink Trees (2)

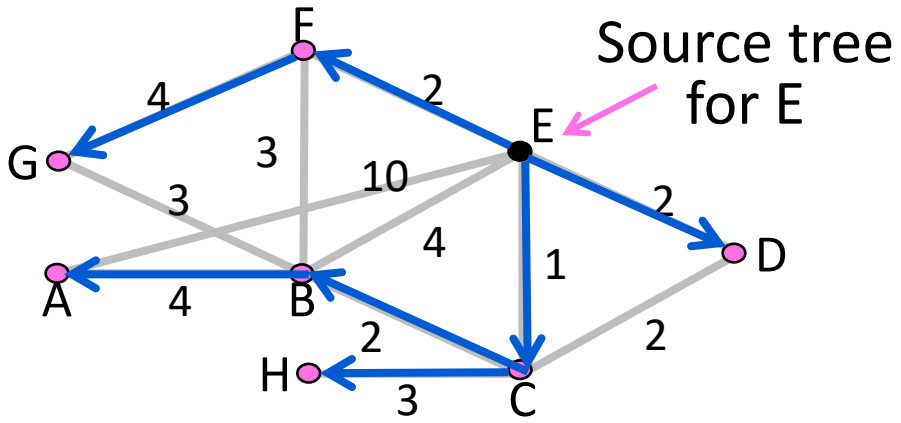
- Implications:
  - Only need to use destination to follow shortest paths
  - Each node only need to send to the next hop

- Forwarding table at a node
  - Lists next hop for each destination
  - Routing table may know more



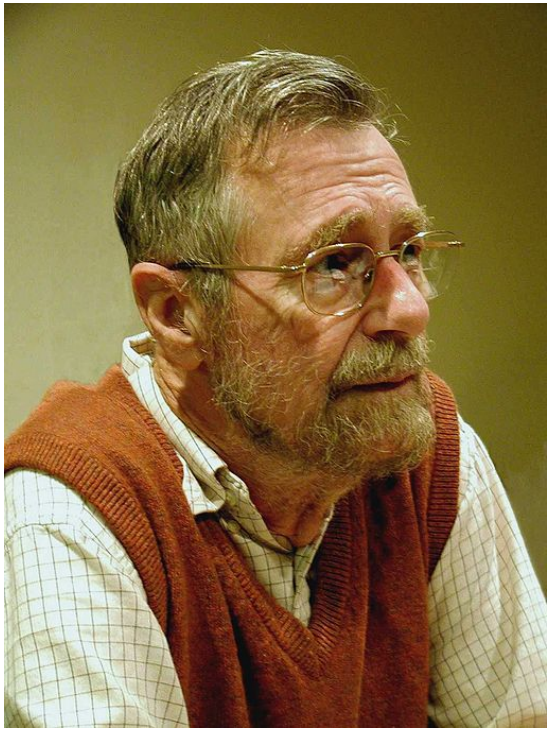
Topic

- How to compute shortest paths given the network topology
  - With Dijkstra's algorithm



Edsger W. Dijkstra (1930-2002)

- Famous computer scientist
  - Programming languages
  - Distributed algorithms
  - Program verification
- Dijkstra's algorithm, 1969
  - Single-source shortest paths, given network with non-negative link costs



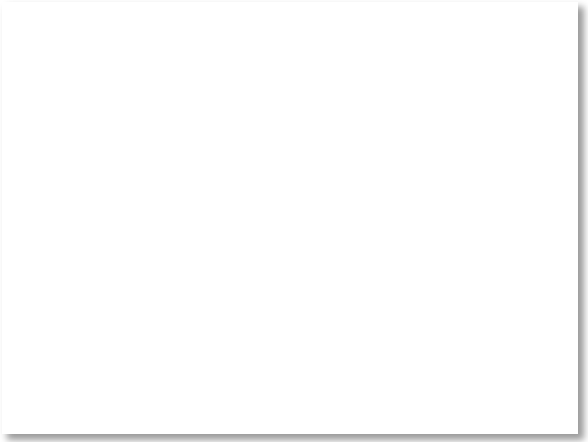
By Hamilton Richards, CC-BY-SA-3.0, via Wikimedia Commons

Dijkstra's Algorithm

Algorithm:

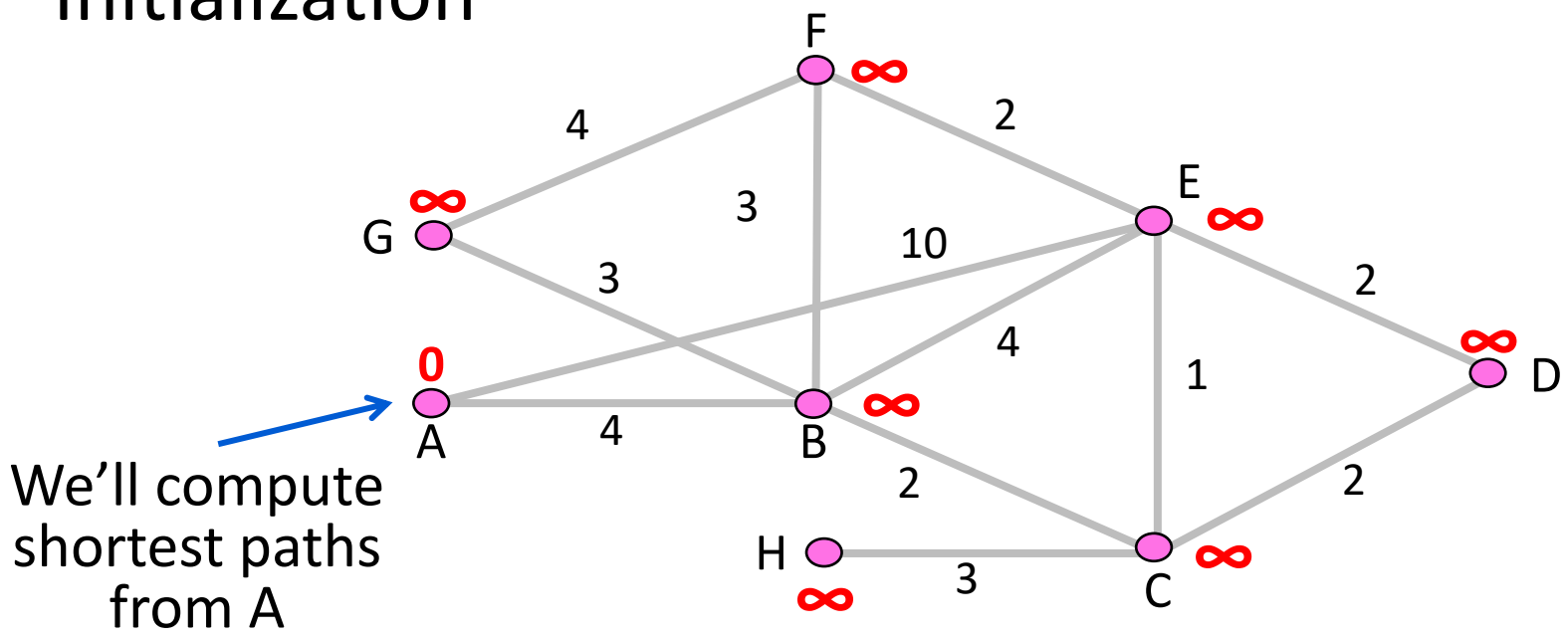
- Mark all nodes tentative, set distances from source to 0 (zero) for source, and ∞ (infinity) for all other nodes

- While tentative nodes remain:
  - Extract N, a node with lowest distance
  - Add link to N to the shortest path tree
  - Relax the distances of neighbors of N by lowering any better distance estimates



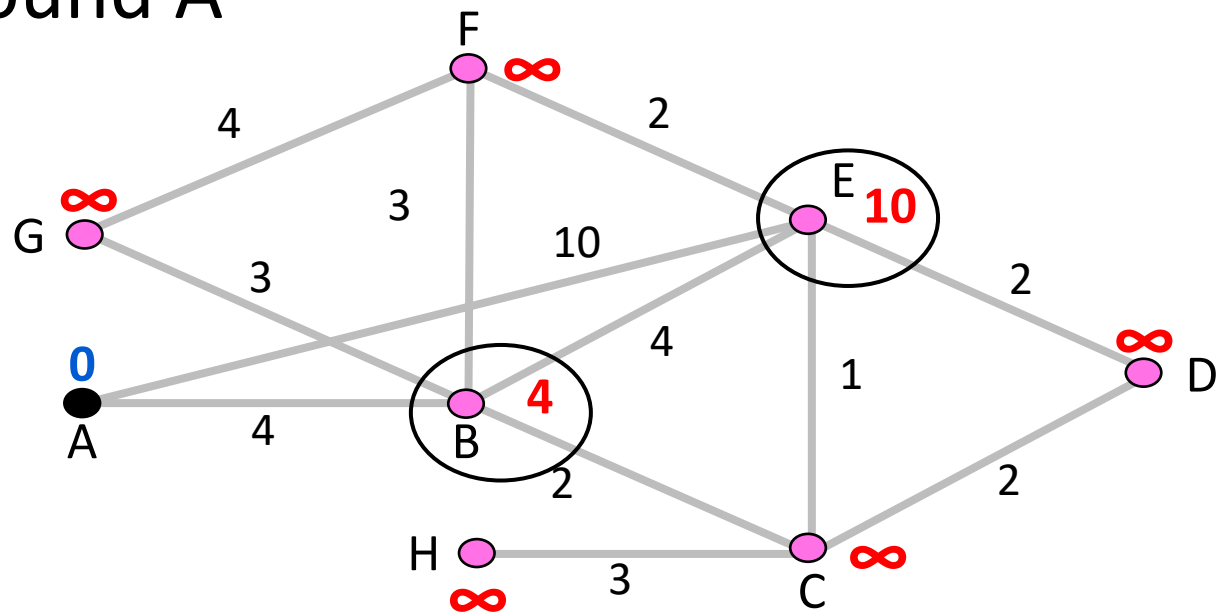
Dijkstra's Algorithm (2)

- Initialization



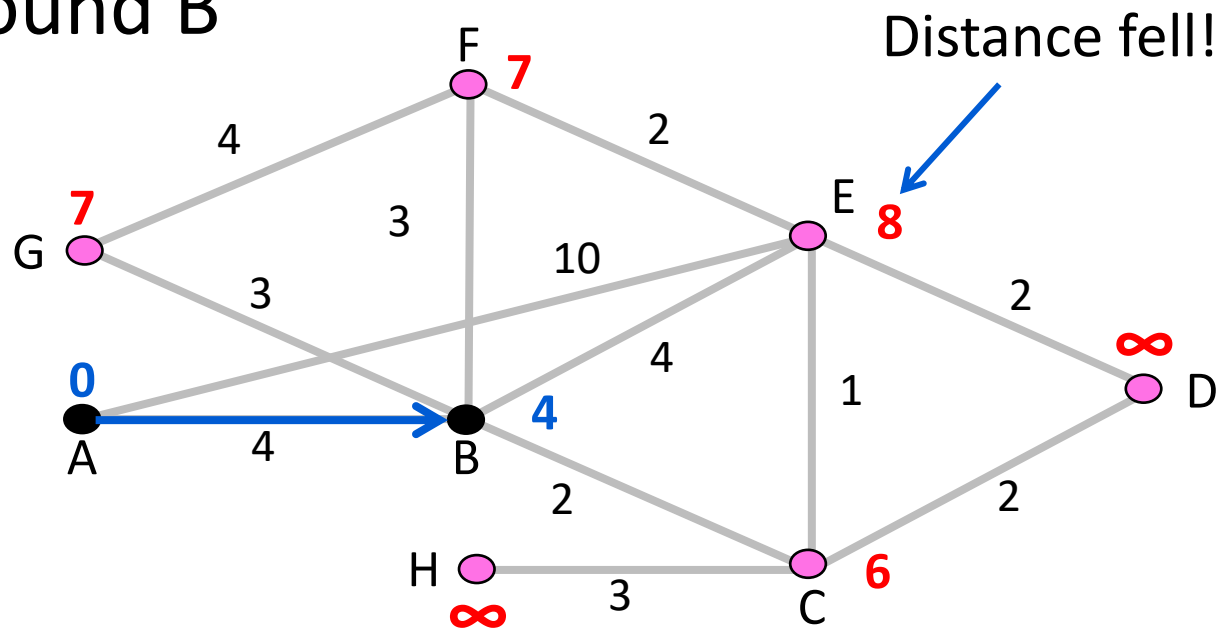
Dijkstra's Algorithm (3)

- Relax around A



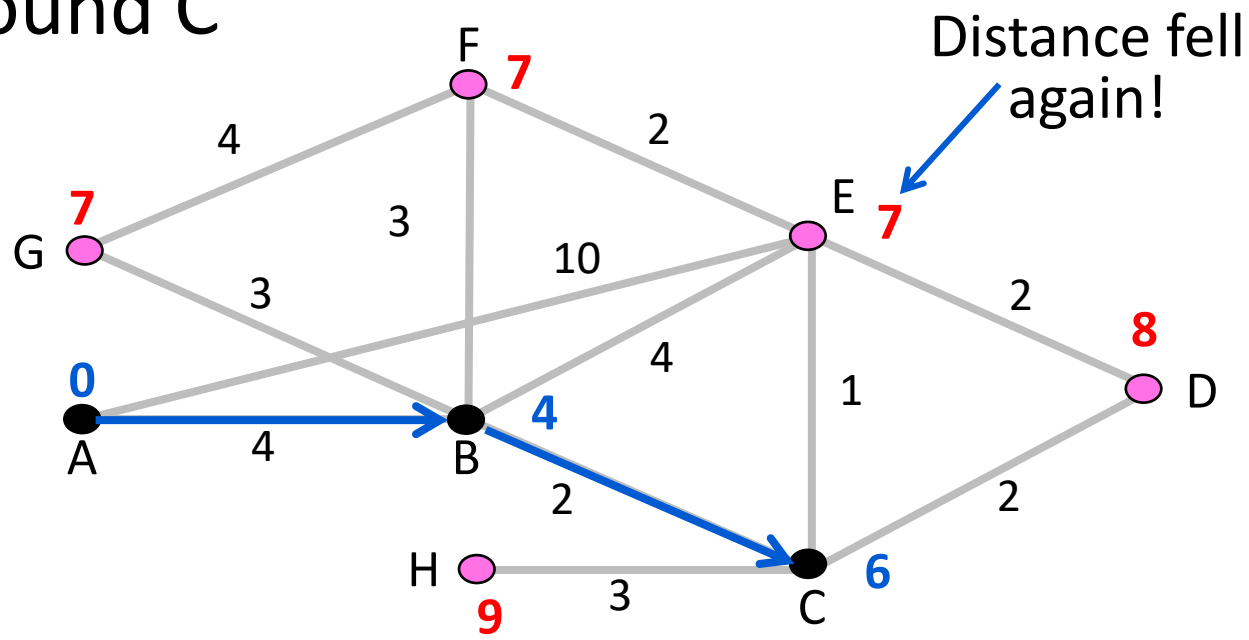
Dijkstra's Algorithm (4)

- Relax around B



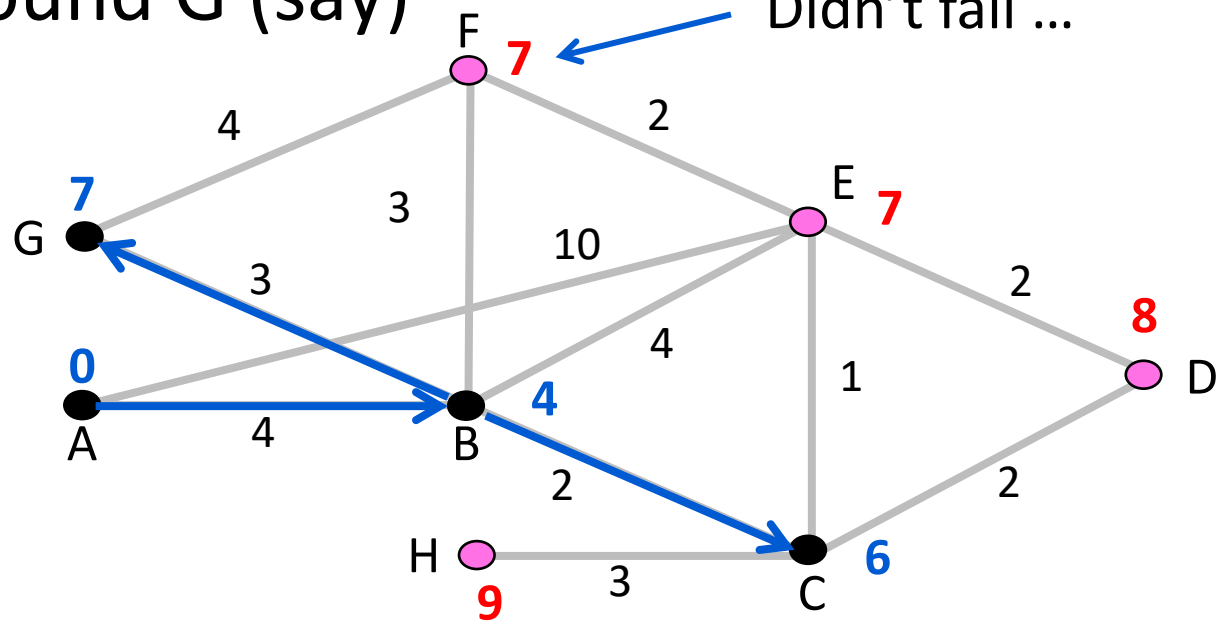
Dijkstra's Algorithm (5)

- Relax around C



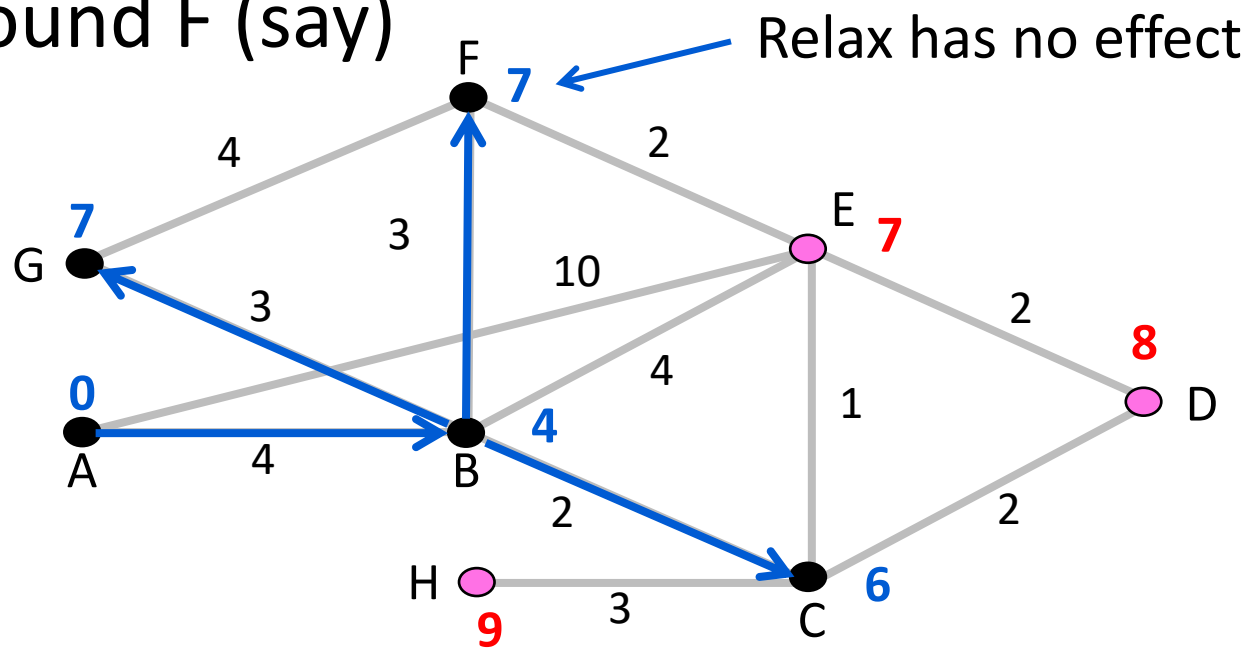
Dijkstra's Algorithm (6)

- Relax around G (say)



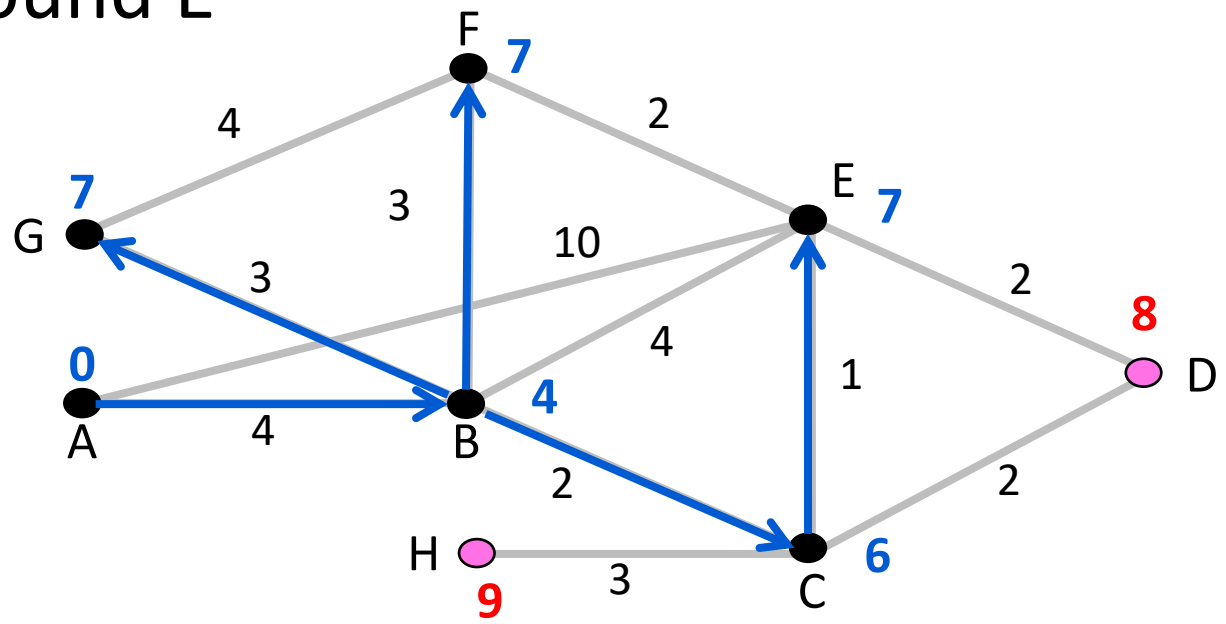
Dijkstra's Algorithm (7)

- Relax around F (say)



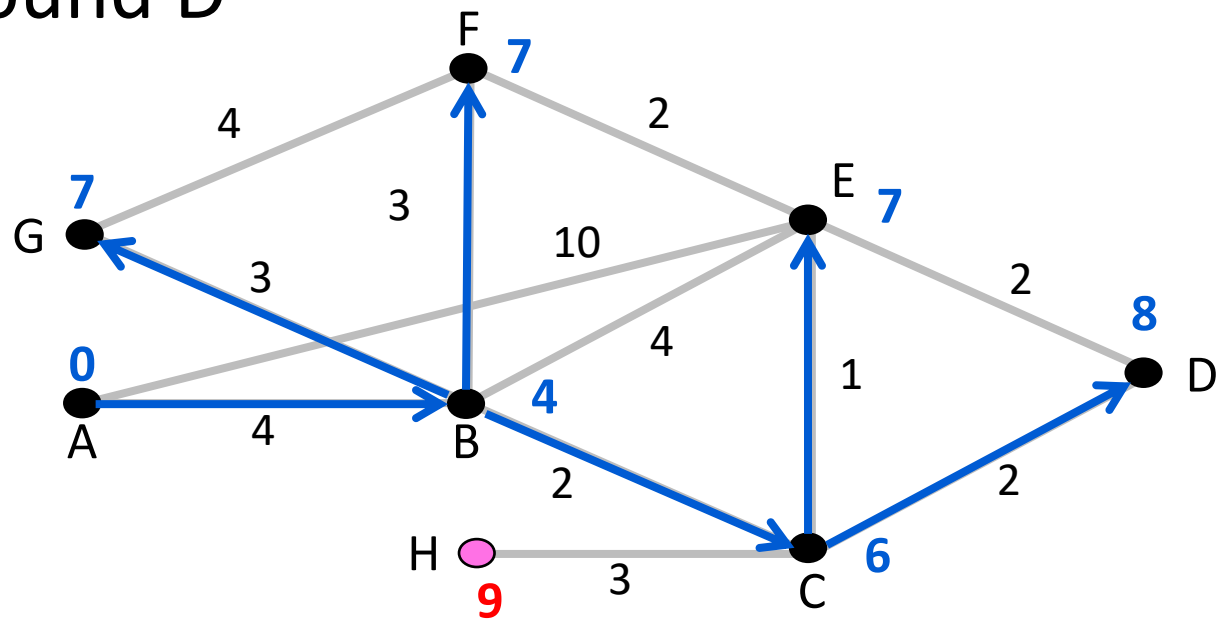
Dijkstra's Algorithm (8)

- Relax around E



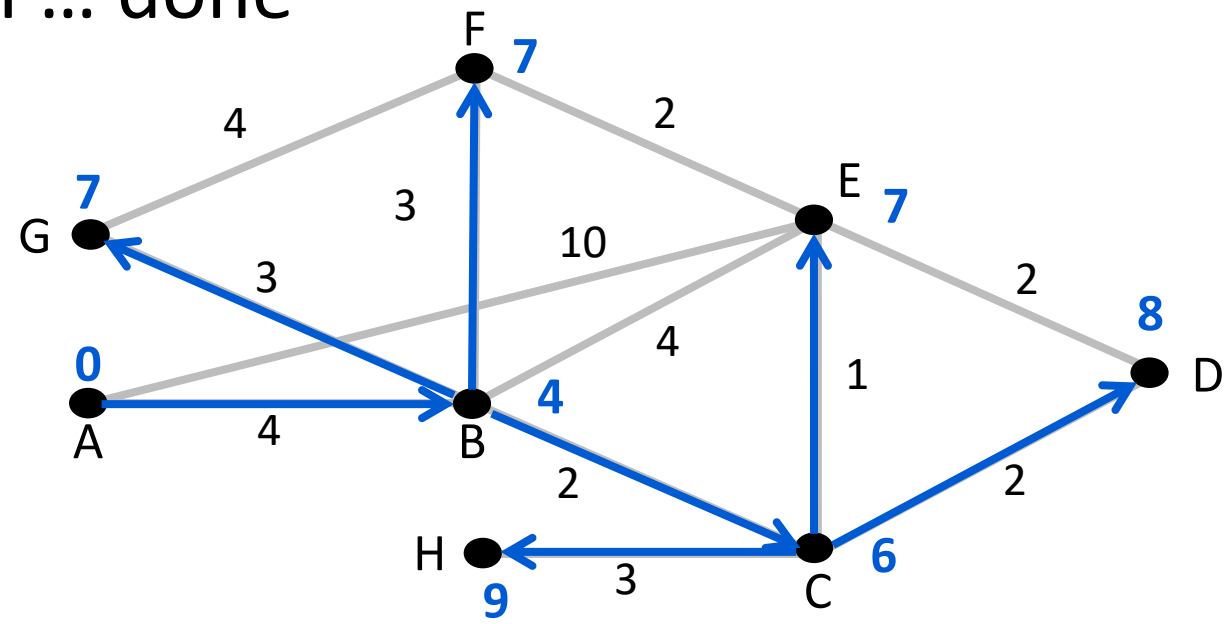
Dijkstra's Algorithm (9)

- Relax around D



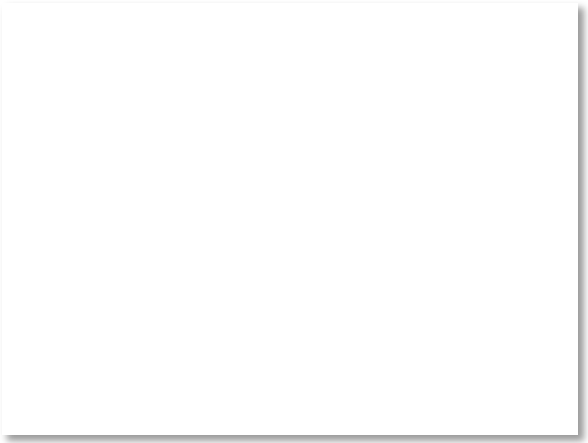
Dijkstra's Algorithm (10)

- Finally, H … done



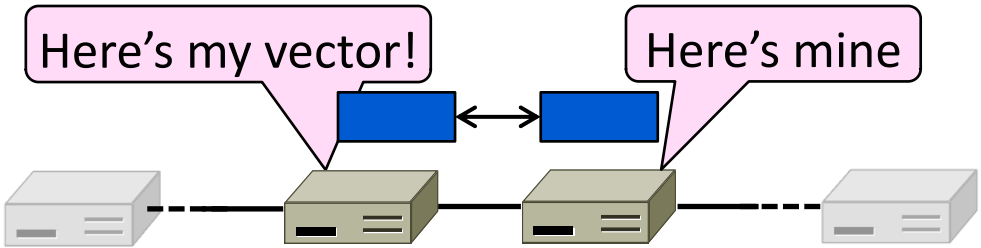
Dijkstra Comments

- Finds shortest paths in order of increasing distance from source
  - Leverages optimality property
- Runtime depends on efficiency of extracting min-cost node
  - Superlinear in network size (grows fast)
- Gives complete source/sink tree
  - More than needed for forwarding!
  - But requires complete topology



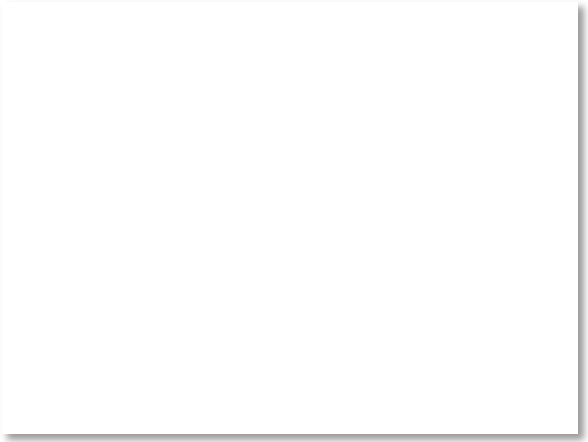
Topic

- How to compute shortest paths in a distributed network
  - The Distance Vector (DV) approach



Distance Vector Routing

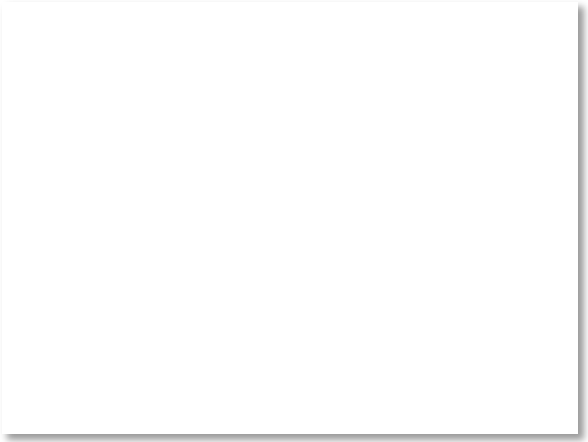
- Simple, early routing approach
  - Used in ARPANET, and RIP
- One of two main approaches to routing
  - Distributed version of Bellman-Ford
  - Works, but very slow convergence after some failures
- Link-state algorithms are now typically used in practice
  - More involved, better behavior



Distance Vector Setting

Each node computes its forwarding table in a distributed setting:

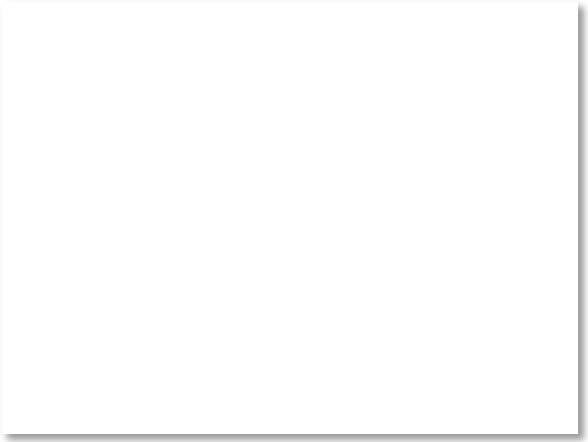
1. Nodes know only the cost to their neighbors; not the topology
2. Nodes can talk only to their neighbors using messages
3. All nodes run the same algorithm concurrently
4. Nodes and links may fail, messages may be lost



Distance Vector Algorithm

Each node maintains a vector of distances (and next hops) to all destinations

1. Initialize vector with 0 (zero) cost to self, ∞ (infinity) to other destinations
2. Periodically send vector to neighbors
3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
   - Use the best neighbor for forwarding

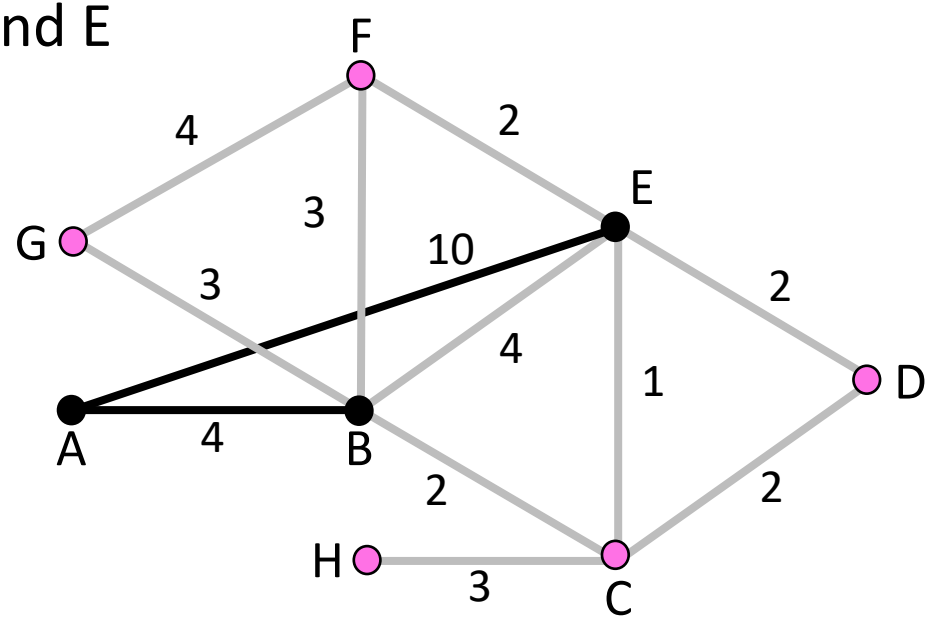


Distance Vector (2)

- Consider from the point of view of node A
  - Can only talk to nodes B and E

Initial vector →

To	Cost
A	0
B	∞
C	∞
D	∞
E	∞
F	∞
G	∞
H	∞



Distance Vector (3)

- First exchange with B, E; learn best 1-hop routes

To	B says	E says
A	∞	∞
B	0	∞
C	∞	∞
D	∞	∞
E	∞	0
F	∞	∞
G	∞	∞
H	∞	∞

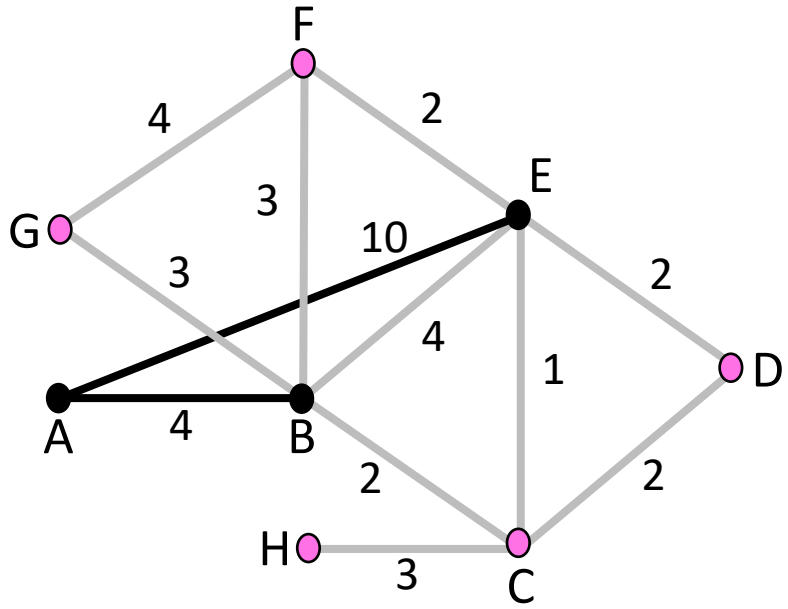
→

B +4	E +10
∞	∞
4	∞
∞	∞
∞	∞
∞	10
∞	∞
∞	∞
∞	∞

→

A's Cost	A's Next
0	--
4	B
∞	--
∞	--
10	E
∞	--
∞	--
∞	--

Learned better route



Distance Vector (4)

- Second exchange; learn best 2-hop routes

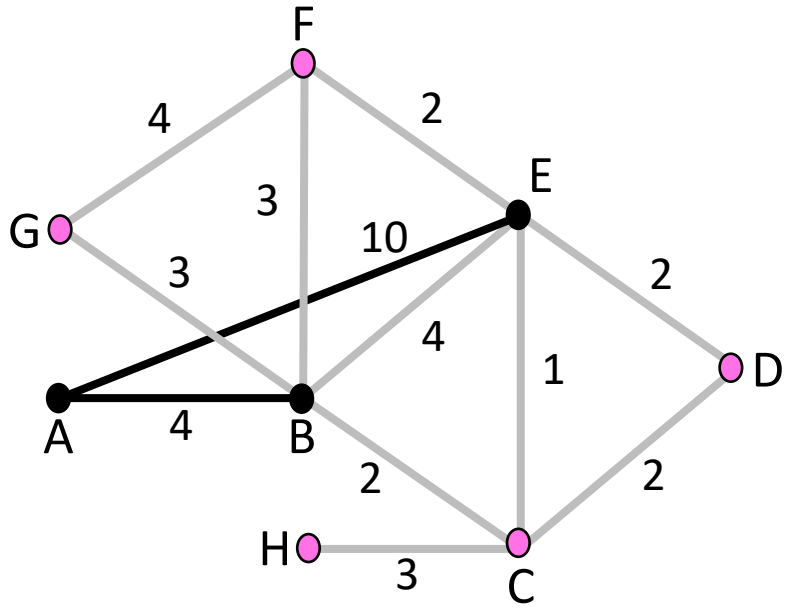
To	B says	E says
A	4	10
B	0	4
C	2	1
D	∞	2
E	4	0
F	3	2
G	3	∞
H	∞	∞

→

B +4	E +10
8	20
4	14
6	11
∞	12
8	10
7	12
7	∞
∞	∞

→

A's Cost	A's Next
0	--
4	B
6	B
12	E
8	B
7	B
7	B
∞	--



Distance Vector (4)

- Third exchange; learn best 3-hop routes

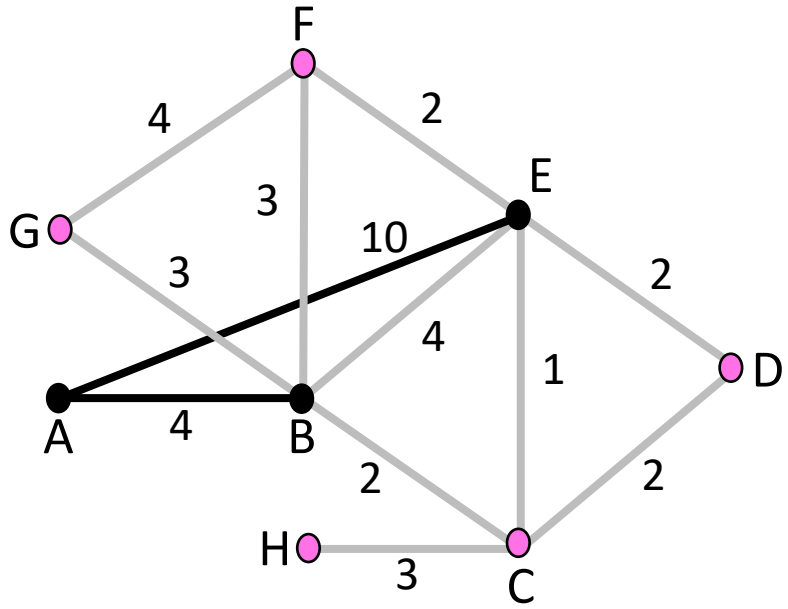
To	B says	E says
A	4	8
B	0	3
C	2	1
D	4	2
E	3	0
F	3	2
G	3	6
H	5	4

→

B +4	E +10
8	18
4	13
6	11
8	12
7	10
7	12
7	16
9	14

→

A's Cost	A's Next
0	--
4	B
6	B
8	B
7	B
7	B
7	B
9	B



Distance Vector (5)

- Subsequent exchanges; converged

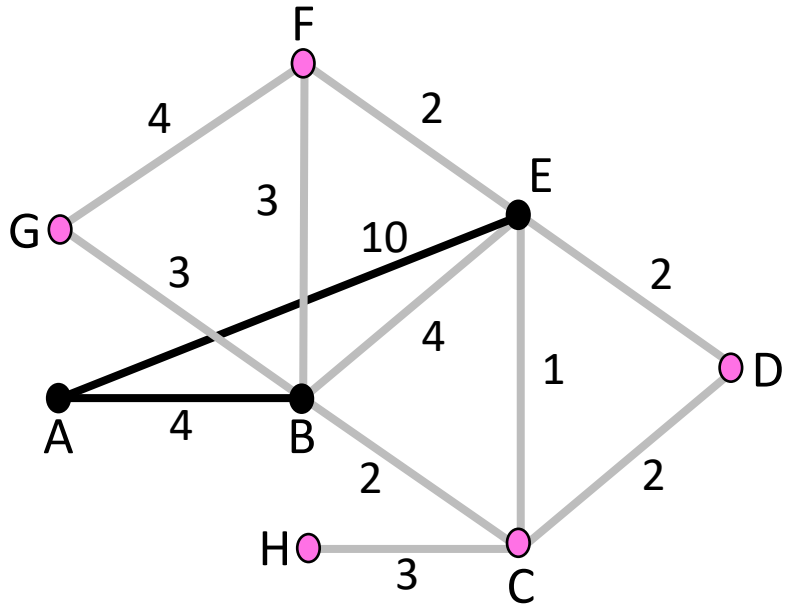
To	B says	E says
A	4	7
B	0	3
C	2	1
D	4	2
E	3	0
F	3	2
G	3	6
H	5	4

→

B +4	E +10
8	17
4	13
6	11
8	12
7	10
7	12
7	16
9	14

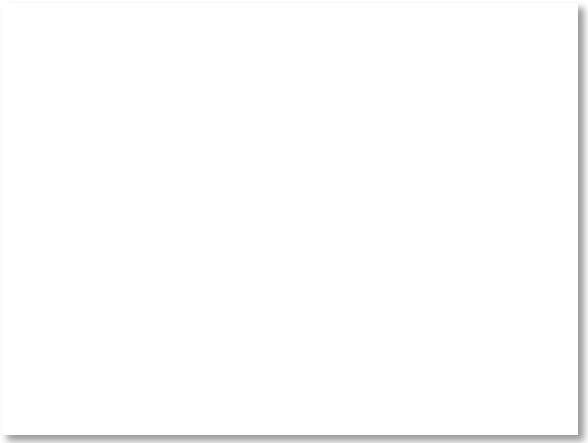
→

A's Cost	A's Next
0	--
4	B
6	B
8	B
8	B
7	B
7	B
9	B



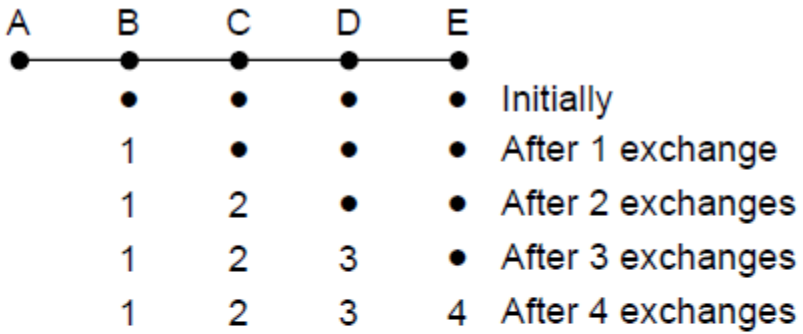
Distance Vector Dynamics

- Adding routes:
  - News travels one hop per exchange
- Removing routes
  - When a node fails, no more exchanges, other nodes forget
- But partitions (unreachable nodes in divided network) are a problem
  - "Count to infinity" scenario

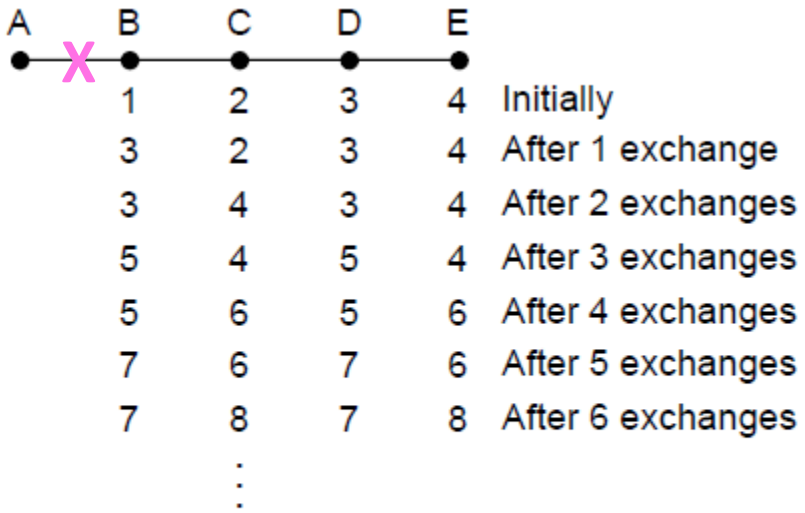


DV Dynamics (2)

- Good news travels quickly, bad news slowly (inferred)



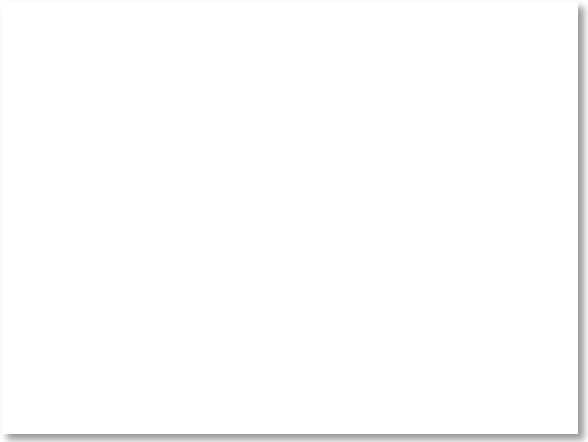
Desired convergence



"Count to infinity" scenario

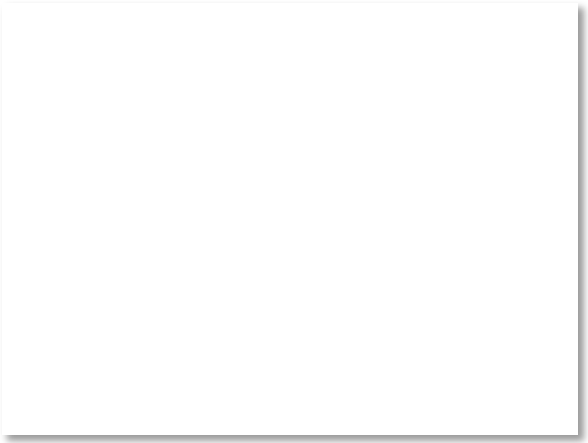
DV Dynamics (3)

- Various heuristics to address
  - e.g., “Split horizon, poison reverse” (Don’t send route back to where you learned it from.)
- But none are very effective
  - Link state now favored in practice
  - Except when very resource-limited



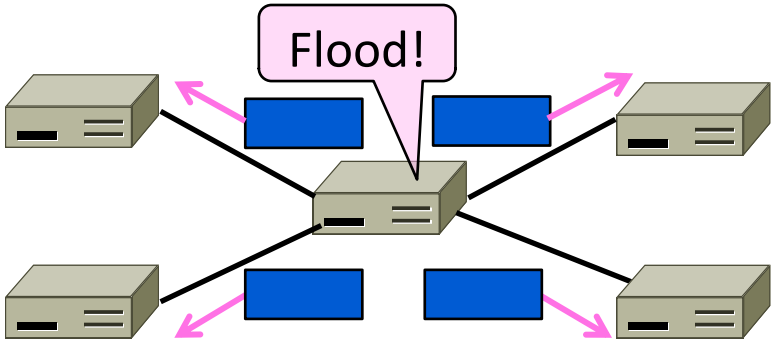
RIP (Routing Information Protocol)

- DV protocol with hop count as metric
  - Infinity is 16 hops; limits network size
  - Includes split horizon, poison reverse
- Routers send vectors every 30 seconds
  - Runs on top of UDP
  - Time-out in 180 secs to detect failures
- RIPv1 specified in RFC1058 (1988)



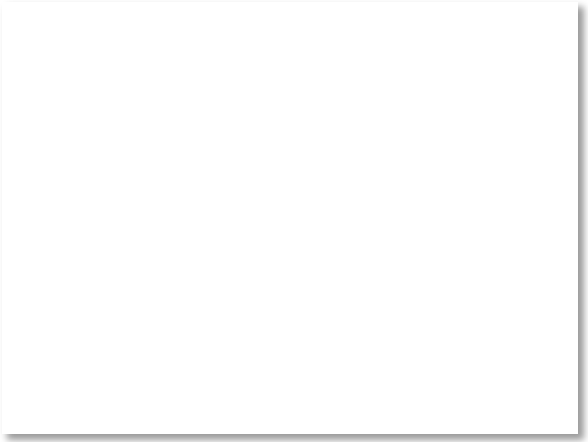
Topic

- How to broadcast a message to all nodes in the network with flooding
  - Simple mechanism, but inefficient



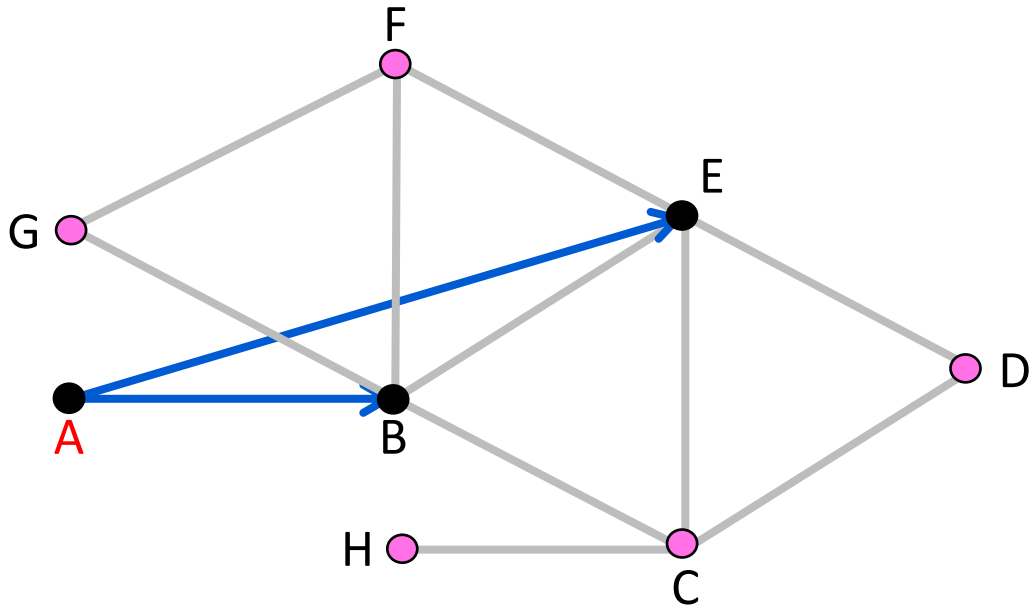
Flooding

- Rule used at each node:
  - Sends an incoming message on to all other neighbors
  - Remember the message so that it is only flood once
- Inefficient because one node may receive multiple copies of message



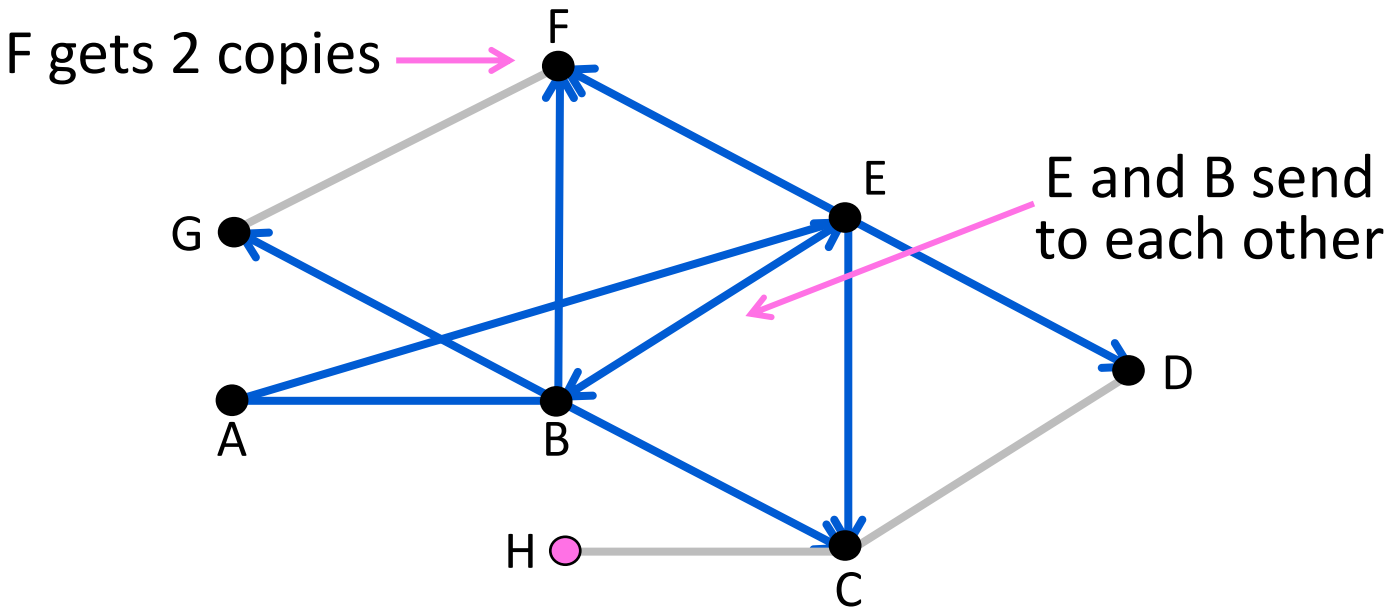
Flooding (2)

- Consider a flood from A; first reaches B via AB, E via AE



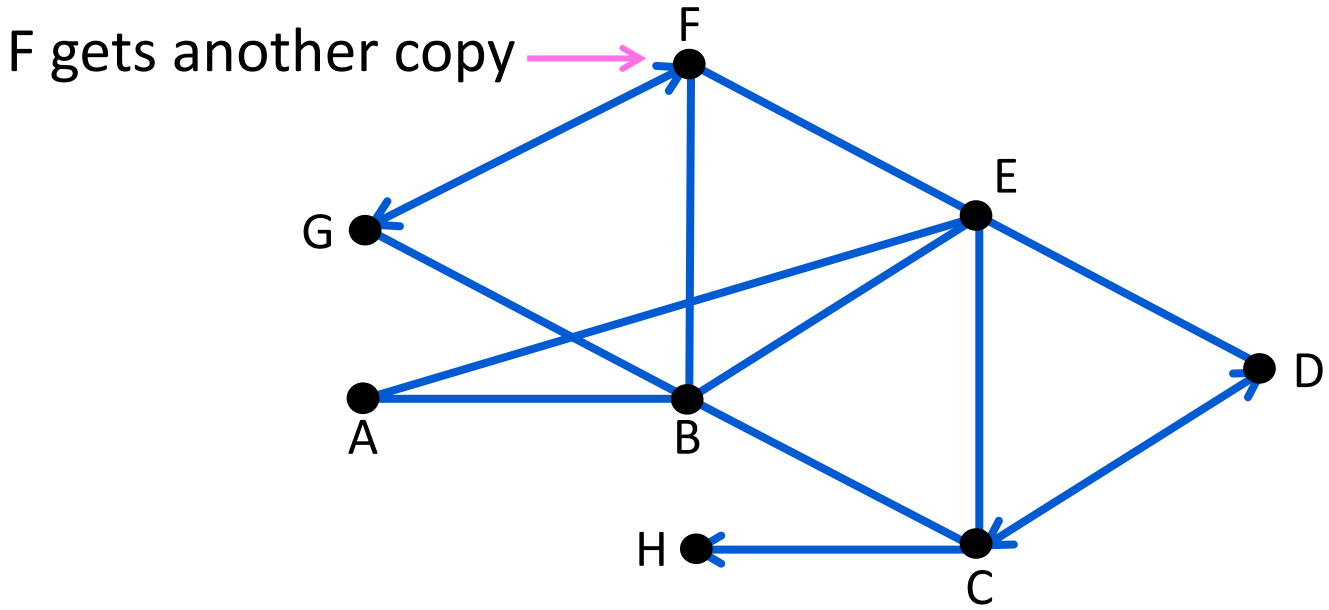
Flooding (3)

- Next B floods BC, BE, BF, BG, and E floods EB, EC, ED, EF



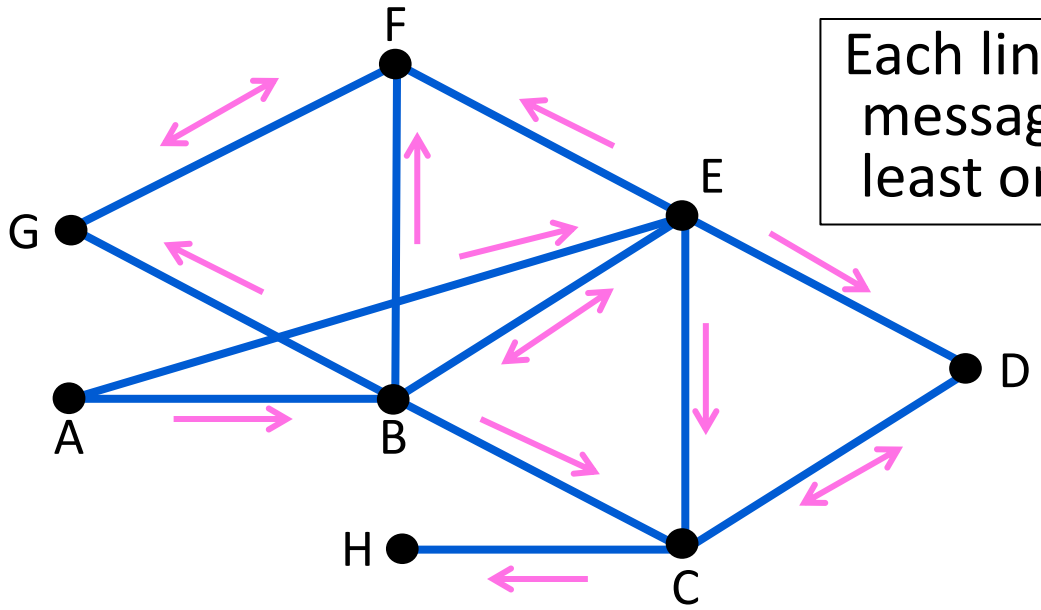
Flooding (4)

- C floods CD, CH; D floods DC; F floods FG; G floods GF



Flooding (5)

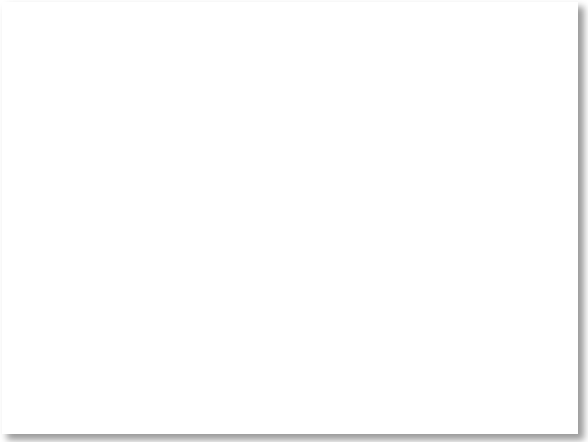
- H has no-one to flood … and we're done



Each link carries the message, and in at least one direction

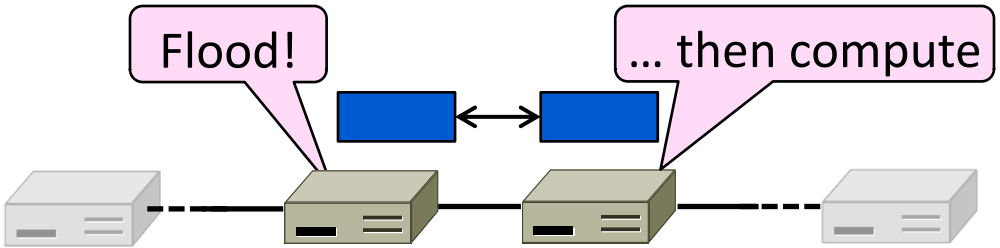
Flooding Details

- Remember message (to stop flood) using source and sequence number
  - So next message (with higher sequence number) will go through
- To make flooding reliable, use ARQ
  - So receiver acknowledges, and sender resends if needed



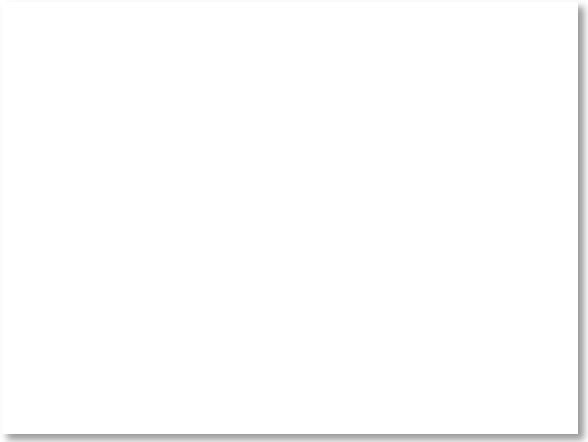
Topic

- How to compute shortest paths in a distributed network
  - The Link-State (LS) approach



Link-State Routing

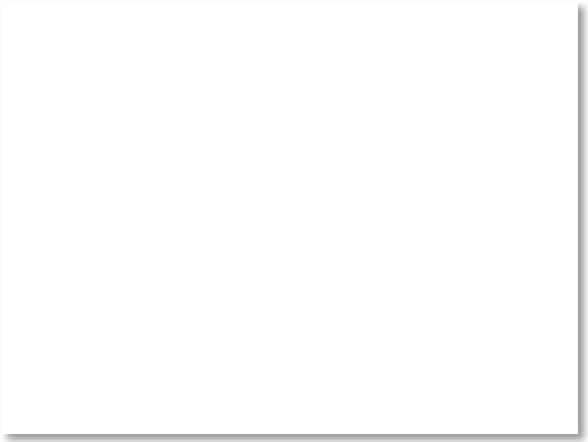
- One of two approaches to routing
  - Trades more computation than distance vector for better dynamics
- Widely used in practice
  - Used in Internet/ARPANET from 1979
  - Modern networks use OSPF and IS-IS



Link-State Setting

Nodes compute their forwarding table in the same distributed setting as for distance vector:

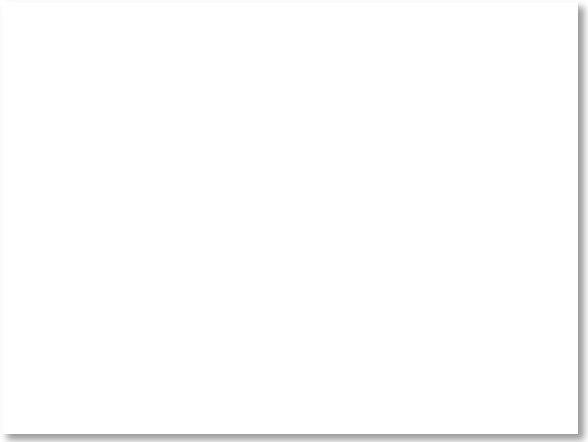
1. Nodes know only the cost to their neighbors; not the topology
2. Nodes can talk only to their neighbors using messages
3. All nodes run the same algorithm concurrently
4. Nodes/links may fail, messages may be lost



Link-State Algorithm

Proceeds in two phases:

1. Nodes flood topology in the form of link state packets
   - Each node learns full topology
2. Each node computes its own forwarding table
   - By running Dijkstra (or equivalent)

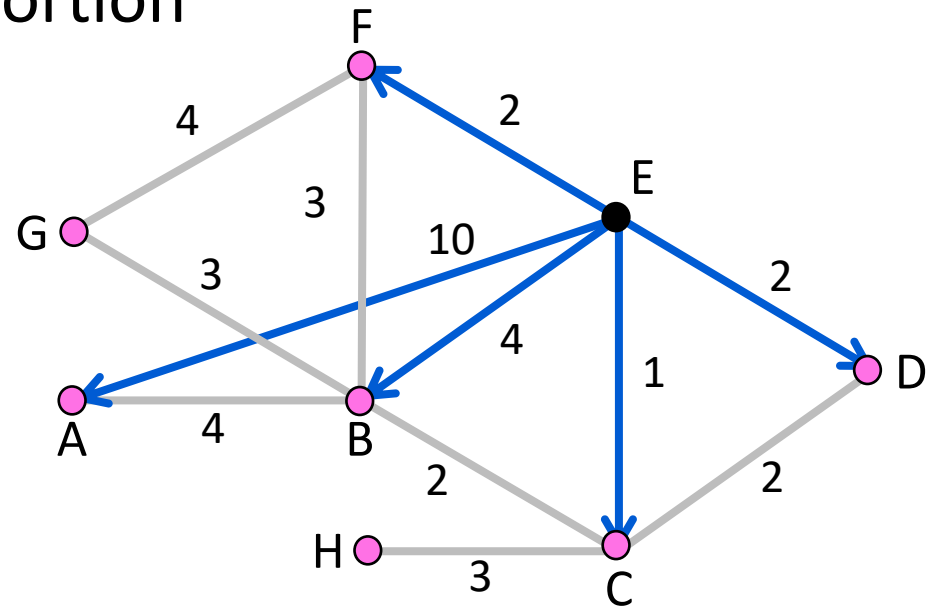


Phase 1: Topology Dissemination

- Each node floods link state packet (LSP) that describes their portion of the topology

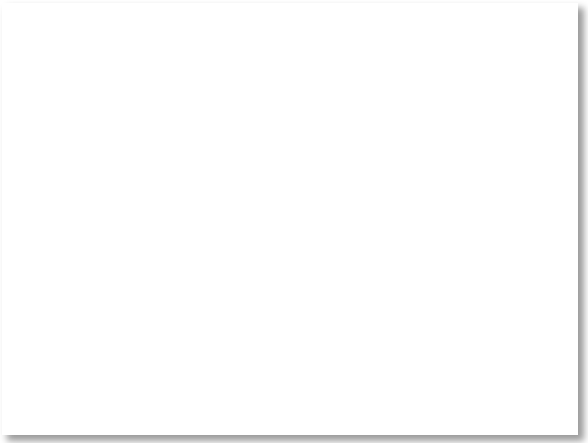
Node E's LSP flooded to A, B, C, D, and F

Seq. #	
A	10
B	4
C	1
D	2
F	2



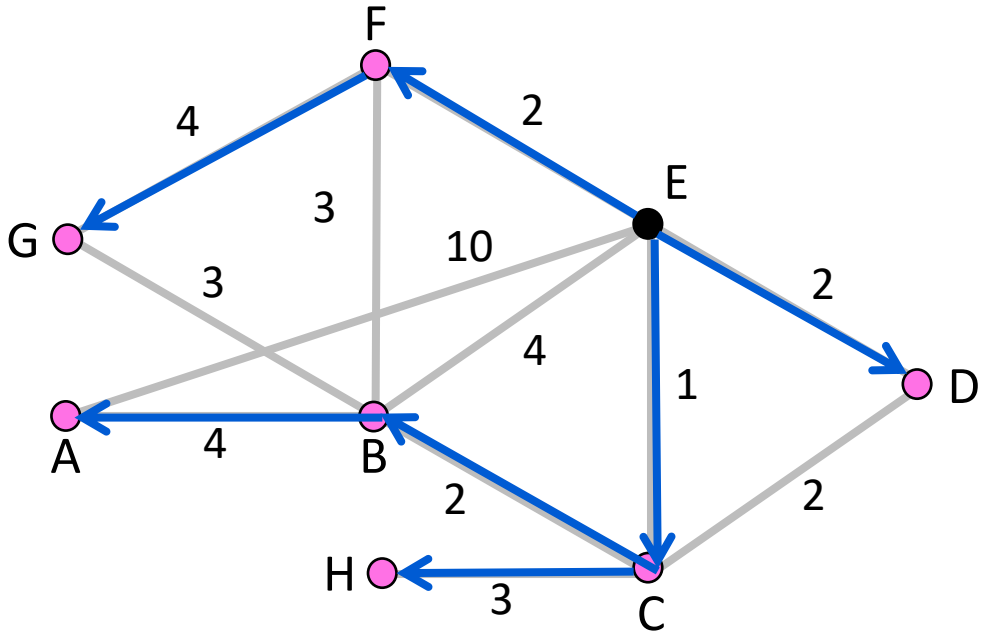
Phase 2: Route Computation

- Each node has full topology
  - By combining all LSPs
- Each node simply runs Dijkstra
  - Some replicated computation, but finds required routes directly
  - Compile forwarding table from sink/source tree
  - That's it folks!



Forwarding Table

Source Tree for E (from Dijkstra)



E's Forwarding Table

To	Next
A	C
B	C
C	C
D	D
E	--
F	F
G	F
H	C

Handling Changes

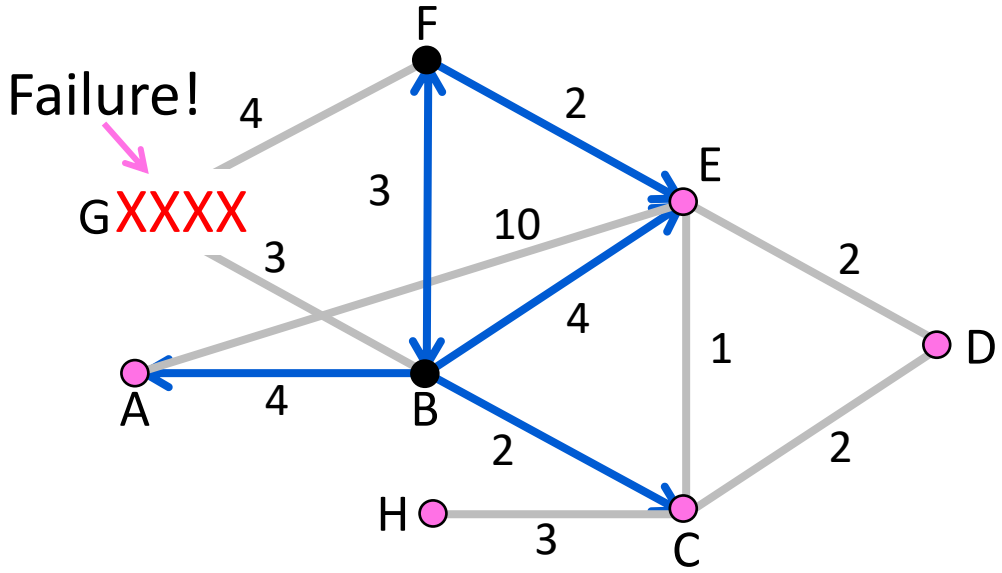
- On change, flood updated LSPs, and re-compute routes
  - E.g., nodes adjacent to failed link or node initiate

B's LSP

Seq. #	
A	4
C	2
E	4
F	3
G	∞

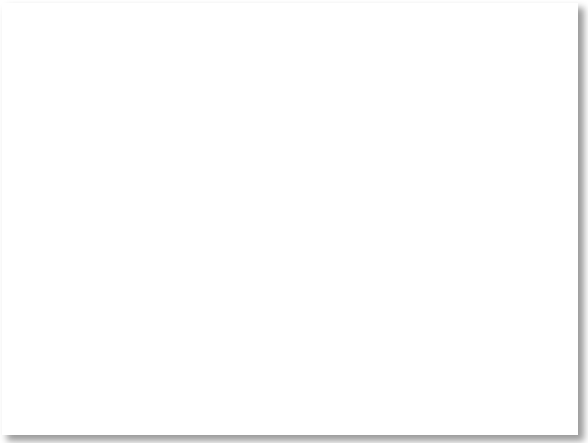
F's LSP

Seq. #	
B	3
E	2
G	∞



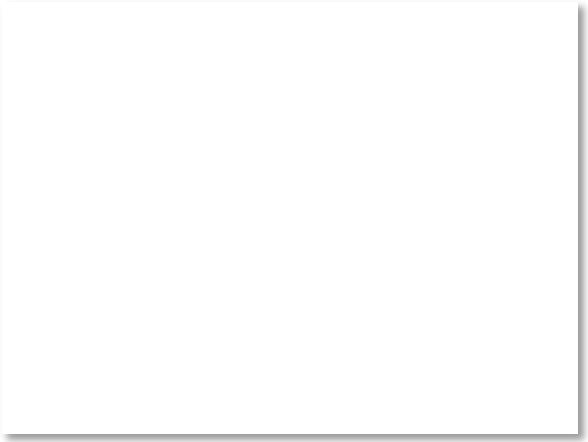
Handling Changes (2)

- Link failure
  - Both nodes notice, send updated LSPs
  - Link is removed from topology
- Node failure
  - All neighbors notice a link has failed
  - Failed node can't update its own LSP
  - But it is OK: all links to node removed



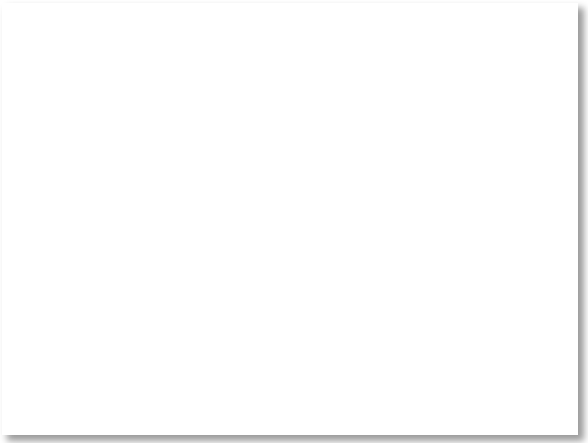
Handling Changes (3)

- Addition of a link or node
  - Add LSP of new node to topology
  - Old LSPs are updated with new link
- Additions are the easy case …



Link-State Complications

- Things that can go wrong:
  - Seq. number reaches max, or is corrupted
  - Node crashes and loses seq. number
  - Network partitions then heals
- Strategy:
  - Include age on LSPs and forget old information that is not refreshed
- Much of the complexity is due to handling corner cases (as usual!)

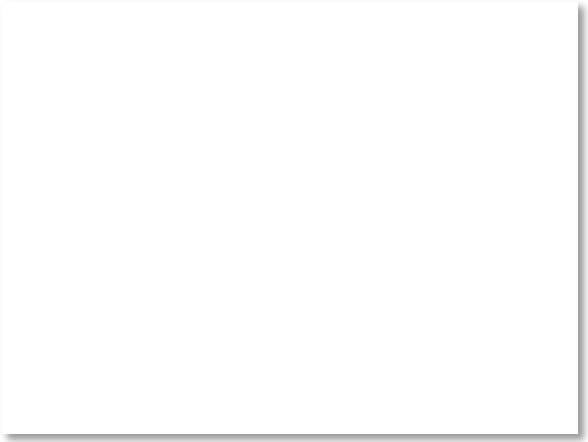


DV/LS Comparison

Goal	Distance Vector	Link-State
Correctness	Distributed Bellman-Ford	Replicated Dijkstra
Efficient paths	Approx. with shortest paths	Approx. with shortest paths
Fair paths	Approx. with shortest paths	Approx. with shortest paths
Fast convergence	Slow – many exchanges	Fast – flood and compute
Scalability	Excellent – storage/compute	Moderate – storage/compute

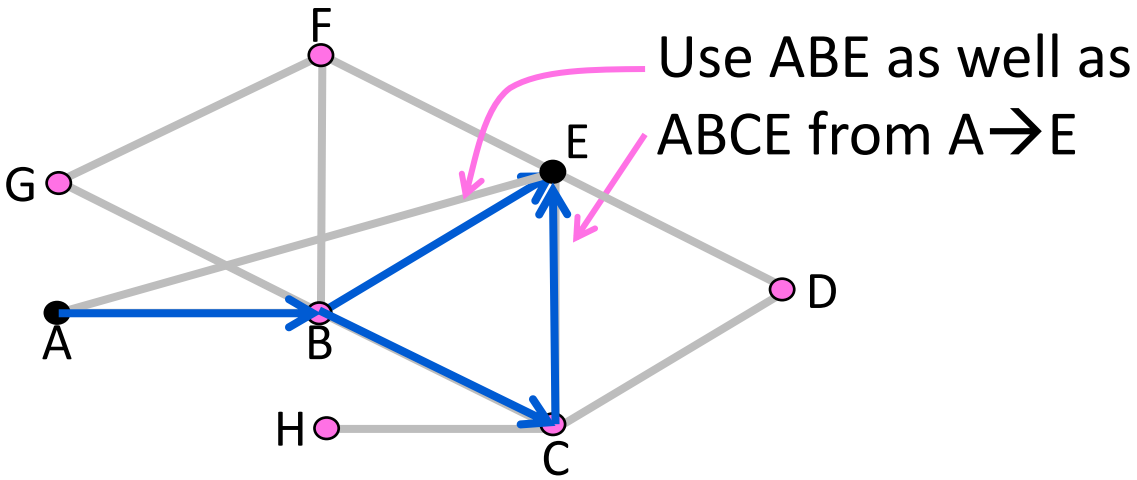
IS-IS and OSPF Protocols

- Widely used in large enterprise and ISP networks
  - IS-IS = Intermediate System to Intermediate System
  - OSPF = Open Shortest Path First
- Link-state protocol with many added features
  - E.g., “Areas” for scalability



Topic

- More on shortest path routes
  - Allow multiple shortest paths



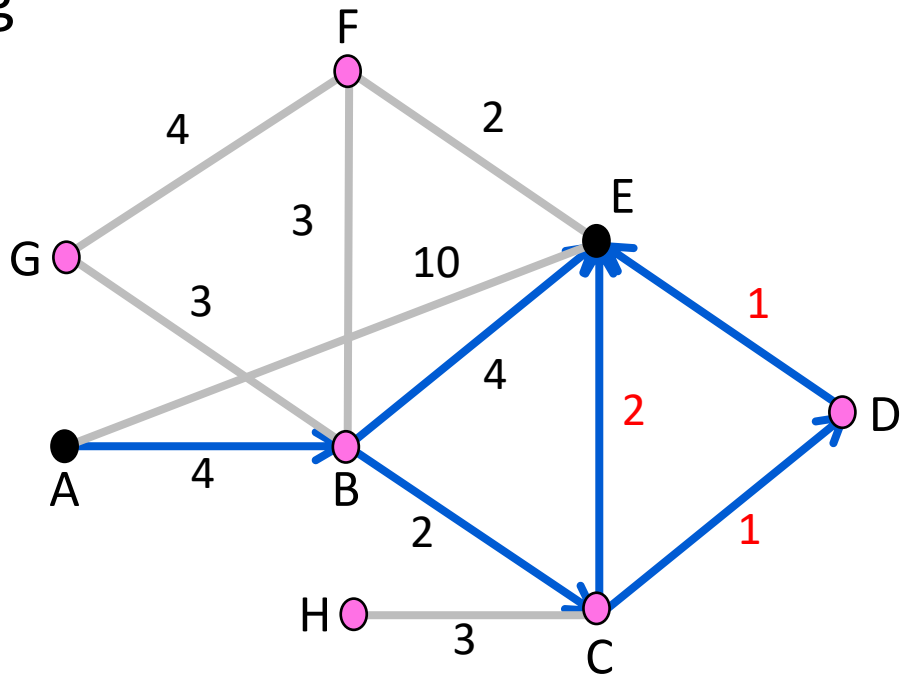
Multipath Routing

- Allow multiple routing paths from node to destination be used at once
  - Topology has them for redundancy
  - Using them can improve performance

- Questions:
  - How do we find multiple paths?
  - How do we send traffic along them?

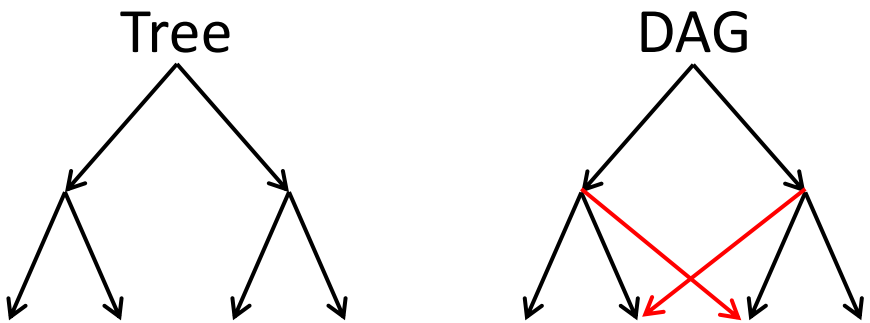
Equal-Cost Multipath Routes

- One form of multipath routing
  - Extends shortest path model by keeping set if there are ties
- Consider A→E
  - ABE = 4 + 4 = 8
  - ABCE = 4 + 2 + 2 = 8
  - ABCDE = 4 + 2 + 1 + 1 = 8
  - Use them all!



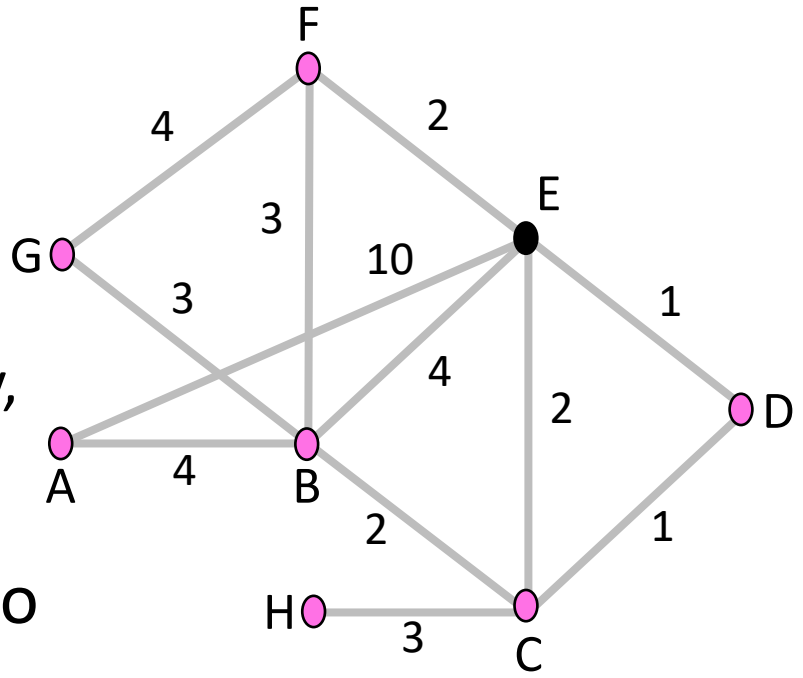
Source "Trees"

- With ECMP, source/sink "tree" is a directed acyclic graph (DAG)
  - Each node has set of next hops
  - Still a compact representation



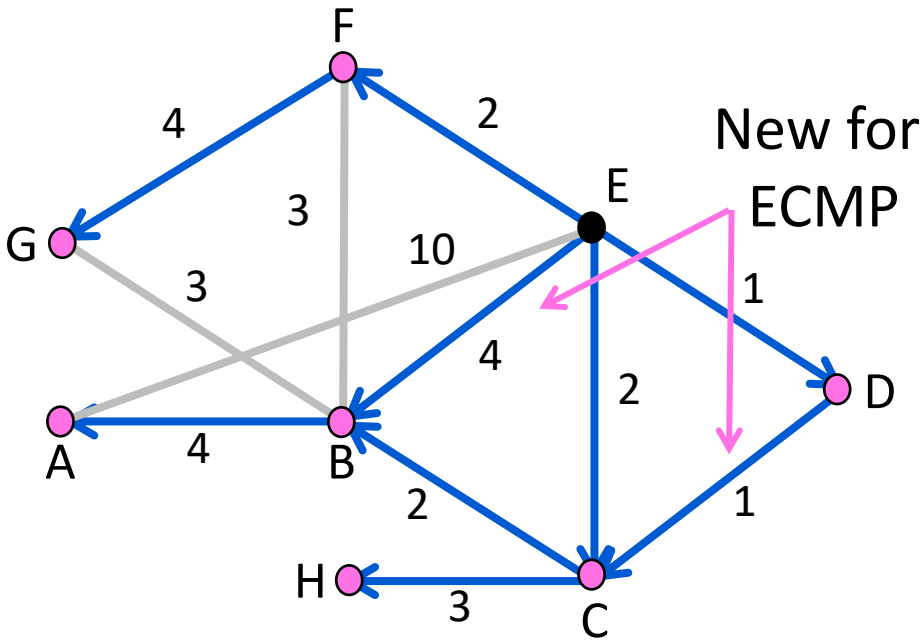
Source "Trees" (2)

- Find the source "tree" for E
  - Procedure is Dijkstra, simply remember set of next hops
  - Compile forwarding table similarly, may have set of next hops
- Straightforward to extend DV too
  - Just remember set of neighbors



Source "Trees" (3)

Source Tree for E

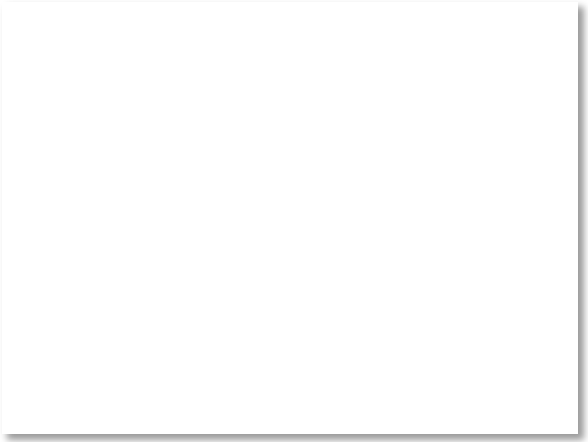


E's Forwarding Table

Node	Next hops
A	B, C, D
B	B, C, D
C	C, D
D	D
E	--
F	F
G	F
H	C, D

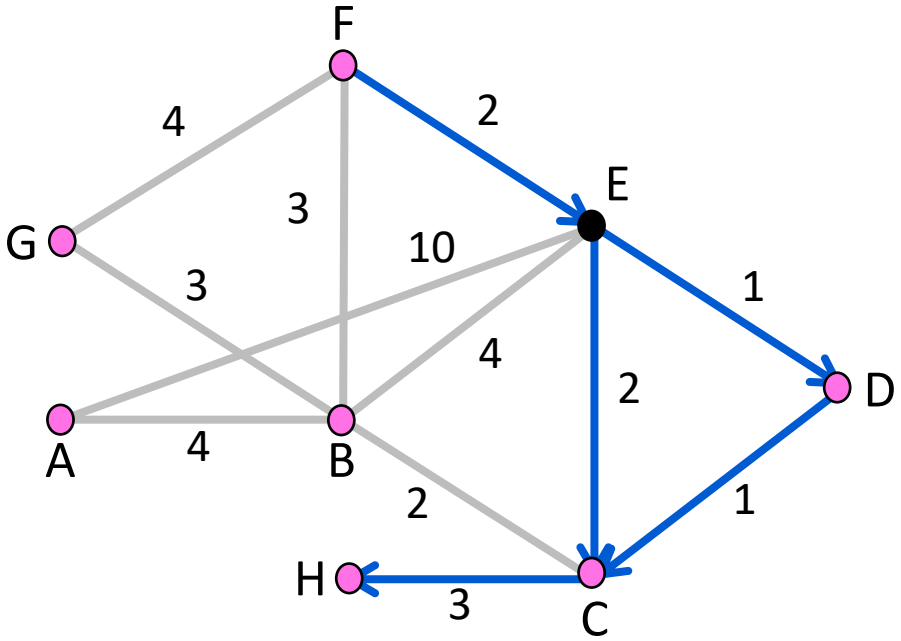
Forwarding with ECMP

- Could randomly pick a next hop for each packet based on destination
  - Balances load, but adds jitter
- Instead, try to send packets from a given source/destination pair on the same path
  - Source/destination pair is called a flow
  - Map flow identifier to single next hop
  - No jitter within flow, but less balanced



Forwarding with ECMP (2)

Multipath routes from F/E to C/H



E's Forwarding Choices

Flow	Possible next hops	Example choice
F → H	C, D	D
F → C	C, D	D
E → H	C, D	C
E → C	C, D	C

Use both paths to get to one destination