

# CSE160: Computer Networks

## Lecture #09 – Intro to Routing Protocols

**2020-09-24**



**Professor  
Alberto E. Cerpa**



# Last Time

---

- Introduction to the Network layer
  - Datagram and virtual circuit services
  - Internetworks
  - Internet Protocol (IP) packet format
  - Packet Fragmentation and Path Discovery
  - ICMP
- The Network layer
  - Provides end-to-end data delivery between networks
  - Issues of scale and heterogeneity

Application
Presentation
Session
Transport
Network
Data Link
Physical



# This Time

---

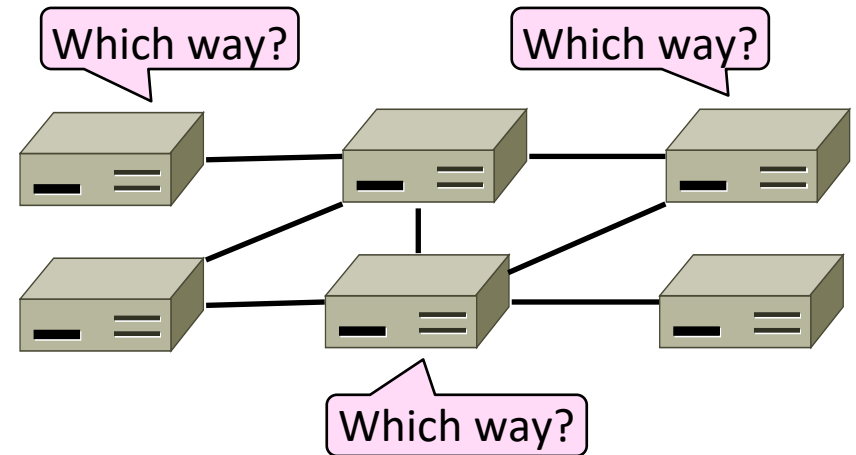
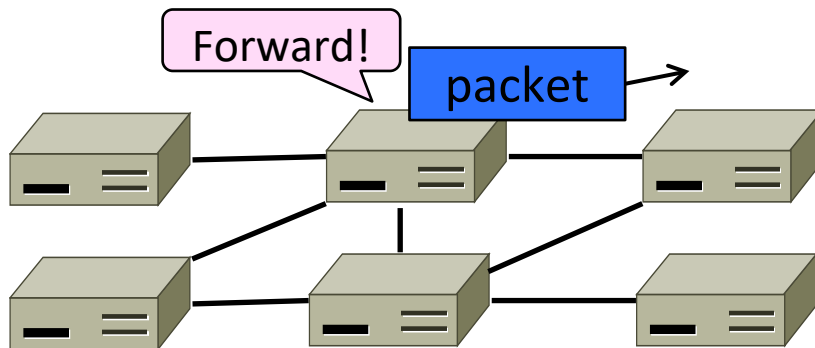
- Focus
  - How do we calculate routes for packets?
  - Routing is a network layer function
- Routing Algorithms
  - Intro to routing
  - Best paths
  - Dijkstra SP Algorithm
  - Link-State routing (OSPF)
  - Cost metrics and cost estimation

Application
Presentation
Session
Transport
Network
Data Link
Physical



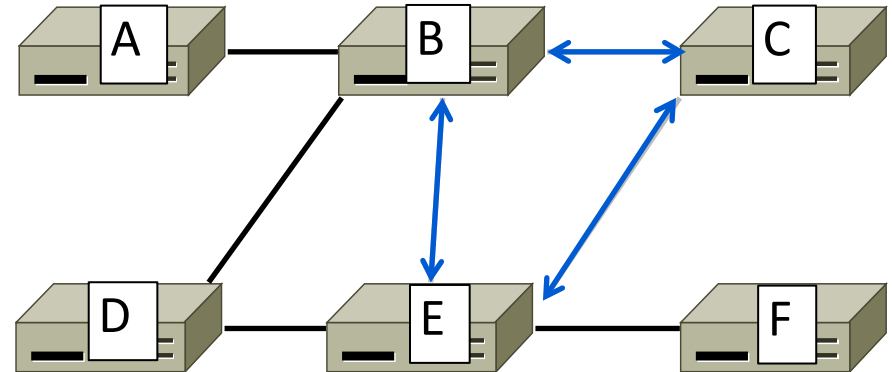
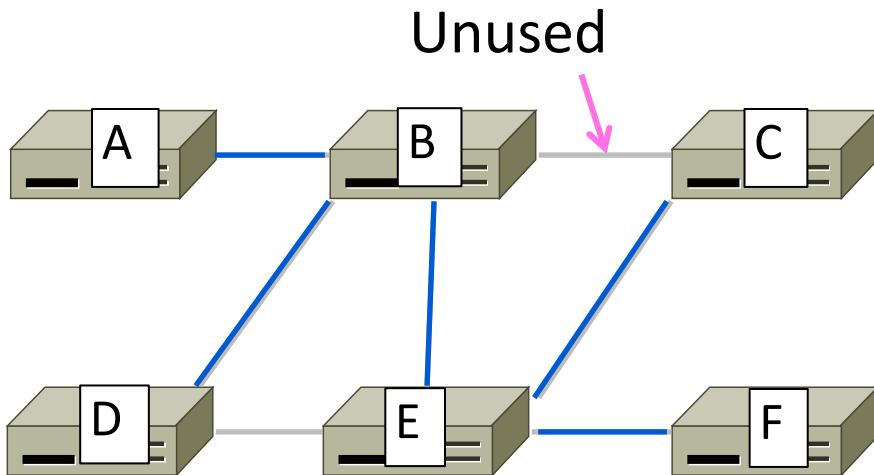
# Forwarding vs. Routing

- Forwarding is the process of sending a data packet
  - Nodes process (local) and fast
- Routing is the process of deciding in which direction to send traffic
  - Network wide (global) and expensive



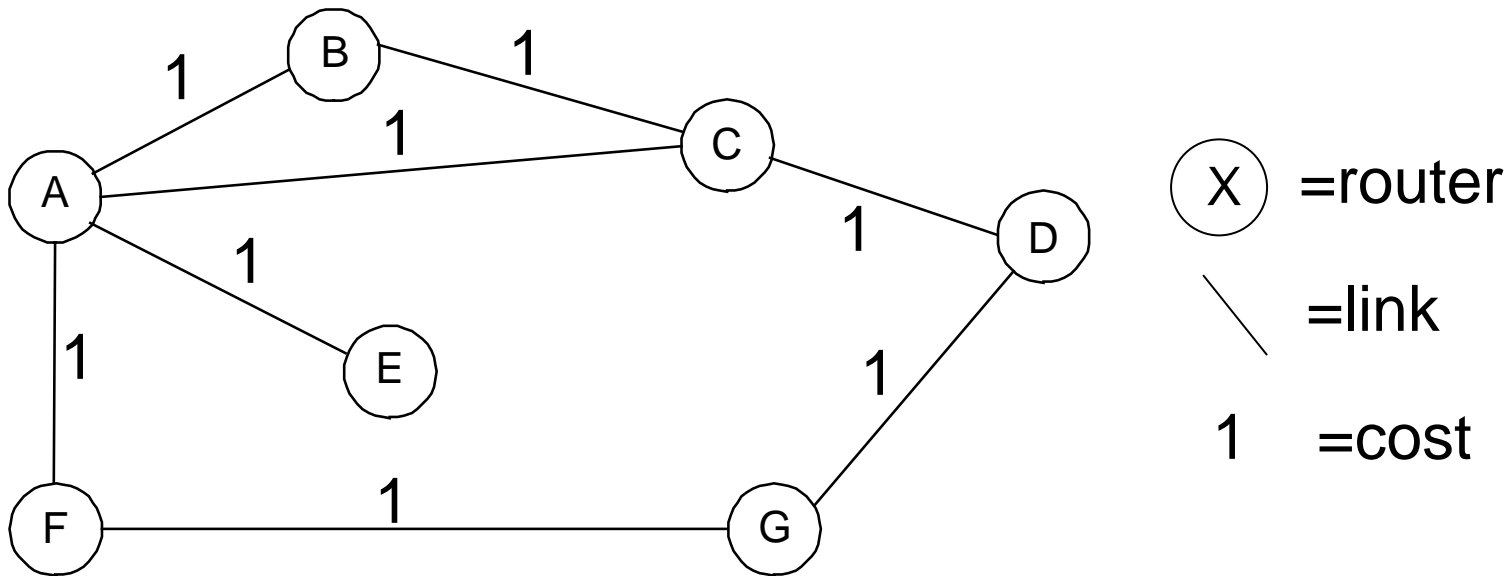
# Improving on the Spanning Tree

- Spanning tree provides basic connectivity
  - e.g., some path  $B \rightarrow C$
- Routing uses all links to find “best” paths
  - e.g., use BC, BE, and CE



# Network as a Graph

- Routing is essentially a problem in graph theory



# Perspective on Bandwidth Allocation

---

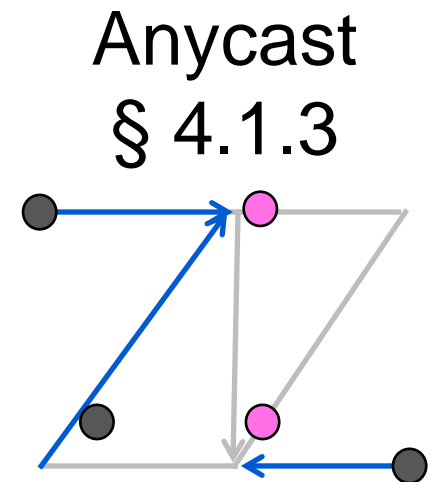
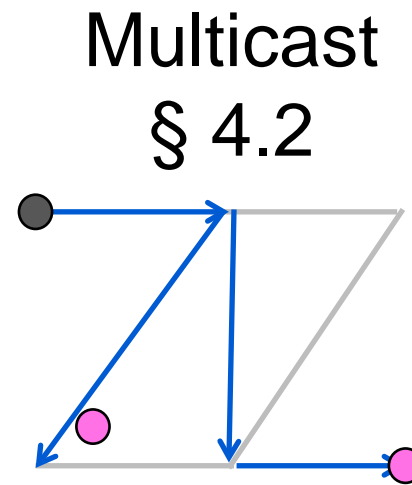
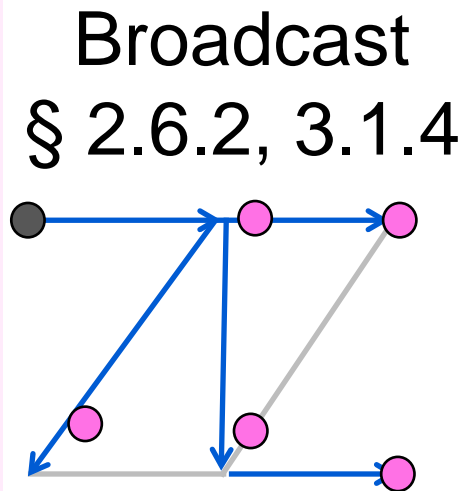
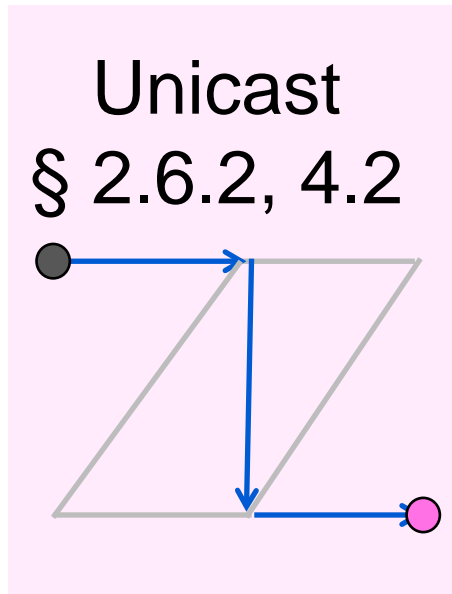
- Routing allocates network bandwidth adapting to failures; other mechanisms used at other timescales

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





# 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



# Kinds of Routing Schemes

---

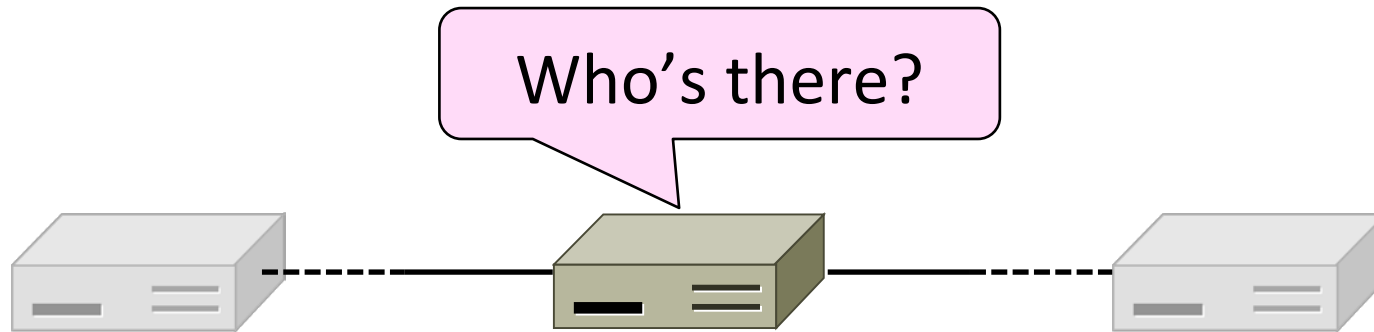
- Many routing schemes have been proposed/explored!
- Distributed or centralized
- Hop-by-hop or source-based
- Deterministic or stochastic
- Single or multi-path
- Dynamic or static route selection
- Internet is to the left ☺



# Rules of Routing Algorithms

---

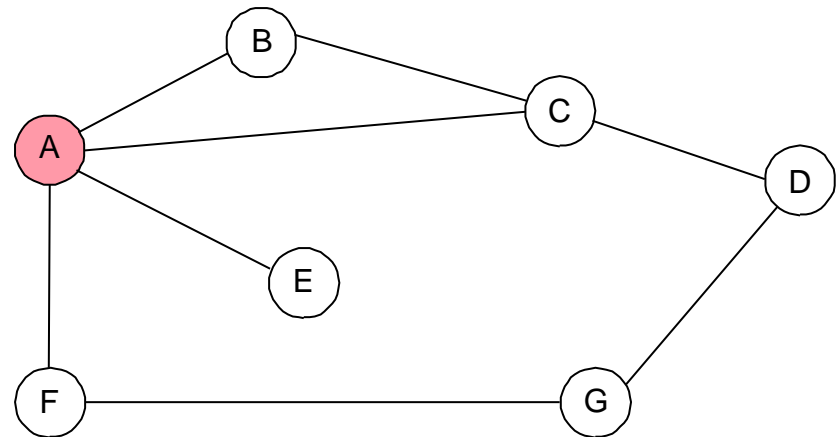
- Decentralized, distributed setting
  - All nodes are alike; no central controller
  - Nodes only know what they learn by exchanging messages with neighbors
  - Nodes operate concurrently
  - May be node/link/message failures



# What's in a Routing Table?

- The routing table at A, for example, lists *at a minimum* the next hops for the different destinations

Dest	Next Hop
B	B
C	C
D	C
E	E
F	F
G	F



# Routing Questions

---

- How to choose best path?
  - Defining “best” is slippery
- How to scale to millions of users?
  - Minimize control messages and routing table size
- How to adapt to failures or changes?
  - Node and link failures, plus message loss
  - We will use distributed algorithms



# Some Pitfalls

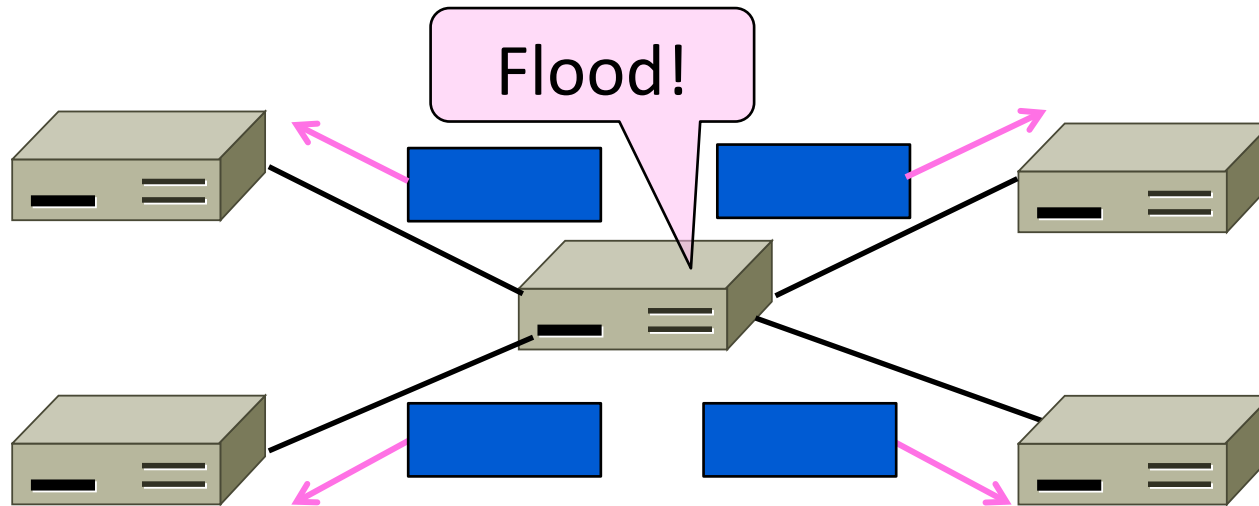
---

- Using global knowledge is challenging
  - Hard to collect
  - Can be out-of-date
  - Needs to summarize in a locally-relevant way
- Inconsistencies in local /global knowledge can cause:
  - Loops (black holes)
  - Oscillations, esp. when adapting to load



# Flooding

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



# Flooding Basics

---

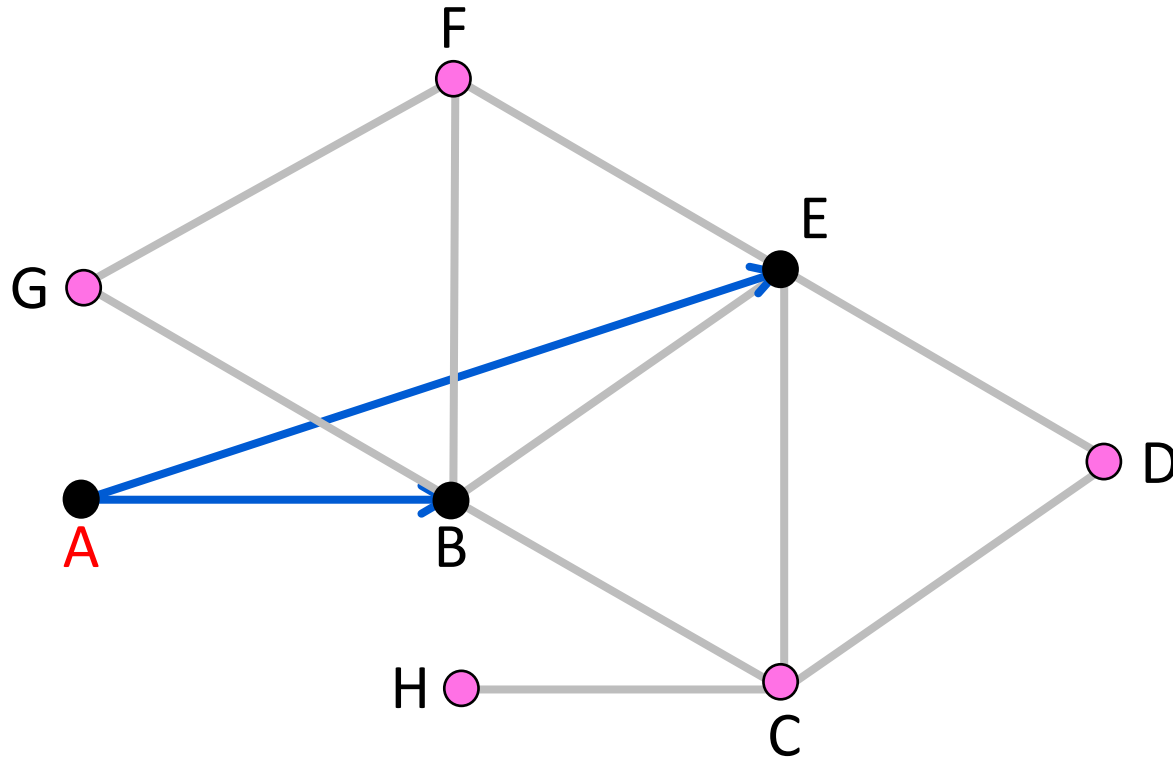
- Rule used at each node:
  - Sends an incoming message on to all other neighbors
  - Remember the message so that it is only flood once
- Inefficient because one node may receive multiple copies of the message





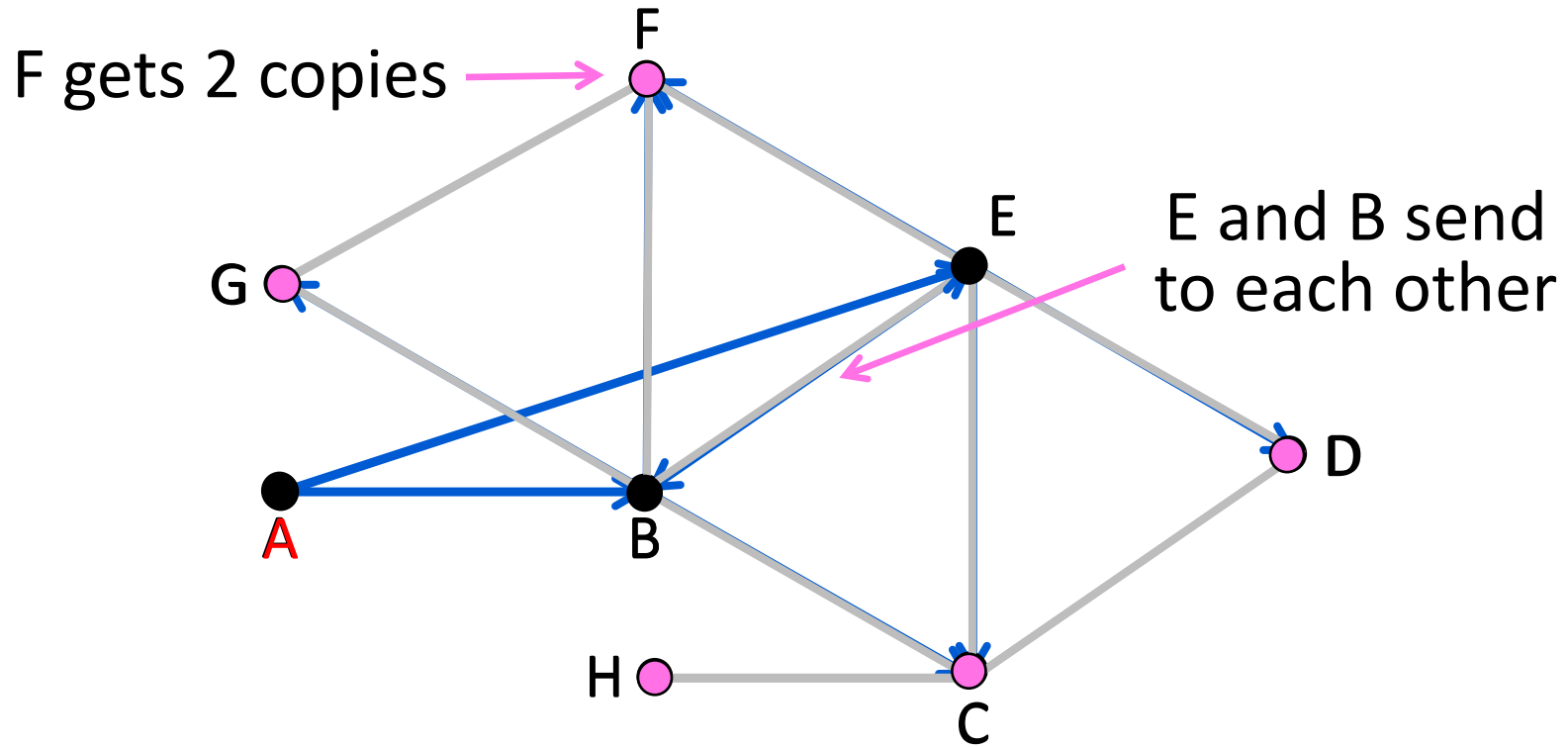
# Flooding Example

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



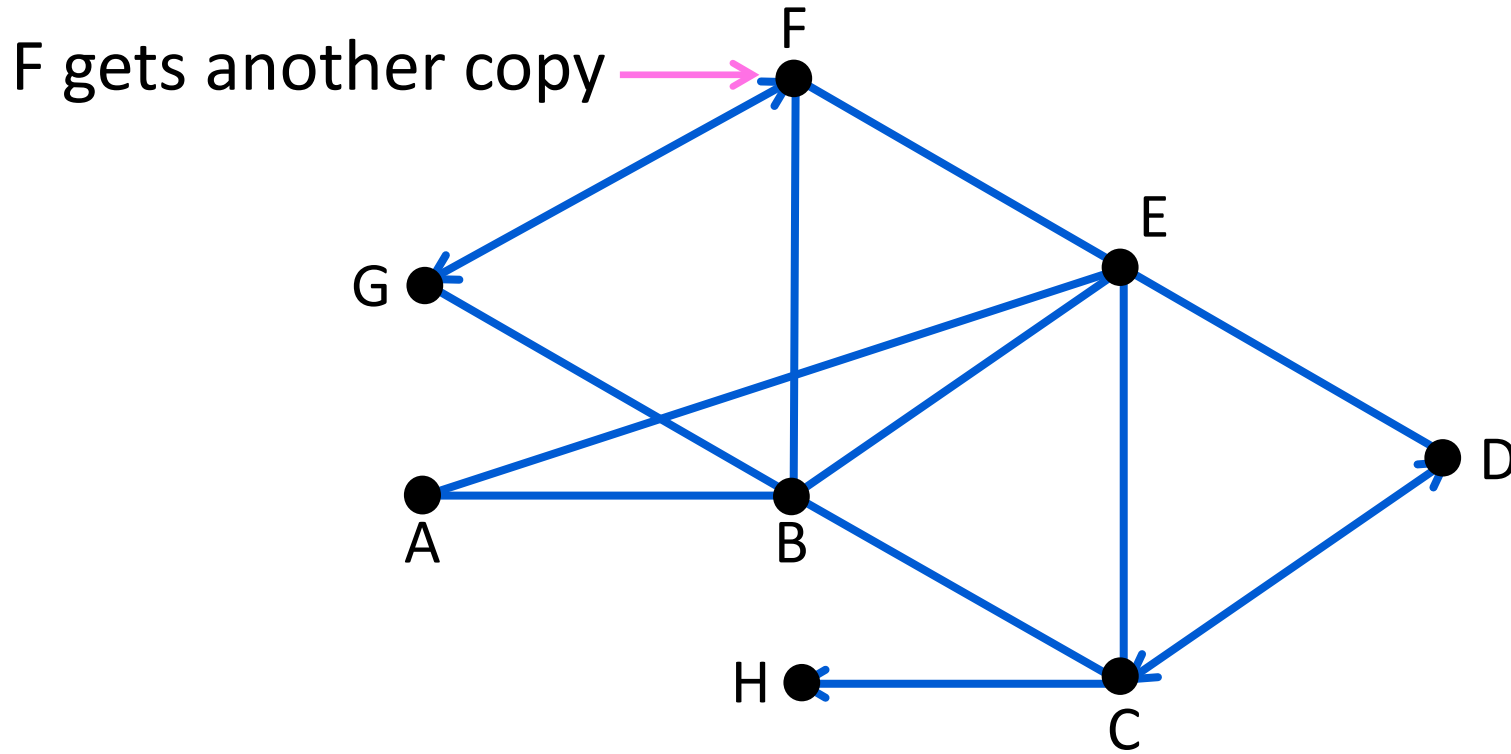
# Flooding Example (2)

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



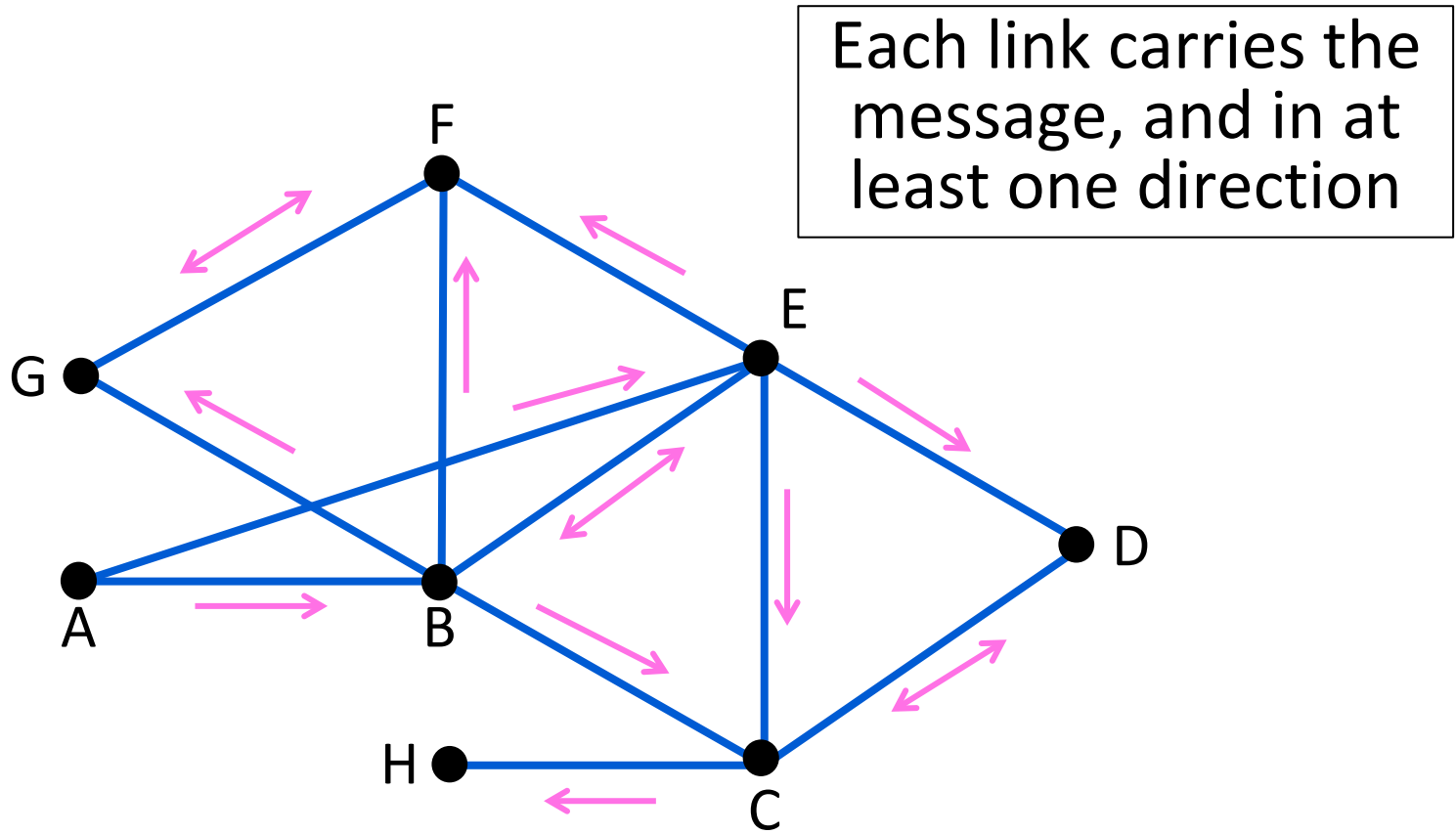
# Flooding Example (3)

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



# Flooding Example (4)

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



# Flooding Details

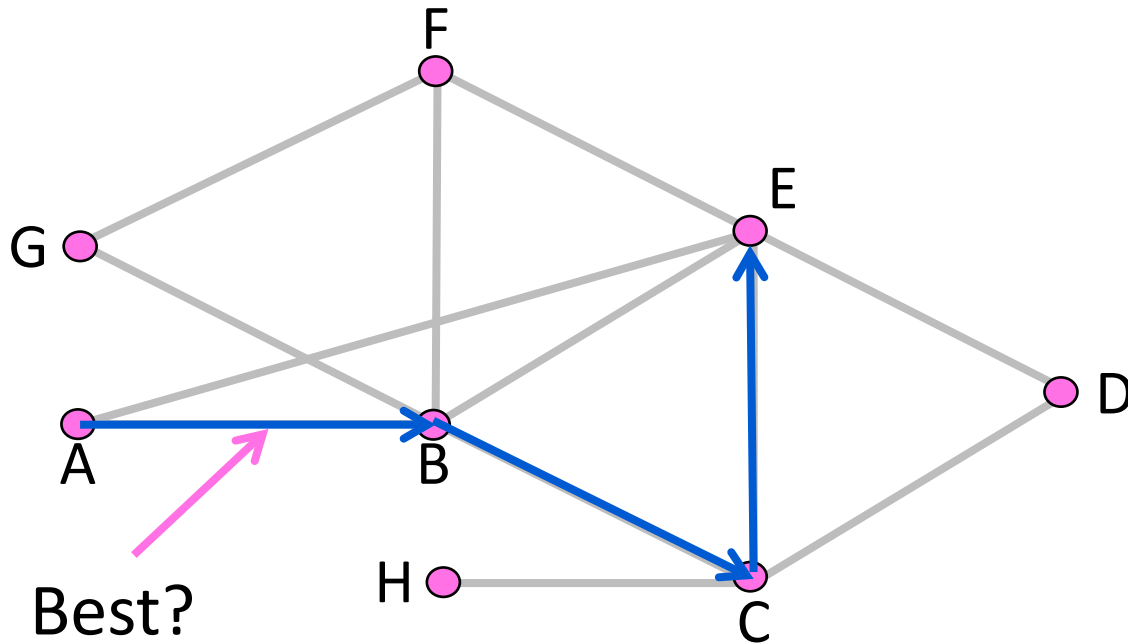
---

- 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



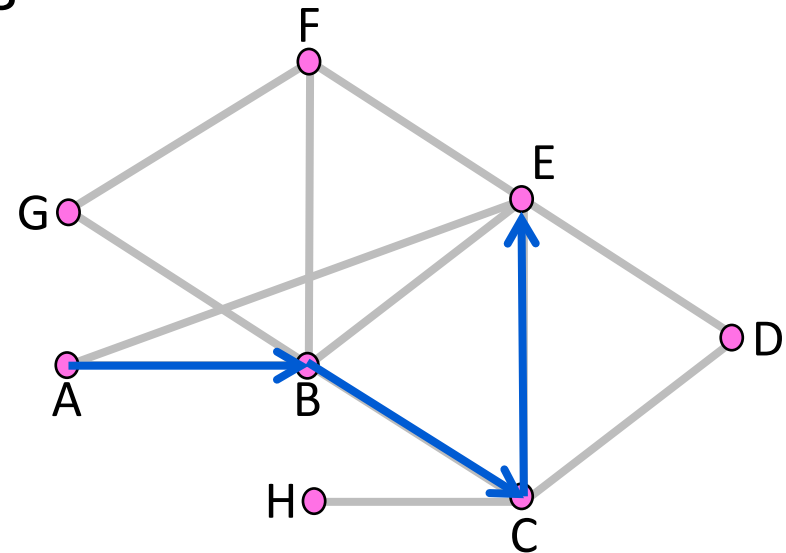
# Shortest Path Routing

- 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

---

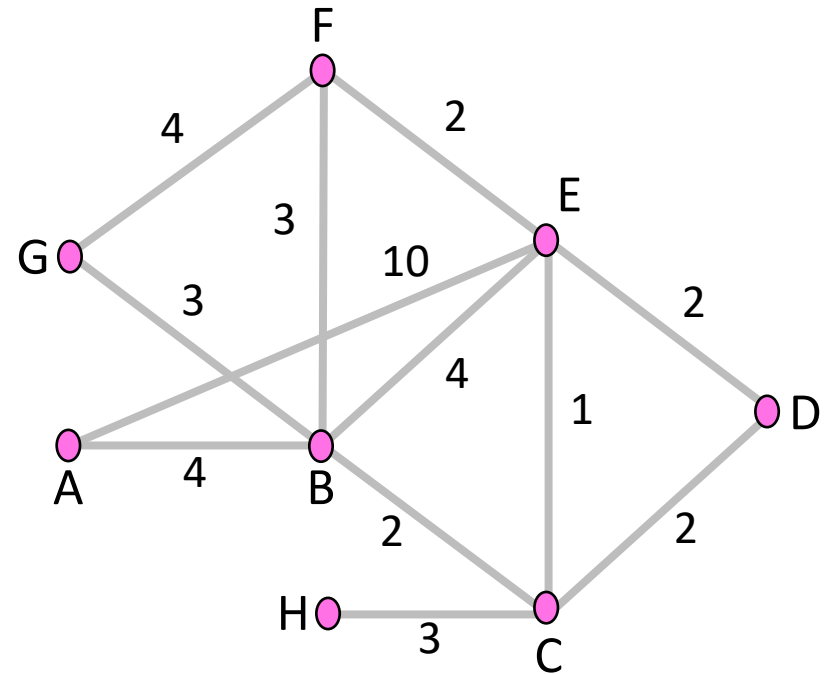
- We'll approximate “best” by a cost function that captures the factors
    - Often call lowest “shortest”
1. Assign each link a cost (distance)
  2. Define best path between each pair of nodes as the path that has the lowest total cost (or is the shortest)
  3. Pick randomly to break ties





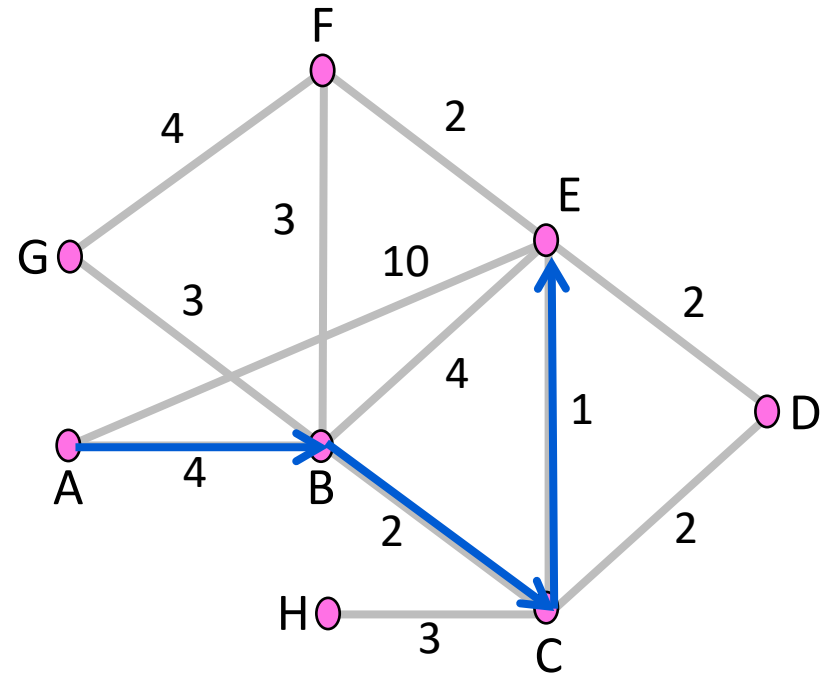
# Shortest Paths (2)

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



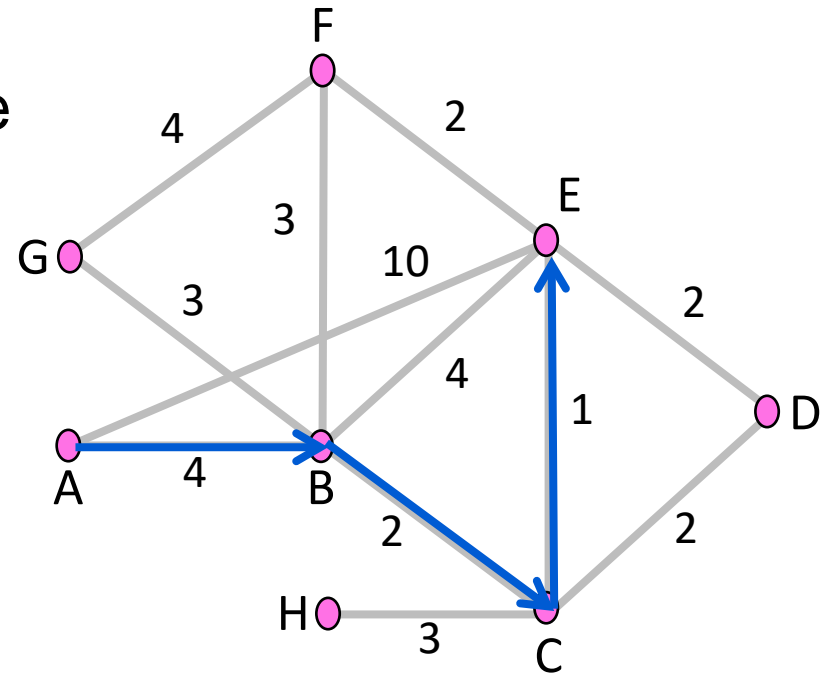
# Shortest Paths (3)

- ABCE is a shortest path
- $\text{dist}(\text{ABCE}) = 4+2+1 = 7$
- This is less than:
  - $\text{dist}(\text{ABE}) = 8$
  - $\text{dist}(\text{ABFE}) = 9$
  - $\text{dist}(\text{AE}) = 10$
  - $\text{dist}(\text{ABCDE}) = 10$



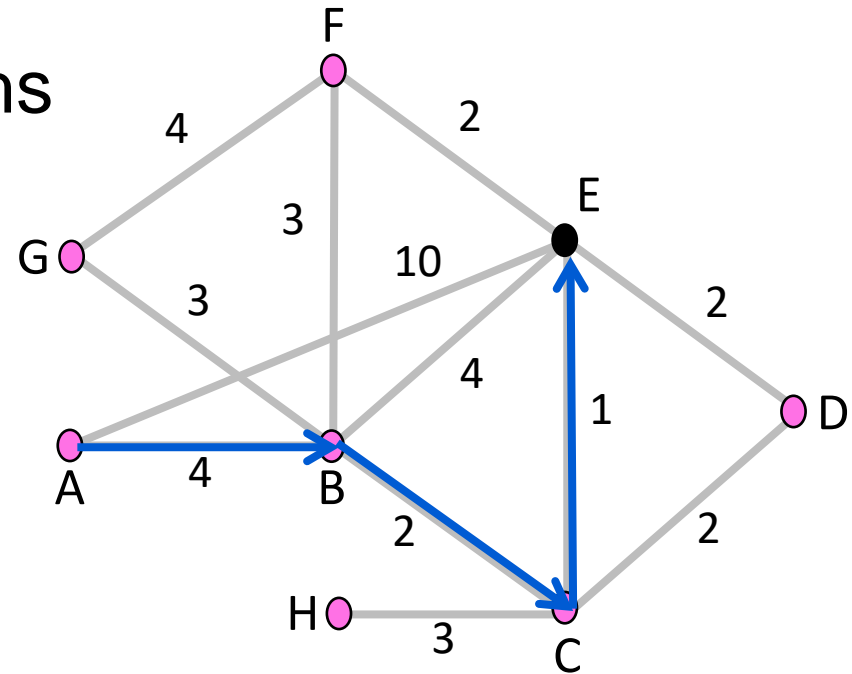
# Shortest Paths (4)

- Optimality property:
  - Subpaths of shortest paths are also shortest paths
- ABCE is a shortest path
  - So are ABC, AB, BCE, BC, CE



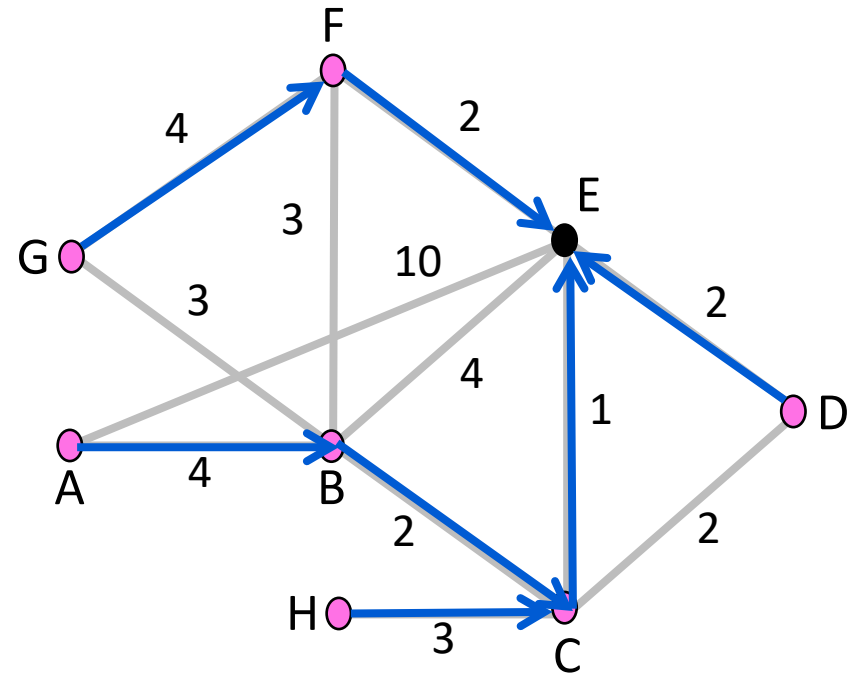
# Sink Trees

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



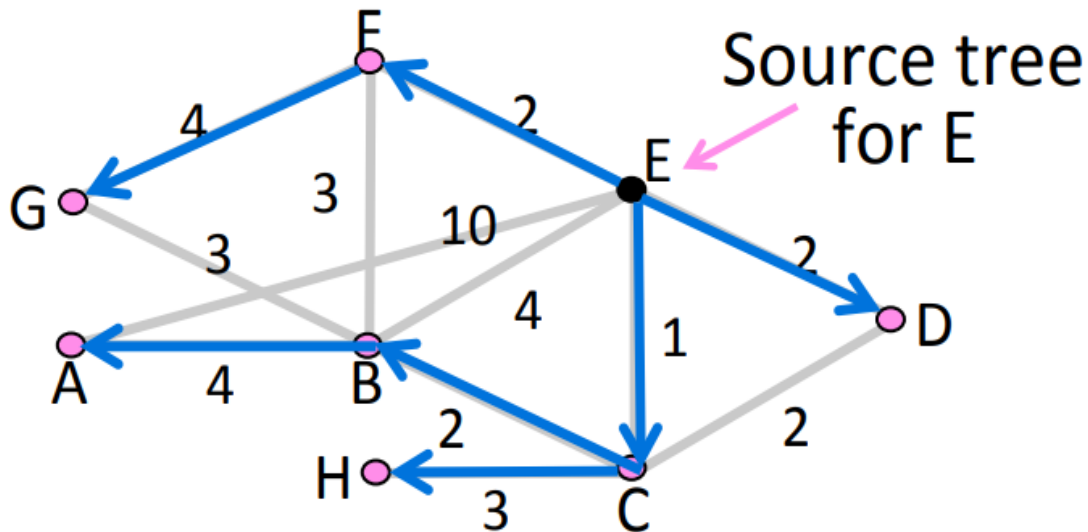
# Sink Trees (2)

- Implications:
  - Only need to use destination to follow shortest paths
  - Each node only needs to send to the next hop
- Forwarding table at a node
  - Lists next hop for each destination
  - Routing table may know more



# Computing Shortest Path

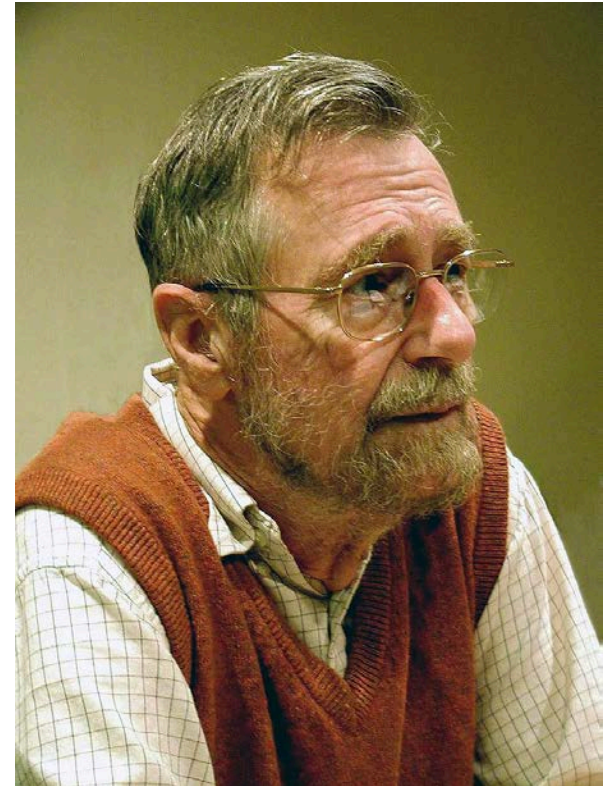
- How to compute shortest paths given a network topology
  - With Dijkstra's algorithm
  - With Bellman-Ford's algorithm



# Edsger W. Dijkstra (1930-2002)

---

- Famous computer scientist
  - Programming languages
  - Distributed algorithms
  - Program verification
- Dijkstra's algorithm, 1959
  - Single-source shortest paths, given network with non-negative link costs



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



# Shortest Paths: Dijkstra's Algorithm

---

- $N$ : Set of all nodes
- $M$ : Set of nodes for which we think we have a shortest path
- $s$ : The node executing the algorithm
- $L(i,j)$ : cost of edge  $(i,j)$  ( $\infty$  if no edge connects)
- $C(i)$ : Cost of the path from  $s$  to  $i$ .
- Two phases:
  - Initialize  $C(n)$  according to  $s$ ' neighbors
  - Compute shortest path to all nodes from  $s$





# The Algorithm

---

// Initialization

$M = \{s\}$  //  $M$  is the set of all nodes considered so far.

For each  $n$  in  $N - \{s\}$

$$C(n) = L(s, n)$$

// Find Shortest paths

Forever {

*Unconsidered* =  $N - M$

If *Unconsidered* == {} break

$M = M + \{w\}$  such that  $C(w)$  is the smallest in  
*Unconsidered*

For each  $n$  in *Unconsidered*

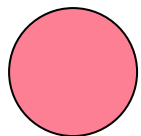
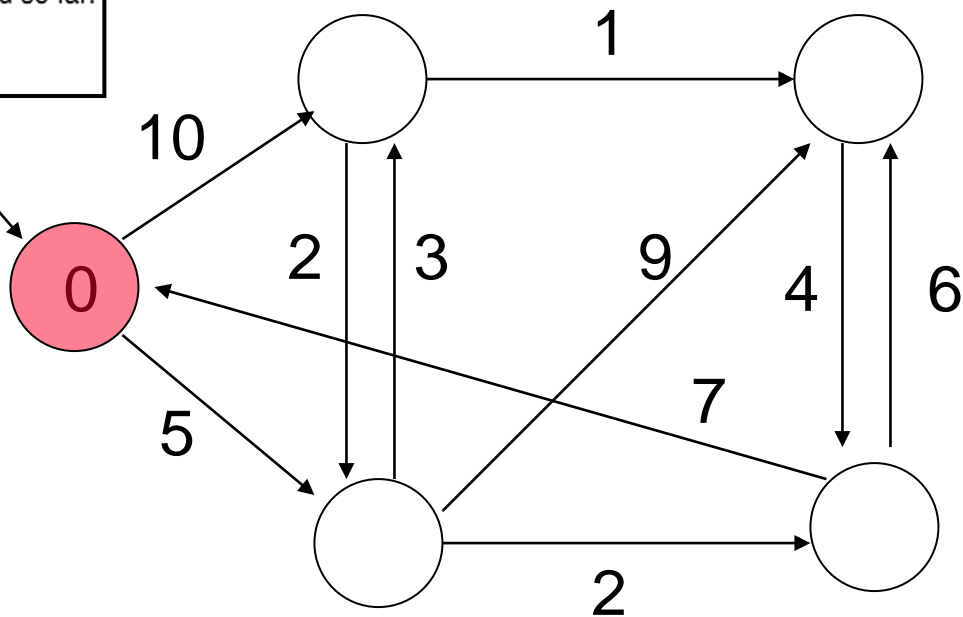
$$C(n) = \text{MIN}(C(n), C(w) + L(w, n))$$

}

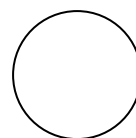


# Dijkstra Example – Initialization

\* // Initialization  
M = {s} // M is the set of all nodes considered so far. \*  
For each n in N - {s}  
C(n) = L(s,n)



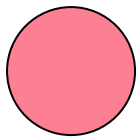
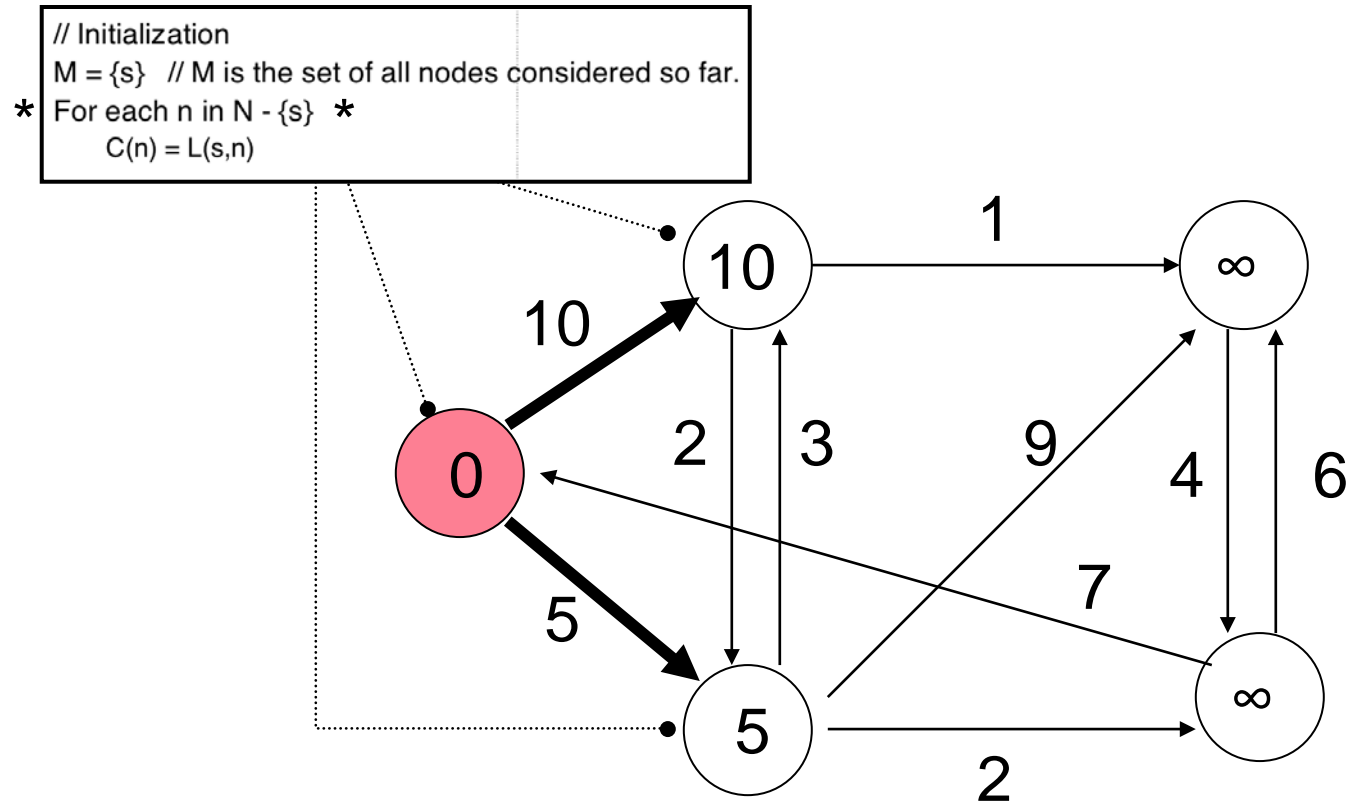
The Considered



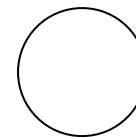
The Unconsidered



# Dijkstra Example – Post Initialization



The Considered

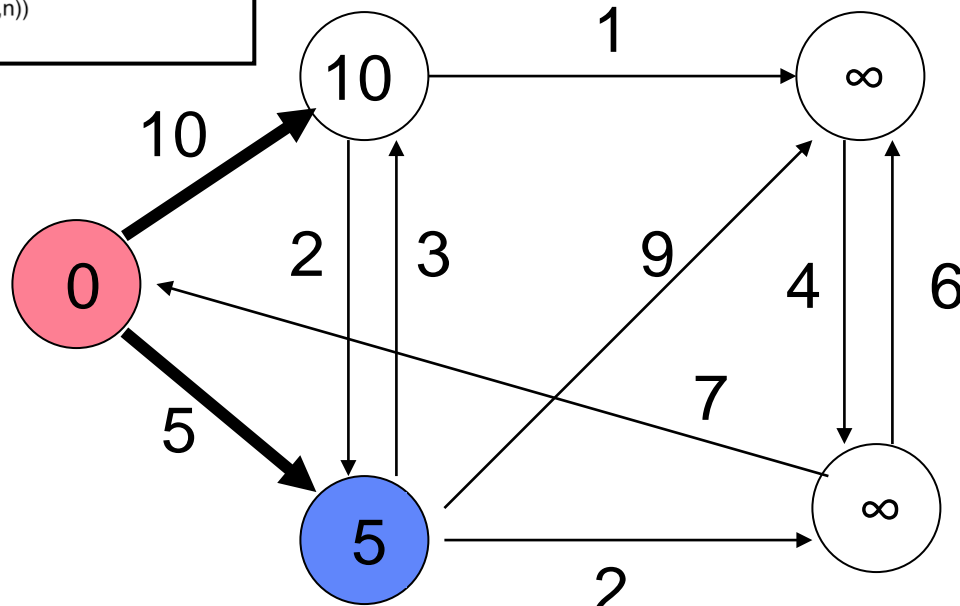


The Unconsidered

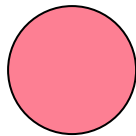


# Considering a Node

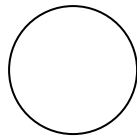
```
// Find Shortest paths
Forever {
  Unconsidered = N-M
  If Unconsidered == {} break
  M = M + {w} such that C(w) is the smallest in Unconsidered
  For each n in Unconsidered
    C(n) = MIN(C(n), C(w) + L(w,n))
}
```



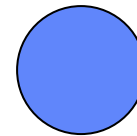
*Cost updates of 8, 14, and 7*



The Considered



The Unconsidered

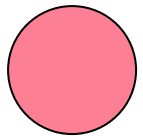
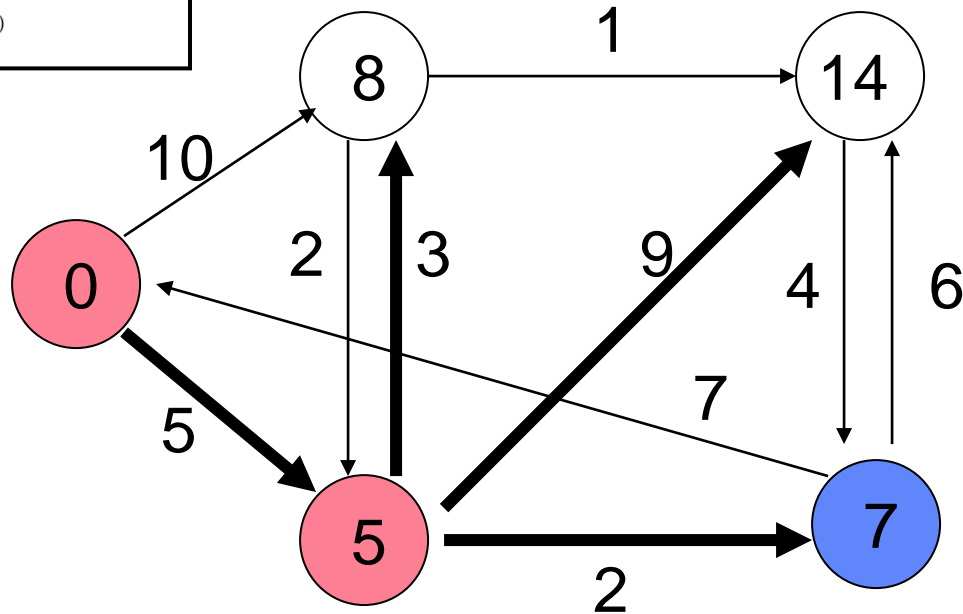


The Under Consideration (w)

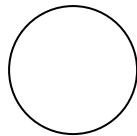


# Pushing out the horizon

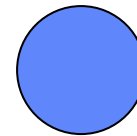
```
// Find Shortest paths
Forever {
  Unconsidered = N-M
  If Unconsidered == {} break
  M = M + {w} such that C(w) is the smallest in Unconsidered
  For each n in Unconsidered
    C(n) = MIN(C(n), C(w) + L(w,n))
}
```



The Considered



The Unconsidered

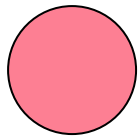
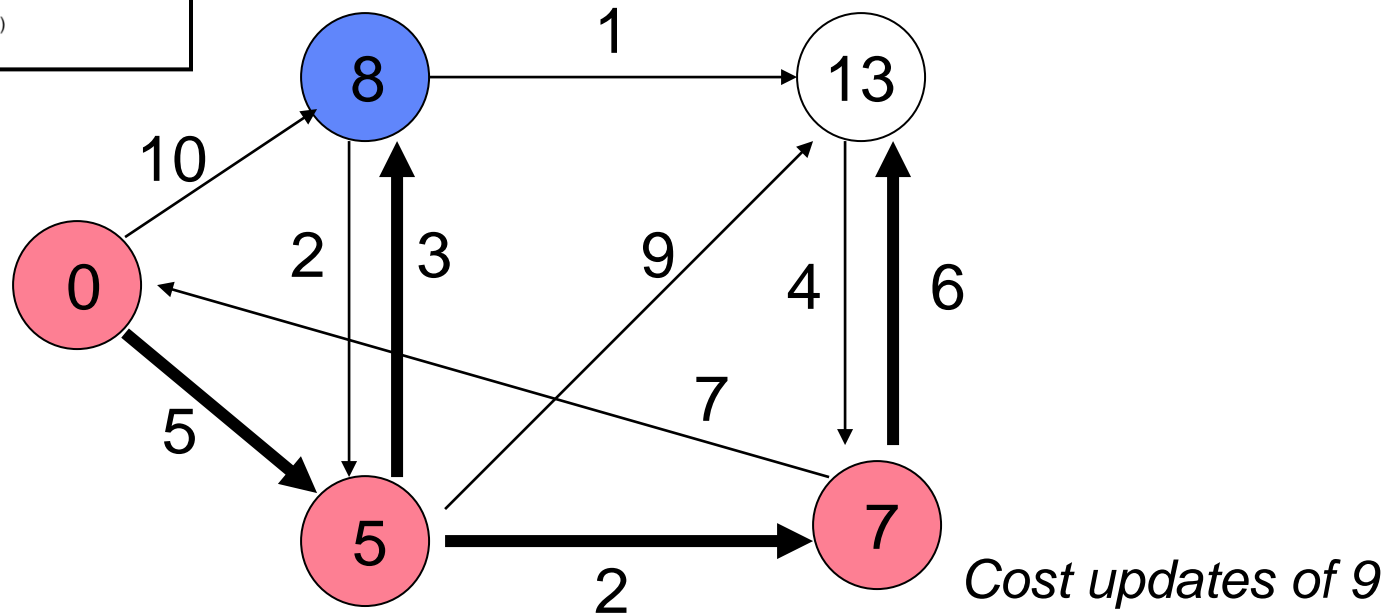


The Under Consideration (w)

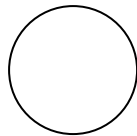


# Next Phase

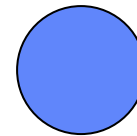
```
// Find Shortest paths
Forever {
  Unconsidered = N-M
  If Unconsidered == {} break
  M = M + {w} such that C(w) is the smallest in Unconsidered
  For each n in Unconsidered
    C(n) = MIN(C(n), C(w) + L(w,n))
}
```



The Considered



The Unconsidered

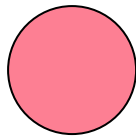
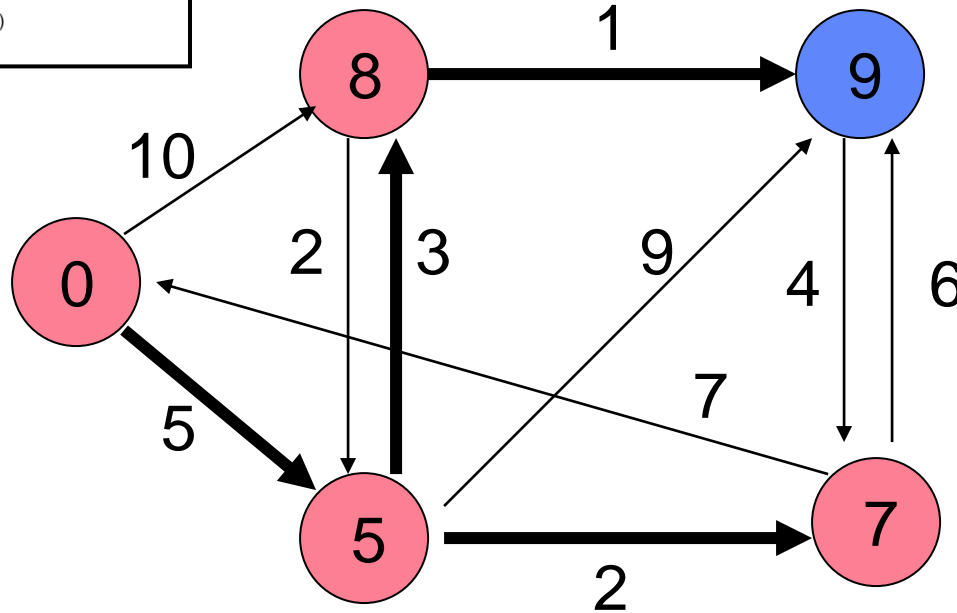


The Under Consideration (w)

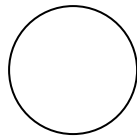


# Considering the last node

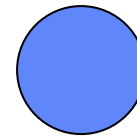
```
// Find Shortest paths
Forever {
  Unconsidered = N-M
  If Unconsidered == {} break
  M = M + {w} such that C(w) is the smallest in Unconsidered
  For each n in Unconsidered
    C(n) = MIN(C(n), C(w) + L(w,n))
}
```



The Considered



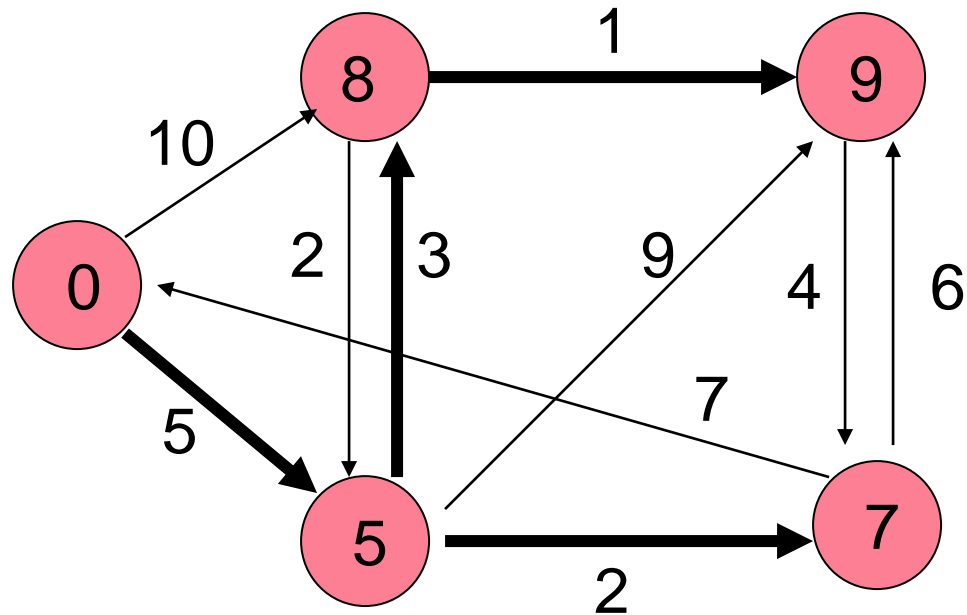
The Unconsidered



The Under Consideration (w)



# Dijkstra Example – Done





# Dijkstra Comments

---

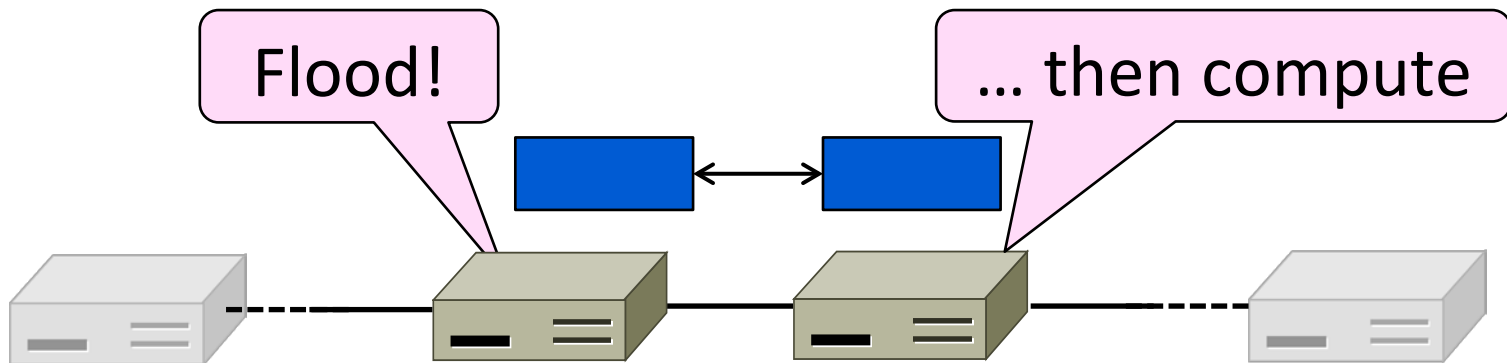
- Finds shortest paths in order of increasing distance from source
  - Leverages optimality property
- Runtime depends on efficiency of extracting min-cost node
  - Superlinear with network size (grows fast)
  - Using Fibonacci Heaps the complexity turns out to be  $O(|E| + |V| \log |V|)$  (E: # edges, V: # vertices)
- Gives complete source/sink tree
  - More than needed for forwarding!
  - But requires **complete topology**



# Link-State Routing Approach

---

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



# Link-State Protocols

---

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



# Link-State Routing

---

- Assume:
  - Each router knows only address/cost of neighbors
- Goal:
  - Calculate routing table of next hop information for each destination at each router
- Idea:
  - Each node tells all nodes in the network about the distances to its neighbors



# Link-State 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



# Link State Routing Process

---

- Tell all routers the topology and have each compute best paths
- Two phases:
  1. Nodes flood topology in the form of link state packets (LSPs)
    - Each node learns full topology
  2. Each node computes its own forwarding table
    - By running Dijkstra's algorithm



# Phase 1: Topology Dissemination

---


- Each router maintains link state database and periodically floods link state packets (LSPs)
  - LSPs contain [router, neighbors, costs]
- Each router forwards/floods LSPs not already in its database on all ports except where received
  - Each LSP will travel over the same link at most once in each direction
- Flooding is fast, and can be made reliable with acknowledgments



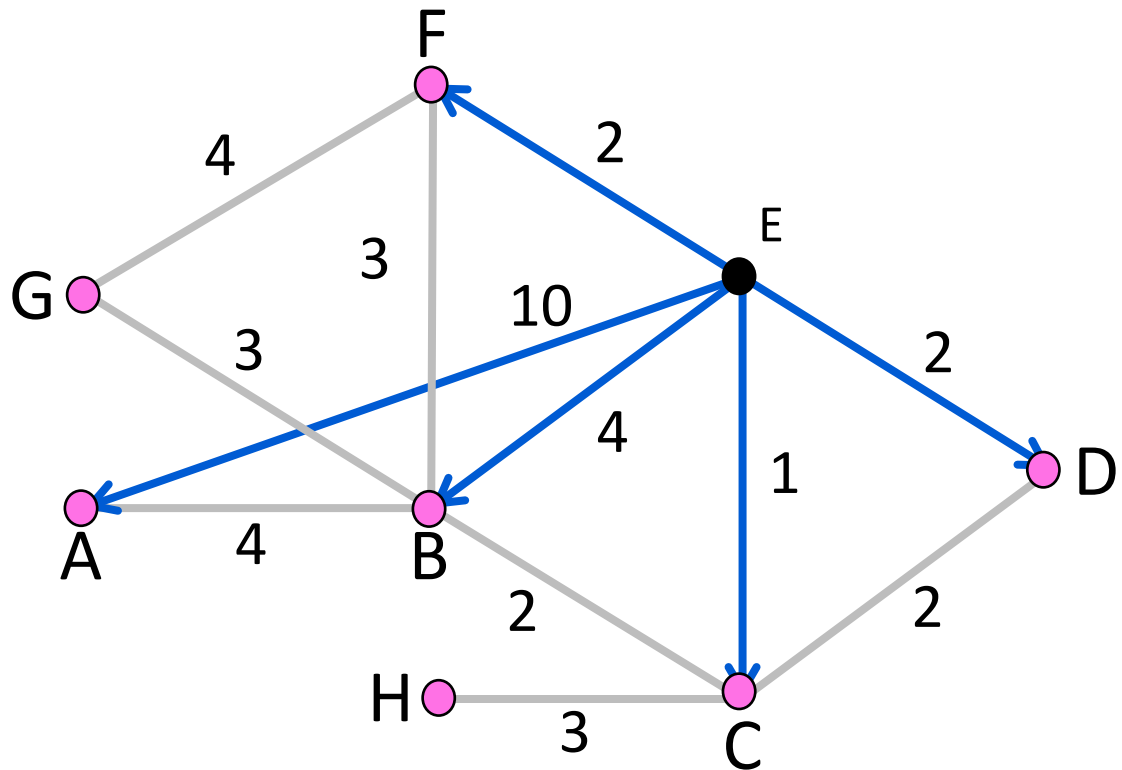
# Phase 1: Topology Dissemination

- 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



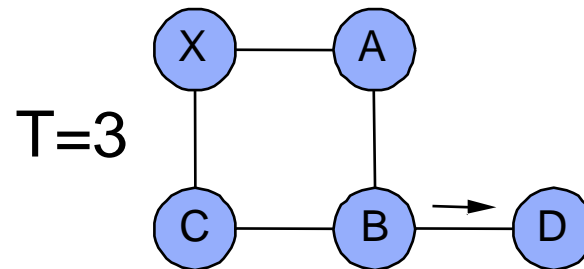
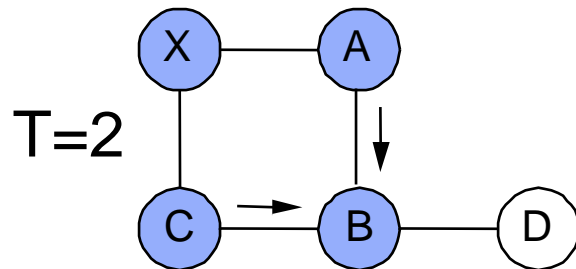
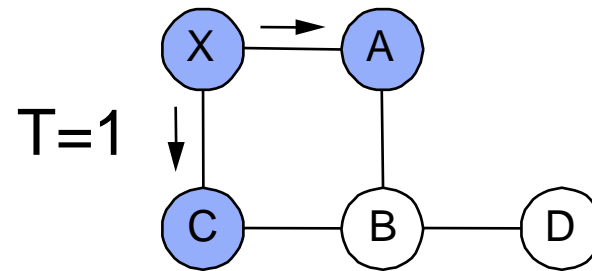
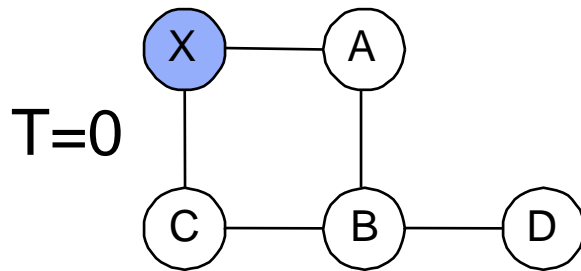
Seq. #	
A	10
B	4
C	1
D	2
F	2





# Example

- LSP generated by X at  $T=0$
- Nodes become blue as they receive it



# Phase 2: Route Computation

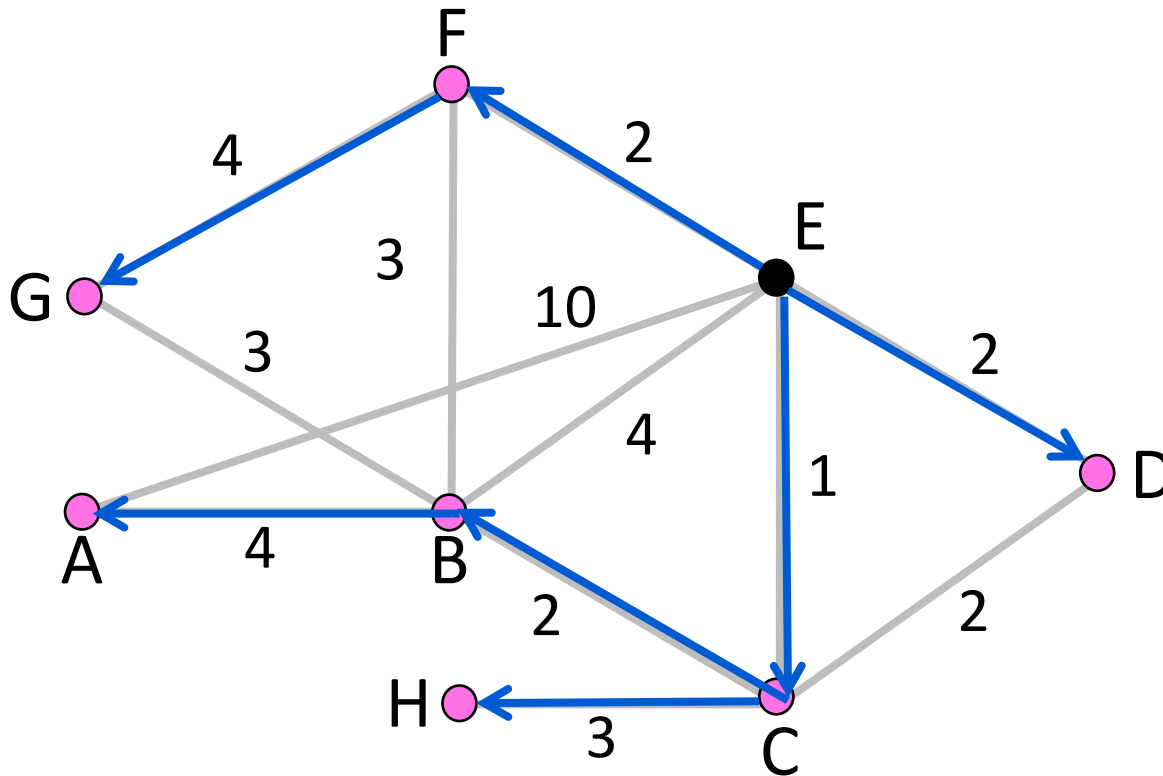
---

- Each node has full topology
  - By combining all LSPs from all nodes
- Each node 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

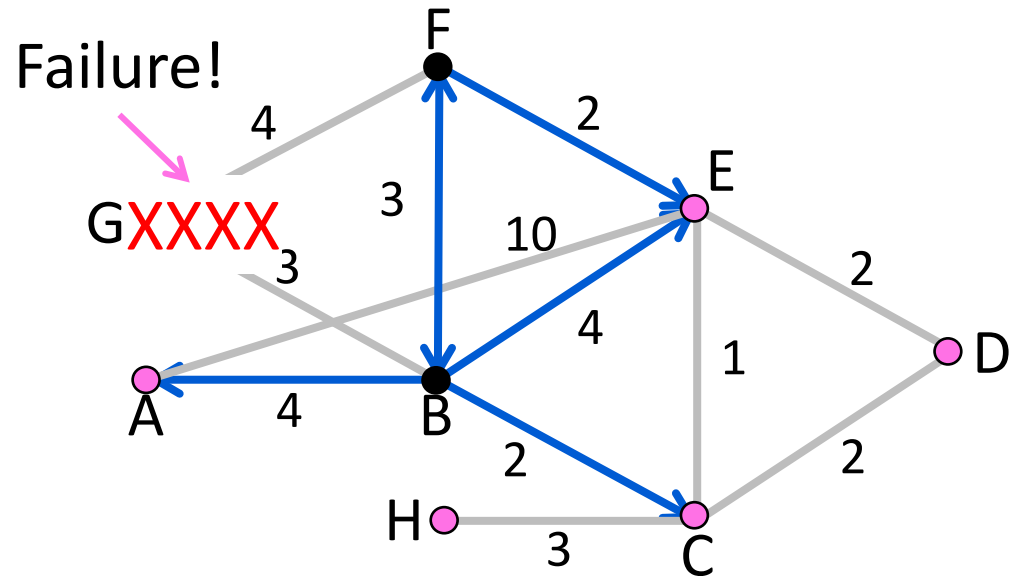
To	Next
A	C
B	C
C	C
D	D
E	--
F	F
G	F
H	C



# Handling Changes

- Nodes adjacent to failed link or node will notice
  - Flood updated LSP with less connectivity

B's LSP		F's LSP	
Seq. #		Seq. #	
A	4	B	3
C	2	E	2
E	4	<del>G</del>	<del>4</del>
F	3		
<del>G</del>	<del>3</del>		



# Handling Changes (2)

---

- Link failure
  - Both nodes sharing the link notice, send updated LSPs
  - Link is removed from the topology
- Node failure
  - All neighbors of the failed node notice link(s) failed
  - Failed node can't update its own LSP
  - But it is OK: all links to node removed
- When link/node fails need to remove old data  
How?
  - LSPs carry sequence numbers to determine new data
  - Send a new LSP with cost infinity to signal a link down



# Handling Changes (2)

---

- 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
- What happens if the network is partitioned and heals?
  - Different LS databases must be synchronized
  - A version number is used!
- Much of complexity is due to handling corner cases (as usual!)



# Open Shortest Path First (OSPF)

---

- Most widely-used Link State protocol today
- Basic link state algorithms plus many features:
  - Authentication of routing messages
  - Extra hierarchy: partition into routing areas
    - Only bordering routers send link state information to another area
      - Reduces chatter
      - Border router “summarizes” network costs within an “Area” by making it appear as though it is directly connected to all interior routers
    - Load balancing





# Cost Metrics

---

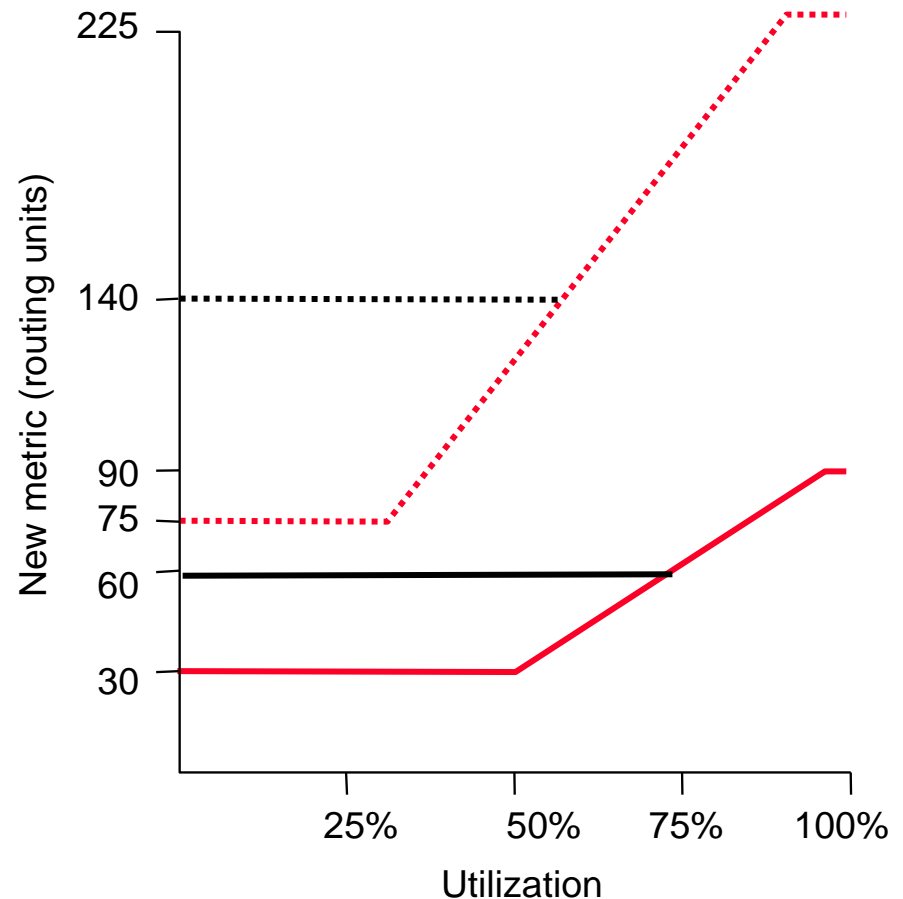
- How should we choose cost?
  - To get high bandwidth, low delay or low loss?
  - Do they depend on the load?
- Static Metrics
  - Hopcount is easy but treats OC3 (155 Mbps) and T1 (1.5 Mbps)
  - Can tweak result with manually assigned costs
- Dynamic Metrics
  - Depend on load; try to avoid hotspots (congestion)
  - But can lead to oscillations (damping needed)



# Revised ARPANET Cost Metric

- Based on load and link
- Variation limited (3:1) and change damped
- Capacity dominates at low load; we only try to move traffic if high load

9.6-Kbps satellite link	-----
9.6-Kbps terrestrial link	-----
56-Kbps satellite link	-----
56-Kbps terrestrial link	-----



# Cost Estimation

---

- How we calculate the cost of a stochastic variable?
  - Simple cumulative average
    - 2 variables, avg and # samples
  - Simple moving average
    - W variables, the size of window W
  - Weighted moving average
    - W variables + weight function
  - Exponentially weighted moving average
    - 1 variable + alpha factor
    - $EWMA_n = \alpha * S_n + (1 - \alpha) * EWMA_{n-1}$



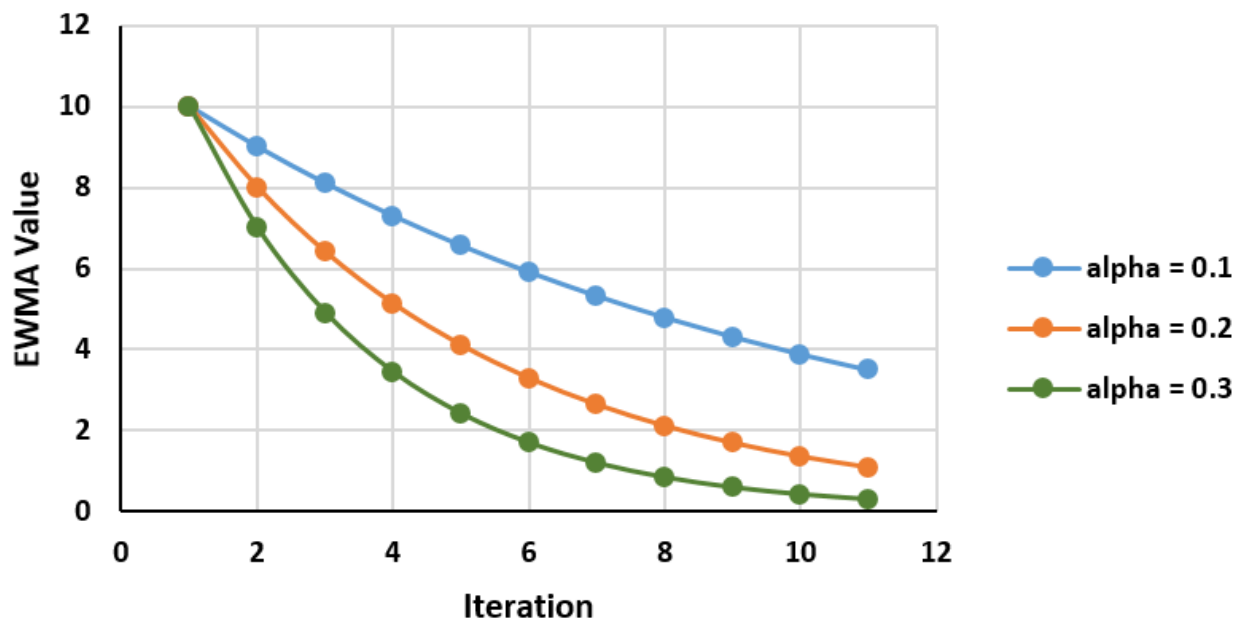
# EWMA Example

$$EWMA_n = \alpha * S_n + (1 - \alpha) * EWMA_{n-1}$$

$$EWMA_0 = 10$$

Iteration	Samples	Alpha Values		
		0.1	0.2	0.3
0		10	10	10
1	0	9	8	7
2	0	8.1	6.4	4.9
3	0	7.29	5.12	3.43
4	0	6.561	4.096	2.401
5	0	5.9049	3.2768	1.6807
6	0	5.31441	2.62144	1.17649
7	0	4.782969	2.097152	0.823543
8	0	4.3046721	1.6777216	0.5764801
9	0	3.87420489	1.34217728	0.40353607
10	0	3.486784401	1.073741824	0.282475249

EWMA for different Alpha values



# Key Concepts

---

- Routing uses global knowledge; forwarding is local
- Many different algorithms address the routing problem
  - We have looked at one class of algorithms:
    - Link State (OSPF)
      - Distributed data dissemination but centralized computation
      - Fast convergence, limited scalability (due to flooding)
- Challenges:
  - Handling failures/changes
  - Defining “best” paths
  - Scaling to millions of users
  - Cost metrics and cost estimation
- Next: DV Routing and IP Helpers

