Network Routing
CS 118
Computer Network Fundamentals
Peter Reiher

Routing Outline

- Background
- Key algorithms

Background

- What we’re doing
- Collecting our thoughts
- Goal
- Info requirements

What we're doing

- Using the network to run the network
  - Runs on top of an existing network
- What can we assume?
  - Who can you talk to?
  - What kind of messages can you send?
  - Who's in charge of setting this up?

Relaying and routing

- If we don't have a direct channel to the receiver, we ultimate must relay
  – Send our messages through some other node
  – Which forwards them towards the destination
- Easy if there's only one choice
  – You only connect to one other node
- For non-trivial topologies, some relaying involves choice
- Routing describes how we choose to relay

I’ll do it myself!

- Static routes
  - Manual entry by network operator
  - Boot-time configuration file
  - Boot-time initialization (DHCP)

- Default routes
  - Pass the buck
    (move the problem)

Limits of going solo

- Requires external reconfiguration
  - When a node joins, leaves
  - When a link is added or removed (dies)
- Bootstrapping is difficult
  - Need to deploy incrementally
  - Can't reach nodes that need configuration until some routing works
- Must assume others do it right
  - If you relay more than one hop

Automated routing

- Adaptive
  - No need to intervene externally
- Bootstraps itself
  - Each node can initiate discovery and relay

Collecting our thoughts

- Assume we have our “stack” DAG
  - I.e., maps between protocol name spaces
  - I.e., layers we can “stack”
- What other information do we need?
  - Who’s connected to whom
  - Who we can reach through whom
  - A way to differentiate paths
    - Weight, cost, delay, etc.

Terminology

- *Relaying*
  - Moving messages based on the DAG tables
  - Forwarding (typically IP)
  - Switching (typically Ethernet, ATM)

- *Routing*
  - Computing the relay tables
  - Route computation
  - Path computation

More terminology

- Two approaches to routing
  - Link state
  - Distance vector
- But both:
  - Depend on link state (up/down/load)
  - Calculate distance vectors (path costs)

Names are a pain sometimes!

How do we collect that info?

- Neighbors
  - We don’t need no stinkin’ relays!
  - Won’t get you far
- Six degrees of flooding
  - Your neighbors’ neighbors
  - Neighbors’ neighbors’ neighbors
  - Etc...

What do we flood?

- The topology
  - Who we are, who we're connected to
  - "Link state"
- Our decisions
  - Who we think we can reach

When do we flood

- In the beginning, all at once
  - Flood link state
  - Everyone computes their own routing
- In between each step of route computation
  - Who we can reach
  - Ends up flooding reachability

Goal

- Information to guide DAG traversal
  - A way to pick alternate next-layer tables
    - When both have viable translations
  - A way to pick from among proxies
    - I.e., multiple resolutions within one table
  - A way to populate the DAG tables
    - Relays are proxies for their destinations

Optimization

- Beyond just getting there…
  - Getting there in the best way
    - Lowest delay, highest BW, greatest reliability, etc.
  - Getting there without a loop

Information requirements

- Node name
  - A way to identify the node itself
- Link name
  - A way to identify each link
  - A single node may have many attached links
  - A single link may have many attached nodes
- Costs
  - To visit a node
  - To traverse a link
  - Cost != price in dollars
  - Usually expressed in delay units

Key algorithms

- Basic flooding
- Distance vector
- Link state

Basic flooding

- Start:
  - Get a request on interface A
- Relay out:
  - Send a copy on every interface

Does this include A?

When will this terminate?

Goals of flooding for routing

1. Get request to everyone reliably
2. Get responses back to the entity that needs them
   – In particular, let him know when he has all responses
3. Minimize the cost

- Assuming connectivity, of course

Limiting the flood

- Track the messages
- Track the nodes

Hopcount in messages

- At each relay
  - Drop count one
  - Stop flooding when zero

Will this work? Under what conditions?

What do we have to know?

Checkbox at nodes

- On receive
  - Set visited = TRUE
- Once visited
  - Don’t relay any more

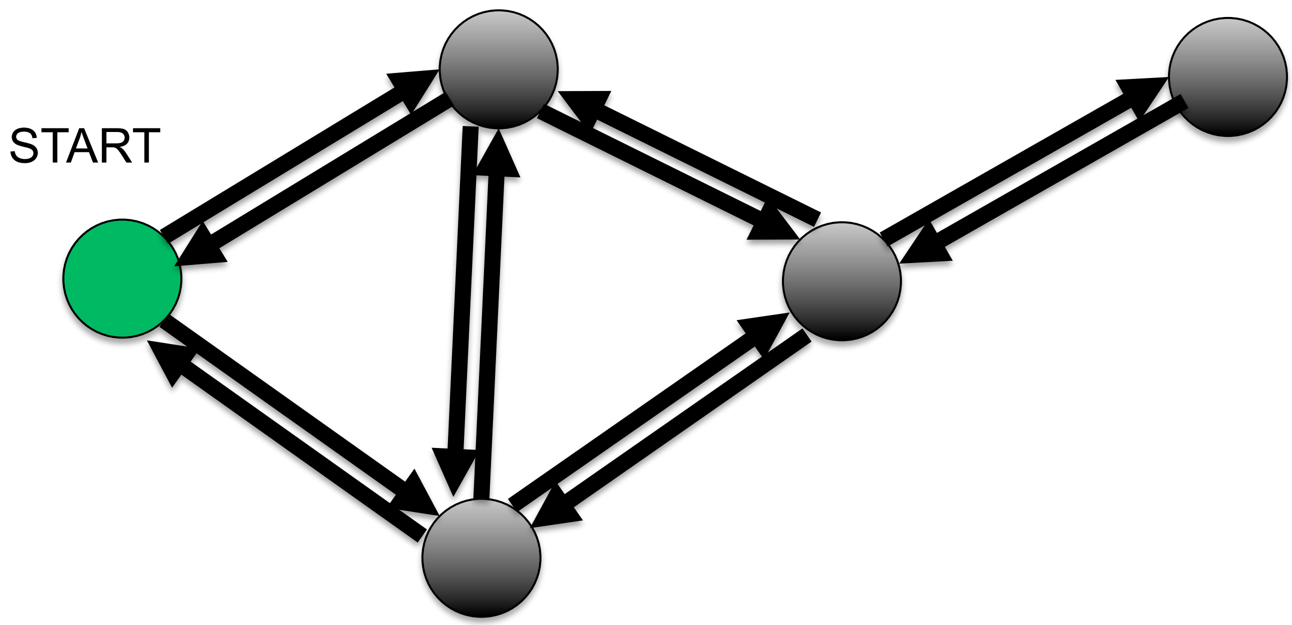
Will this work?

How will initiator know when it’s done?

Controlled flooding

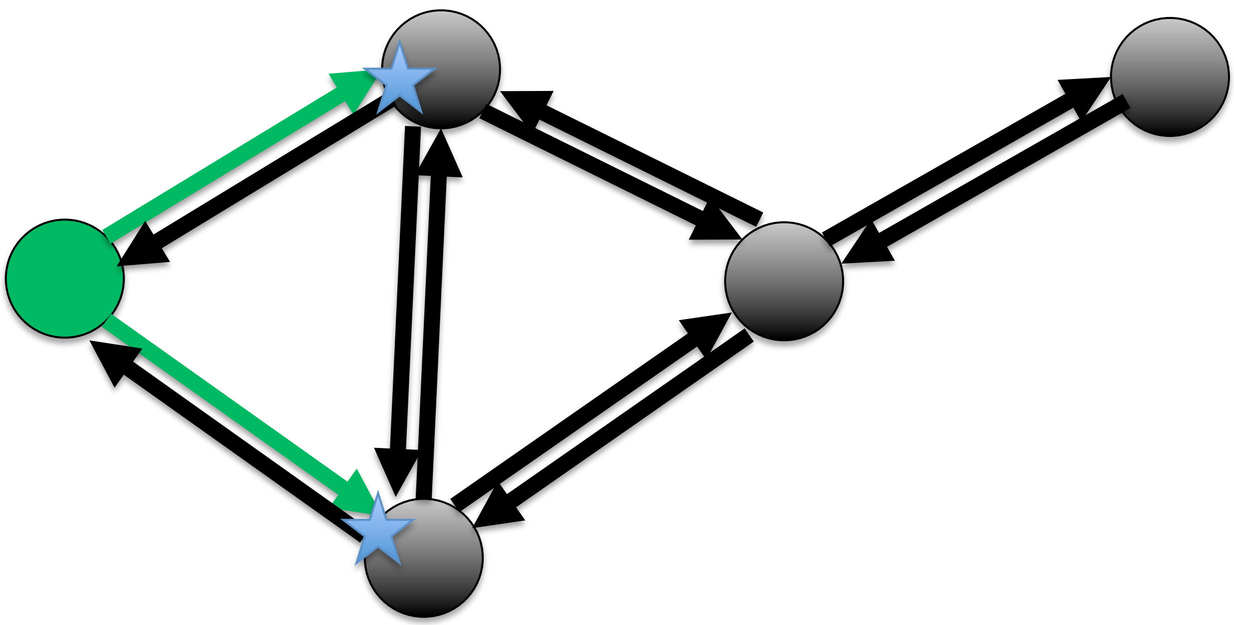
- Chang's Echo algorithm (1982)
  – Start:
    - Get the message on interface A
  – Relay out:
    - Send a copy on every interface except A
  – Relay in:
    - Wait for a copy on every interface except A
  – End:
    - Send the message back to A

A picture of Echo

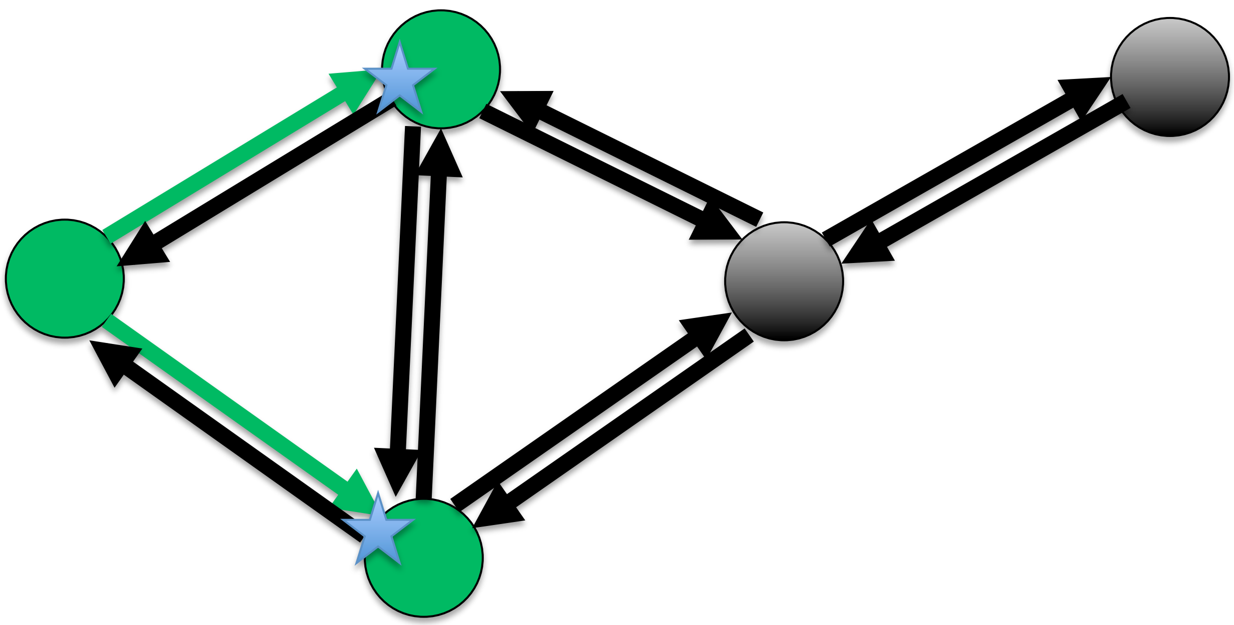


A picture of Echo

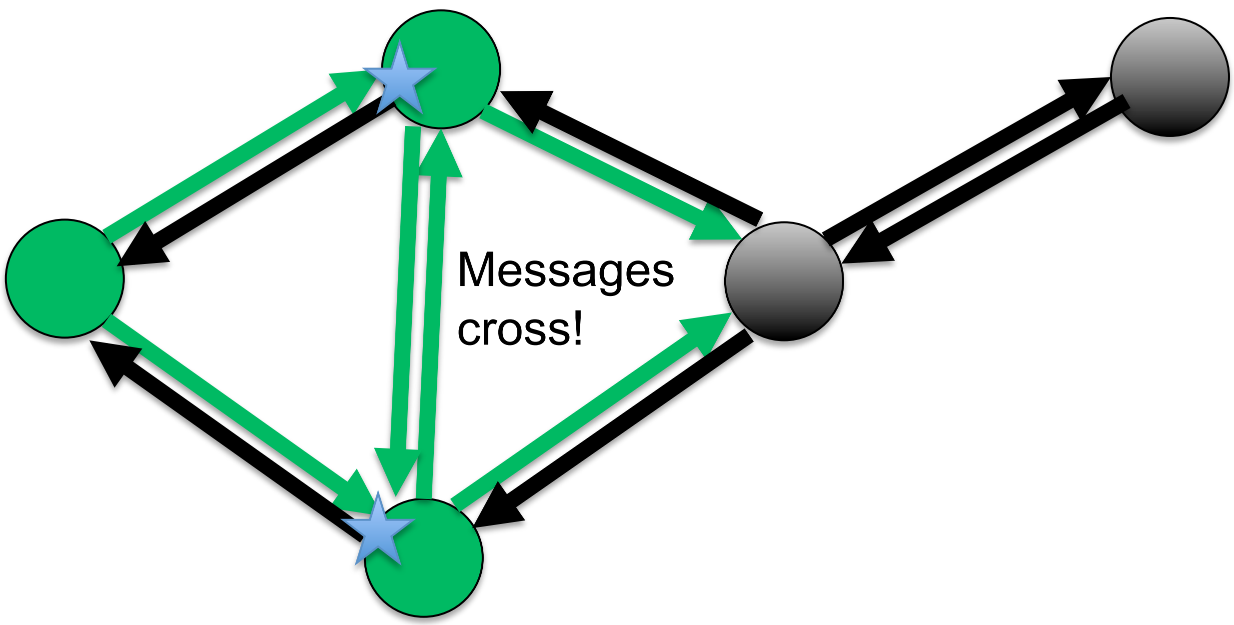
Mark incoming links



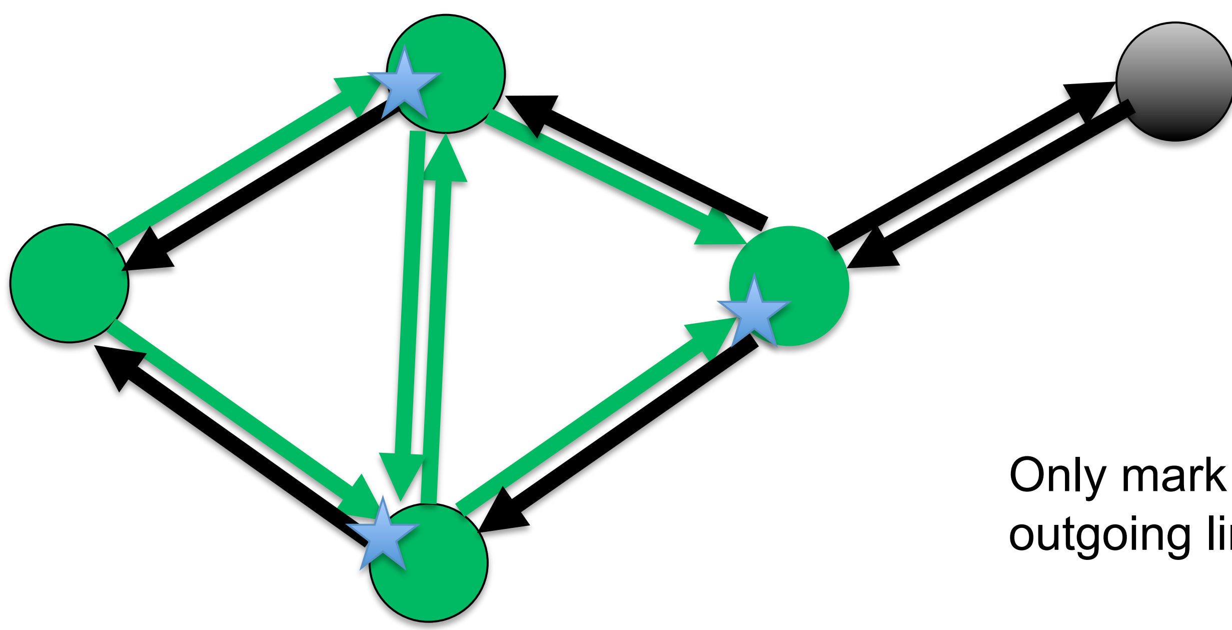
A picture of Echo



A picture of Echo

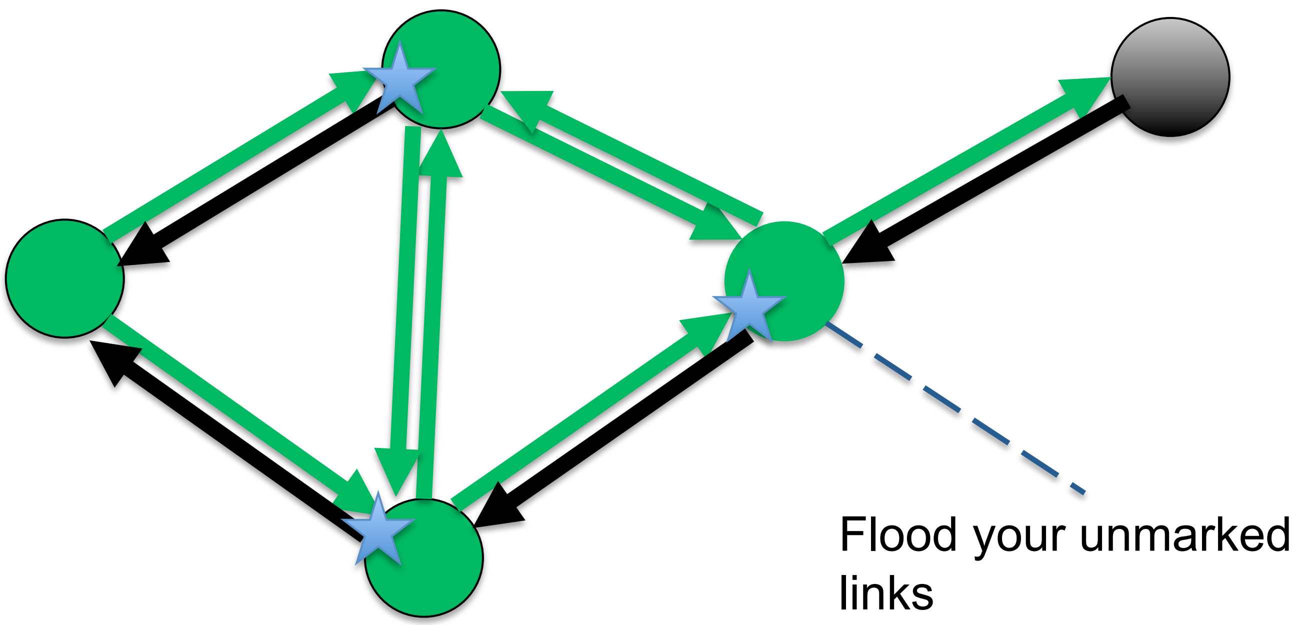


A picture of Echo



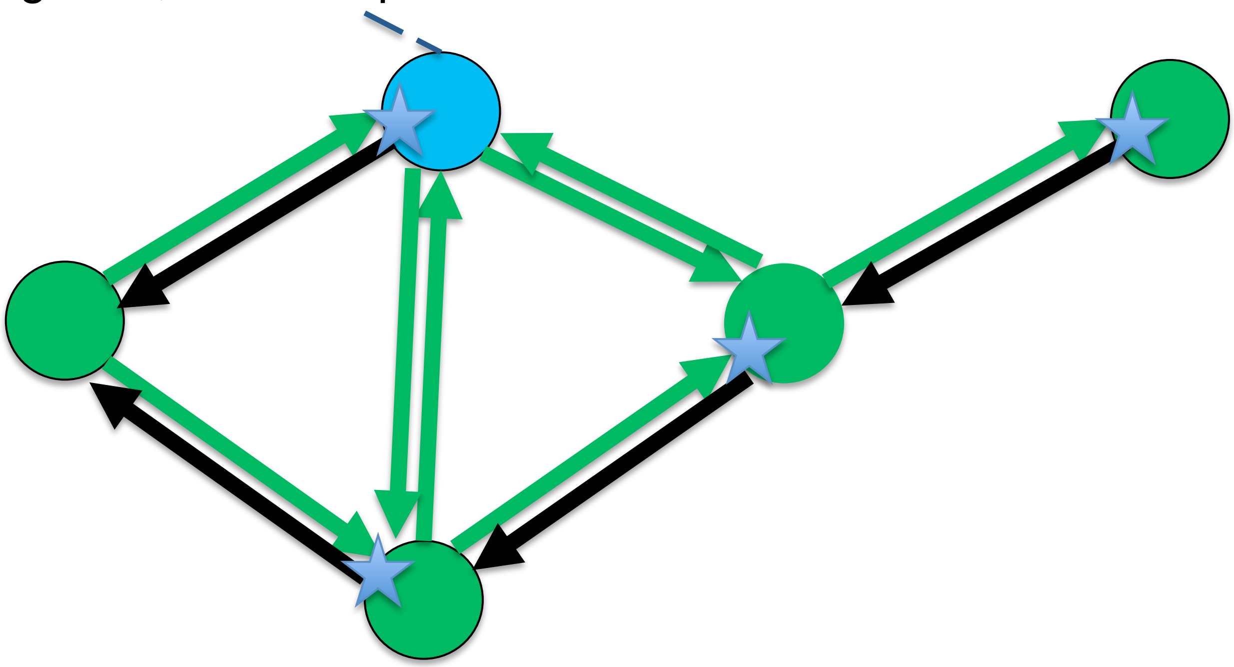
Only mark one outgoing link

A picture of Echo

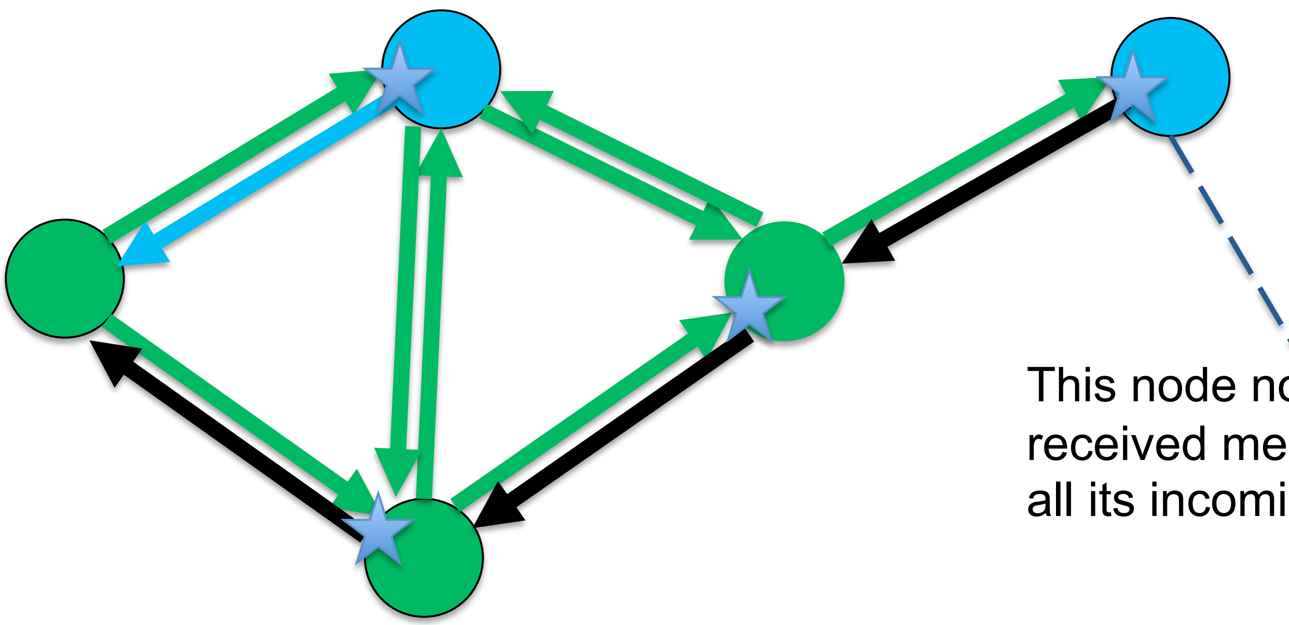


A picture of Echo

This node received messages on all its incoming links; it can respond on its marked link

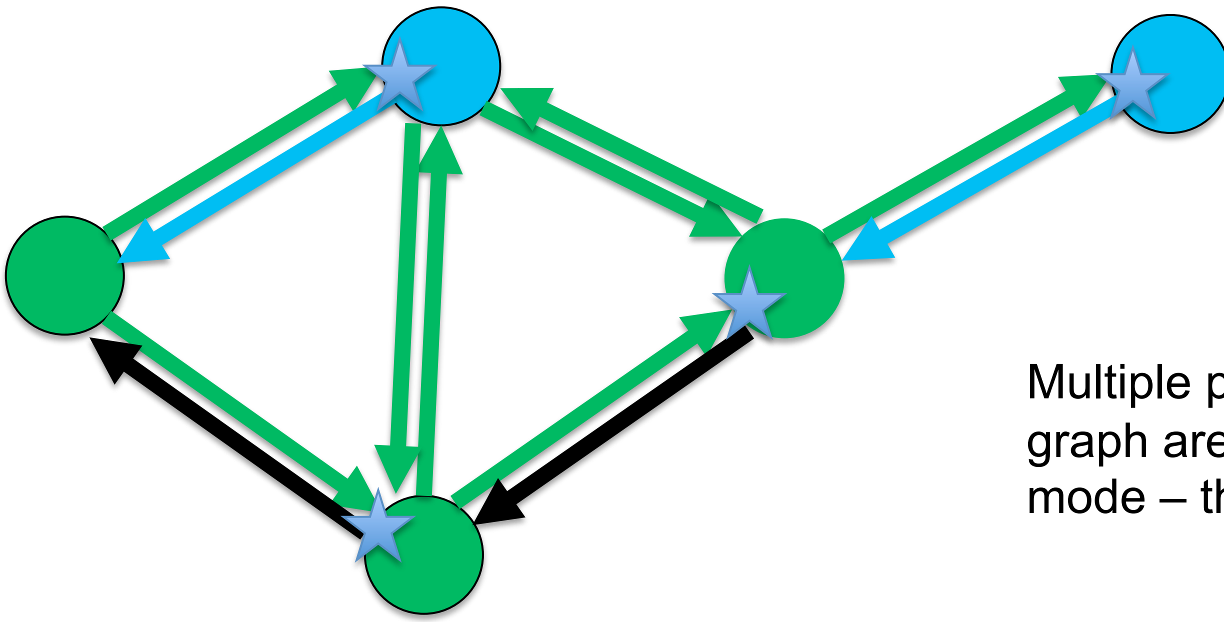


A picture of Echo



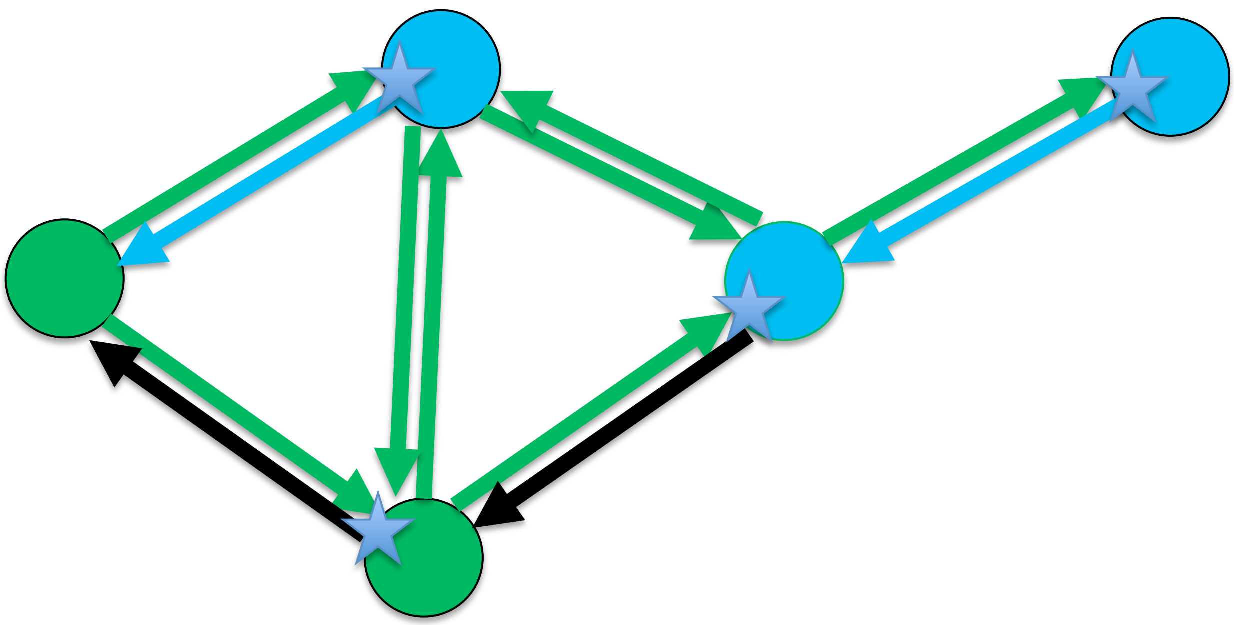
This node now has received messages on all its incoming links too

A picture of Echo

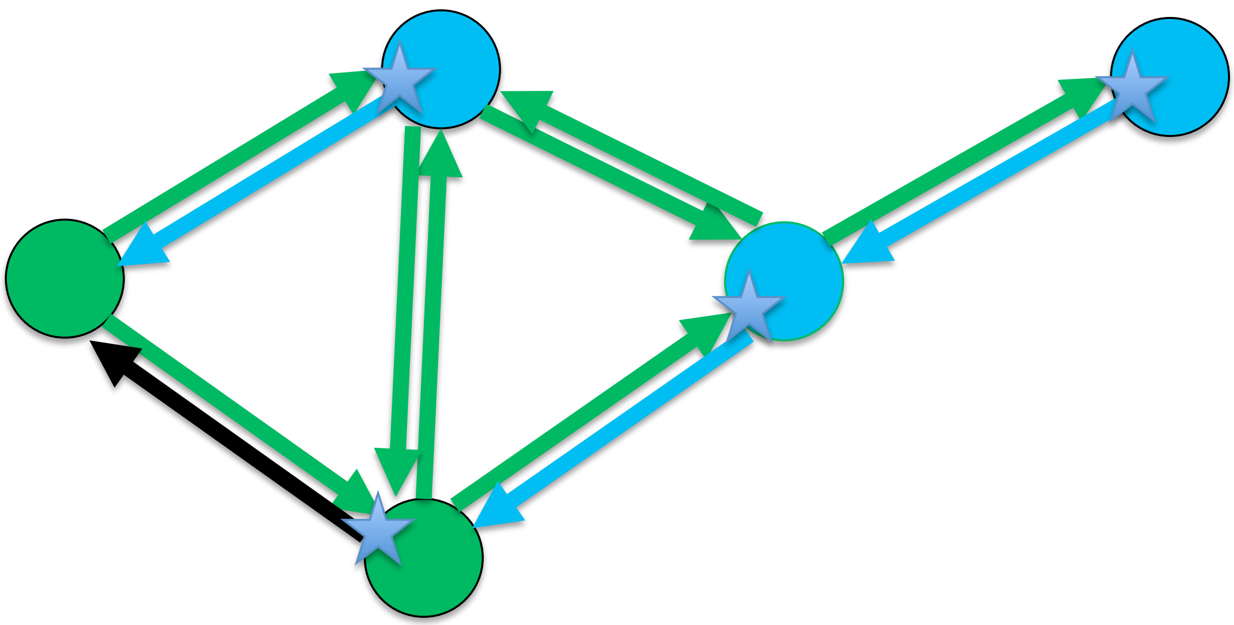


Multiple parts of the graph are in “ACK” mode – that’s OK

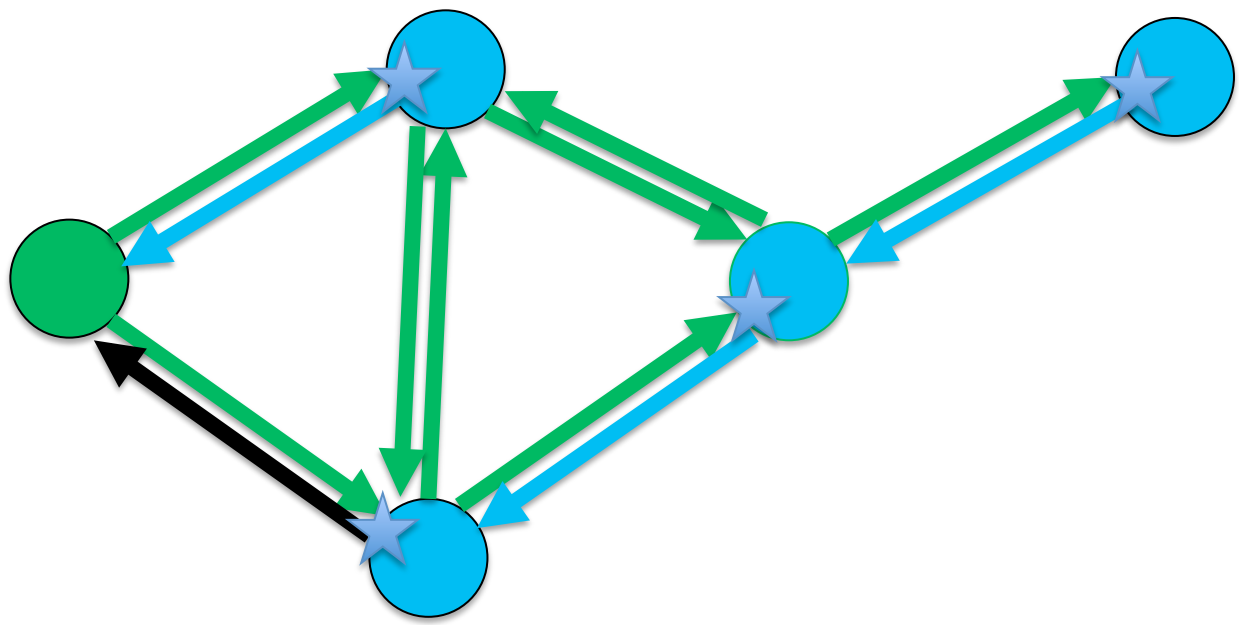
A picture of Echo



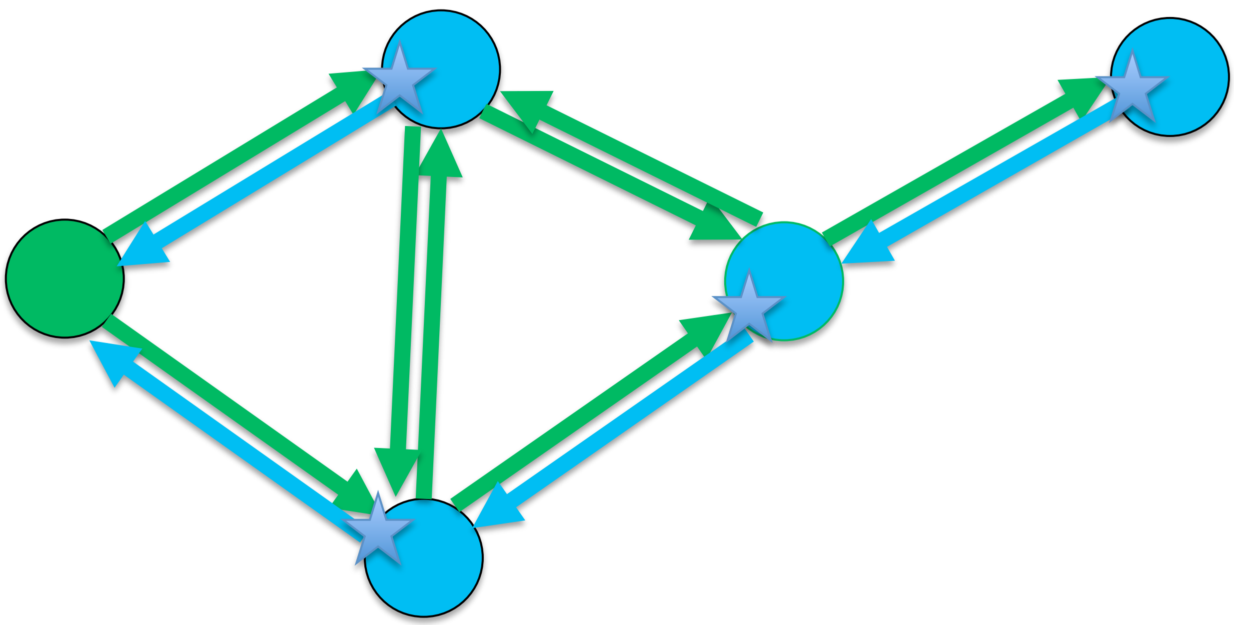
A picture of Echo



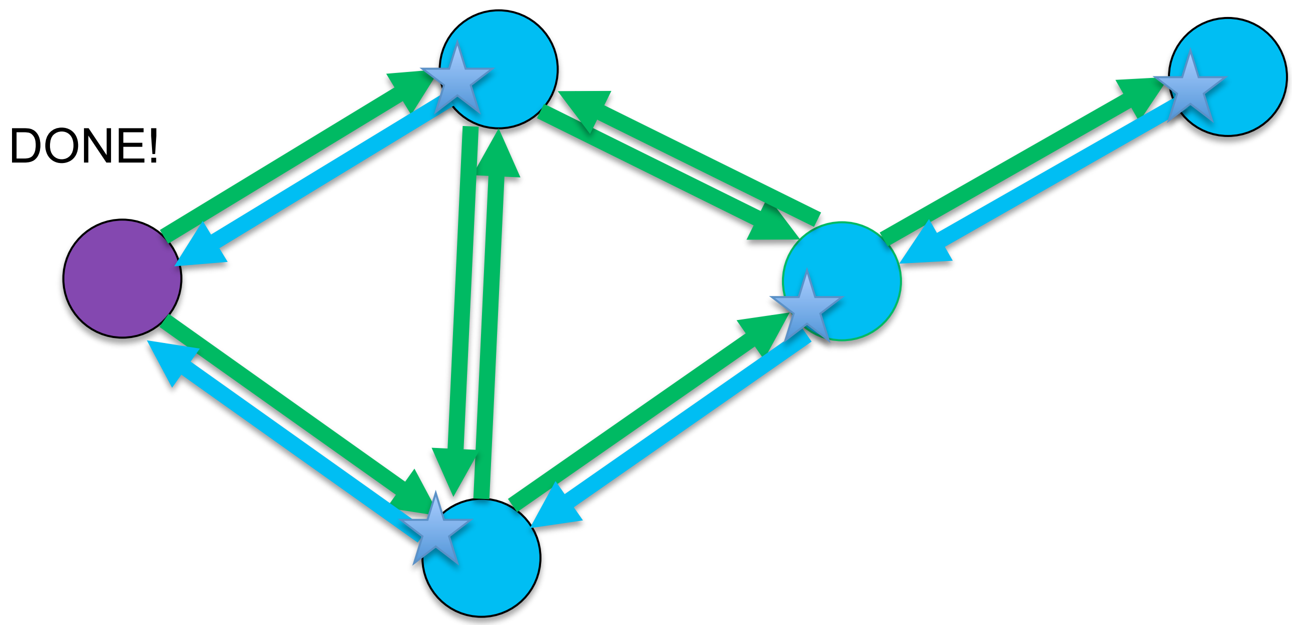
A picture of Echo



A picture of Echo



A picture of Echo



Properties of the echo algorithm

- Assumes
  - Bidirectional links
  - Connected graph (no isolated subgraphs)

- Exactly E messages
  - One message on each link in each direction

- Scalably confirms a flood
  - Without counts in the messages OR counts in the nodes!
  - I.e., with a single message and one flag per interface at each node (finite state), it can confirm the flood of a network of arbitrary size

What did all that get us?

- Flooding
  - With confirmation
- Now what?
  - What do we DO with that capability?

Two phase flooding

- Phase 1
  - Outgoing messages start the algorithm
  - Incoming messages (starred links) list everyone you've heard from
  - At end of phase 1, initiator has complete map

- Phase 2
  - Initiator floods the map
  - When the algorithm is done, everyone knows everyone has the complete map

What map do we flood?

- The entire map
  - Expensive to flood
  - Each node has to calculate connectivity
- The shortest paths
  - Sure, but how do we get <u>*those*</u>?

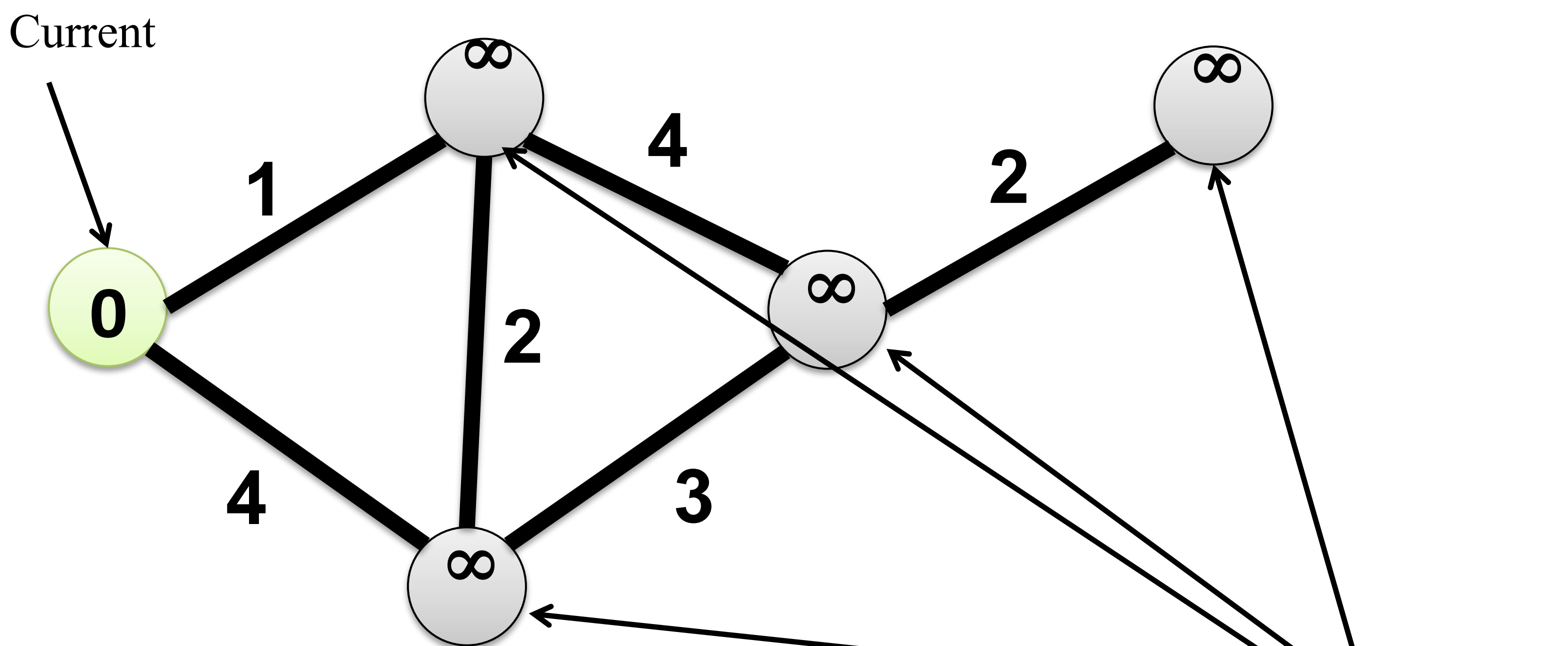
Link state

- Flood the entire map
- Calculate shortest paths
  – Dijkstra's algorithm

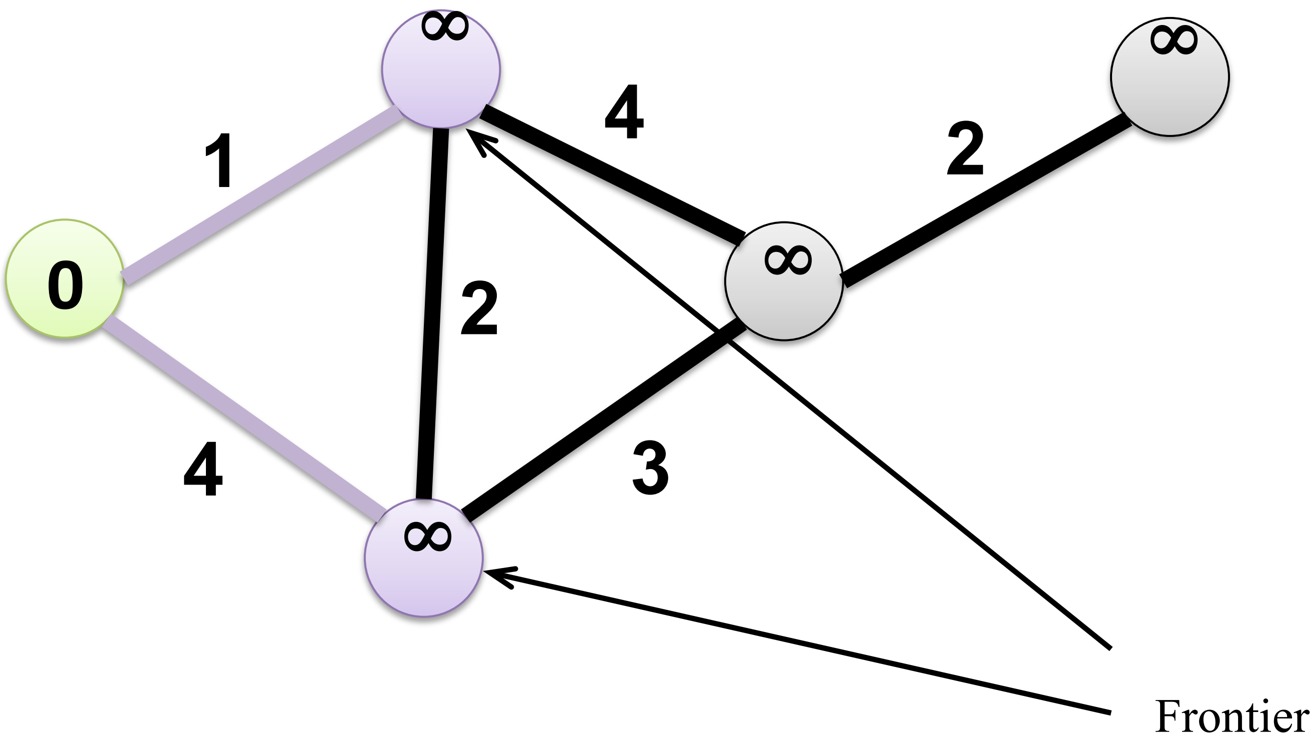
Dijkstra's algorithm

- Not a distributed algorithm!
- Start with one node in the CURRENT set
  - Mark it as zero cost
- For the CURRENT node
  - Check its links for UNVISITED or FRONTIER neighbors
    - Add each UNVISITED node it can reach to the FRONTIER set with a new cost of "link" + CURRENT node cost
    - If the node is already in the FRONTIER set, compare the new cost to the previous cost; update the cost if it is lower
  - Once done, mark the CURRENT node as VISITED
  - Find the FRONTIER node with the smallest cost; move it to CURRENT and repeat
- Continue until there are no more FRONTIER nodes

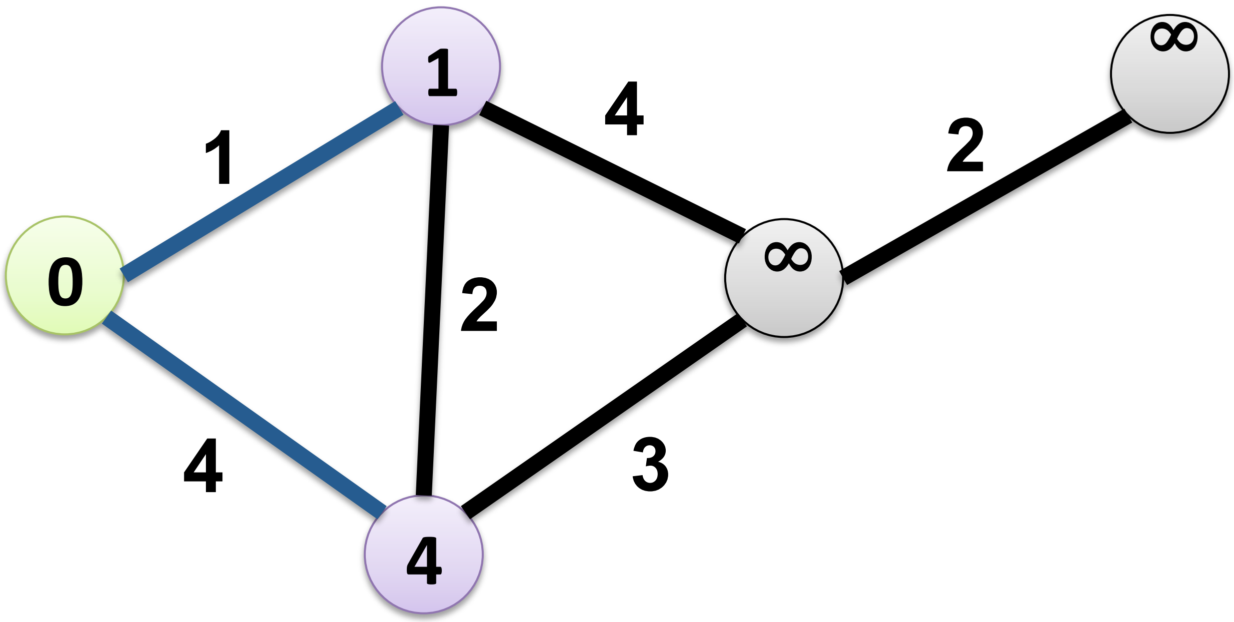
Dijkstra's Algorithm at work



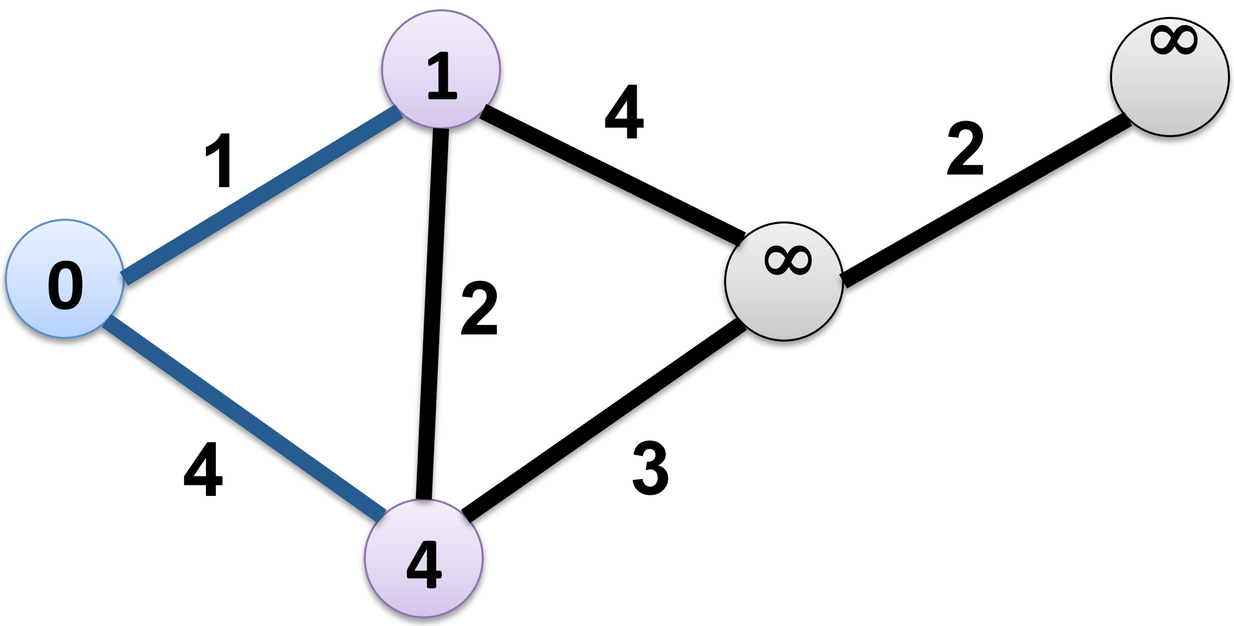
Dijkstra's Algorithm at work



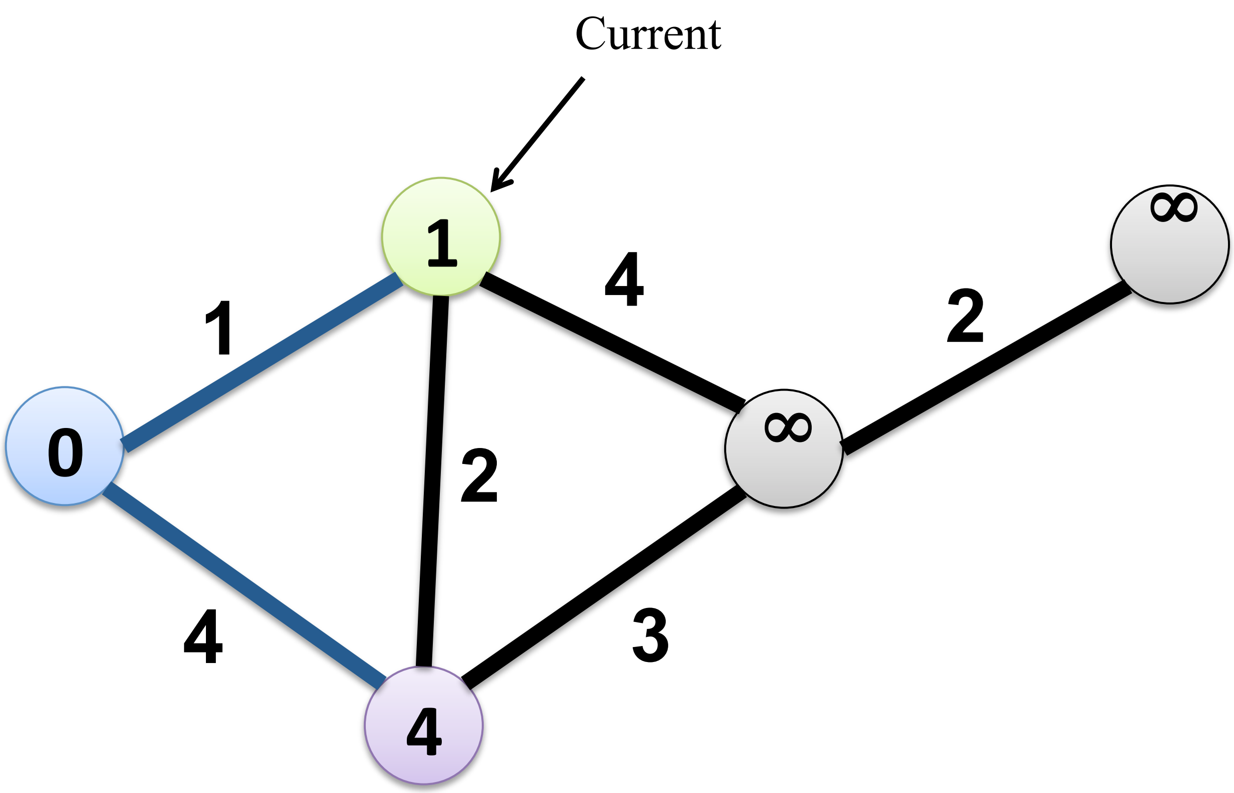
Dijkstra’s Algorithm at work



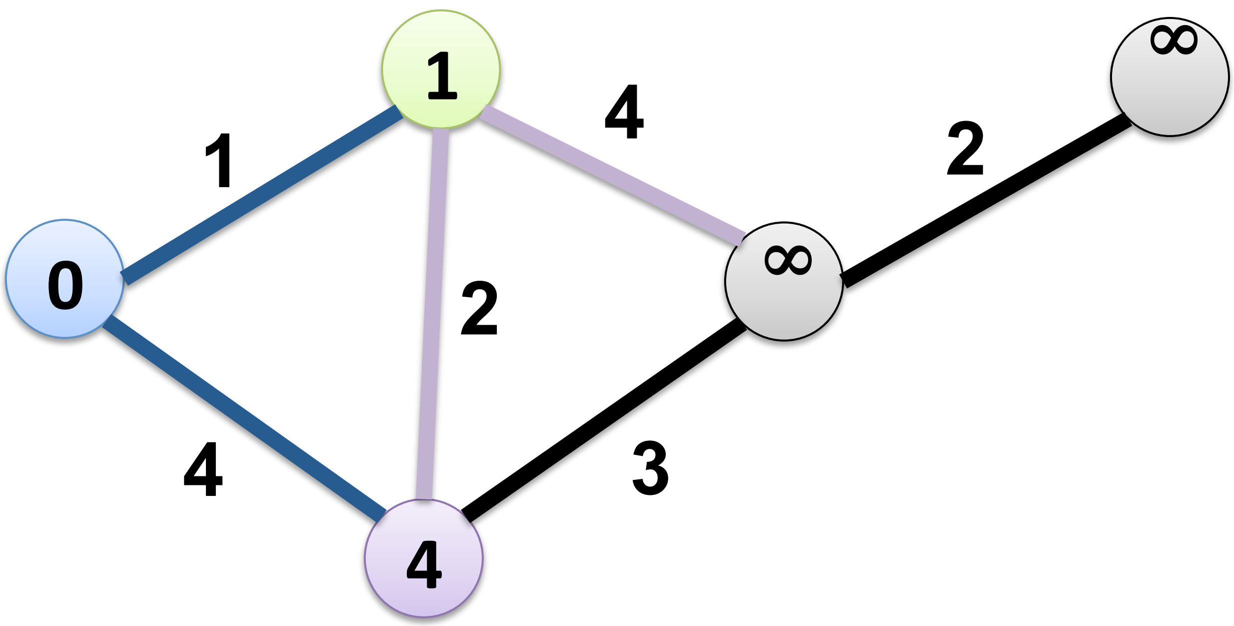
Dijkstra's Algorithm at work



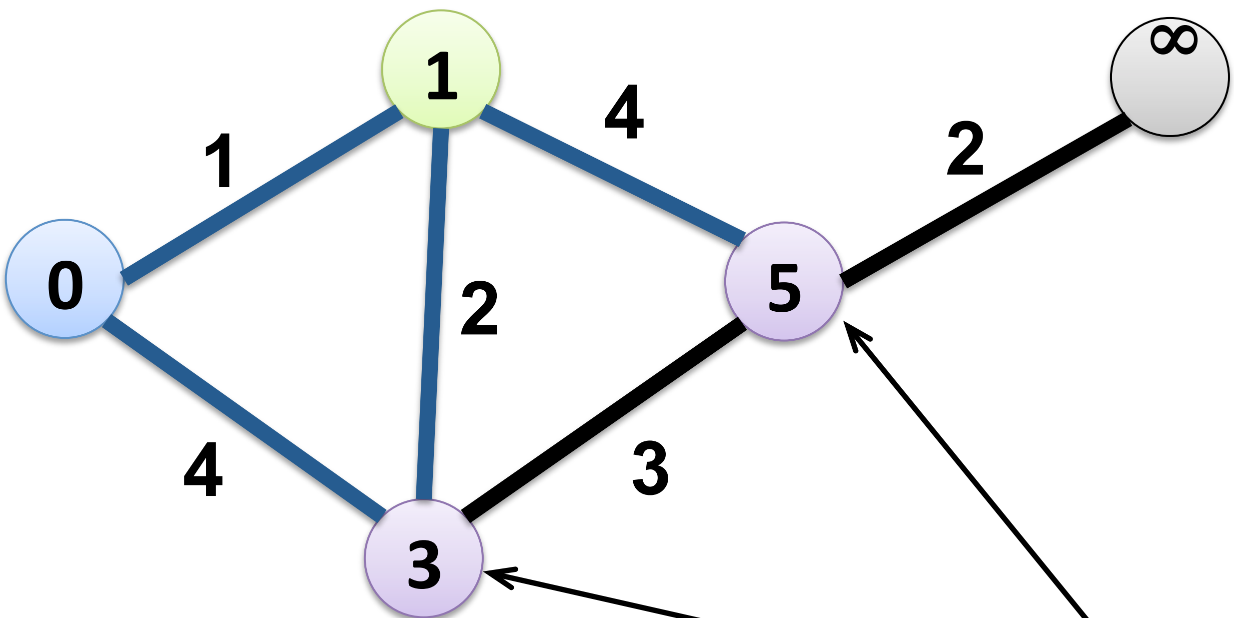
Dijkstra's Algorithm at work



Dijkstra's Algorithm at work



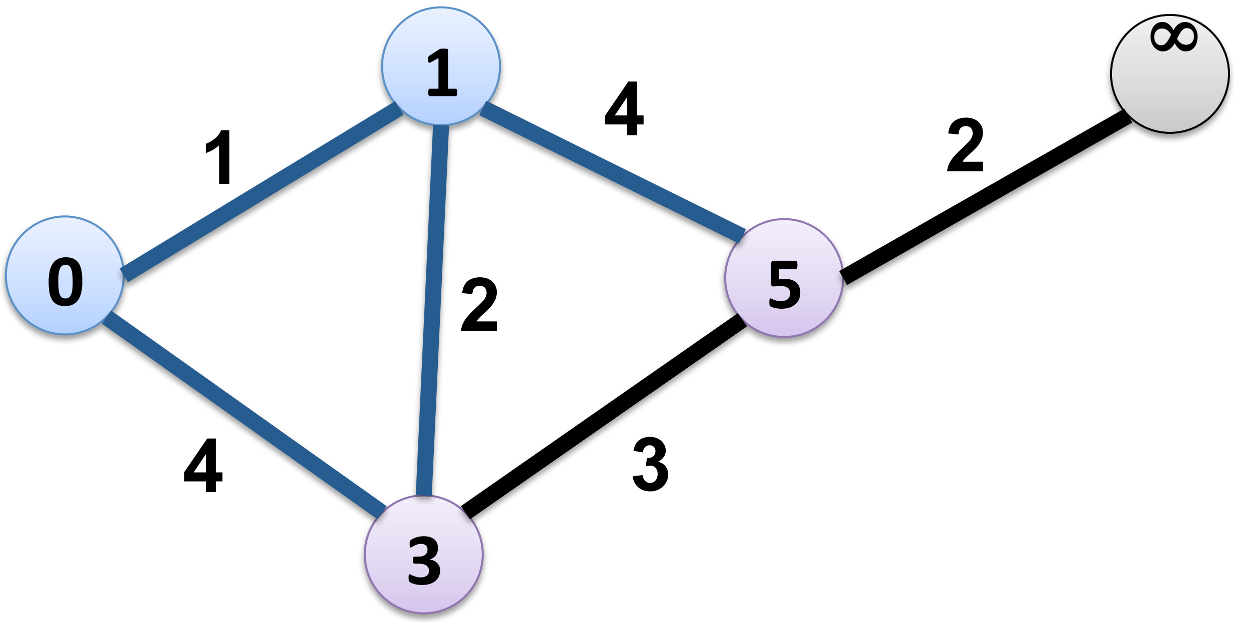
Dijkstra's Algorithm at work



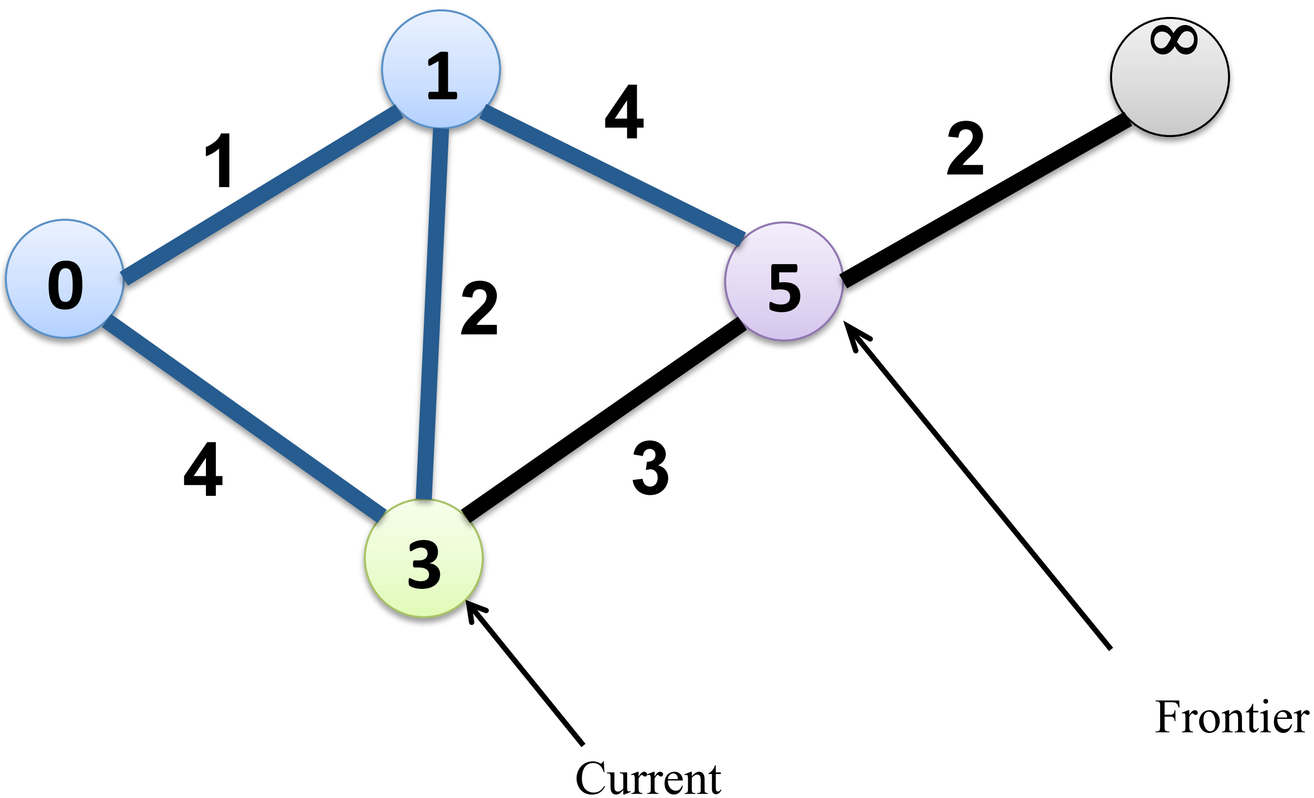
Note: This node's cost dropped at this step

Frontier

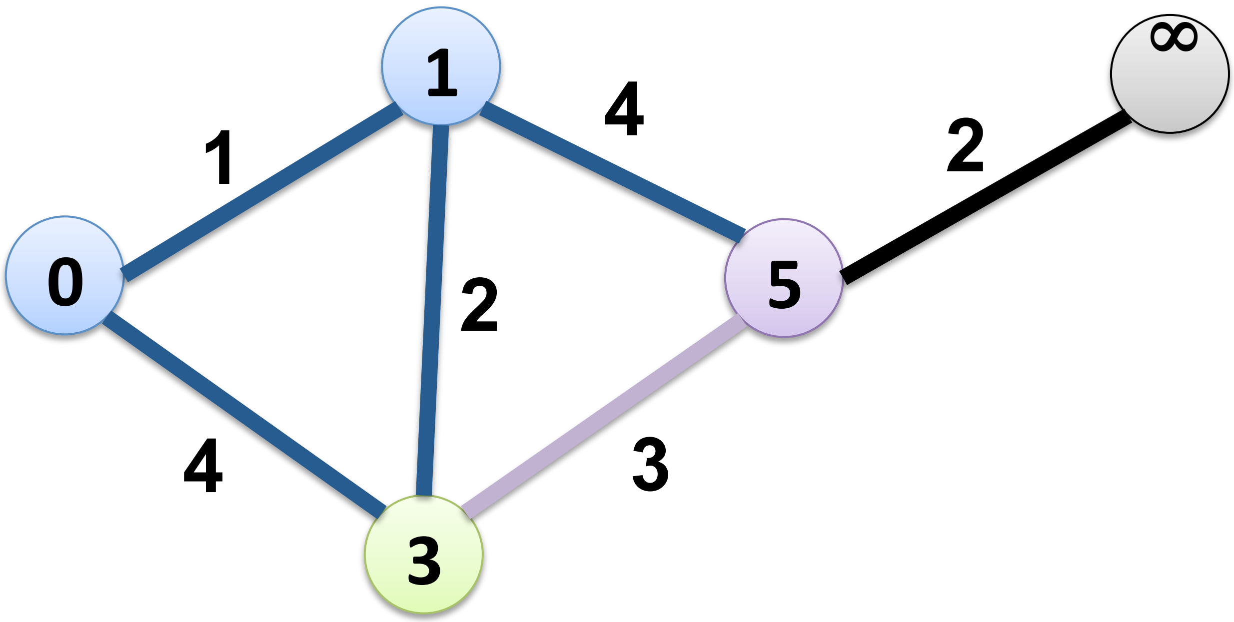
Dijkstra’s Algorithm at work



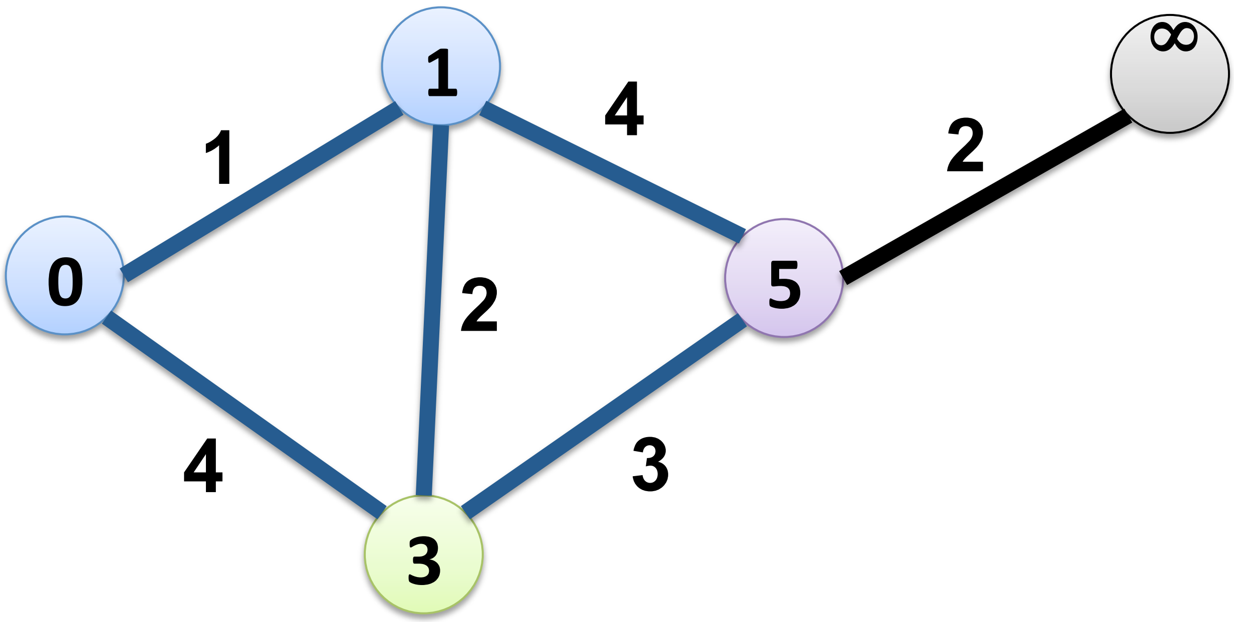
Dijkstra's Algorithm at work



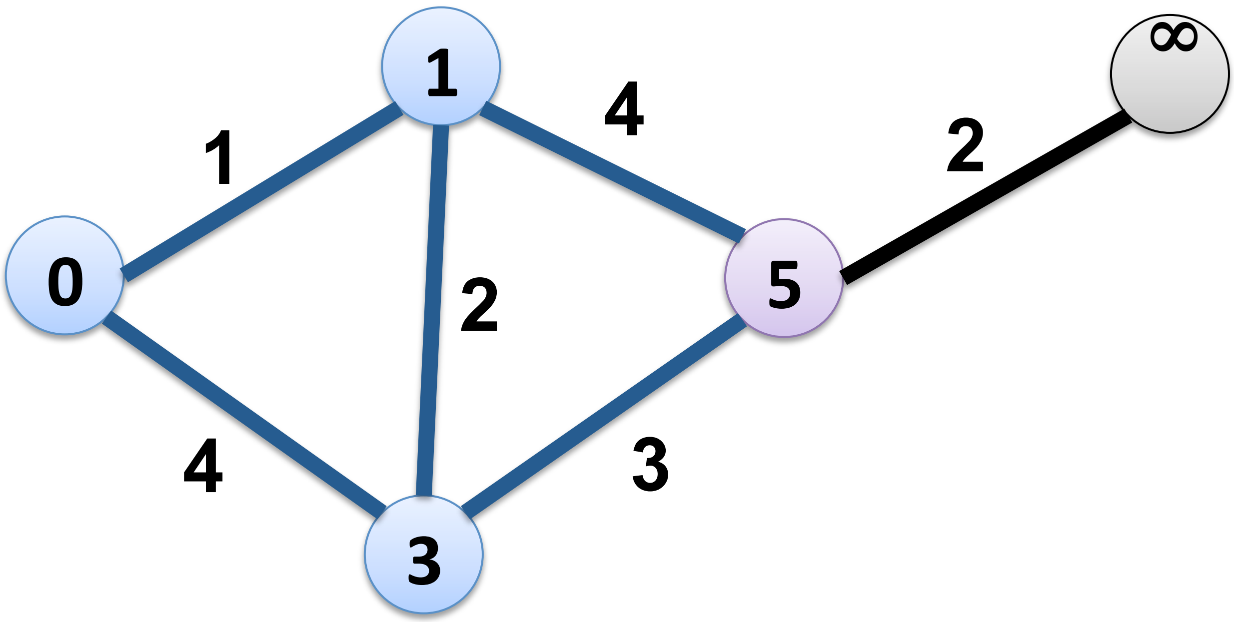
Dijkstra's Algorithm at work



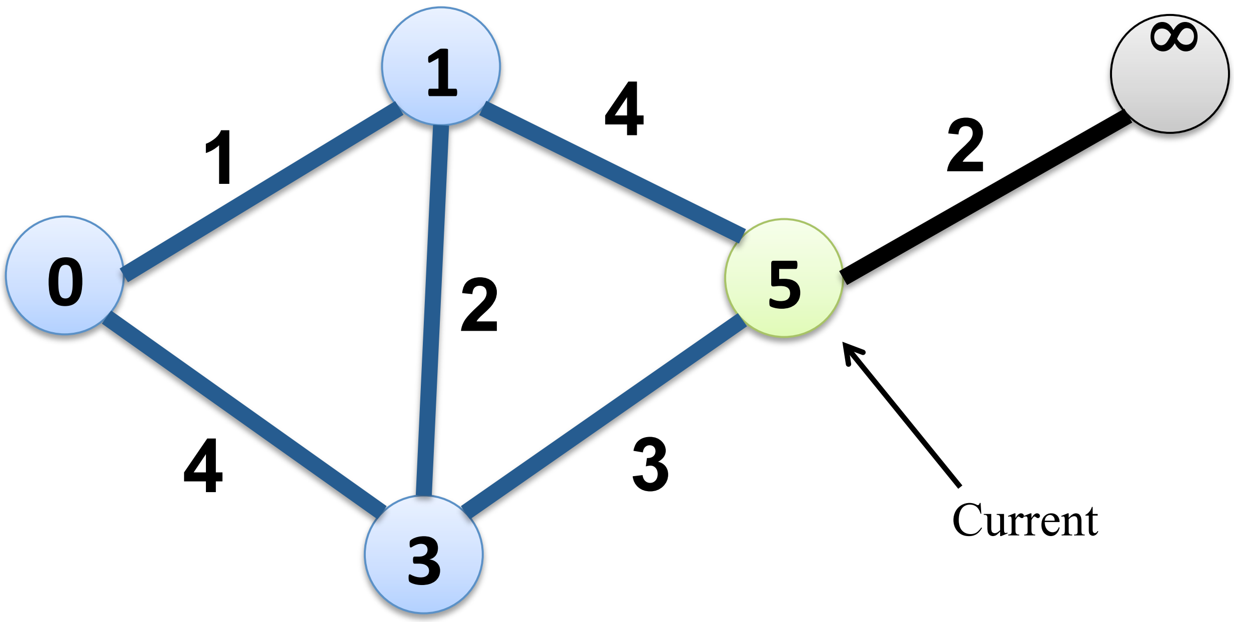
Dijkstra's Algorithm at work



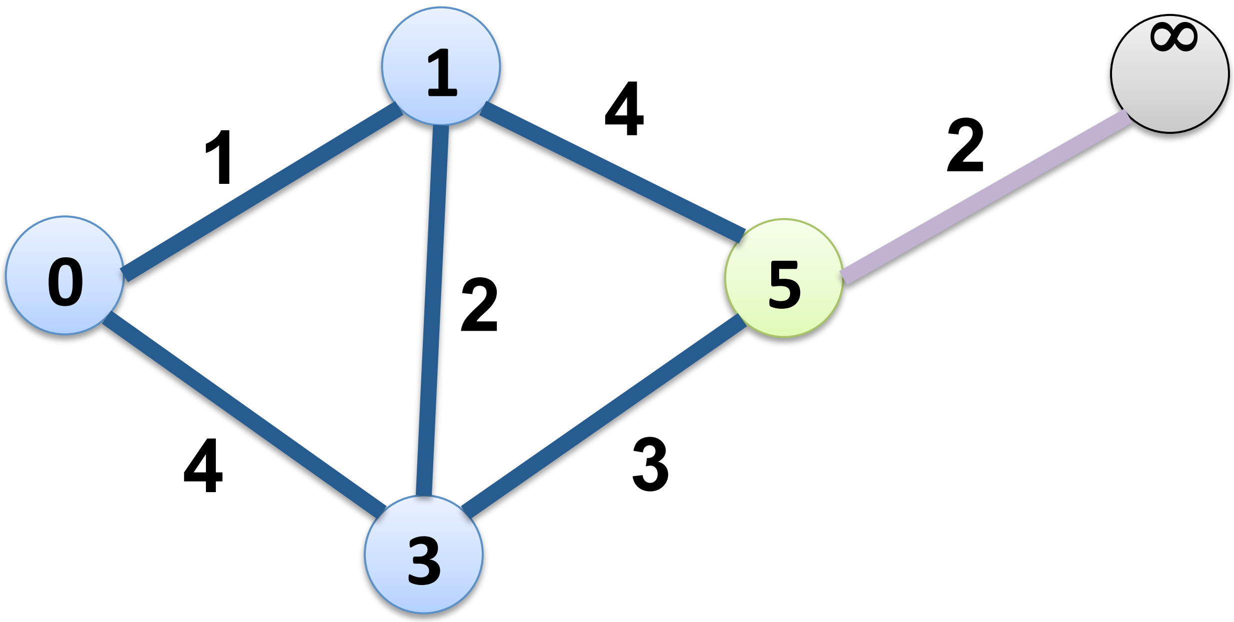
Dijkstra's Algorithm at work



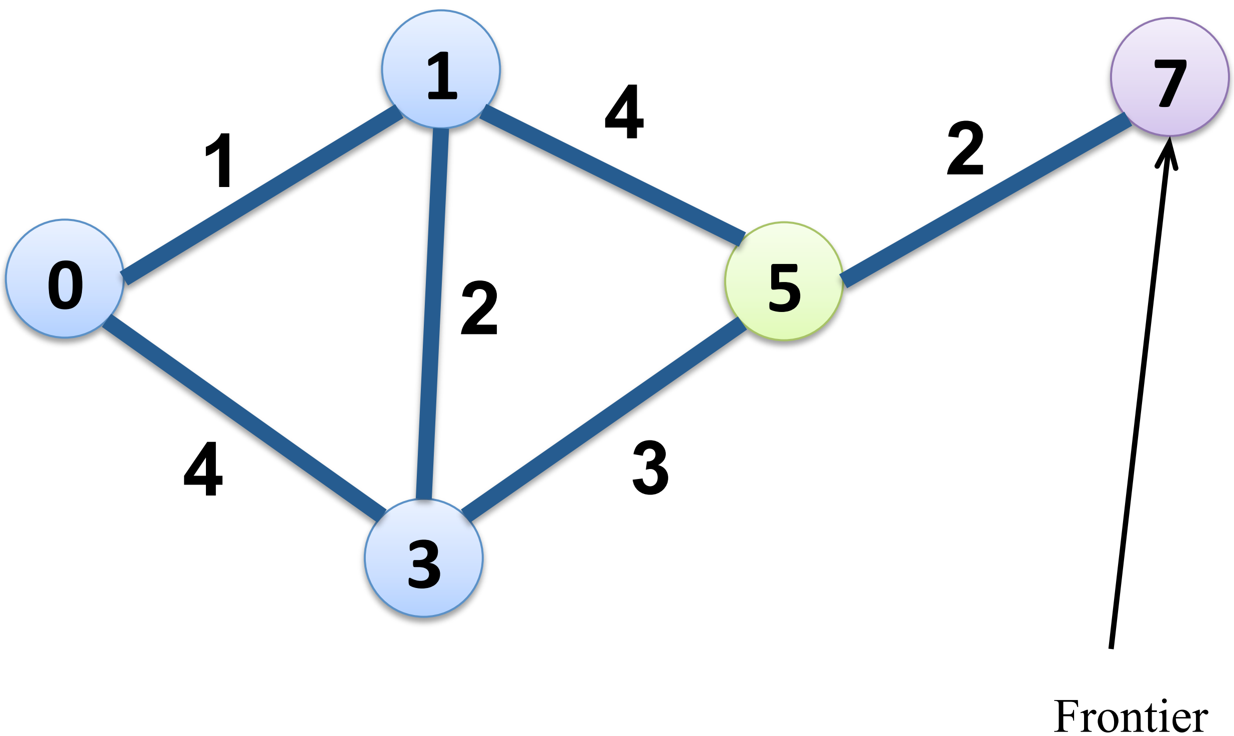
Dijkstra's Algorithm at work



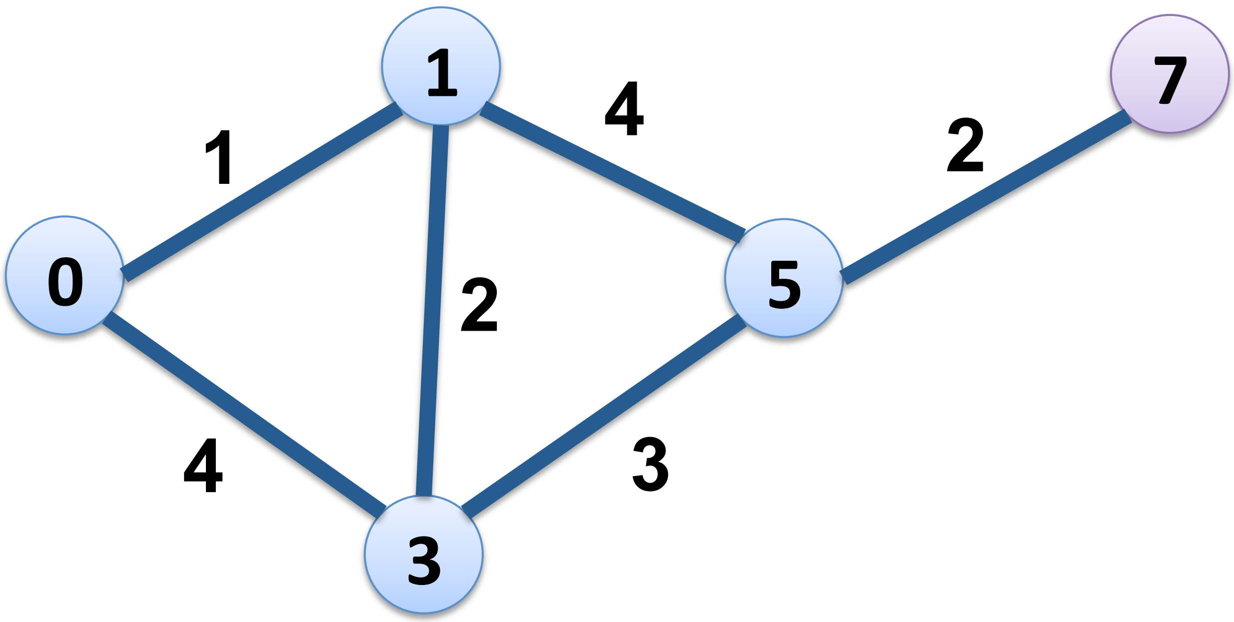
Dijkstra's Algorithm at work



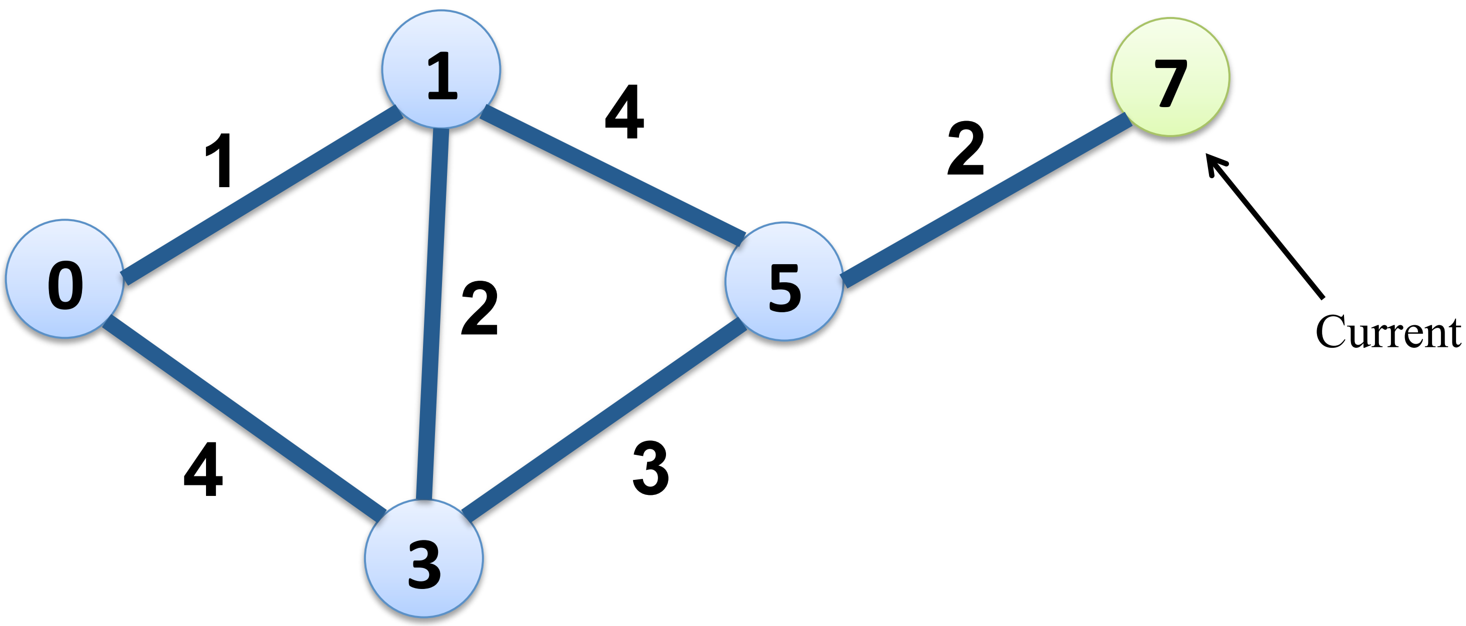
Dijkstra's Algorithm at work



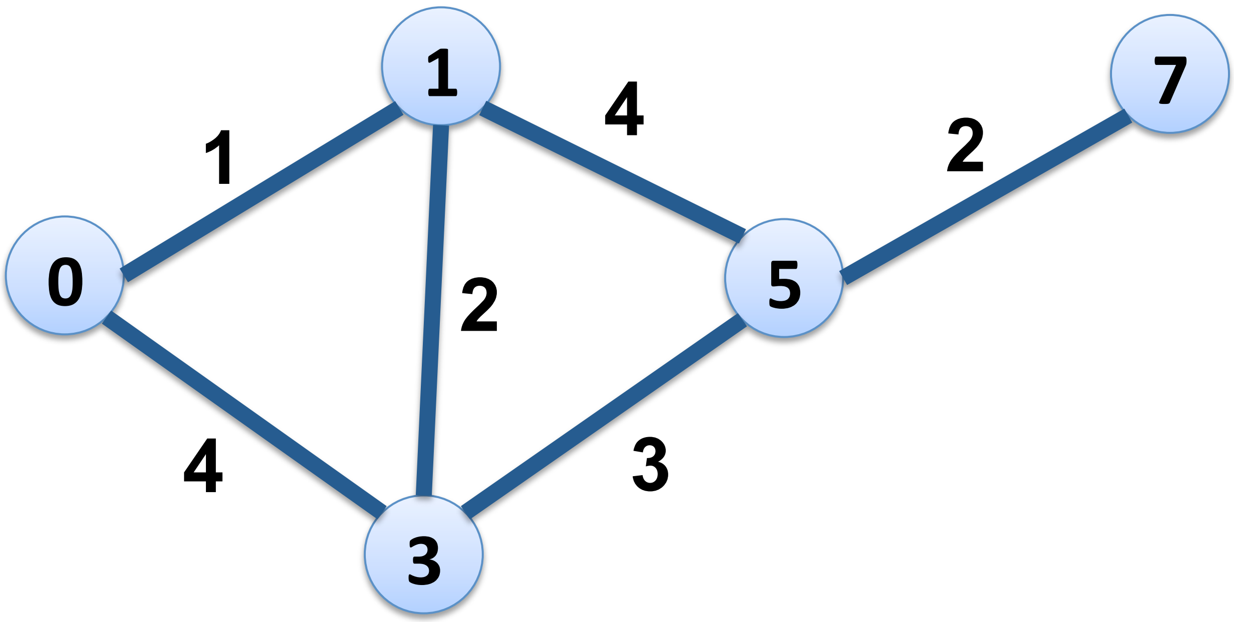
Dijkstra's Algorithm at work



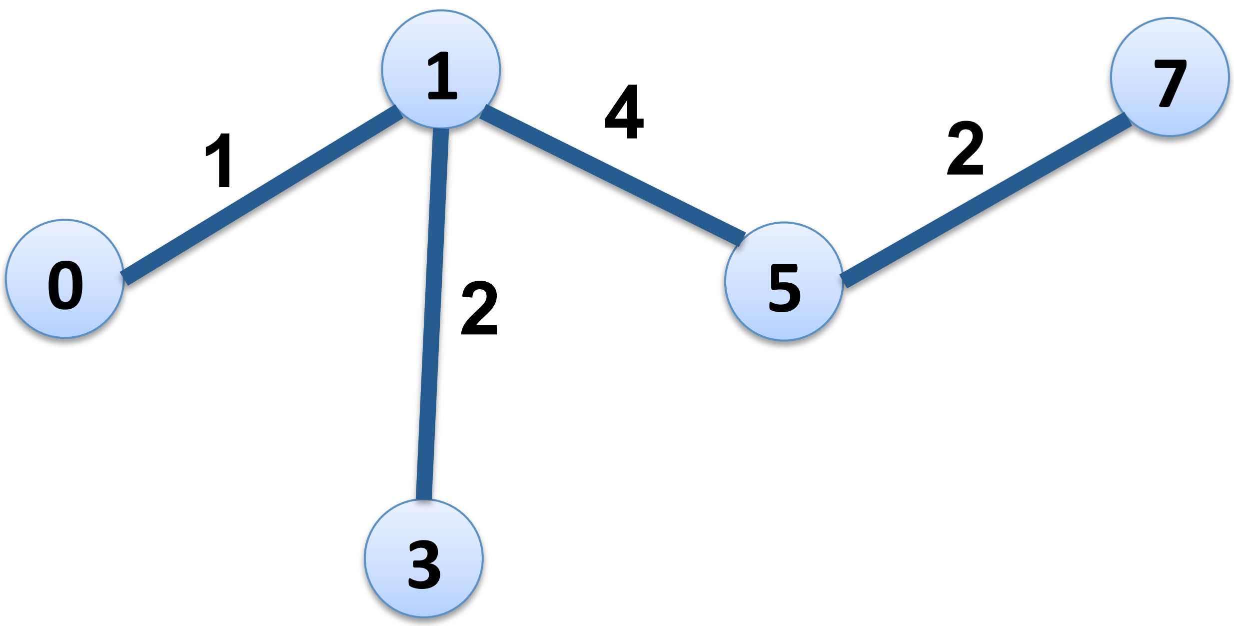
Dijkstra’s Algorithm at work



Dijkstra's Algorithm at work



Which paths are used?



What does Dijkstra compute?

- Shortest path
  - Between two nodes
- A shortest rooted tree
  - Between the root (initial) node and all others
  - I.e., N-1 routes between root:node pairs
  - There might be other trees with same cost

Dijkstra: pros and cons

- Pros
  - Simple to implement
    - Broadcast to everyone
    - Everyone runs the same algorithm
- Cons
  - Requires broadcast flooding
  - Not everyone might compute the same tree
  - Everyone has to compute the full path everywhere

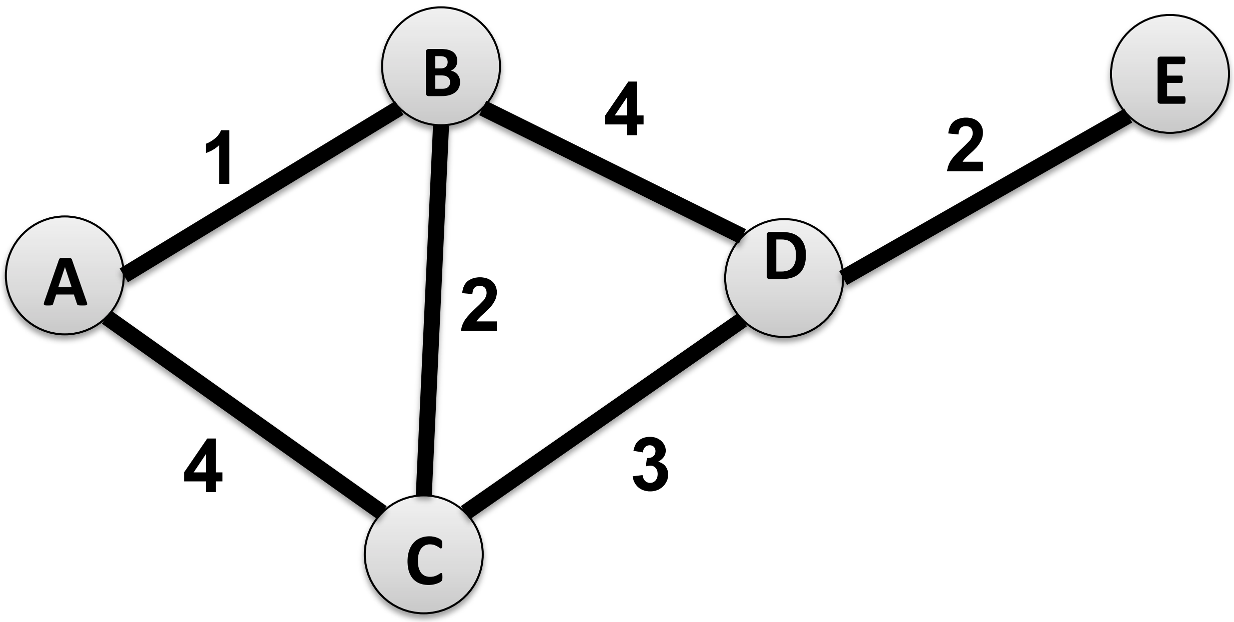
Distance vector

- Not always flooding
- Bellman-Ford algorithm
  - Shortest path
- Ford-Fulkerson
  - Max-flow
- DUAL
  - Current popular variant
- We won't look at Ford-Fulkerson or DUAL in detail

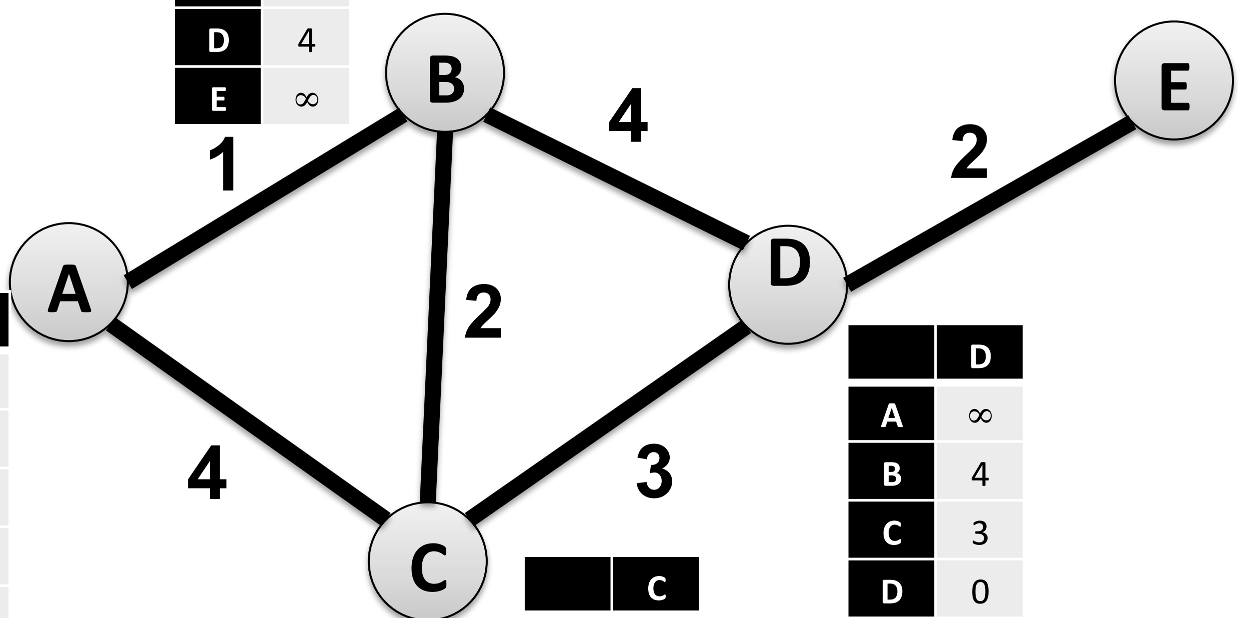
Basic distance vector algorithm

- Routing by sending only useful info
  - Tell neighbors who you can reach and cost
  - Everyone updates their table by transitive closure rules
- Effect
  - Walking the nodes while calculating Dijkstra
  - Still floods – just not everything

Example of Bellman-Ford



Example of Bellman-Ford



	A
A	0
B	1
C	4
D	∞
E	∞

	B
A	1
B	0
C	2
D	4
E	∞

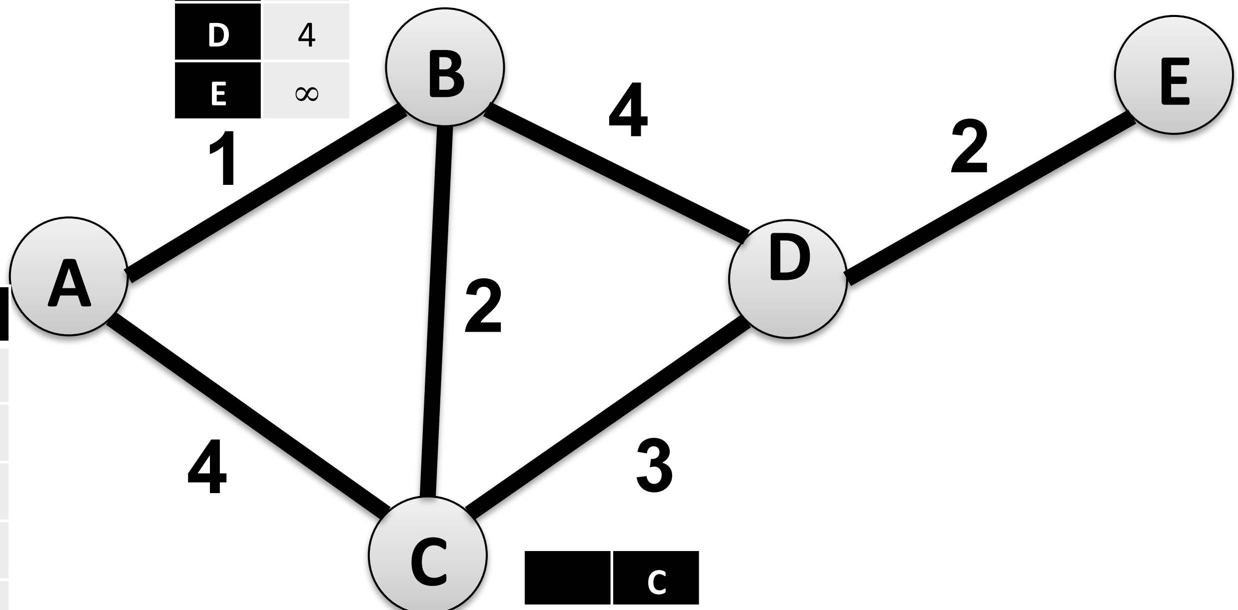
	C
A	4
B	2
C	0
D	3
E	∞

	D
A	∞
B	4
C	3
D	0
E	2

	E
A	∞
B	∞
C	∞
D	2
E	0

Example of Bellman-Ford

	B
A	1
B	0
C	2
D	4
E	∞



	A
A	0
B	1
C	4
D	∞
E	∞

	C
A	4
B	2
C	0
D	3
E	∞

A look at A

- A looks at the tables it has received

	B
A	1
B	0
C	2
D	4
E	∞

	C
A	4
B	2
C	0
D	3
E	∞

A look at A

- A looks at tables it has received

	B
A	1
B	0
C	2
D	4
E	∞

	C
A	4
B	2
C	0
D	3
E	∞

- Updates them with the cost to get to there

	A
A	0
B	1
C	4
D	∞
E	∞

→

	B+1
A	1
B	0
C	2
D	4
E	∞

	C+4
A	4
B	2
C	0
D	3
E	∞

A look at A

- A looks at tables it has received

	B
A	1
B	0
C	2
D	4
E	∞

	C
A	4
B	2
C	0
D	3
E	∞

- Updates them with the cost to get to there

	A
A	0
B	1
C	4
D	∞
E	∞

	B+1
A	2
B	1
C	3
D	5
E	∞

	C+4
A	8
B	6
C	4
D	7
E	∞

A look at A

- A looks at tables it has received

	B
A	1
B	0
C	2
D	4
E	∞

	C
A	4
B	2
C	0
D	3
E	∞

- Updates its own table with the row min

	A
A	0
B	1
C	3
D	5
E	∞

	B+1
A	2
B	1
C	3
D	5
E	∞

	C+4
A	8
B	6
C	4
D	7
E	∞

Bellman-Ford

- Converges over time
  - Keep exchanging tables and updating them
- Each step
  - Faster – O(N), not O(E)
  - Less state – O(N), not O(E)
  - Works while it's running

Bellman-Ford

- Pros
  - Fewer and smaller messages
  - Send only changes, stops flood when changes stop
  - Keeps less state per node
  - Fast convergence when link improves/comes up
- Cons
  - Decentralized (benign errors or malicious attacks)
  - Slow convergence on link failure

Link state vs. distance vector

- Link state
  - Sees the entire graph
  - Reacts fast to changes
  - *Provides complete path*
- But…
  - Always floods
  - Large local table
  - O(N^2) computation
- Distance vector
  - Floods only where changes affect route
  - Smaller table
  - O(N) computation
  - Reacts faster to some changes
  - *Provides next-hop*
- But…
  - No global view, so no global optimization

Other algorithms

- Hierarchical routing
  - Use structure in the name
  - See the DNS
- Geographic routing
  - See phone calls

Hierarchical

- Go up when you don't know
  - Go towards the root
- Go down based on what you know
  - If target is a leaf on a subtree, go to that subtree

This describes a lot of Internet routing (except that the root is a graph)

Geographic

- Requires
  – Spatial geometry (line, ring, plane, etc.)
  – Node locations
- Use geometry to get you there
  – Works great when it works
  – Hard to get it to work

Landmark

- Some geographic and hierarchical routing
- Subset of nodes/locations called “landmarks”
  - You must know how to get to landmarks
  - Go towards the landmark closest to your target
  - Once close enough, some other routing will help

Who uses what?

- Link state (Dijkstra)
  - OSPF (runs over IP)
  - IS-IS (runs over its own protocol)
- Distance vector (Bellman-Ford, etc.)
  - RIP (runs over UDP)
  - BGP (runs over TCP) but with complete path!
  - EIGRP (runs over its own protocol)

Issues

- Split horizon
- Loop avoidance
- Cost metrics

Split horizon

- DV algorithms converge slowly
  - But link failure = ∞
  - How long does it take to count to ∞?
- Problem
  - DV doesn't keep track of path, only cost
- Solutions
  - Don't send back info you just got (split horizon)
  - Send back the info as bad (poison reverse)

Loop avoidance

- Prevention
  - Ensure loops are never created
- Correction
  - Check for loops and remove them
- Accommodation
  - Add a hopcount so messages can loop a little without causing a big problem

Cost metrics

- Lowest propagation delay?
  - Not the shortest message delivery time
- Highest available capacity?
  - Not the shortest delivery time either
- Lowest price?
  - I.e., minimize an external cost

How to compose cost

- Various equations
  - Sum
  - Weighted sum
  - Min or max
- Rules for composition?
  - Depend on routing algorithm

Metrics for success

- Algorithm performance
- Backups and then some
- Other details

Algorithm performance

- Time
  - To initial table (can start relaying)
  - To convergence
  - To add new routes
  - To delete dead routes
- Bandwidth
  - Number of messages
  - Size of messages
- Fairness / equality
  - Will everyone have the same result?
- Local costs
  - Computation
  - Storage

Solutions to performance

- Use simple topologies
  - Original Ethernet
  - Token rings
  - Wireless LAN
- Compartmentalize
  - Break graph into regions
    - Route within the regions
    - Route between the regions separately

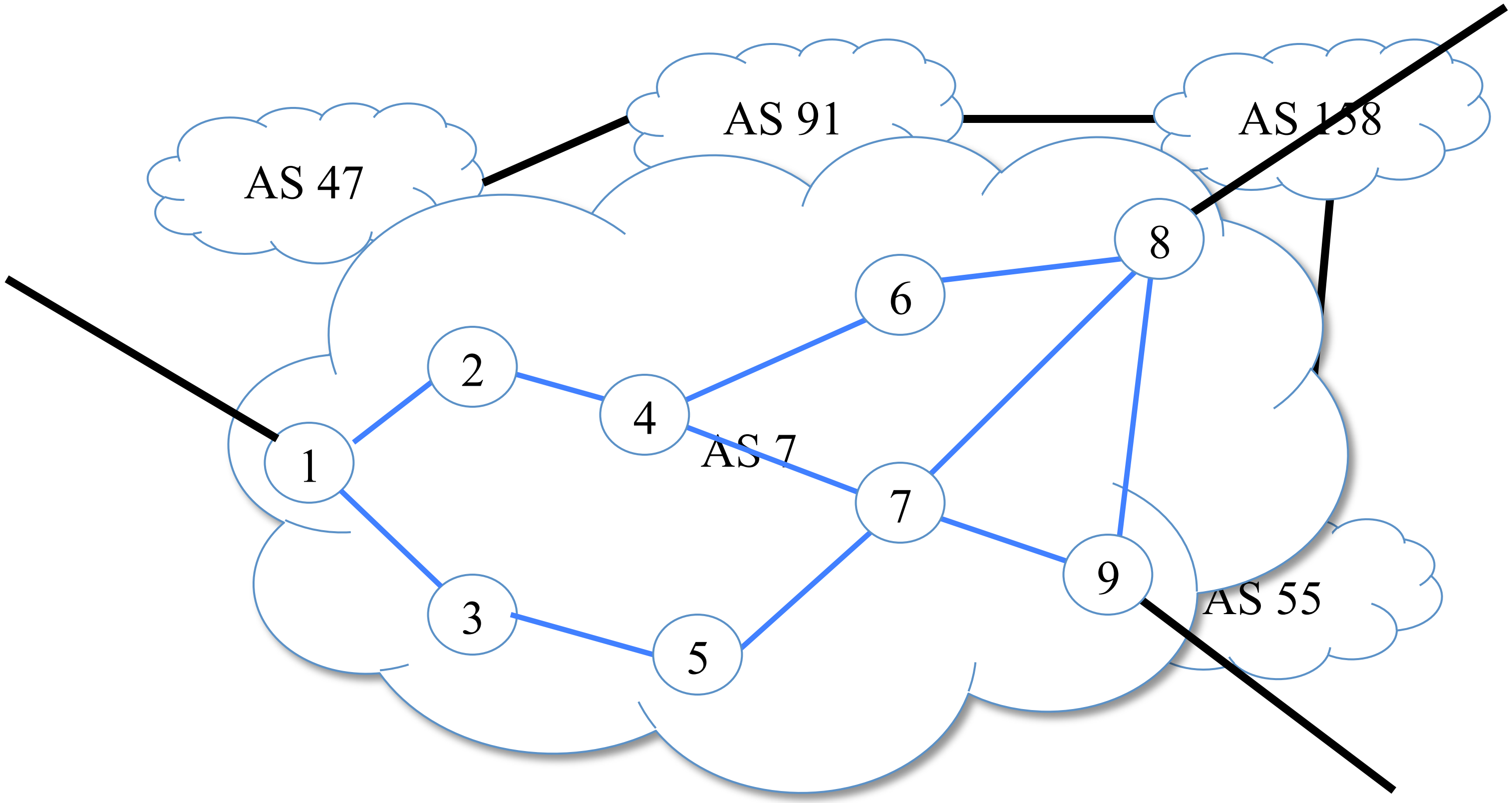
Compartmentalization and Internet routing

- How does the Internet route?
- It breaks the graph up
  - Subgraphs connected at ingress/egress
  - Name each subgraph (“Autonomous system”)
- Route within the subgraph
  - Typically OSPF (link state)
- Route between the subgraphs
  - Typically BGP (distance vector, sort of)

BGP and autonomous systems

- BGP doesn't route between nodes
- It routes at a higher level
  - The autonomous system level
- What is an autonomous system (AS)?
  - A connected subnet controlled by one party
  - E.g., Verizon or AT&T
- An AS contains multiple routers

Graphically,



BGP routes at AS level AS47, AS7, AS55

Each AS routes internally as it pleases 1,2,4,7,9

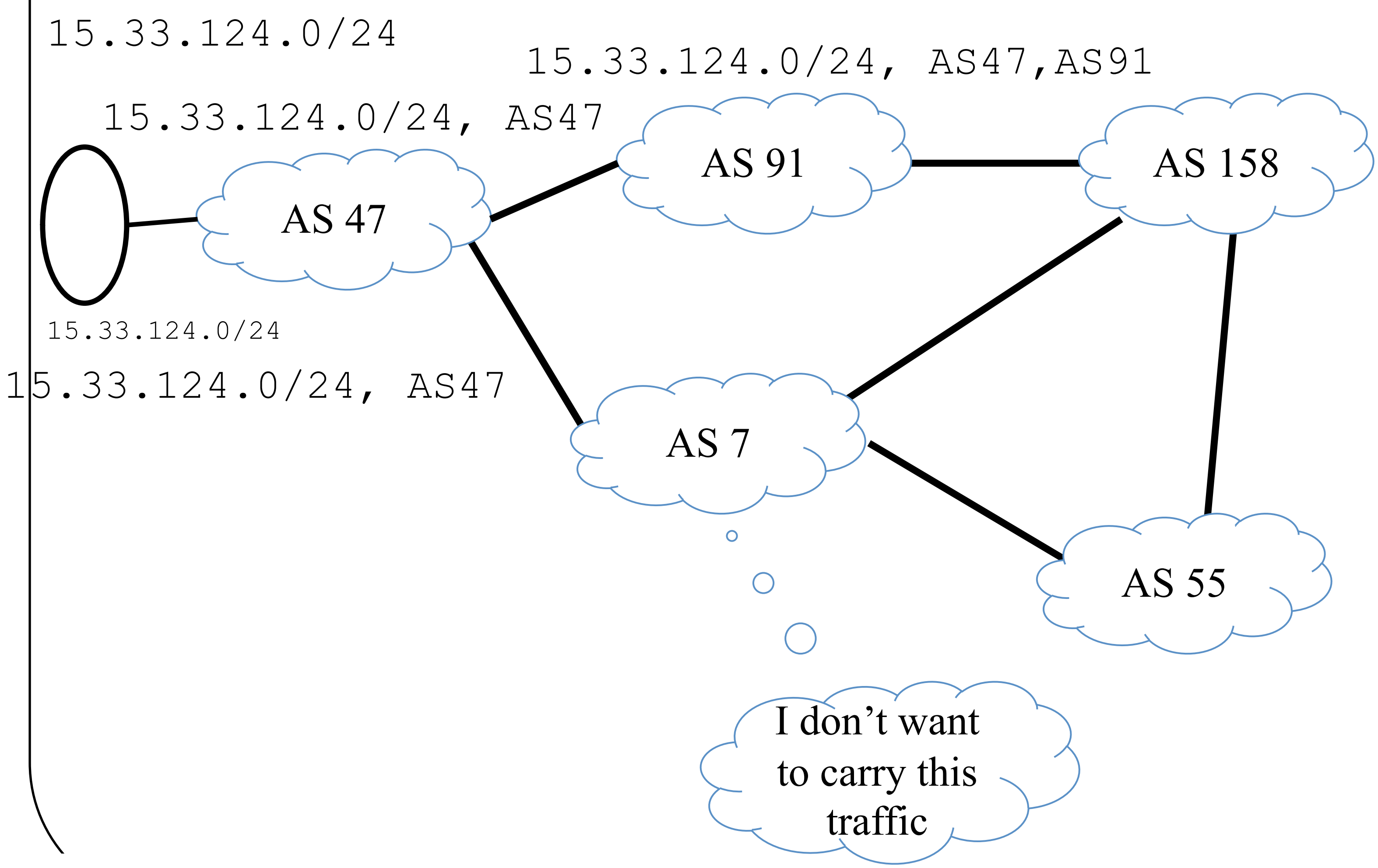
BGP and policy-based routing

- BGP essentially routes at a business-relevant level
- BGP routing decisions are thus made by policy
- Each AS learns of routing options
- The AS uses local policy to choose an option
- Not necessarily shortest or computationally cheapest
  - Perhaps the business partner who gave you the best deal

Building BGP paths

- AS that handles traffic to an IP prefix advertises that fact to neighboring ASes
  - E.g., "I can deliver to 15.33.124.0/24"
- Each neighbor AS remembers that advertisement
- If those neighbors choose, they advertise a route to their neighbors
  - Adding themselves to the path

For example,


15.33.124.0/24
15.33.124.0/24, AS47,AS91
15.33.124.0/24, AS47
AS 91
AS 158
AS 47
15.33.124.0/24
15.33.124.0/24, AS47
AS 7
AS 55
I don't want to carry this traffic

Some BGP implications

- No centralized decisions
  - Either by authority or single algorithm
  - ASes don’t even know all possible choices
- Decisions changeable dynamically
  - At the AS level
- Constraints on routing based not just on physical connectivity
  - Also on business arrangements
- Only a partial description of the routes

Backups and then some

- One route might not be enough
  - "Hot spare" – equivalent backup link ready for immediate use
  - Multipath – for increased capacity
  - Alternate path – to route around a dead link

Summary

- Many ways to route
  - All variations of transitive closure
  - Vary in performance, convergence time, etc.
- Primary alternatives
  - Link state (i.e., central computation)
  - Distance vector (i.e., distributed computation)
- The hardest parts
  - Are the details – how to assign cost, how to compose cost, etc.