CS551
Unicast Routing

Bill Cheng

http://merlot.usc.edu/cs551-f12

Forwarding V.S. Routing

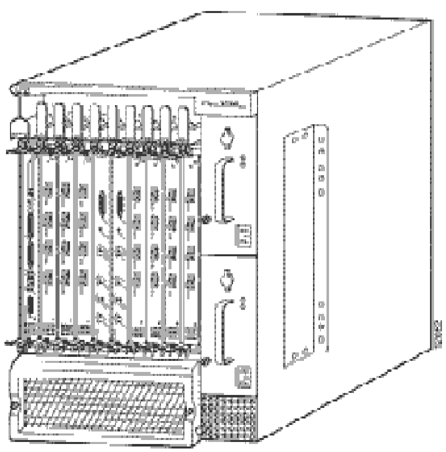
- ***Forwarding:*** **the process of moving packets from input to output based on:**
  - **the forwarding table**
  - **information in the packet**
- ***Routing:*** **process by which the forwarding table is built and maintained:**
  - **one or more routing protocols**
  - **procedures (algorithms) to convert routing info to forwarding table**

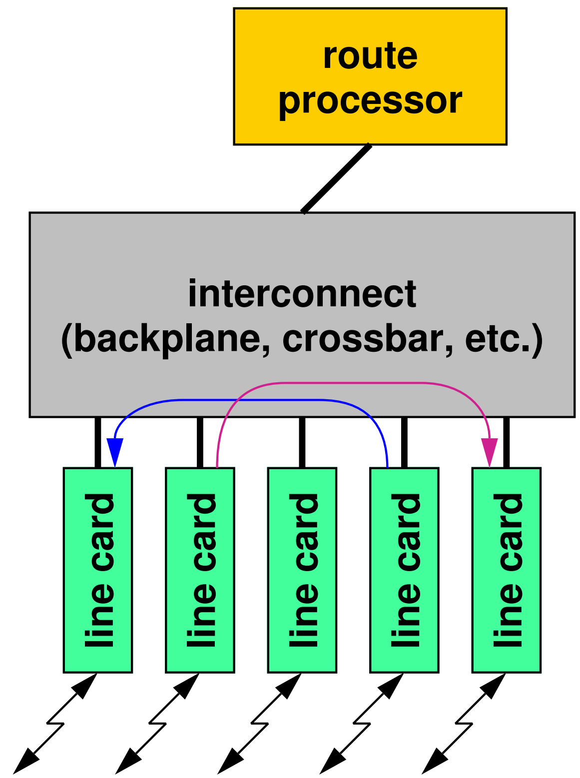
Forwarding Examples

- To forward unicast packets a router uses:
  - destination IP address
  - longest matching prefix in forwarding table
- To forward multicast packets:
  - source + destination IP address and incoming interface
  - longest and exact match algorithms

A Router And Its Components

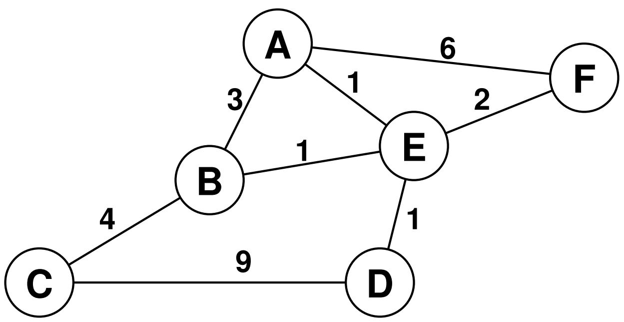
Cisco 7xxx Router





Factors Affecting Routing

- Routing algorithms view the network as a graph
- Problem: find lowest cost path between two nodes
- Factors
  - static: topology
  - dynamic: load
  - policy



Two Main Approaches

- **DV: Distance-vector protocols**
  - **you tell your neighbors what you know about everyone**
- **LS: Link state protocols**
  - **you tell everyone about your neighbors**

Distance Vector Protocols

- Employed in the early Arpanet
- Distributed next hop computation
  - adaptive
- Asynchronous, iterative
- Unit of information exchange
  - vector of distances to destinations
- Distributed Bellman-Ford Algorithm

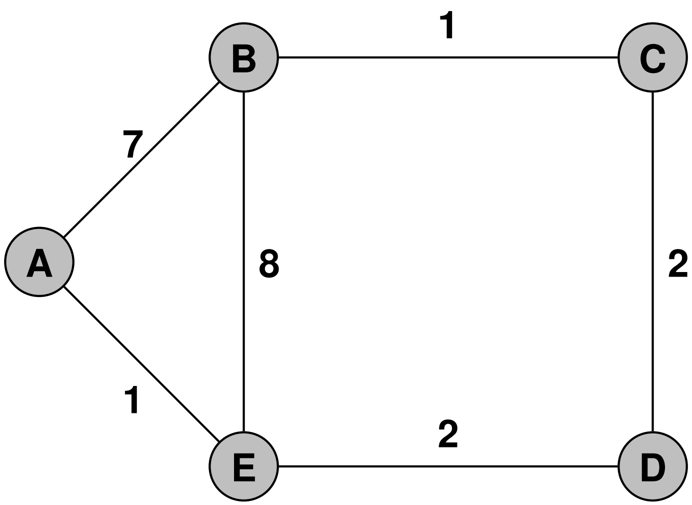
Distance Vector

- $D^X(Y,Z)$: distance to Y via Z in node X's distance table (Z is X's direct neighbor)
- $c(X,Z)$: cost from X to X's direct neighbor Z
- $D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\}$
  where w is a direct neighbor of Z

Distributed Bellman-Ford

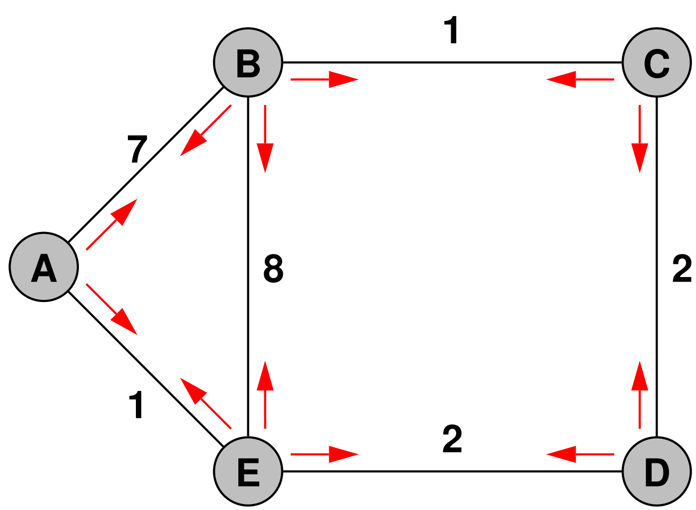
- **Start Conditions:**
  - each router starts with a vector of distances to all directly attached networks
- **Send step:**
  - each router advertises its current vector to all neighboring routers
- **Receive step:**
  - upon receiving vectors from each of its neighbors, router computes its own *distance* to each neighbor
  - then, for every network X, router finds that neighbor who is closer to X than to any other neighbor
  - router updates its cost to X. After doing this for all X, router goes to send step if routing information has changed

Example - Initial Distances



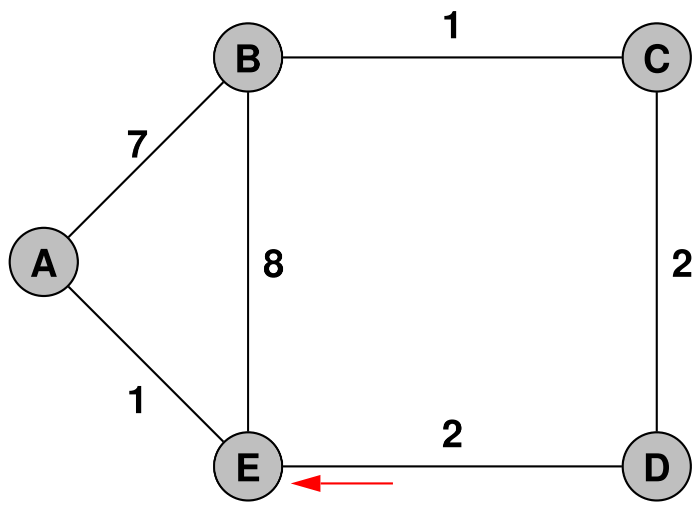
Info at node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

Example - Initial Distances



Info at node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

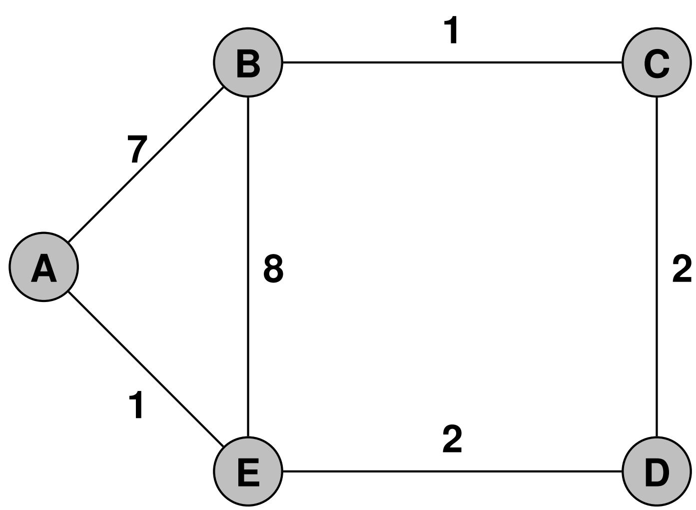
E Receives D's Routes



Info at node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

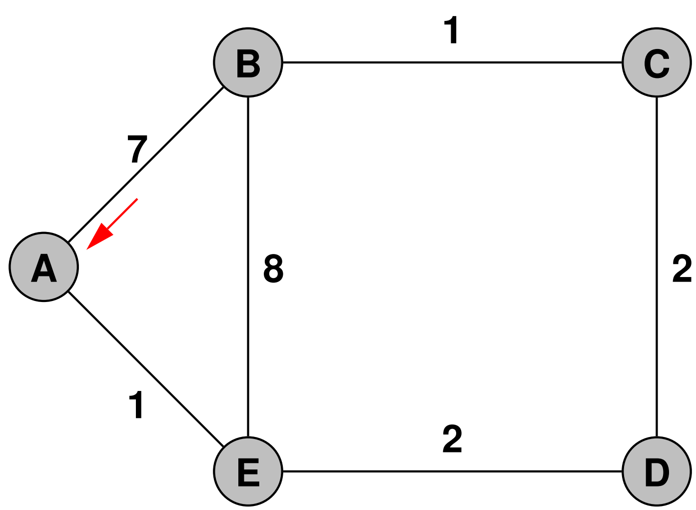
c(D,E) = 2

E Updates Cost to C



Info at node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

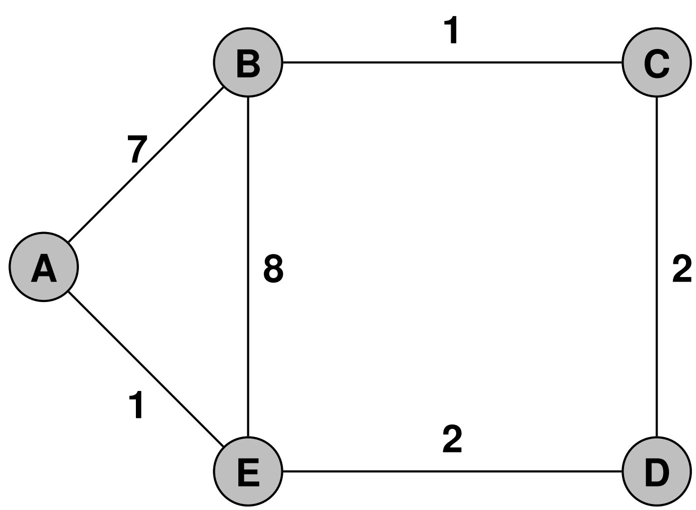
A Receives B's Routes



c(A,B) = 7

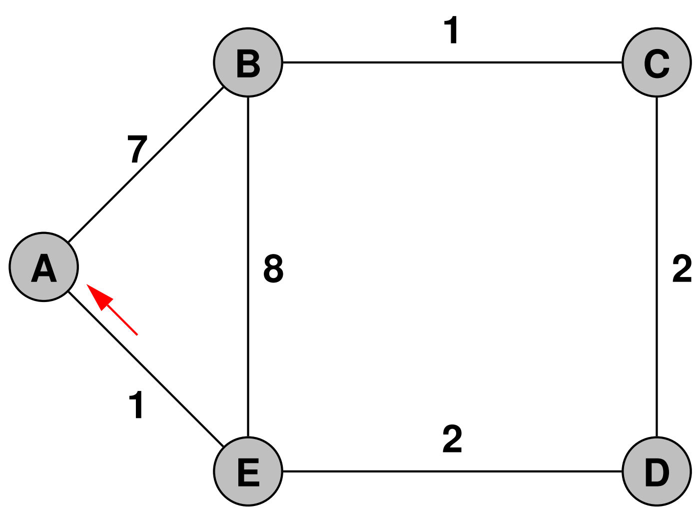
Info at node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A Updates Cost to C



Info at node	A	B	C	D	E
A	0	7	8	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

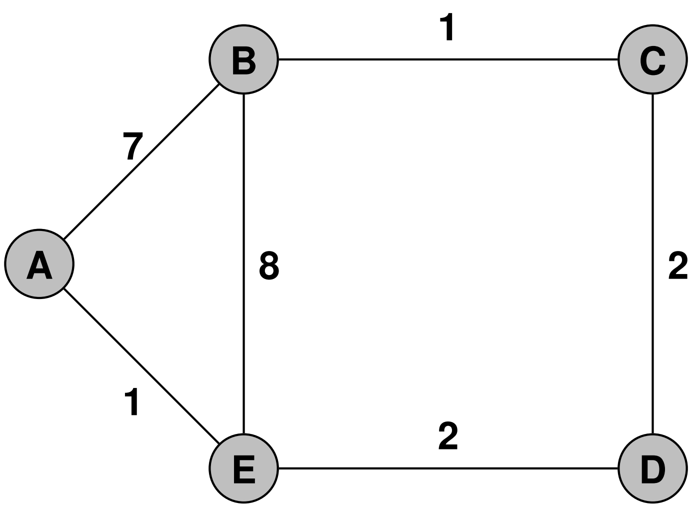
A Receives E's Routes



c(A,E) = 1

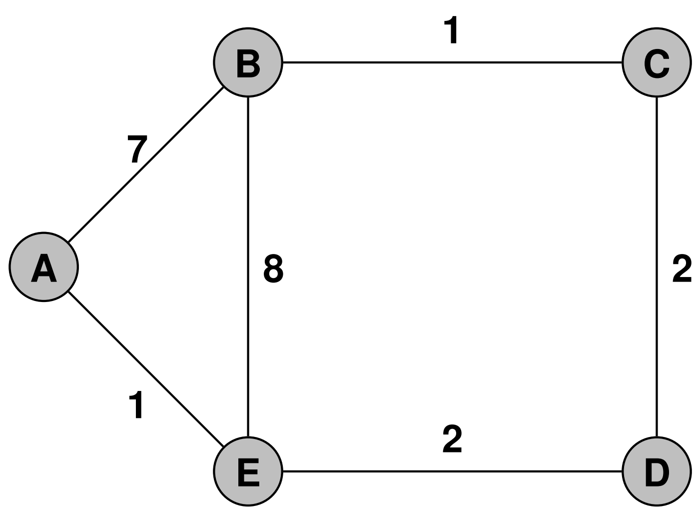
Info at node	A	B	C	D	E
A	0	7	8	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A Updates Cost to C & D



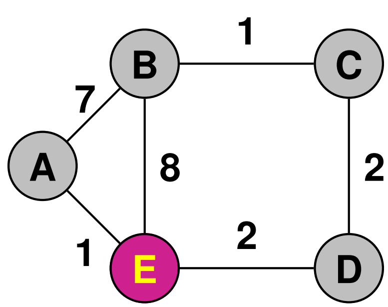
Info at node	A	B	C	D	E
A	0	7	5	3	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

Final Distances



Info at node	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

View From a Node

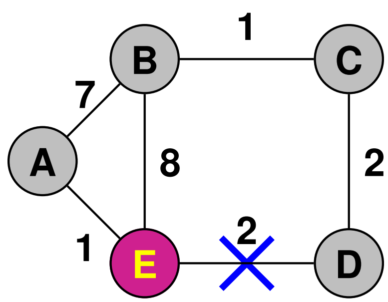


Info at node	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

E's routing table:

	Next hop		
dest	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Final Distances After Link Failure

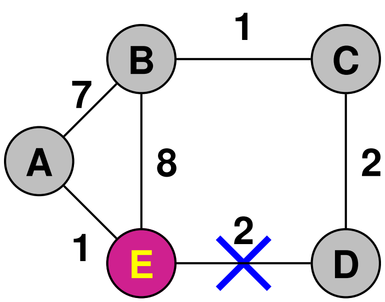


Info at node	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

E's routing table:

dest	Next hop A	Next hop B	Next hop D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Final Distances After Link Failure (Cont...)

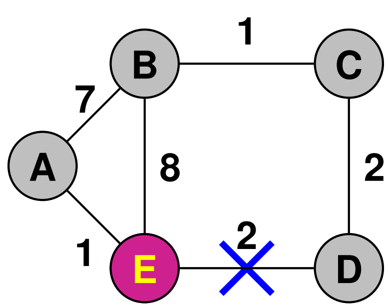


Info at node	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	~~2~~	0

E's routing table:

	Next hop		
dest	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Final Distances After Link Failure (Cont...)



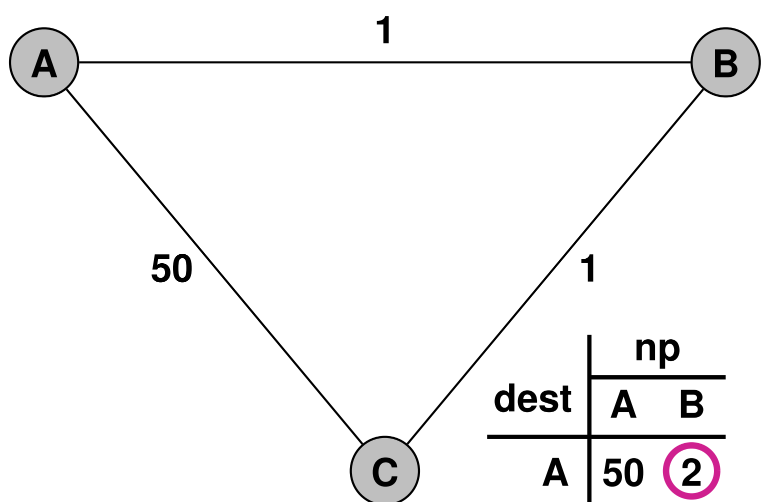
Info at node	A	B	C	D	E
A	0	7	8	10	1
B	7	0	1	3	8
C	8	1	0	2	9
D	10	3	2	0	11
E	1	8	9	11	0

E's routing table:

dest	Next hop A	Next hop B	Next hop ~~D~~
A	1	15	~~5~~
B	8	8	~~5~~
C	9	9	~~4~~
D	11	11	~~2~~

The Bouncing Effect (a.k.a. Count to Infinity Problem)

dest	np: B	np: C
B	1	51
C	2	50

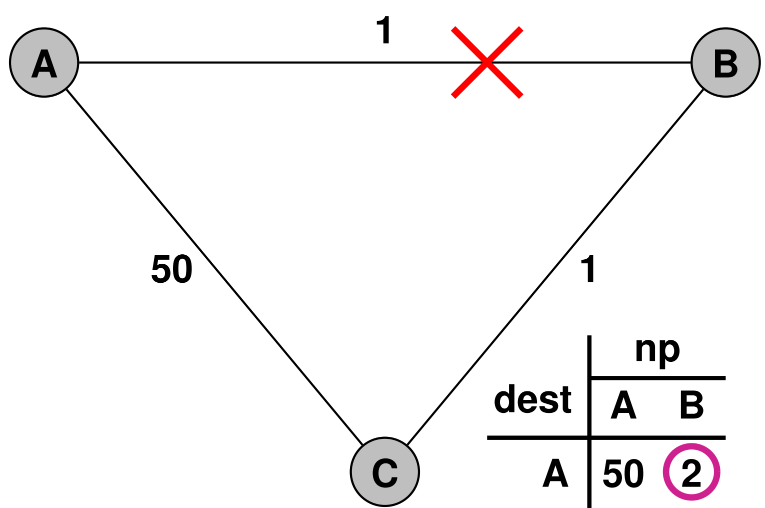


dest	np: A	np: C
A	1	3
C	3	1

dest	np: A	np: B
A	50	2
B	51	1

The Bouncing Effect (a.k.a. Count to Infinity Problem)

dest	np: B	np: C
B	1	51
C	2	50

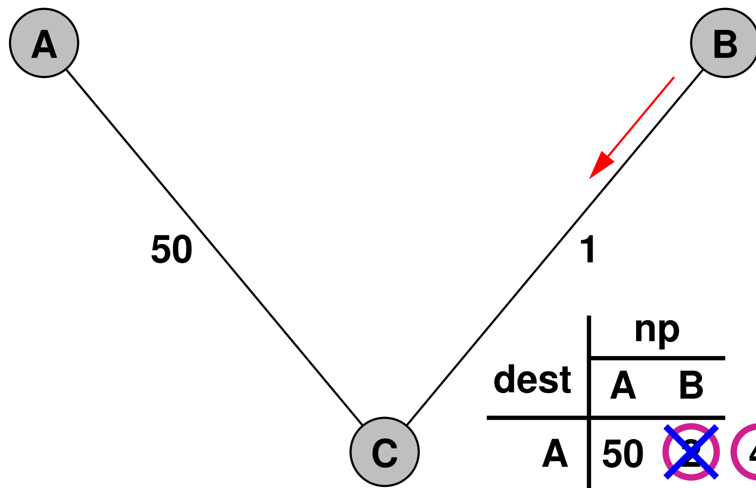


dest	np: A	np: C
A	1	3
C	3	1

dest	np: A	np: B
A	50	2
B	51	1

B Sends Routes to C
C Updates Distance to A

dest	np: B	np: C
B	1	51
C	2	50

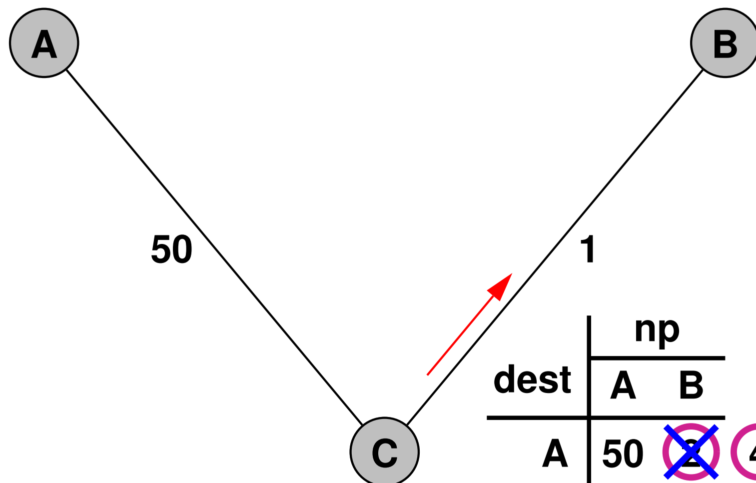


dest	np: A	np: C
A	1	3
C	3	1

dest	np: A	np: B	
A	50	~~2~~	4
B	51	1	

C Sends Routes to B
B Updates Distance to A

dest	np B	np C
B	1	51
C	2	50



dest	np A	np C
A	1	~~8~~ 5
C	3	1

dest	np A	np B
A	50	~~2~~ 4
B	51	1

B Sends Routes to C
C Updates Distance to A

Table at A:

dest	np: B	np: C
B	1	51
C	2	50

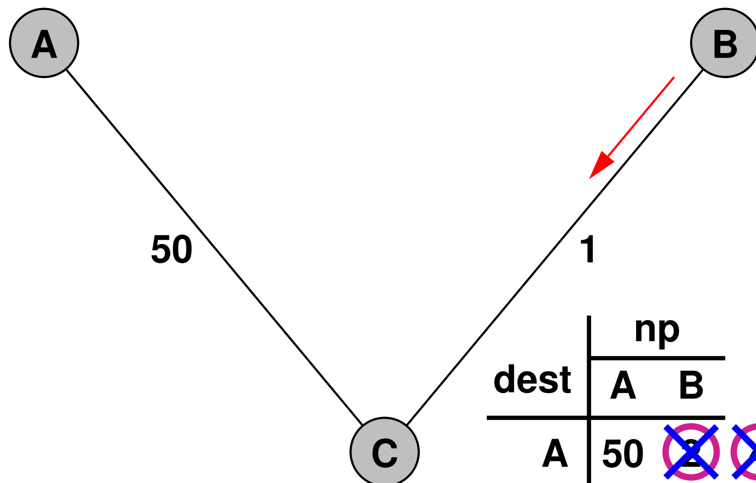


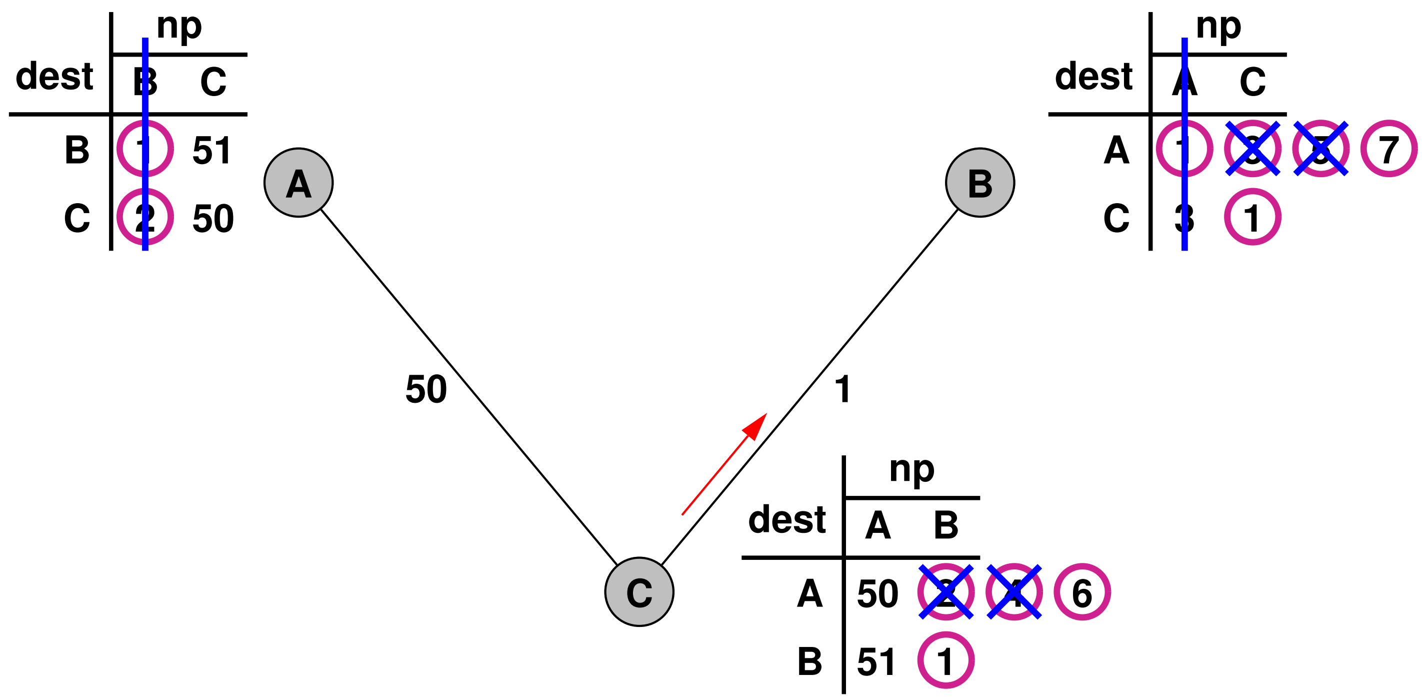
Table at B:

dest	np: A	np: C
A	1	~~8~~ 5
C	3	1

Table at C:

dest	np: A	np: B
A	50	~~2~~ ~~4~~ 6
B	51	1

C Sends Routes to B
B Updates Distance to A



This is known as the *count to infinity* problem

How Are These Loops Caused?

- **Observation 1:**
  - B's metric *increases*
- **Observation 2:**
  - C picks B as next hop to A
  - But, the *implicit path* from C to A includes itself!

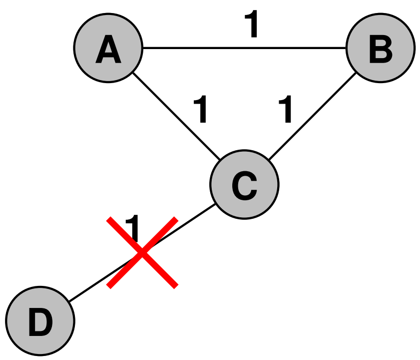
Solution 1: Holddowns

- If metric increases, delay propagating information
  - in our example, B delays advertising route
  - C eventually thinks B’s route is gone, picks its own route
  - B then selects C as next hop
- Adversely affects convergence

Other "Solutions"

- Split horizon
  - B does not advertise route to C
- Poisoned reverse
  - B advertises route to C with infinite distance
    - works for two node loops
    - does not work for loops with more nodes

Example Where Split Horizon Fails



- When link breaks, C marks D as unreachable and reports that to A and B.
- Suppose A learns it first. A now thinks best path to D is through B. A reports D unreachable to B and a route of cost=3 to C.
- C thinks D is reachable through A at cost 4 and reports that to B.
- B reports a cost 5 to A who reports new cost to C.
- etc...

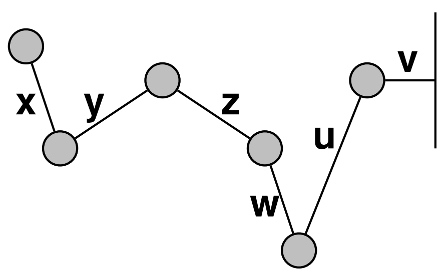
Avoiding the Bouncing Effect

- Select loop-free *paths*
  - One way of doing this:
    - each route advertisement carries entire path
    - if a router sees itself in path, it rejects the route
- BGP does it this way
- Space proportional to diameter

[Cheng, Riley et al]

Computing Implicit Paths

- To reduce the space requirements
  - propagate for each destination not only the cost but also its predecessor
  - can recursively compute the path
  - space requirements independent of diameter



v	u
u	w
w	z
z	y
y	x

Loop Freedom at Every Instant

- Does bouncing effect avoid loops?
  - No! *Transient* loops are still possible
  - Why? Because implicit path information may be stale
- Only way to fix this
  - ensure that you have up-to-date information by explicitly querying

Distance Vector in Practice

- RIP and RIP2
  - uses split-horizon/poison reverse
- BGP/IDRP
  - propagates entire path
  - path also used for effecting policies

Link State Algorithms

Basic Steps

- Each node assumed to know state of links to its neighbors
  - *Step 1:* Each node broadcasts its state to all other nodes
  - *Step 2:* Each node locally computes shortest paths to all other nodes from global state

Building Blocks

- Reliable broadcast mechanism
  - flooding
  - sequence number issues
- Shortest path tree (SPT) algorithm
  - Dijkstra's SPT algorithm

Link State Packets (LSPs)

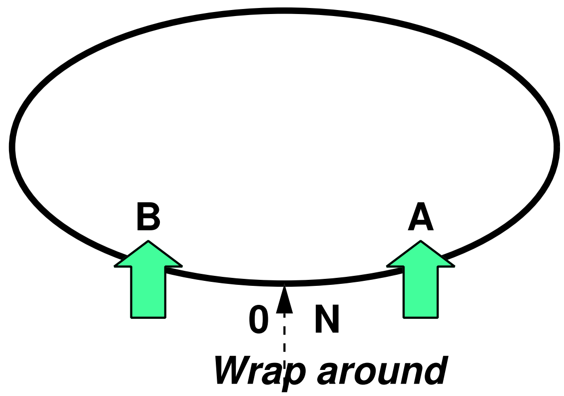
- Periodically, each node creates a Link state packet containing:
  - Node ID
  - List of neighbors and link cost
  - Sequence number
  - Time to live (TTL)
- Node outputs LSP on *all* its links

Reliable Flooding

- When node i receives LSP from node j:
  - If LSP is the most recent LSP from j that i has seen so far, i saves it in database and forwards a copy on all links except link LSP was received on.
  - Otherwise, discard LSP.

Sequence Number Space Issues

- Problem: sequence number may wrap around
- Solution: treat space as circular, continue after wrap around:
  - A is less than B if
    - A<B and B-A < N/2, or
    - A>B and A-B > N/2



Problem: Router Failure

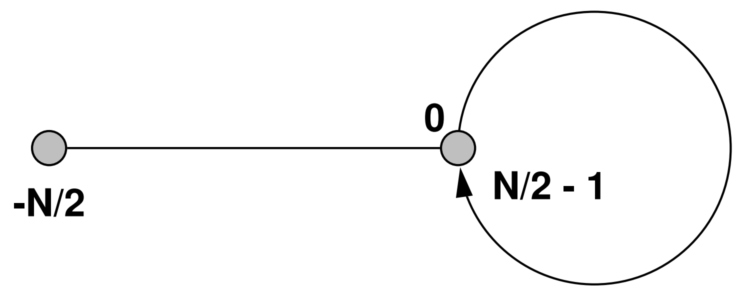
- A failed router and comes up but does not remember the last sequence number it used before it crashed
- New LSPs may be ignored if they have lower sequence number

One Solution: LSP Aging

- Nodes periodically decrement age (TTL) of stored LSPs
- LSPs expire when TTL reaches 0
  - LSP is re-flooded once TTL = 0
    - (haven't heard from you for a while, how are you doing? by the way, this is our last conversation)
- Rebooted router waits until all LSPs have expired
- Trade-off between frequency of LSPs and router wait after reboot

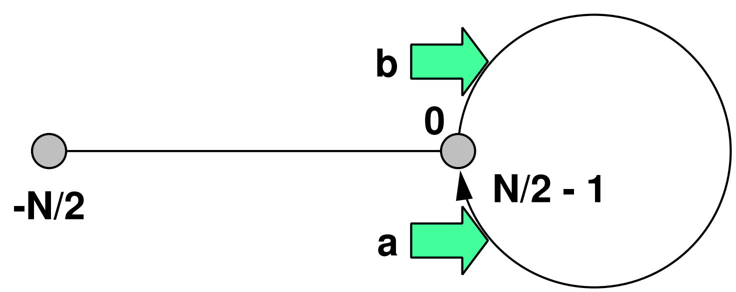
A Better Solution

- Lollipop Sequence space [Perlman83]
  - Divide sequence space N into 3 spaces:
    - Negative space: -N/2 - 0
    - The number 0
    - Positive space: 0 to N/2 -1



Lollipop Operation

- Router comes up and starts with - N/2, then -N/2 + 1, -N/2 + 2, etc.
- When seq number becomes positive, wrap around as before
- a is older than b if:
  - a < 0 and a < b, or
  - a > 0, a < b and b - a < N/4
  - a > 0, b > 0, a > b, and a - b > N/4



Lollipop Operation (Cont...)

- Newly booted router always starts with oldest seq num (-N/2)
- New rule:
  - if router R1 gets older LSP from router R2, R1 informs R2 of the sequence number in R1's LSP
- Newly booted router discovers its seq num before it crashed and resumes

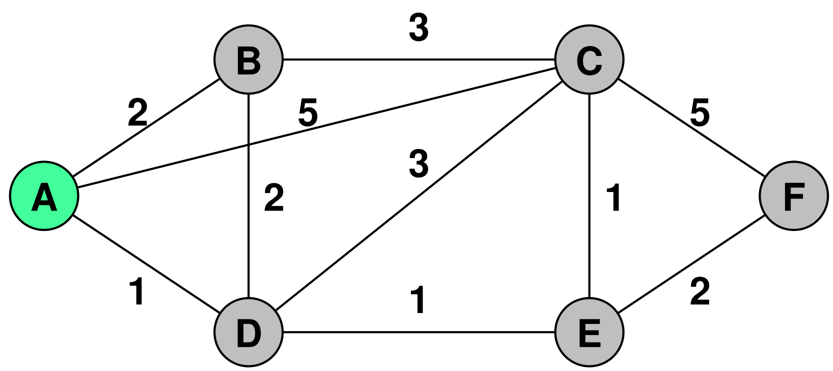
Is Aging Still Needed?

- Yes! Stale LSPs are still possible
  - suppose a router is down but not detected
  - net partitions and then heals
- Aging ensures that old state is eventually flushed out of the network

SPT Algorithm (Dijkstra)

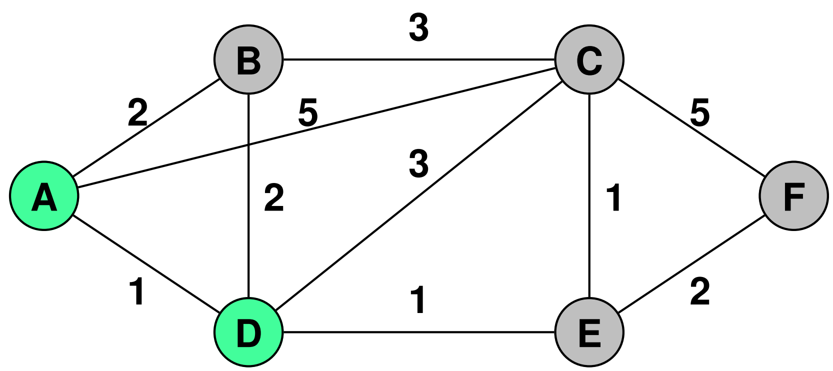
```
SPT = {a}
for all nodes v
    if v adjacent to a then D(v) = cost (a, v)
    else D(v) = infinity
Loop
    find w not in SPT, where D(w) is min
    add w in SPT
    for all v adjacent to w and not in SPT
        D(v) = min (D(v), D(w) + C(w, v))
until all nodes are in SPT
```

Example



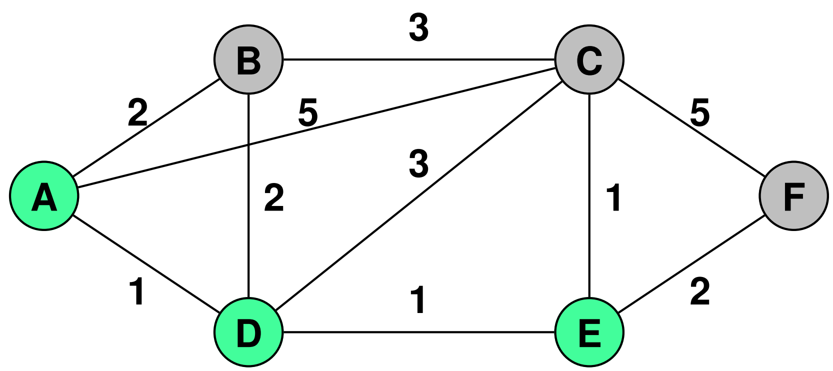
		B	C	D	E	F
step	SPT	D(b),P(b)	D(c),P(c)	D(d),P(d)	D(e),P(e)	D(f),P(f)
0	A	2,A	5,A	1,A	~	~

Example



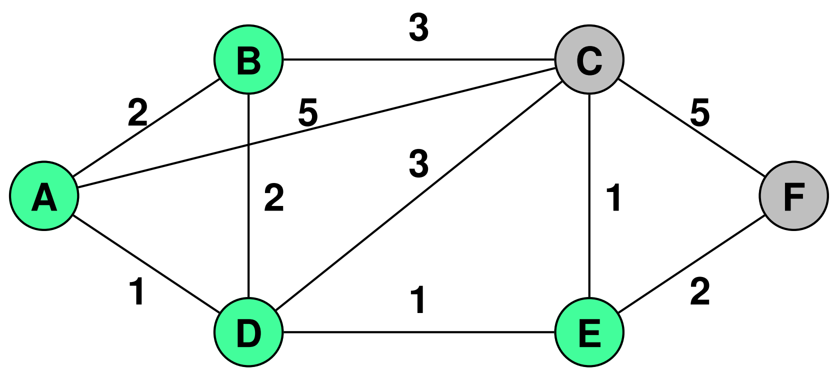
		B	C	D	E	F
step	SPT	D(b),P(b)	D(c),P(c)	D(d),P(d)	D(e),P(e)	D(f),P(f)
0	A	2,A	5,A	1,A	~	~
1	AD	2,A	4,D		2,D	~

Example



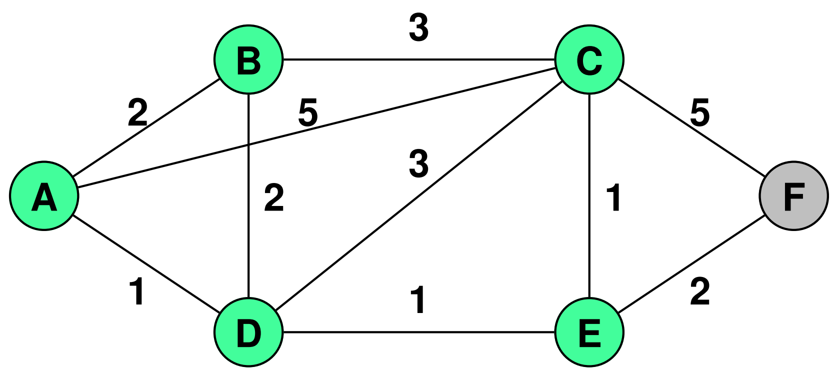
		B	C	D	E	F
step	SPT	D(b),P(b)	D(c),P(c)	D(d),P(d)	D(e),P(e)	D(f),P(f)
0	A	2,A	5,A	1,A	~	~
1	AD	2,A	4,D		2,D	~
2	ADE	2,A	3,E			4,E

Example



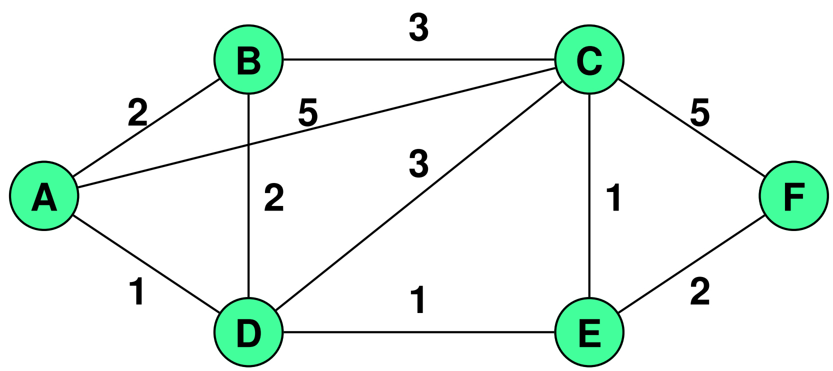
		B	C	D	E	F
step	SPT	D(b),P(b)	D(c),P(c)	D(d),P(d)	D(e),P(e)	D(f),P(f)
0	A	2,A	5,A	1,A	~	~
1	AD	2,A	4,D		2,D	~
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E

Example



		B	C	D	E	F
step	SPT	D(b),P(b)	D(c),P(c)	D(d),P(d)	D(e),P(e)	D(f),P(f)
0	A	2,A	5,A	1,A	~	~
1	AD	2,A	4,D		2,D	~
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E

Example



		B	C	D	E	F
step	SPT	D(b),P(b)	D(c),P(c)	D(d),P(d)	D(e),P(e)	D(f),P(f)
0	A	2,A	5,A	1,A	~	~
1	AD	2,A	4,D		2,D	~
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					

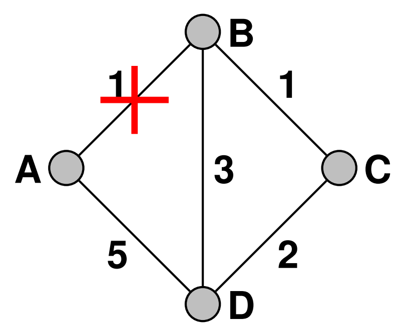
Link State Algorithm

- **Flooding:**
  1) **Periodically distribute link-state advertisement (LSA) to neighbors**
     - **LSA contains delays to each neighbor**
  2) **Install received LSA in LS database**
  3) **Re-distribute LSA to all neighbors**

- **Path Computation**
  1) **Use Dijkstra’s shortest path algorithm to compute distances to all destinations**
  2) **Install *<destination, nexthop>* pair in forwarding table**

Link State Characteristics

- With consistent LSDBs, all nodes compute consistent loop-free paths
- Limited by Dijkstra computation overhead, space requirements
- Can still have transient loops



Packet from C->A may loop around BDC

LS v.s. DV

- In DV send everything you know to your neighbors
- In LS send info about your neighbors to everyone

LS v.s. DV (Cont...)

- Msg size:
  - LS: small
  - DV: potentially large
- Msg exchange:
  - LS: O(nE)
  - DV: only to neighbors
- Convergence speed:
  - LS: fast
  - DV: fast with triggered updates
- Space requirements:
  - LS maintains entire topology
  - DV maintains only neighbor state

LS v.s. DV (Cont...)

- Robustness:
  - LS can broadcast incorrect/corrupted LSP
    - localized problem
  - DV can advertise incorrect paths to all destinations
    - incorrect calculation can spread to entire network
- In LS, nodes must compute consistent routes independently
  - must protect against LSDB corruption
- In DV, routes are computed relative to other nodes

LS v.s. DV (Cont...)

- DV risks:
  - looping, convergence time, corrupted host can get all routes
  - solutions are split horizon, poison reverse, path vectors
- LS risks:
  - flooding of information, must know whole topology (hierarchy and aggregation are forces against this)
- Bottom line: no clear winner, but we see more frequent use of LS in the Internet

What Makes Routing Hard?

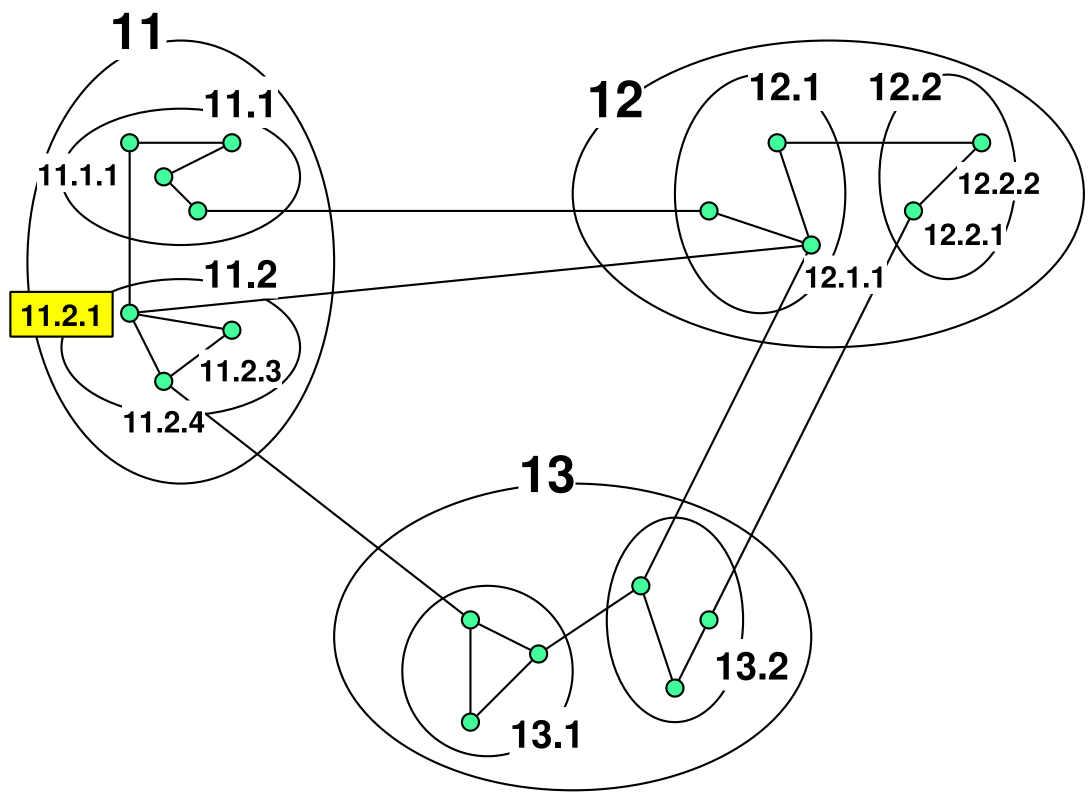
- Scalability to many hosts
- Reliability and robustness
- Dealing with changes
  - some changes (link goes down) should be dealt with ASAP, some (link goes up and down) should be surpressed
- Congestion
  - why not route around congestion?
    - routing algorithm takes too long to react to congestion
- Distributed computation (and debugging)
- Routing and business/policy issues

Scaling to Big Networks

- Internet today is on the order of 1 million networks
- Approaches
  - hierarchy and aggregation
- Key idea: want to know about *you* and not about some networks far away
- Two approaches:
  - area hierarchy
    - approach used in the Internet
    - semi-manual aggregation
  - landmark hierarchy
    - not directly used

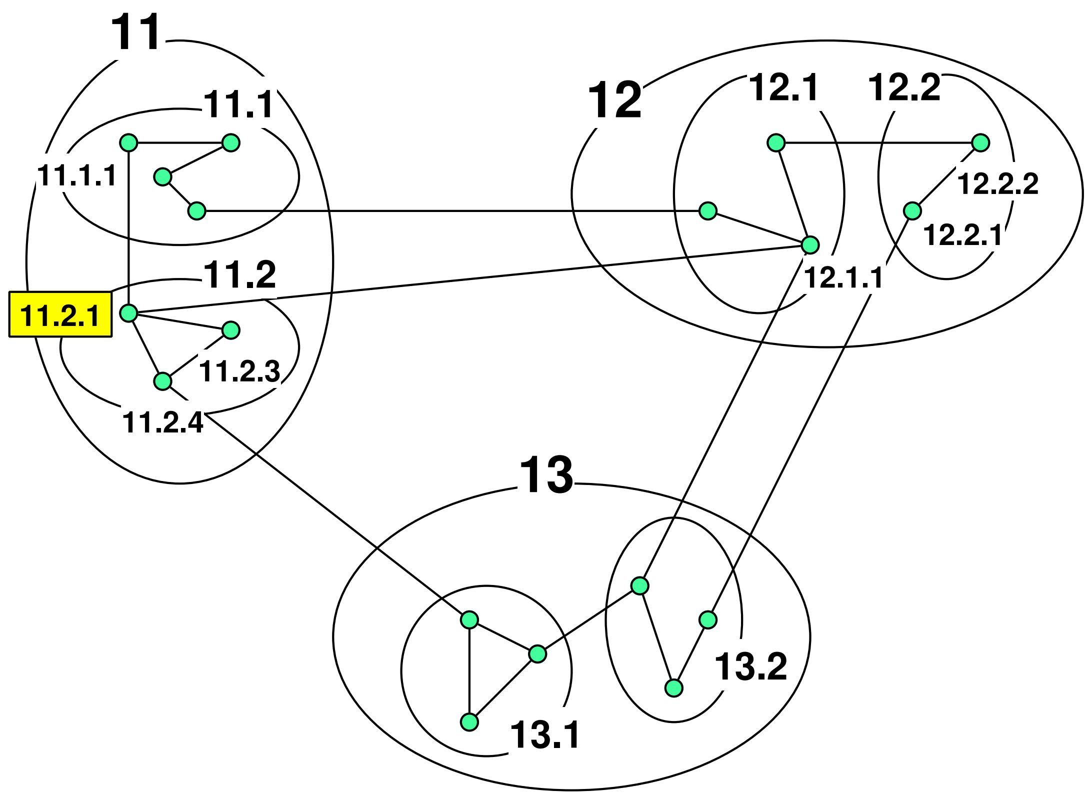
Example Area Hierarchy

- AS's (11, 12, 13)
- sub-AS's (11.1, etc.)
- networks (11.2.1/24, etc.)
- routing table at 11.2.1:
  - 11.2.3/24: ?



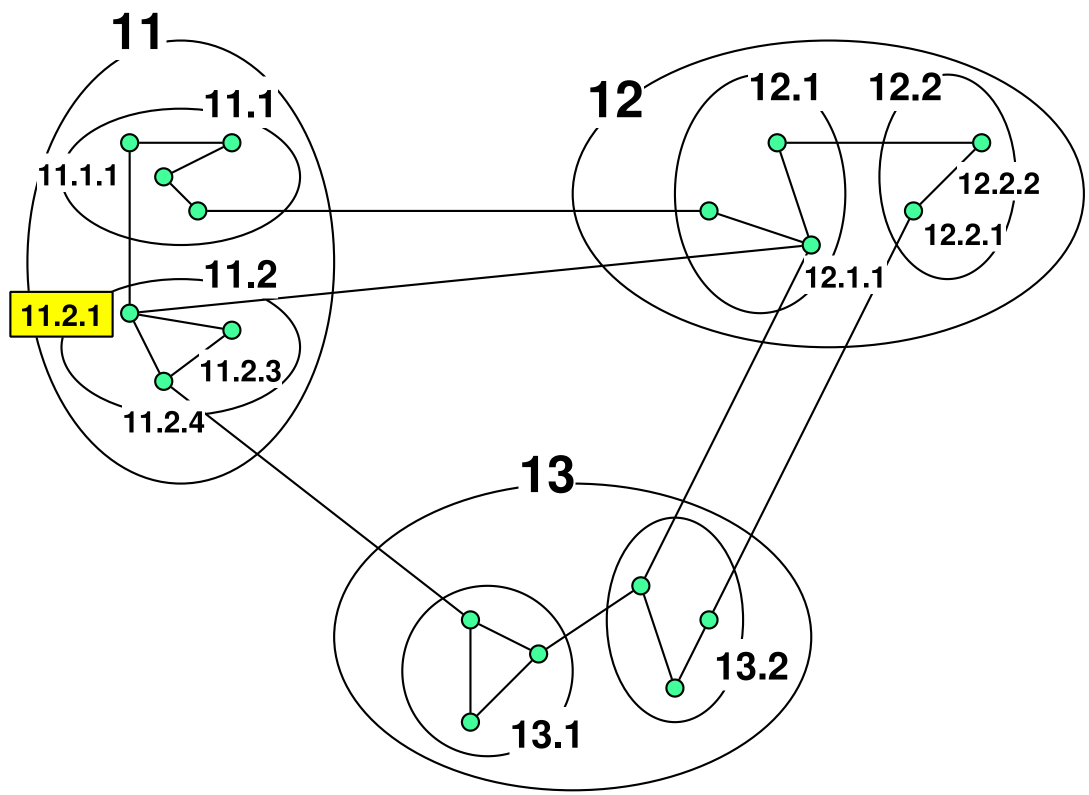
Example Area Hierarchy

- AS's (11, 12, 13)
- sub-AS's (11.1, etc.)
- networks (11.2.1/24, etc.)
- routing table at 11.2.1:
  - 11.2.3/24: 11.2.1.x
  - 11.2.4/24: ?



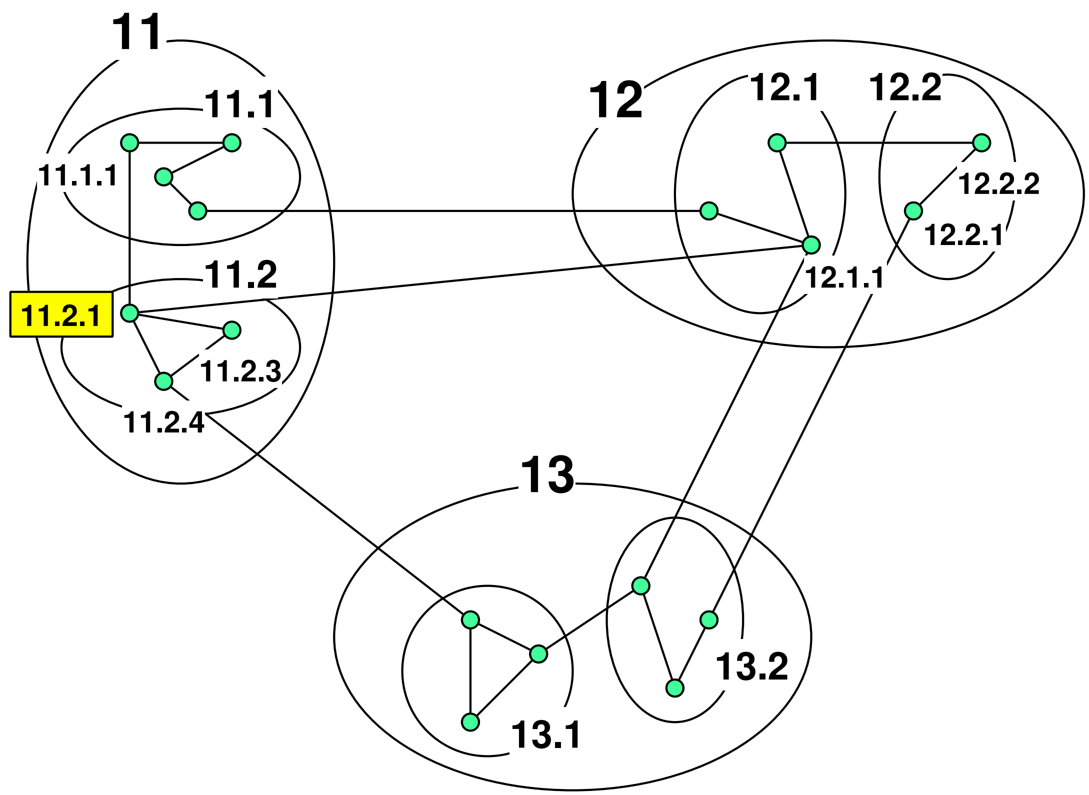
Example Area Hierarchy

- AS's (11, 12, 13)
- sub-AS's (11.1, etc.)
- networks (11.2.1/24, etc.)
- routing table at 11.2.1:
  - 11.2.3/24: 11.2.1.x
  - 11.2.4/24: 11.2.1.y
  - 11.1/16: ?



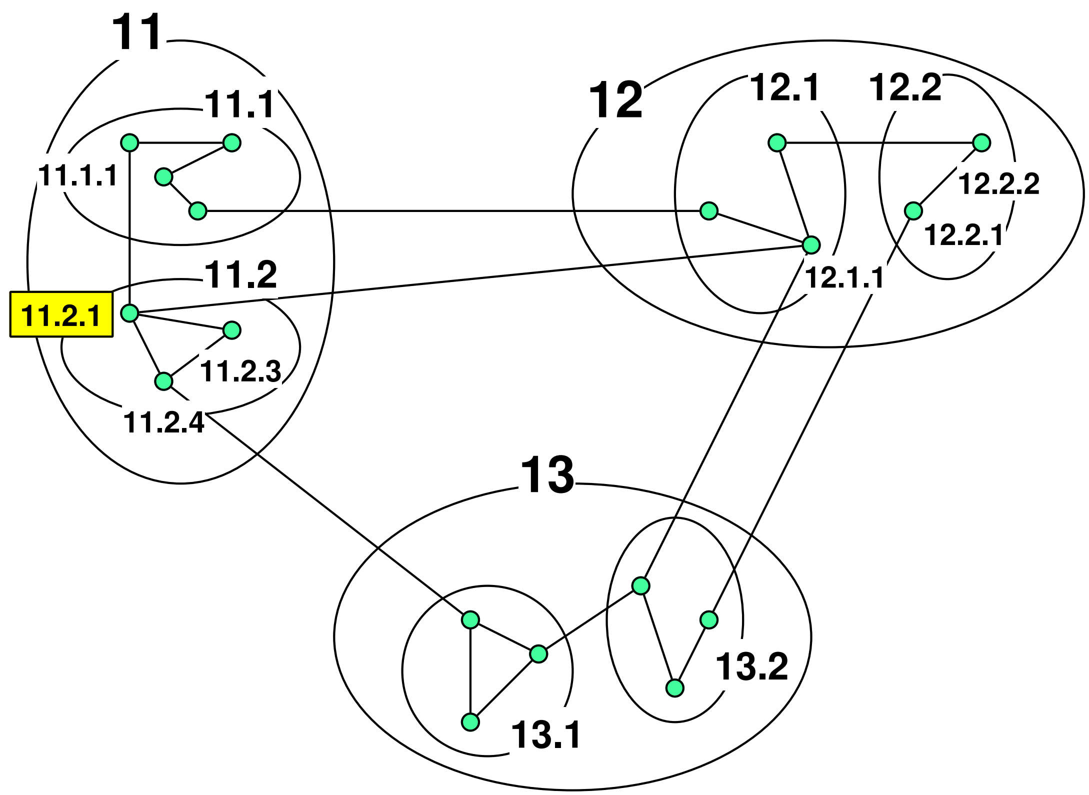
Example Area Hierarchy

- AS's (11, 12, 13)
- sub-AS's (11.1, etc.)
- networks (11.2.1/24, etc.)
- routing table at 11.2.1:
  - 11.2.3/24: 11.2.1.x
  - 11.2.4/24: 11.2.1.y
  - 11.1/16: 11.2.1.z
  - 12/8: ?



Example Area Hierarchy

- AS's (11, 12, 13)
- sub-AS's (11.1, etc.)
- networks (11.2.1/24, etc.)
- routing table at 11.2.1:
  - 11.2.3/24: 11.2.1.x
  - 11.2.4/24: 11.2.1.y
  - 11.1/16: 11.2.1.z
  - 12/8: 11.2.1.w
  - 13/8: ?



Example Area Hierarchy

- AS's (11, 12, 13)
- sub-AS's (11.1, etc.)
- networks (11.2.1/24, etc.)
- routing table at 11.2.1:
  - 11.2.3/24: 11.2.1.x
  - 11.2.4/24: 11.2.1.y
  - 11.1/16: 11.2.1.z
  - 12/8: 11.2.1.w
  - 13/8: 11.2.1.y

is this the only way?