Routing 1

Intra-Domain Routing

Forwarding v.s. Routing

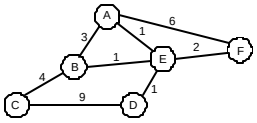
- Forwarding: the process of moving packets from input to output based on:
  - the forwarding table
  - information in the packet.
- Routing: process by which the forwarding table is built and maintained:
  - one or more routing protocols
  - procedures (algorithms) to convert routing info to forwarding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

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
- LS: Link state protocols

Distance Vector Protocols

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

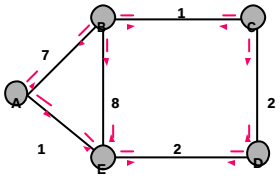
Distributed Bellman-Ford

Start Conditions:
Each router starts with a vector of (zero) distances to all directly attached networks

Send step:
Each router advertises its current vector to all neighboring routers.

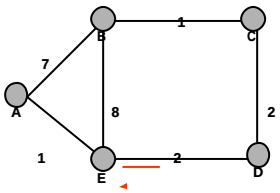
Receive step:
Upon receiving vectors from each of its neighbors, router computes its own distance to each neighbor.
Then, for every network X, router finds that neighbor who is closer to X than to any other neighbor.
Router updates its cost to X. After doing this for all X, router goes to send step.

Example - initial distances



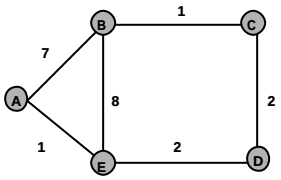
Info at node	Distance to node A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

E receives D's routes



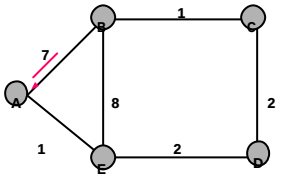
Info at node	Distance to node A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

E updates cost to C



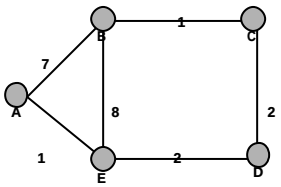
Info at node	Distance to node A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A receives B's routes



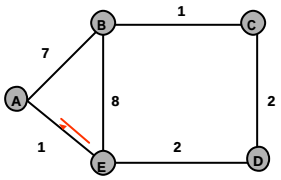
Info at node	Distance to node A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A updates cost to C



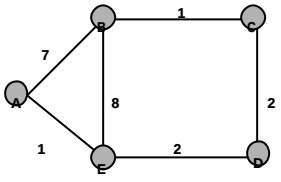
Info at node	Distance to node A	B	C	D	E
A	0	7	8	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A receives E's routes



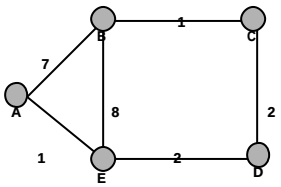
Info at node	Distance to node A	B	C	D	E
A	0	7	8	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

A updates cost to C and D



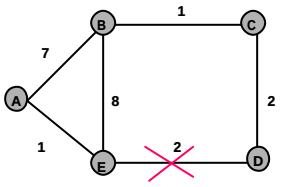
Info at node	Distance to node A	B	C	D	E
A	0	7	5	3	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

Final distances



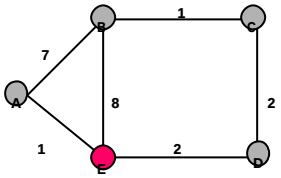
Info at node	Distance to node A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

Final distances after link failure



Info at node	Distance to 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

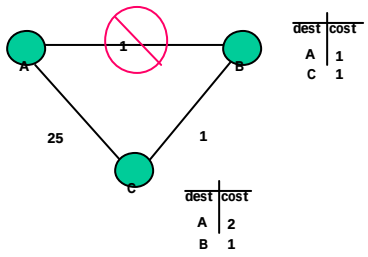
View from a node



E's routing table

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

The bouncing effect

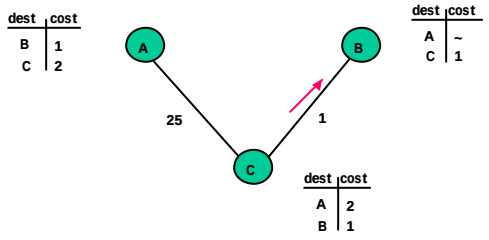


dest	cost
B	1
C	2

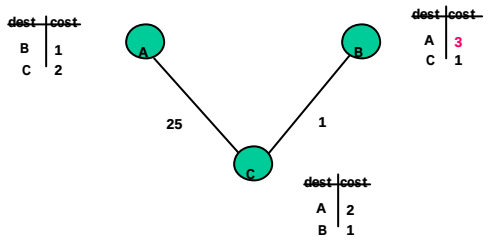
dest	cost
A	1
C	1

dest	cost
A	2
B	1

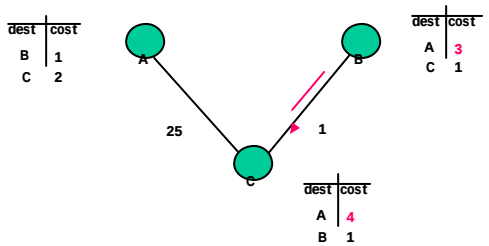
C sends routes to B



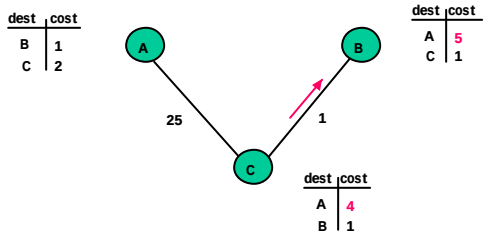
B updates distance to A



B sends routes to C



C sends routes to B



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

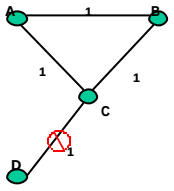
Other "solutions"

- 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	z

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

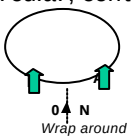
Reliable flooding

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

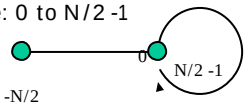
One solution: LSP Aging

- Nodes periodically decrement age (TTL) of stored LSPs
- LSPs expire when TTL reaches 0
  - LSP is re-flooded once TTL = 0
- 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

- 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

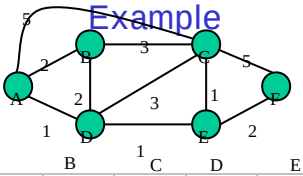
Is aging still needed?

- 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

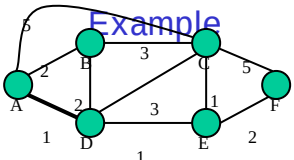
Example



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	~	~

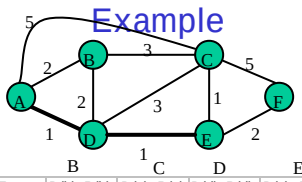
Example



B C D E F

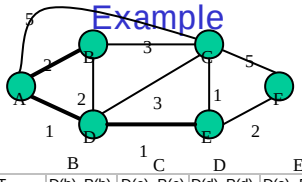
step	SPT	D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~

Example



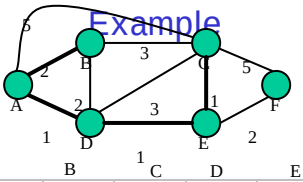
step	SPT	D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E

Example



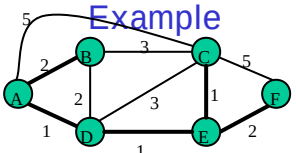
step	SPT	D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E
3	ADEB		3, E			4, E

Example



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



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**

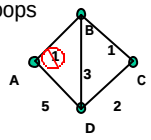
Path Computation

- **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.

- Msg size: small with LS, potentially large with DV
- Msg exchange: LS: O(nE), DV: only to neighbors

LS v.s. DV

- Convergence speed:
  - LS: fast
  - DV: fast with triggered updates
- Space requirements:
  - LS maintains entire topology
  - DV maintains only neighbor state

LS v.s. DV

Robustness:

- LS can broadcast incorrect/corrupted LSP
  - localized problem
- DV can advertise incorrect paths to all destinations
  - incorrect calculation can spread to entire network

LS v.s. DV

- In LS nodes must compute consistent routes independently - must protect against LSDB corruption
- In DV routes are computed relative to other nodes

Bottom line: no clear winner, but we see more frequent use of LS in the Internet