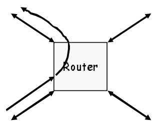
# CS 640: Introduction to Computer Networks

Aditya Akella

Lecture 10 -
Intra-Domain Routing

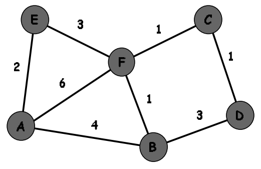
## The Road Ahead

- Past two lectures
  - IP addresses are structured
  - IP packet headers carry these addresses
  - When packet arrives at router
    - Examine header for intended destination
    - Look up next hop in table
    - Send packet out appropriate port
- This lecture:
  - How these forwarding tables are built?
  - Routing algorithms



## A Model of the Problem

- Network as a Graph:
  - Represent each router as node
  - Direct link between routers represented by edge
  - Symmetric links ⇒ undirected graph
- Edge "cost" c(x,y) denotes measure of difficulty of using link
  - delay, $ cost, or congestion level
- Task
  - Determine *least cost path* from every node to every other node
    - Path cost d(x,y) = sum of link costs



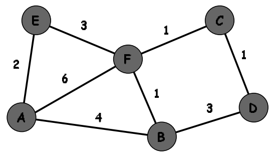
## Ways to Compute Shortest Paths

- Centralized
  - Collect graph structure in one place
  - Use standard graph algorithm
  - Disseminate routing tables
- Distributed
  - Routers perform local computation
  - Converge to a globally consistent routing state
  - *"Global": Link-state*
    - Every node collects complete graph structure
    - Each computes shortest paths from it
    - Each generates own routing table
  - *Local: Distance-vector*
    - No one has copy of graph
    - Nodes construct their own tables iteratively
    - Each sends information about its table to neighbors

4

## Distance-Vector Method

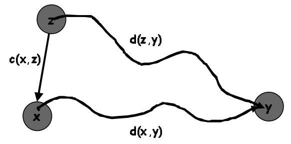
| Initial Table for A | | |
|---|---|---|
| Dest | Cost | Next Hop |
| A | 0 | A |
| B | 4 | B |
| C | ∞ | – |
| D | ∞ | – |
| E | 2 | E |
| F | 6 | F |



- Idea
  - At any time, have cost/next hop of best known path to destination
  - Use cost ∞ when no path known
- Initially
  - Only have entries for directly connected nodes

5

## Distance-Vector Update



```
Update(x,y,z)
d ← c(x,z) + d(z,y)    /* Cost of path from x to y with first hop z */
if d < d(x,y)
    /* Found better path */
    return d,z    /* Updated cost / next hop */
else
    return d(x,y), nexthop(x,y)    /* Existing cost / next hop */
```

6

## Algorithm

Each node x stores:
- c(x,v) for each neighbor v
- Distance vector of node x: estimate of d(x,y) for all y
- Distance vectors heard from each neighbor

Initialization:
1. d(x,y) = c(x,y) for all y.
2. Send distance vector to each neighbor

Repeat:

Whenever link cost to neighbor changes or distance vector received from neighbor

```
For every neighbor z
    For every destination y
        d(x,y) ← Update(x,y,z)
```

If d(x,y) changed for any y, send distance vector to all neighbors

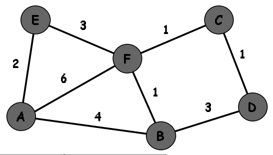
7

## Start

**Optimum 1-hop paths**

| Table for A | | | Table for B | | |
|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop |
| A | 0 | A | A | 4 | A |
| B | 4 | B | B | 0 | B |
| C | ∞ | - | C | ∞ | - |
| D | ∞ | - | D | 3 | D |
| E | 2 | E | E | ∞ | - |
| F | 6 | F | F | 1 | F |

| Table for C | | | Table for D | | | Table for E | | | Table for F | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| A | ∞ | - | A | ∞ | - | A | 2 | A | A | 6 | A |
| B | ∞ | - | B | 3 | B | B | ∞ | - | B | 1 | B |
| C | 0 | C | C | 1 | C | C | ∞ | - | C | 1 | C |
| D | 1 | D | D | 0 | D | D | ∞ | - | D | ∞ | - |
| E | ∞ | - | E | ∞ | - | E | 0 | E | E | 3 | E |
| F | 1 | F | F | ∞ | - | F | 3 | F | F | 0 | F |



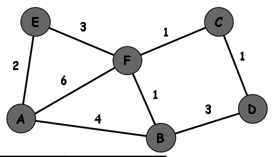
8

## Iteration #1

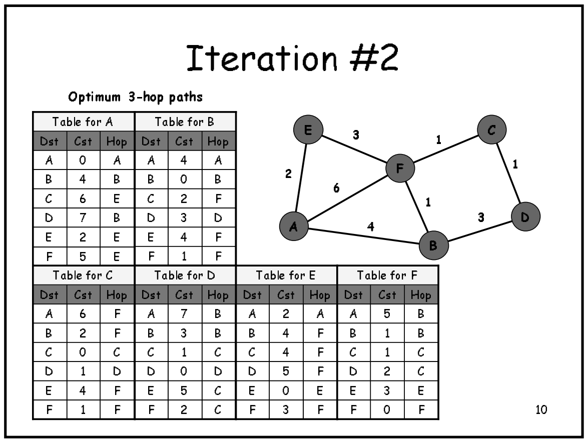
**Optimum 2-hop paths**

| Table for A | | | Table for B | | |
|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop |
| A | 0 | A | A | 4 | A |
| B | 4 | B | B | 0 | B |
| C | 7 | F | C | 2 | F |
| D | 7 | B | D | 3 | D |
| E | 2 | E | E | 4 | F |
| F | 5 | E | F | 1 | F |

| Table for C | | | Table for D | | | Table for E | | | Table for F | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| A | 7 | F | A | 7 | B | A | 2 | A | A | 5 | B |
| B | 2 | F | B | 3 | B | B | 4 | F | B | 1 | B |
| C | 0 | C | C | 1 | C | C | 4 | F | C | 1 | C |
| D | 1 | D | D | 0 | D | D | ∞ | - | D | 2 | C |
| E | 4 | F | E | ∞ | - | E | 0 | E | E | 3 | E |
| F | 1 | F | F | 2 | C | F | 3 | F | F | 0 | F |



9

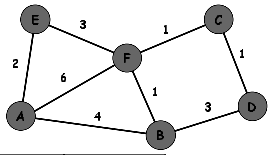


## Iteration #2

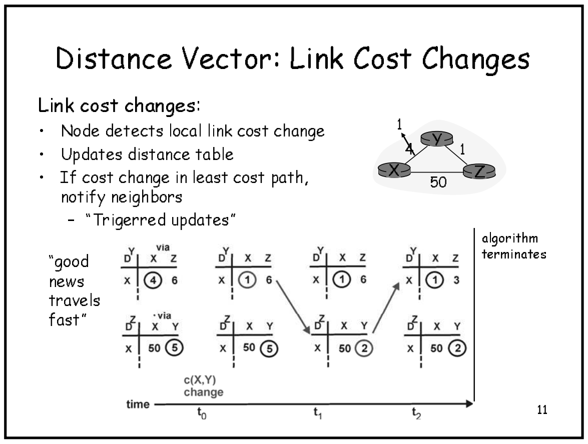
**Optimum 3-hop paths**

| Table for A | | | Table for B | | |
|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop |
| A | 0 | A | A | 4 | A |
| B | 4 | B | B | 0 | B |
| C | 6 | E | C | 2 | F |
| D | 7 | B | D | 3 | D |
| E | 2 | E | E | 4 | F |
| F | 5 | E | F | 1 | F |

| Table for C | | | Table for D | | | Table for E | | | Table for F | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| A | 6 | F | A | 7 | B | A | 2 | A | A | 5 | B |
| B | 2 | F | B | 3 | B | B | 4 | F | B | 1 | B |
| C | 0 | C | C | 1 | C | C | 4 | F | C | 1 | C |
| D | 1 | D | D | 0 | D | D | 5 | F | D | 2 | C |
| E | 4 | F | E | 5 | C | E | 0 | E | E | 3 | E |
| F | 1 | F | F | 2 | C | F | 3 | F | F | 0 | F |



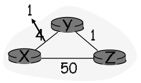
10



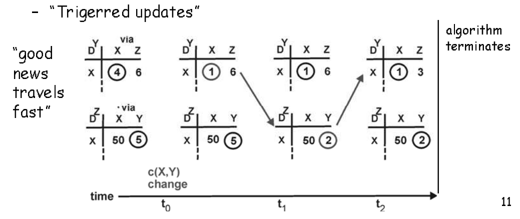
## Distance Vector: Link Cost Changes

Link cost changes:

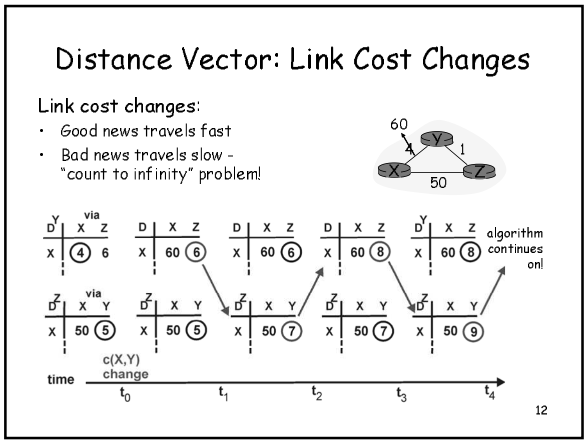
- Node detects local link cost change
- Updates distance table
- If cost change in least cost path, notify neighbors
  - "Trigerred updates"



"good news travels fast"



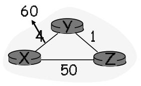
11

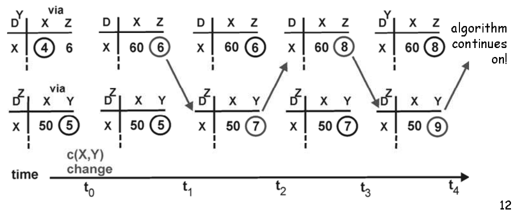


## Distance Vector: Link Cost Changes

Link cost changes:

- Good news travels fast
- Bad news travels slow - "count to infinity" problem!



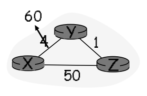


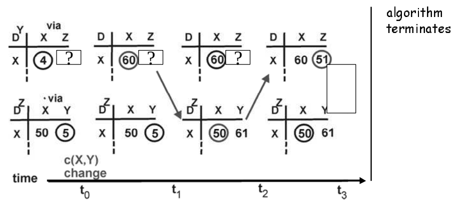
12

## Distance Vector: Split Horizon

If Z routes through Y to get to X :

- Z does not advertise its route to X back to Y



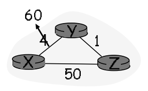


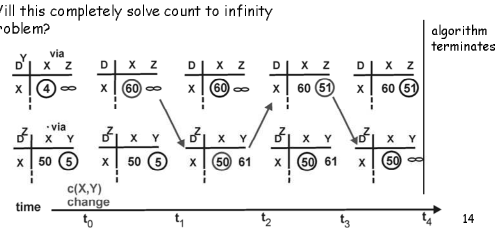
13

## Distance Vector: Poison Reverse

If Z routes through Y to get to X :

- Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- Eliminates some possible timeouts with split horizon
- Will this completely solve count to infinity problem?





14

## Poison Reverse Failures

| Table for A | | | Table for B | | | Table for D | | | Table for F | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| C | 7 | F | C | 8 | A | C | 9 | B | C | 1 | C |

| Table for A | | |
|---|---|---|
| Dst | Cst | Hop |
| C | ∞ | - |

**Forced Update**

**Forced Update**

| Table for F | | |
|---|---|---|
| Dst | Cst | Hop |
| C | ∞ | - |

| Table for A | | |
|---|---|---|
| Dst | Cst | Hop |
| C | 13 | D |

**Better Route**

**Forced Update**

| Table for B | | |
|---|---|---|
| Dst | Cst | Hop |
| C | 14 | A |

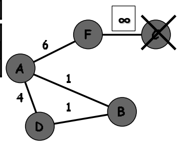
**Forced Update**

| Table for D | | |
|---|---|---|
| Dst | Cst | Hop |
| C | 15 | B |

| Table for A | | |
|---|---|---|
| Dst | Cst | Hop |
| C | 19 | D |

**Forced Update**

⋮



- Iterations don't converge
- "Count to infinity"
- Solution
  - Make "infinity" smaller
  - What is upper bound on maximum path length?

15

## Routing Information Protocol (RIP)

- Earliest IP routing protocol (1982 BSD)
  - *Current standard is version 2 (RFC 1723)*
- Features
  - Every link has cost 1 → Hop count
  - "Infinity" = 16
    - Limits to networks where everything reachable within 15 hops
- Sending Updates
  - Every router listens for updates on UDP port 520
  - Initial
    - When router first starts, asks for copy of table for every neighbor
    - Uses it to iteratively generate own table
  - Triggered
    - When every entry changes, send copy of entry to neighbors
      - Except for one causing update (split horizon rule)
  - Periodic
    - Every 30 seconds, router sends copy of its table to each neighbor

16

## RIP Staleness / Oscillation Control

- Small Infinity
  - Count to infinity doesn't take very long
- Route Timer
  - Every route has timeout limit of 180 seconds
    - Reached when haven't received update from next hop for 6 periods
  - If not updated, set to infinity
  - "Soft-state refresh"
- Behavior
  - When router or link fails, can take minutes to stabilize

17

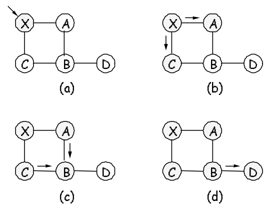
## Link State Protocol Concept

- Every node gets complete copy of graph
  - Every node "floods" network with data about its outgoing links
- Every node computes routes to every other node
  - Using single-source, shortest-path algorithm
- Process performed whenever needed
  - When interconnections die / reappear

18

## Sending Link States by "Flooding"

- X wants to send information
  - Sends on all outgoing links
- When node Y receives information from Z
  - Resend on all links other than Z



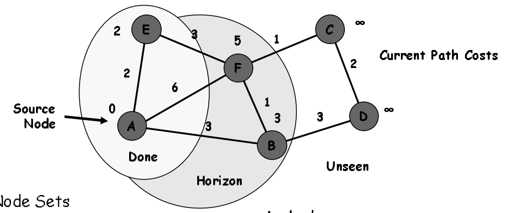
19

## Dijkstra's Algorithm

- Given
  - Graph with source node s and edge costs c(u,v)
  - Determine least cost path from s to every node v
- Single source shortest Path Algorithm
  - Traverse graph in order of least cost from source

20

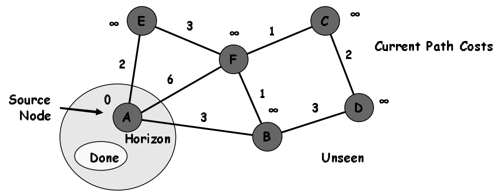
## Dijkstra's Algorithm



- Node Sets
  - Done
    - Already have least cost path to it
  - Horizon:
    - Reachable in 1 hop from node in Done
  - Unseen:
    - Cannot reach directly from node in Done
- Label
  - d(v) = path cost
    - From s to v
- Path
  - Keep track of last link in path

21

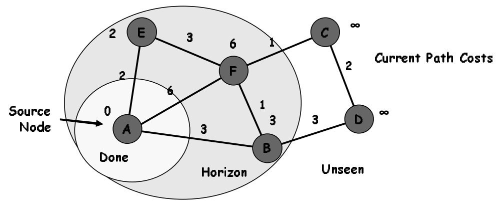
## Dijkstra's Algorithm: Initially



- No nodes "done"
- Source in "horizon"

22

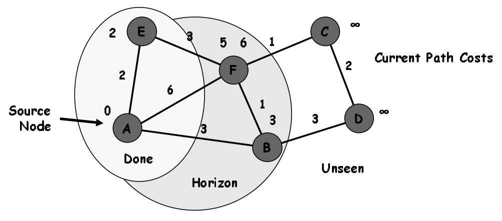
## Dijkstra's Algorithm: Initially



- d(v) to node A shown in red
  - Only consider links from done nodes

23

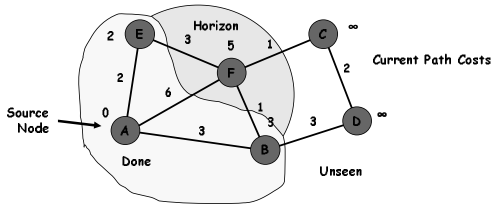
## Dijkstra's Algorithm



- Select node v in horizon with minimum d(v)
- Add link used to add node to shortest path tree
- Update d(v) information

24

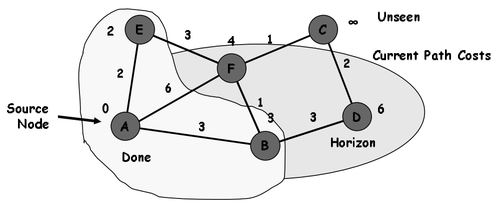
## Dijkstra's Algorithm



• Repeat…

25

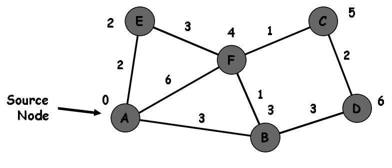
## Dijkstra's Algorithm



• Addition of node can add new nodes to horizon

26

## Dijkstra's Algorithm

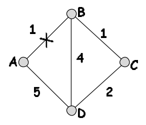


• Final tree shown in green

27

## Link State Characteristics

- With consistent LSDBs*, all nodes compute consistent loop-free paths
- Can still have transient loops



Packet from C→A may loop around BDC if B knows about failure and C & D do not

*Link State Data Base

28

## OSPF Routing Protocol

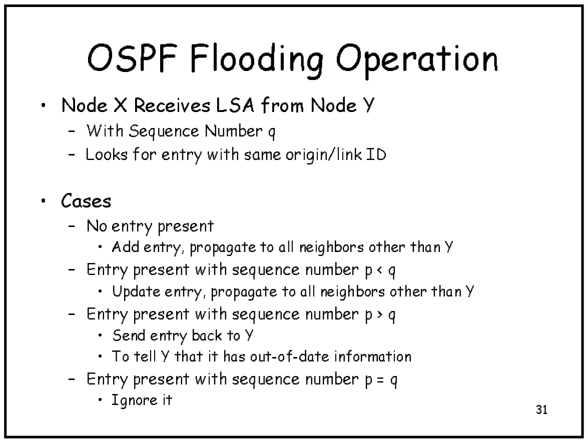
- Open
  - Open standard created by IETF
- Shortest-path first
  - Another name for Dijkstra's algorithm
- More prevalent than RIP

29

## OSPF Messages

- Transmit link state advertisements
  - Originating router
    - Typically, IP address for router
  - Link ID
    - ID of router at other end of link
  - Metric
    - Cost of link
  - Link-state age
    - Incremented each second
    - Packet expires when reaches 3600
  - Sequence number
    - Incremented each time sending new link information

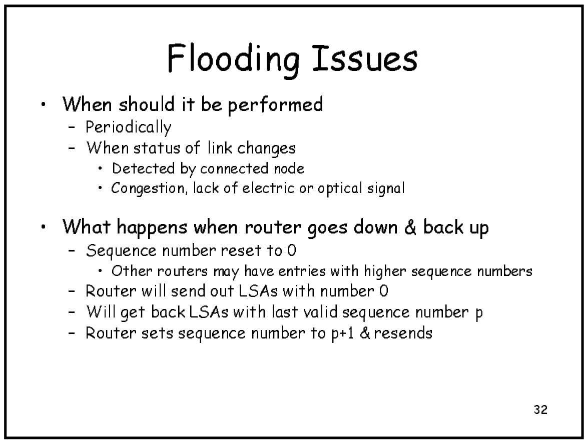
30



## OSPF Flooding Operation

- Node X Receives LSA from Node Y
  - With Sequence Number q
  - Looks for entry with same origin/link ID
- Cases
  - No entry present
    - Add entry, propagate to all neighbors other than Y
  - Entry present with sequence number p < q
    - Update entry, propagate to all neighbors other than Y
  - Entry present with sequence number p > q
    - Send entry back to Y
    - To tell Y that it has out-of-date information
  - Entry present with sequence number p = q
    - Ignore it

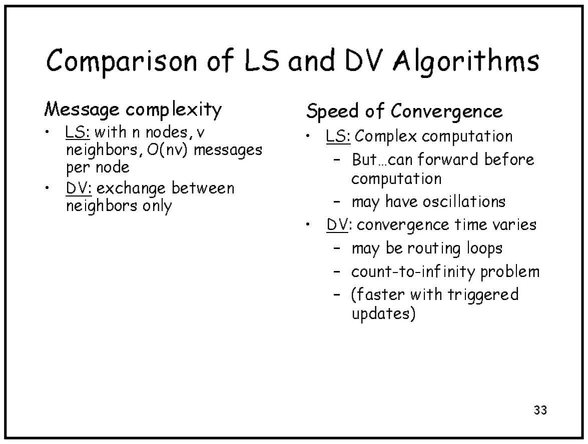
31



## Flooding Issues

- When should it be performed
  - Periodically
  - When status of link changes
    - Detected by connected node
    - Congestion, lack of electric or optical signal
- What happens when router goes down & back up
  - Sequence number reset to 0
    - Other routers may have entries with higher sequence numbers
  - Router will send out LSAs with number 0
  - Will get back LSAs with last valid sequence number p
  - Router sets sequence number to p+1 & resends

32



## Comparison of LS and DV Algorithms

Message complexity
- LS: with n nodes, v neighbors, O(nv) messages per node
- DV: exchange between neighbors only

Speed of Convergence
- LS: Complex computation
  - But…can forward before computation
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem
  - (faster with triggered updates)

33

## Adoption of OSPF

- RIP viewed as outmoded
  - Good when networks small and routers had limited memory & computational power
- OSPF Advantages
  - Fast convergence when configuration changes
  - Full topology map helps

34

## Comparison of LS and DV Algorithms

Robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - errors propagate thru network

- Other tradeoffs
  - Making LSP flood reliable difficult
  - Prioritize routing packets?

35

## Next Lecture

LS and DV are used in intra-domain routing

"Inter-domain" routing is much more complex

- Path vector
- BGP

36