NAT: Network Address Translation

Advantages:

- can change address of devices in local network without notifying outside world
- devices inside local net not explicitly addressable by or visible to the outside world (a security plus)

Disadvantage:

- devices inside local net not explicitly addressable by or visible to the outside world, making peer-to-peer networking that much harder
- routers should only process up to layer 3 (port#'s are app layer objects)
- address shortage should instead be solved by IPv6, instead NAT hinders the adoption of IPv6 (nothing wrong with that?)

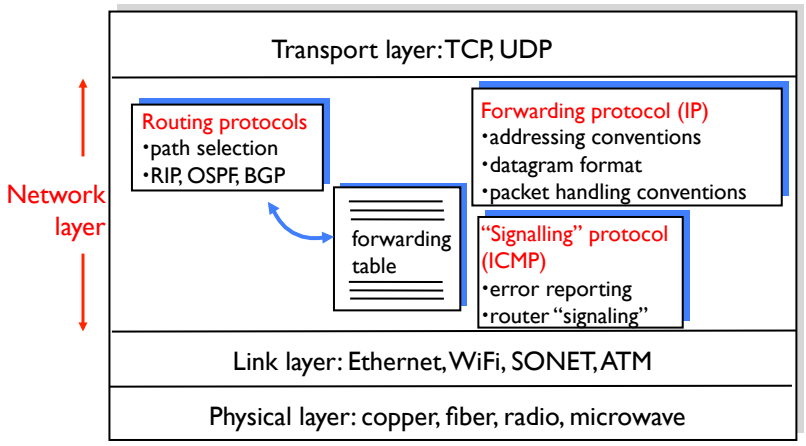
Lesson:

Be careful what you propose as a "temporary" patch,
"temporary" solutions have a tendency to stay around beyond expiration date

"The evil that men do lives after them,
the good is oft interred with their bones."
– Shakespeare, *Julius Caesar* [1]

The Internet Network Layer

Host, router network layer functions:

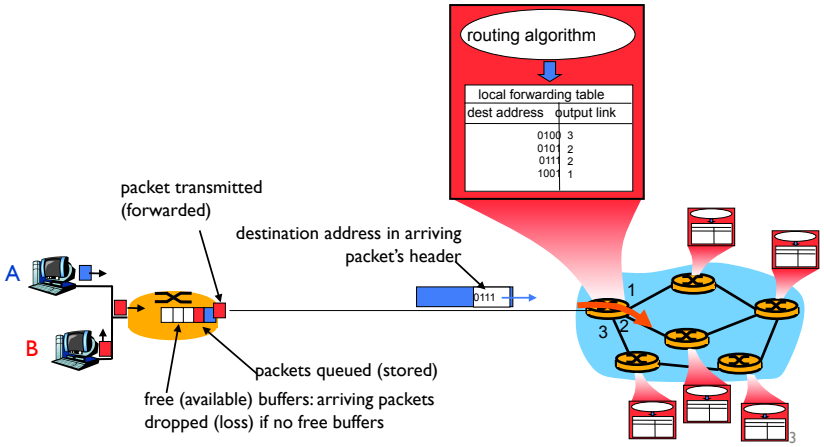


2

Routing on the Internet

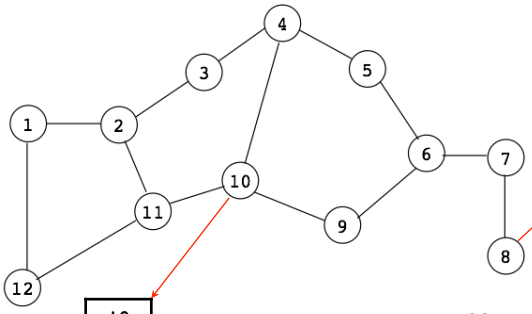
Routers on the Internet are store and forward routers:

- each incoming packet is buffered
- packet's destination is looked up in the routing table
- packet is forwarded to the next hop towards the destination



3

Routing on the Internet: Example



8	
dest	next
default	7

10			
dest	next	dest	next
1	11	7	9
2	11	8	9
3	4	9	9
4	4	10	-
5	4	11	11
6	9	12	11

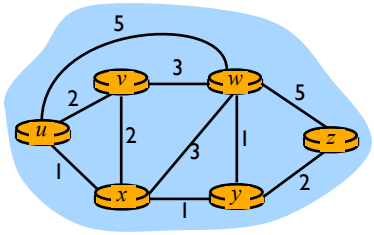
How does a router construct its routing table?

How does a router know which is the next hop towards a destination?

Use a routing protocol to propagate (and update) *reachability* information

4

Graph Abstraction



Graph: $G = \{N, E\}$

N = set of nodes = $\{u, v, w, x, y, z\}$

E = set of links = $\{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$

$c(u,v)$ = cost of link (u,v), assume full-duplex (bidirectional)

- e.g., $c(w,z) = 5$
- cost could always manually assigned, e.g., based on price
- could be hop count, or inversely related to bandwidth, reliability
- or could be dynamic, e.g., proportionally related to congestion

Cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

What is the least-cost path between node u and z ?

Routing algorithm: algorithm that finds least-cost path

5

Routing Algorithm Classification

Centralized or decentralized algorithm?

Global or distributed, local information?

Global info:

- all routers have complete topology, link cost info
- "link state" algorithms

Local info:

- routers know of only physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Static or dynamic routing?

Static routing:

⇒ routes change slowly over time

Dynamic routing:

⇒ routes change more quickly

⇒ periodic update in response to link cost changes

6

Dynamic Programming

- Used when a problem can be divided into subproblems that overlap
- Solves each subproblem once and store the solution in a table
  - if the same subproblem is encountered again, simply look up its solution in the table
  - reconstruct the solution to the original problem from solutions to the subproblems
  - the more overlap the better, as this reduces the number of subproblems

DP used primarily to solve **optimization problem**, e.g., find the shortest, longest, "best" way of doing something

Requirement: an optimal solution to the problem must be a composition of optimal solutions to all subproblems

In other words, there must not be an optimal solution that contains suboptimal solution to a subproblem

7

Distance Vector Algorithm

Origin of the name "dynamic programming":
- Bellman's shortest path algorithm (1957)
- *dynamic*: multi-stage, time-varying process
- *programming*: planning, decision making by a tabular method

Bellman's shortest path algorithm:
⇒ centralized distance vector algorithm
⇒ route table $D[]$ encodes shortest path

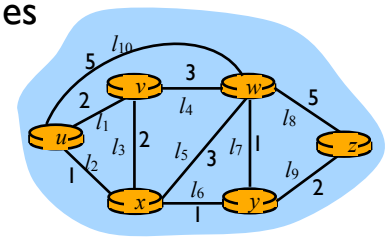
Define: $D[x,y]$:= cost of least-cost path from x to y
Then: $D[x,y] = \min\{c(x,v) + D[v,y]\}$,
where v is a neighbor of x and $\min$ is taken over all neighbors of x

Define two other tables:
⇒ $L[]$: link table
⇒ $H[]$: next hop table

8

Bellman's Algorithm: Initial Values

L	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
n	u	u	v	v	x	x	w	w	y	u
m	v	x	x	w	w	y	y	z	z	w
$c(n,m)$	2	1	2	3	3	1	1	5	2	5



Initial values:

H	u	v	w	x	y	z
u	l_0	l_1	l_{10}	l_2	-	-
v	l_1	l_0	l_4	l_3	-	-
w	l_{10}	l_4	l_0	l_5	l_7	l_8
x	l_2	l_3	l_5	l_0	l_6	-
y	-	-	l_7	l_6	l_0	l_9
z	-	-	l_8	-	l_9	l_0

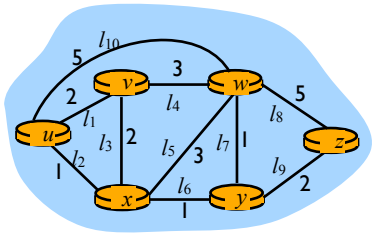
l_0: loopback

Initial values:

D	u	v	w	x	y	z
u	0	2	5	1	∞	∞
v	2	0	3	2	∞	∞
w	5	3	0	3	1	5
x	1	2	3	0	1	∞
y	∞	∞	1	1	0	2
z	∞	∞	5	∞	2	0

Bellman's Algorithm: Example

```
do {
  for each node i in graph do {
    for all node k not i in graph do {
      for each j neighbor of i {
        m = c(i,j)+D[j,k];
        if (m < D[i,k] {
          D[i,k]= m;
          H[i,k]= index of L[] where n=i,m=j;
        }
      }
    }
  }
} while there has been a change in D[];
```



Initial values:

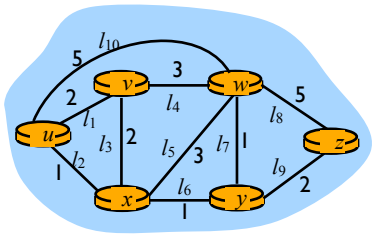
H	u	v	w	x	y	z
u	l_0	l_1	l_{10}	l_2	-	-
v	l_1	l_0	l_4	l_3	-	-
w	-	l_4	l_0	l_5	l_7	l_8
x	l_2	l_3	l_5	l_0	l_6	-
y	-	-	l_7	l_6	l_0	l_9
z	-	-	l_8	-	l_9	l_0

Initial values:

D	u	v	w	x	y	z
u	0	2	5	1	∞	∞
v	2	0	3	2	∞	∞
w	5	3	0	3	1	5
x	1	2	3	0	1	∞
y	∞	∞	1	1	0	2
z	∞	∞	5	∞	2	0

Bellman's Algorithm: Example

```
do {
  for each node i in graph do {
    for all node k not i in graph do {
      for each j neighbor of i {
        m = c(i,j)+D[j,k];
        if (m < D[i,k] {
          D[i,k]= m;
          H[i,k]= index of L[] where n=i,m=j;
        }
      }
    }
  }
} while there has been a change in D[];
```



Final values:

H	u	v	w	x	y	z
u	l_0	l_1	l_2	l_2	l_2	l_2
v	l_1	l_0	l_4	l_3	l_3	l_3
w	l_7	l_4	l_0	l_7	l_7	l_7
x	l_2	l_3	l_6	l_0	l_6	l_6
y	l_6	l_6	l_7	l_6	l_0	l_9
z	l_9	l_9	l_9	l_9	l_9	l_0

Final values:

D	u	v	w	x	y	z
u	0	2	3	1	2	4
v	2	0	3	2	3	5
w	3	3	0	2	1	3
x	1	2	2	0	1	3
y	2	3	1	1	0	2
z	4	5	3	3	2	0

11

Distributed Distance Vector Algorithm

Ford-Fulkerson (1962): modified Bellman's algorithm to a distributed version (a.k.a. Bellman-Ford algorithm)

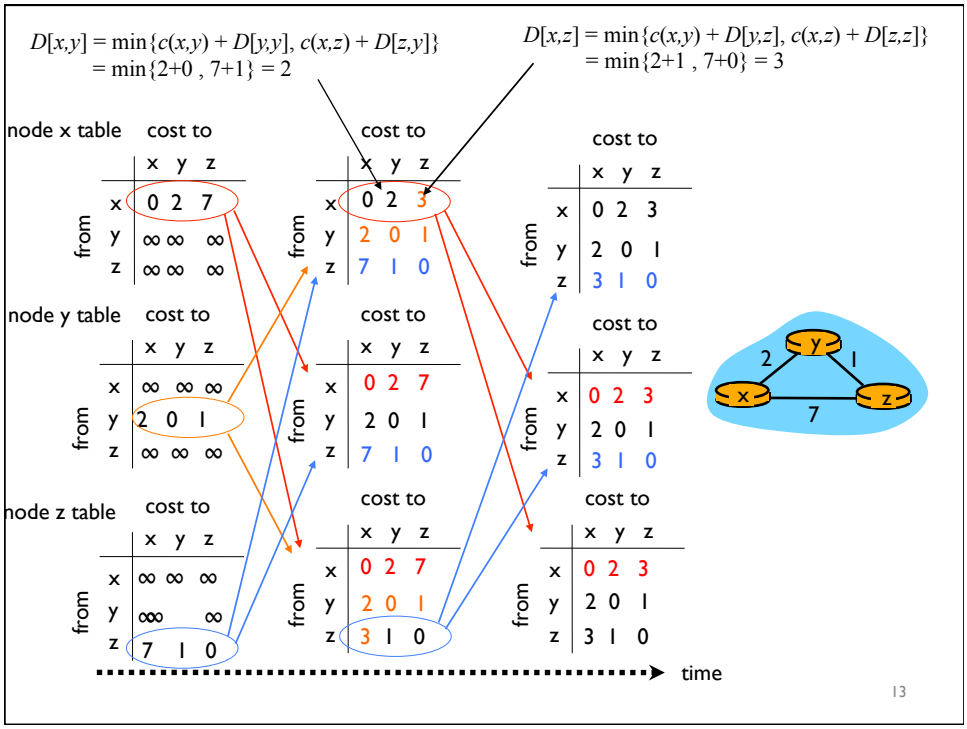
Basic idea:

- Each node periodically sends its own distance estimates to neighbors
- When node i receives new distance estimates from a neighbor, it updates its own distance estimates using the Bellman-Ford equation:

$$D[x,y] = \min\{c(x,v) + D[v,y]\}, \text{ for each node } y \in N$$

- Under stable conditions, the estimate $D[x,y]$ converges to the actual least cost

12



13

Distributed DVA Implementation

Each node i:

- knows the cost to each neighbor
  - keeps entries of L table for local links
- Node i maintains $D[i,*] = \{D[i,*]: k \in N\}$
- i's routing table consists of the i-th row of tables D and H
- sends i-th row of table D as route update from i
- upon receiving a route update from another node, i recomputes its routing table (row i of D and H)

Example:

- u's link table: $[l_1, l_2]$
- u's routing table:

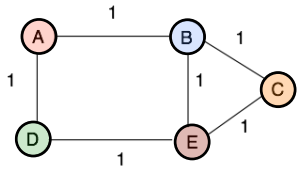
dest	u	v	w	x	y	z
D	0	2	3	1	2	4
H	l_0	l_1	l_2	l_2	l_2	l_2

- u's route update/distance vector:

dest	u	v	w	x	y	z
D	0	2	3	1	2	4

14

Distributed DVA



Scenario:

- time t_0: all nodes "wake up" and send route updates containing their reachability information
- t_1: all nodes have heard from their neighbors and update their routing tables and send out route updates
- t_2: no further changes

time		t_0		t_1		t_2	
A	A	-	0	-	0	-	0
	B	B	1	B	1	B	1
	C			B	2	B	2
	D	D	1	D	1	D	1
	E			B	2	B	2
B	A	A	1	A	1	A	1
	B	-	0	-	0	-	0
	C	C	1	C	1	C	1
	D			A	2	A	2
	E	E	1	E	1	E	1
C	A			B	2	B	2
	B	B	1	B	1	B	1
	C	-	0	-	0	-	0
	D	E	2	E	2	E	2
	E	E	1	E	1	E	1
D	A	A	1	A	1	A	1
	B			A	2	A	2
	C			E	2	E	2
	D	-	0	-	0	-	0
	E	E	1	E	1	E	1
E	A			B	2	B	2
	B	B	1	B	1	B	1
	C	C	1	C	1	C	1
	D	D	1	D	1	D	1
	E	-	0	-	0	-	0

15

Route Updates

Even statically assigned link cost can change over time, e.g., when a link goes down (breaks)

Each node:

waits for change in local link cost or msg from neighbor

recomputes distance estimates

if distance to any dest has increased, notify neighbors

or send periodic update

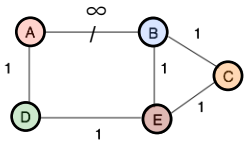
Design question: how does a router communicate changes in link cost to other routers?

- on-demand/triggered updates when cost increases ("bad news")
- "good news" travels slowly with periodic updates, with random periods

Design principle: soft-state protocol

16

Propagating Link Breakage



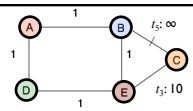
Scenario:

- time t_3: link AB breaks; A and B discover failure, update table, and send out route updates
- t_4: C, D, and E receive A's an B's route updates and in turn update their own routing tables and send out their route updates
- t_5: A, B, D, and E receive route updates sent at time t_4, update their routing tables and send out their route updates
- t_6: A, B, and C receive route updates sent at time t_5, update their routing tables and send out their route updates
- t_7: no more changes

	time	t_2		t_3		t_4		t_5		t_6	
A	A	-	0	-	0	-	0	-	0	-	0
	B	B	1	-	∞	-	∞	-	∞	D	3
	C	B	2	-	∞	-	∞	D	3	D	3
	D	D	1	D	1	D	1	D	1	D	1
	E	B	2	-	∞	-	∞	D	2	D	2
B	A	A	1	-	∞	-	∞	-	∞	E	3
	B	-	0	-	0	-	0	-	0	-	0
	C	C	1	C	1	C	1	C	1	C	1
	D	A	2	-	∞	-	∞	E	2	E	2
	E	E	1	E	1	E	1	E	1	E	1
C	A	B	2	B	2	-	∞	-	∞	E	3
	B	B	1	B	1	B	1	B	1	B	1
	C	-	0	-	0	-	0	-	0	-	0
	D	E	2	E	2	E	2	E	2	E	2
	E	E	1	E	1	E	1	E	1	E	1
D	A	A	1	A	1	A	1	A	1	A	1
	B	A	2	A	2	-	∞	E	2	E	2
	C	E	2	E	2	E	2	E	2	E	2
	D	-	0	-	0	-	0	-	0	-	0
	E	E	1	E	1	E	1	E	1	E	1
E	A	B	2	B	2	-	∞	D	2	D	2
	B	B	1	B	1	B	1	B	1	B	1
	C	C	1	C	1	C	1	C	1	C	1
	D	D	1	D	1	D	1	D	1	D	1
	E	-	0	-	0	-	0	-	0	-	0

17

Bouncing Effect



One big problem with distributed DV algorithm:

- Bounding Effect causes routing loop

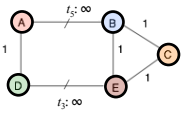
Example scenario (all examples are contrived, to illustrate problems):

- time t_3: cost of link EC jumps to 10 (e.g., due to load increase), shortly after, periodic update from B arrives at E and C, both end up using B as next hop to each other
- t_4: D gets an update from E, A and D both send out updates
- t_5: link BC breaks and A's update arrives at B
- t_6: A and E now goes through B to C
- Events at t_5 and t_6 iterate until cost of B to C through A =10, at which time, E switches to the EC link
  - meanwhile, data packets from A or B to C loop until TTL expires

	time	t_2		t_3		t_4		t_5		t_6	
A	A	-	0	-	0	-	0	-	0	-	0
	B	B	1	B	1	B	1	B	1	B	1
	C	B	2	B	2	B	2	B	2	B	4
	D	D	1	D	1	D	1	D	1	D	1
	E	B	2	B	2	B	2	B	2	B	2
B	A	A	1	A	1	A	1	A	1	A	1
	B	-	0	-	0	-	0	-	0	-	0
	C	C	1	C	1	C	1	A	3	C	1
	D	A	2	A	2	A	2	A	2	A	2
	E	E	1	E	1	E	1	E	1	E	1
C	A	B	2	B	2	B	2	-	∞	-	∞
	B	B	1	B	1	B	1	-	∞	-	∞
	C	-	0	-	0	-	0	-	0	-	0
	D	E	2	B	3	B	3	-	∞	-	∞
	E	E	1	B	2	B	2	-	∞	-	∞
D	A	A	1	A	1	A	1	A	1	A	1
	B	A	2	A	2	A	2	A	2	A	2
	C	E	2	E	2	E	3	E	3	E	3
	D	-	0	-	0	-	0	-	0	-	0
	E	E	1	E	1	E	1	E	1	E	1
E	A	B	2	B	2	B	2	B	2	B	2
	B	B	1	B	1	B	1	B	1	B	1
	C	C	1	B	2	B	2	B	2	B	4
	D	D	1	D	1	D	1	D	1	D	1
	E	-	0	-	0	-	0	-	0	-	0

18

Counting to ∞



As a means to resolve routing loop

Example scenario:

- time t_3: link DE breaks, A sends out a periodic route update
- t_4: D gets an update from A and sends out update, all paths go through A
- t_5: link AB breaks and D's update arrives at A, A updates route table: all paths go through D
- t_6: bouncing effect exists between A and D and routing loops are formed for destinations C and E

Each update brings cost up by one, detect that there's a loop when cost reaches ∞

time		t_2		t_3		t_4		t_5	
A	A	-	0	-	0	-	0	-	0
	B	B	1	B	1	B	1	D	3
	C	B	2	B	2	B	2	D	4
	D	D	1	D	1	D	1	D	1
	E	B	2	B	2	B	2	D	4
B	A	A	1	A	1	A	1	-	∞
	B	-	0	-	0	-	0	-	0
	C	C	1	C	1	C	1	C	1
	D	A	2	A	2	A	2	-	∞
	E	E	1	E	1	E	1	E	1
C	A	B	2	B	2	B	2	B	2
	B	B	1	B	1	B	1	B	1
	C	-	0	-	0	-	0	-	0
	D	E	2	E	2	E	2	B	2
	E	E	1	E	1	E	1	E	1
D	A	A	1	A	1	A	1	A	1
	B	A	2	A	2	A	2	A	2
	C	E	2	-	∞	A	3	A	3
	D	-	0	-	0	-	0	-	0
	E	E	1	-	∞	A	3	A	3
E	A	B	2	B	2	B	2	B	2
	B	B	1	B	1	B	1	B	1
	C	C	1	C	1	C	1	C	1
	D	D	1	-	∞	-	∞	-	∞
	E	-	0	-	0	-	0	-	0

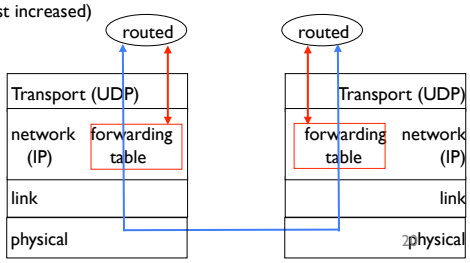
19

RIP (Routing Information Protocol)

An implementation of distance vector algorithm

- distributed with BSD-UNIX in 1982
- distance metric: # of hops (max 15 hops, ∞ = 16 hops)
- distance vectors sent/advertised once every 30 secs
  - ⇒ takes 8 minutes to count to ∞ and detect loop if we rely only on periodic updates
  - ⇒ link failure info quickly propagated with **triggered updates**
  - distance vectors sent using UDP
  - each advertisement lists up to 25 destination nets
- if no advertisement heard after 180 sec => neighbor/link declared down
  - all routes via neighbor invalidated
  - new advertisements sent to all other neighbors (triggered updates)
  - neighbors in turn send out new advertisements (if cost increased)

RIP routing tables managed by **application-level** process called route-d (daemon)



Routing Loop

Problems with distributed DVA:

- bouncing effect
- routing loop
- counting to ∞

Cause of routing loop (in 3 variations):

- inconsistent routing tables
- route updates do not reflect reality
- routers do not know when they are in their neighbor's path to a destination

Heuristics (not solution) to alleviate problem:

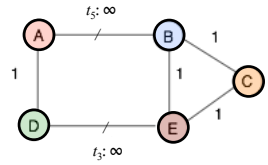
- triggered updates to shorten convergence time
- split horizon
- split horizon with poisonous reverse
- path hold-down
- route poisoning

Loop-free routing:

- path vector
- path finding/source tracing
- diffusing computation
- link reversal

21

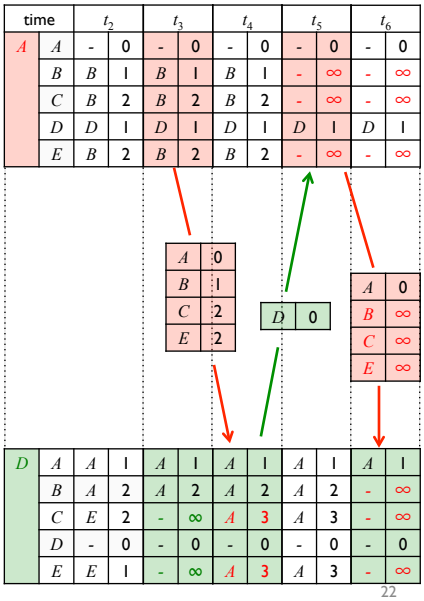
Split Horizon



Idea: don't advertise reachability to next-hop neighbor

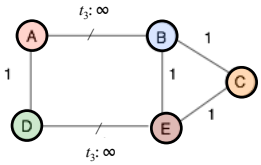
Example scenario:

- time t_3: link DE breaks, D sends out triggered update, A sends out a periodic route update
- t_4: D gets an update from A, all paths go through A. ***Doesn't send reachability to B, C, and E to A because it uses A as next hop to get to these nodes***
- t_5: link AB breaks and D's update arrives at A, A updates route table: only D is now reachable



22

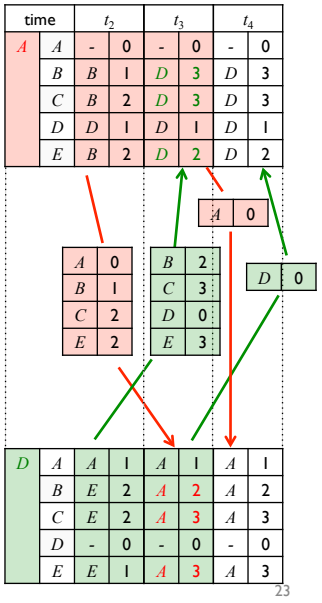
Why Split Horizon is not a Solution



Example scenario:

- time t_2: A and D send periodic updates to each other
- t_3: links AB and DE break simultaneously, right before updates from t_2 arrive, now A and D adopt each other as next hop to B, C, and E
- t_4: D and A **do not update** each other's path to B, C, and E because they're using each other as next hop

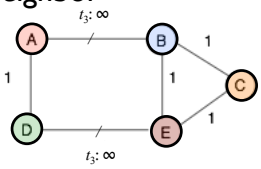
Both must rely on soft-state to stale entries



23

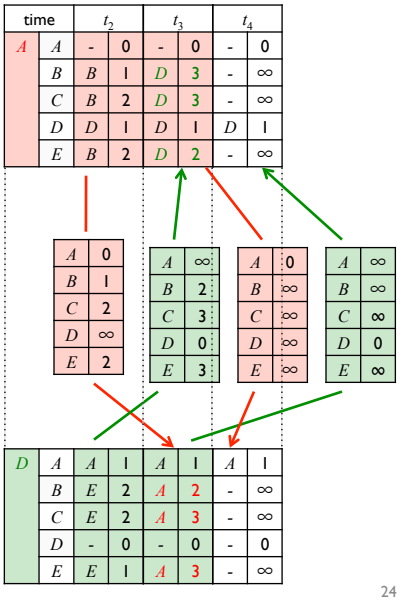
Split Horizon with Poisonous Reverse

Idea: advertise cost ∞ to next-hop neighbor



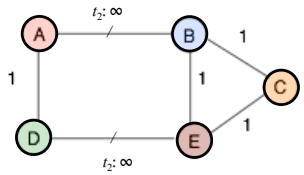
Example scenario:

- time t_2: A and D send periodic updates to each other
- t_3: links AB and DE break simultaneously, right before updates from t_2 arrive, now A and D adopt each other as next hop to B, C, and E
- t_4: D and A **update** each other's path to B, C, and E **as cost** ∞ because they're using each other as next hop



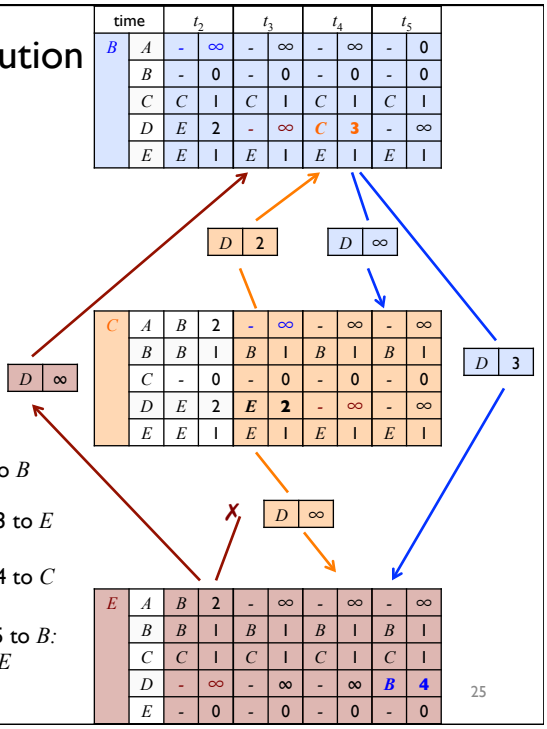
24

Why Both are not Solution



Example scenario (consider only destination D):

- t_2: links AB and DE break simultaneously
- t_3: assumes B receives E's update, but C doesn't:
  C advertises reachability to D at cost 2 to B
- t_4: B advertises reachability to D at cost 3 to E
- t_5: E advertises reachability to D at cost 4 to C
- t_6: C advertises reachability to D at cost 5 to B: bouncing effect exists between B, C, and E



25

Routing Loop

Problems with distributed DVA:

- bouncing effect
- routing loop
- counting to ∞

Cause of routing loop (in 3 variations):

- inconsistent routing tables
- route updates do not reflect reality
- routers do not know when they are in their neighbor's path to a destination

Heuristics (not solution) to alleviate problem:

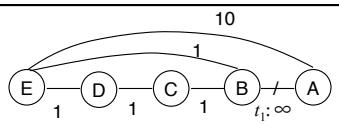
- triggered updates to shorten convergence time
- split horizon
- split horizon with poisonous reverse
- path hold-down
- route poisoning

Loop-free routing:

- path vector
- path finding/source tracing
- diffusing computation
- link reversal

26

Path Hold-down



Idea: do not switch route for n update periods after cost increase

Each node's cost to A,
with hold-down period $n = 2$

time	B	C	D	E
t_0	1	2	3	2
t_1	∞	2	3	2
t_2	∞	∞	3	∞
t_3	∞	∞	∞	∞
t_4	∞	∞	∞	10
t_5	11	∞	11	10
t_6	11	12	11	10
t_7	11	12	11	10

Smaller hold-down period
$n = 1$

time	B	C	D	E
t_0	1	2	3	2
t_1	∞	2	3	2
t_2	3	∞	3	∞
t_3	∞	4	∞	4
t_4	5	∞	5	∞
		∘		
		∘		
		∘		
t_k				10

27

Route Poisoning

Idea: advertise cost ∞ if cost from next hop has been increasing for n updates

- does not actually change cost in routing table entry, only what is advertised

Can be used with both path hold-down and split horizon with poisonous reverse

All heuristics rely on counting to ∞ to detect loop, they differ only in convergence time

28

Path Vector

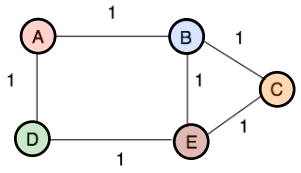
Idea:

- instead of sending only the next hop to a destination in distance vector, send the full path to each destination
- a router adopt a neighbor as the next hop to a destination only if it is not itself in neighbor's path to the destination
- a router prepends itself to all of its paths before propagating them further

Path vector is used in BGP

A's path vector:

dest	*A*	*B*	*C*	*D*	*E*
metric	0	1	2	1	2
path	*A*	*AB*	*ABC*	*AD*	*ADE*



29

Distributed DVA Deployment History

- Early days: GGP, HELLO, Fuzzball (ARPANET, early Internet)
- 1988 (standardized): RIP (routed)
  - v1: 30 secs periodic update with triggered updates and split horizon with poisonous reverse
  - v2 (1993): CIDR
- 1988: IGRP (cisco): Interior Gateway Routing Protocol
  - v1: split horizon, with path hold-down (n=2)
  - v2: 90 secs periodic update with triggered updates, route poisoning
- 1993: EIGRP (cisco): Enhanced IGRP
  - uses DUAL, supports CIDR
- 1994: BGPv4 for inter-domain routing
  - uses path vector, supports CIDR, runs on TCP

30

Link State Routing

Observation: loop can be prevented if each node knows the actual network topology

In link-state routing, each node:

- floods the network with the state (up, down) of its links
- uses Dijkstra's Shortest Path First (SPF) algorithm to compute a shortest-path tree

What is advertised:

- DV: all nodes reachable from me, advertised to all neighbors
- LS: all my immediate neighbors, advertised to all nodes

31

Dijkstra's Shortest Path First (SPF) Algorithm

A greedy algorithm for solving single-source shortest path problem

- assume non-negative edge weights
- even if we're only interested in the path from s to a single destination, d, we need to find the shortest path from s to *all* vertices in G (otherwise, we might have missed a shorter path)
- if the shortest path from s to d passes through an intermediate node u, i.e., $P = \{s, \ldots, u, \ldots, d\}$, then $P' = \{s, \ldots, u\}$ must be the shortest path from s to u

32

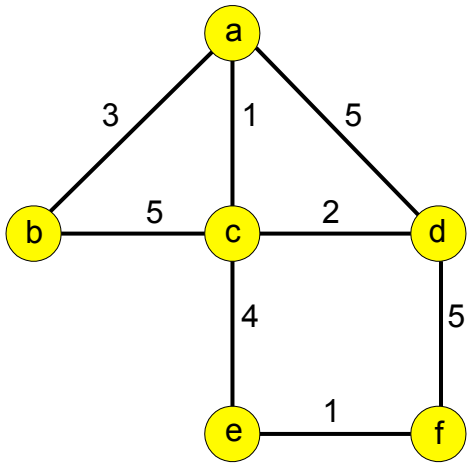
Dijkstra's Shortest Path First (SPF) Algorithm

```
SPF(startnode s)
{ // Initialize
  table = createtable(|V|); // stores spf, cost, predecessor
  table[*].spf = false; table[*].cost = INFINITY;
  pq = createpq(|E|); // empty pq
  table[s].cost = 0;
  pq.insert(0, s); // pq.insert(cost, v)
  while (!pq.isempty()) {
    v = pq.getMin();
    if (!table[v].spf) { // not on sp tree
      table[v].spf = true;
      for each u = v.neighbors() {
        newcost = weight(u, v) + table[v].cost;
        if (table[u].cost > newcost) {
          table[u].cost = newcost;
          table[u].pred = v;
          pq.insert(newcost, u);
        }
      }
    }
  }
  extract SPF from table;
}
```

33

Dijkstra's SPF Example (init)

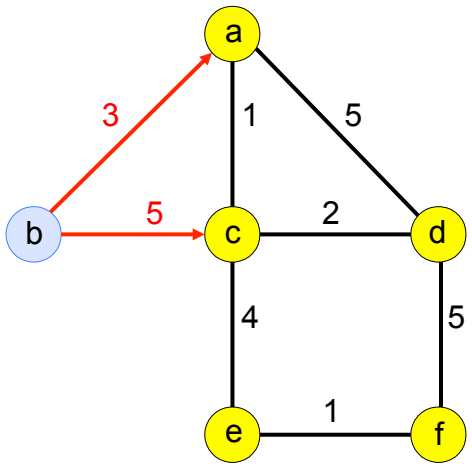
u	spf	cost	pred
a	F	∞	
b	F	0	-
c	F	∞	
d	F	∞	
e	F	∞	
f	F	∞	



34

Dijkstra's SPF Example ($v = s = b$)

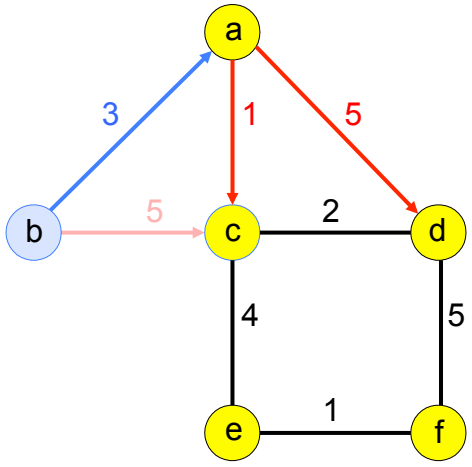
u	spf	cost	pred
a	F	3	b
b	T	0	-
c	F	5	b
d	F	∞	
e	F	∞	
f	F	∞	



35

Dijkstra's SPF Example ($v = a$)

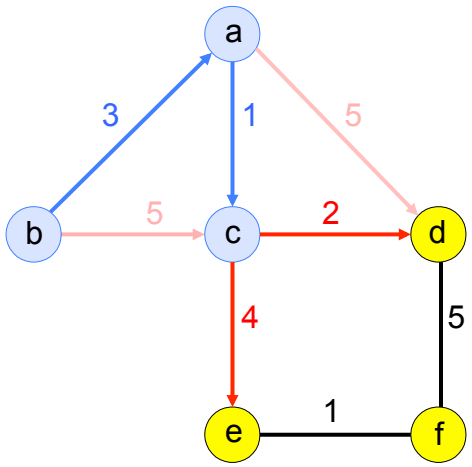
u	spf	cost	pred
a	T	3	b
b	T	0	-
c	F	4	a
d	F	8	a
e	F	∞	
f	F	∞	



36

Dijkstra's SPF Example ($v = c$)

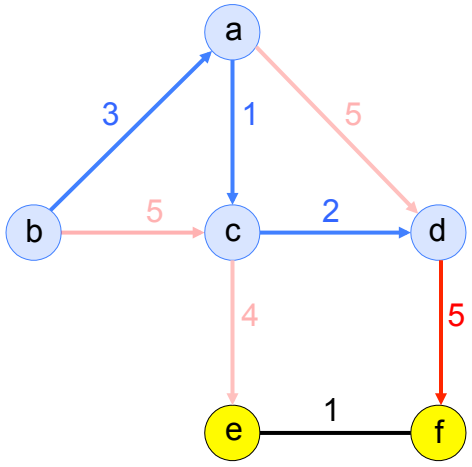
u	spf	cost	pred
a	T	3	b
b	T	0	-
c	T	4	a
d	F	6	c
e	F	8	c
f	F	∞	



37

Dijkstra's SPF Example ($v = d$)

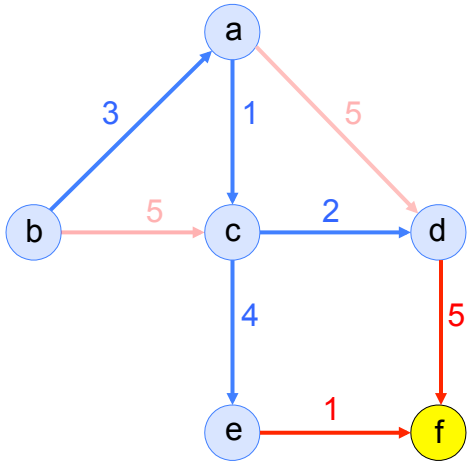
u	spf	cost	pred
a	T	3	b
b	T	0	-
c	T	4	a
d	T	6	c
e	F	8	c
f	F	11	d



38

Dijkstra's SPF Example ($v = e$)

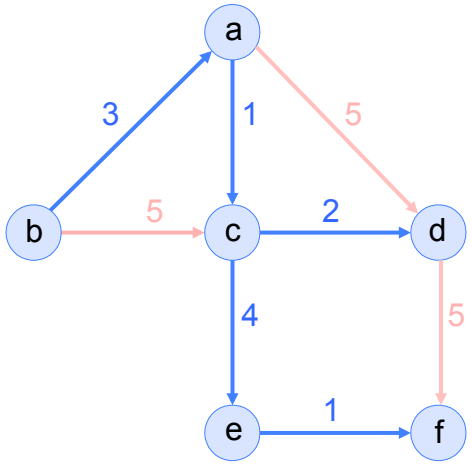
u	spf	cost	pred
a	T	3	b
b	T	0	-
c	T	4	a
d	T	6	c
e	T	8	c
f	F	9	e



39

Dijkstra's SPF Example ($v = f$)

u	spf	cost	pred
a	T	3	b
b	T	0	-
c	T	4	a
d	T	6	c
e	T	8	c
f	T	9	e



40

Dijkstra's SPF Algorithm

Algorithm complexity: N nodes

- each iteration: extract minHeap $O(\log|N|)$
- total $O(|N|\log|N|)$

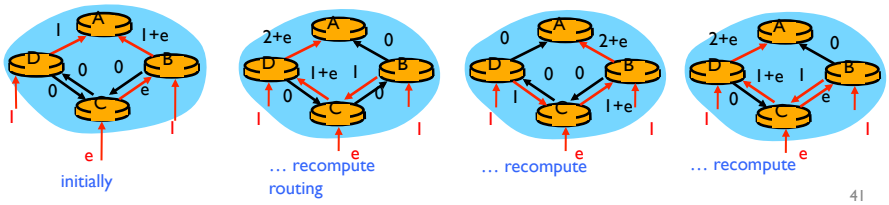
Each neighbor of each node could also potentially go thru the minHeap once: $O(|E|\log|N|)$

Total: $O(|N|\log(|N|)+|E|\log(|N|)) = O(|E|\log|N|)$

- $(|E| \geq |N| - 1$ for a connected graph)

Oscillations possible:

e.g., link cost = amount of carried traffic, asymmetric link cost



41

OSPF (Open Shortest Path First)

"Open": publicly available

Uses Link State algorithm

- LS packet dissemination
  - advertisements disseminated to entire network (via flooding protocol: forward to all interfaces except the incoming one)
  - advertisement carried in OSPF messages directly over IP (rather than TCP or UDP)
- route computation using Dijkstra's algorithm
- topology map at each node
  - OSPF is not loop free due to delay in topology propagation
- maintaining LS database consistency is hard due to router reboot:
  - how to determine which LS is newer?

42

OSPF (Open Shortest Path First)

Advance features (not in RIP):

- security: all OSPF messages authenticated (to prevent fake advertisement)
- multiple same-cost paths allowed (only one path in RIP)
- for each link, multiple cost metrics for different TOS
  (e.g., satellite link cost set to "low" for best effort; high for real time)
- integrated uni- and multicast support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- Hierarchical OSPF in large domains

43