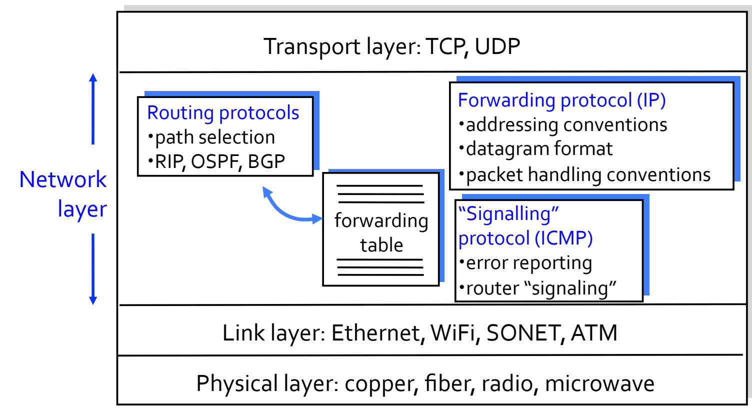
eecs489 Computer Networks

Lecture 15: Routing:
Distance Vector Algorithm

The Internet Network Layer

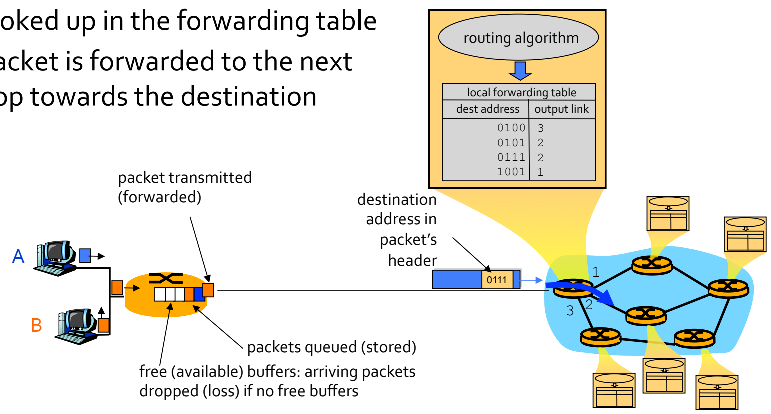
Host, router network layer functions:



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 forwarding table
- packet is forwarded to the next hop towards the destination



Routing vs. Forwarding

Routing: so-called "control plane"

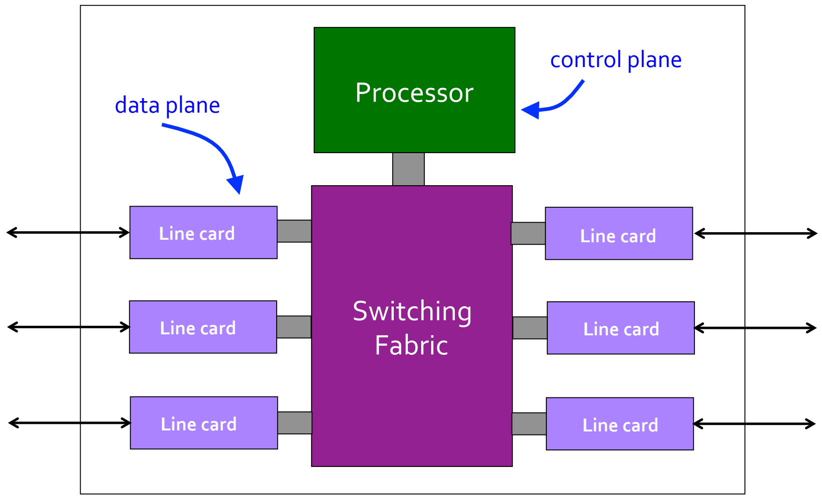
- compute paths that packets follow across an internetwork
- used by routers to talk to each other
- individual router creates a forwarding table from routing data

Forwarding: "data plane"

- individual router uses forwarding table to direct packets from an incoming to an outgoing link inside the router

[after Rexford]

Inside a Router

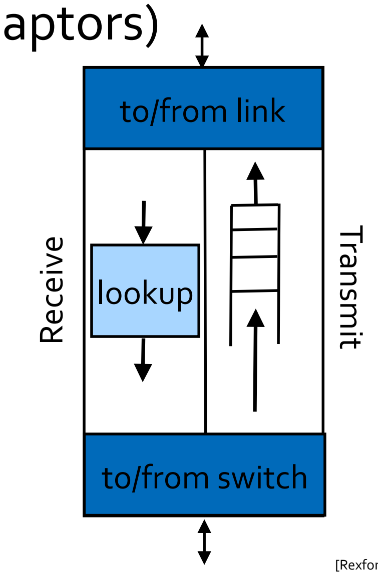


[Rexford]

Line Cards (Interface Cards, Adaptors)

Interfacing
- physical link
- switching fabric

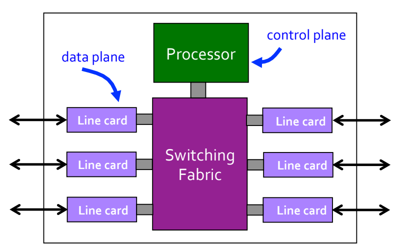
Packet handling
- packet forwarding
- decrement time-to-live
- buffer management
- link scheduling
- packet filtering
- rate limiting
- packet marking
- measurement



[Rexford]

Switching Fabric

Deliver packets inside router
- from incoming to outgoing interfaces
- a network in and of itself

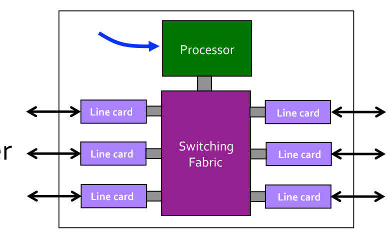


Must operate very quickly
- multitude of packets destined to the same outgoing interface
- switch scheduling to match inputs to outputs

Implementation techniques
- bus, crossbar, interconnection network, ...
- running at a faster speed (e.g., 2X) than links
- divide variable-length packets into fixed-size cells

[after Rexford]

Router Processor



"Loopback" interface
- IP address of the CPU on the router

"Control-plane" software
- implementation of routing protocols
- creation of forwarding table for the line cards

Handling of special data packets
- packets with IP options enabled
- packets with expired Time-to-Live

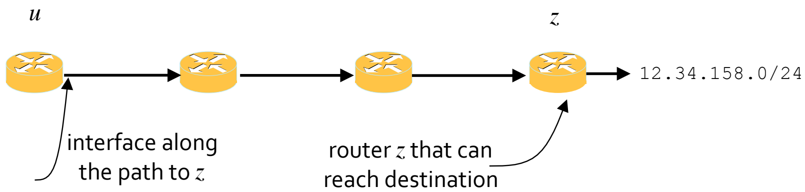
Network management functions:
- command-line interface (CLI) for configuration
- transmission of measurement statistics

[after Rexford]

Computing Paths Between Routers

Routers need to know two things

1. which router to use to reach a destination prefix
2. which outgoing interface to use to reach that router



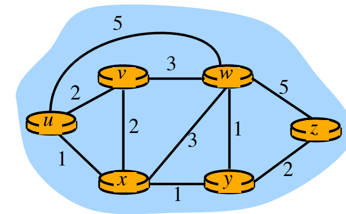
How does a router construct its routing table?

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

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

[after Rexford]

Graph Abstraction



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

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

E = set of edges/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 be manually assigned, e.g., based on price
- or 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

Routing Algorithm Classification

Centralized or decentralized?

- traditionally decentralized preferred for scalability
- software-defined networking (SDN) is centralizing

Global or distributed, local information?

- global: all routers have complete topology
- and link cost information
  - "link state" algorithms
- local: each router knows only of physically-connected neighbors and its link costs to neighbors
  - iterative process of computation, information exchange with neighbors
  - "distance vector" algorithms

Routing Algorithm Classification

Static or dynamic?

- static:
  - ⇒ routes change slowly over time
- dynamic:
  - ⇒ routes change more quickly
  - ⇒ periodic update in response to link cost changes

Dynamic Programming

Used when a problem can be divided into sub-problems that overlap

Solves each sub-problem once and stores the solution in a table

- if the same sub-problem is encountered again, simply looks up its solution in the table
- reconstructs the solution to the original problem from solutions to the sub-problems
- the more overlap the better, as this reduces the number of sub-problems

Dynamic Programming

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 sub-problems

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

Distance Vector Algorithm

Bellman's shortest path algorithm (1957)

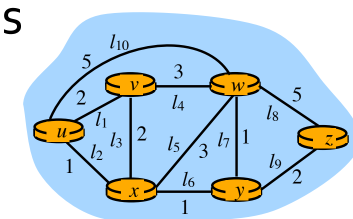
- a centralized distance vector algorithm
- the origin of the name "dynamic programming":
  - dynamic: multi-stage, time-varying process
  - programming: planning, decision making by a tabular method, e.g., TV programming

$D[]$ encodes shortest path between two nodes x and y computed as $D[x, y] = \min\{c(x, v) + D[v, y]\}$, where v is a neighbor of x and min is taken over all neighbors of x

Relies on two other tables:

- $L[]$: link table
- $H[]$: next hop table

Example: 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

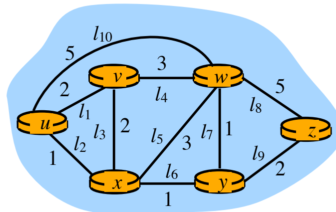
l_0: loopback

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

Example: Final Values

```
do {
  for each node i in graph do {
    for all node k not i in graph do {
      for each j neighbor of i {
        d = c(i,j)+D[j,k];
        if (d < D[i,k]) {
          D[i,k]= d;
          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

Distributed Distance Vector Alg.

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 all its immediate neighbors
- when node i receives new distance estimates from a neighbor, it updates its own distance estimates using the Bellman distance 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

$D[x,y] = \min\{c(x,y) + D[y,y], c(x,z) + D[z,y]\}$
$= \min\{2+0 , 7+1\} = 2$

$D[x,z] = \min\{c(x,y) + D[y,z], c(x,z) + D[z,z]\}$
$= \min\{2+1 , 7+0\} = 3$

x's table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

y's table

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

z's table

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

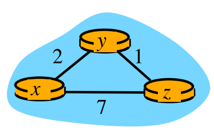
cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

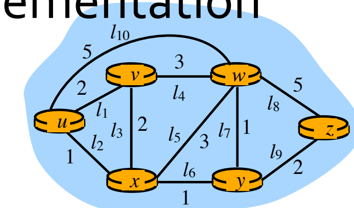
cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

time



Distributed DVA Implementation



Each node i:

- knows the cost to each neighbor
  - keeps entries of L table for local links
- 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

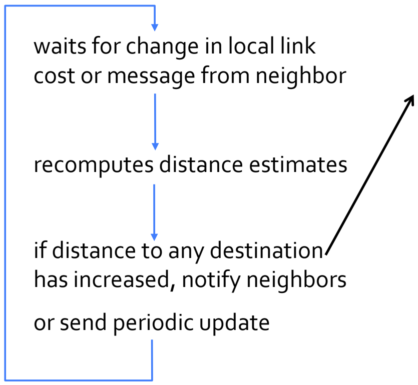
- u's route update/distance vector:

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

Route Updates

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

Each node:



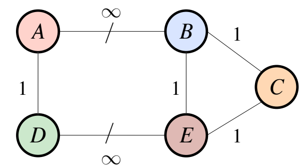
Design question: how does a router communicate changes in link cost to other routers?

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

Design principle: soft-state

Routing Loop

Problems with distributed DVA:

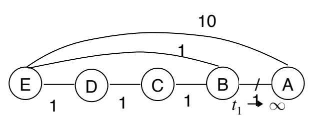


- bouncing effect, when there's alternate path, e.g., between A and D: when AB breaks, A and D (to B) count to 11 before settling on $ADEB$
- counting to ∞, when there's no alternate path, e.g., between B, C, and E (to A or D) when both AB and DE break
- leads to: routing loop

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

Routing Loop



Heuristics (not solution) to alleviate routing loop in distributed distance vector algorithm:

- triggered updates
- split horizon (with poisonous reverse): don't advertise reachability (or advertise ∞) to next-hop neighbor
- path hold-down, route poisoning: don't switch path for n rounds or advertise ∞ if cost has been going up for n rounds

Cost to A, with $n = 2$:

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

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

t_{11}	11	12	11	10

Loop-free Routing

Solutions to routing loop:

- diffusing computation (DUAL)
- path finding/source tracing
- link reversal
- path vector

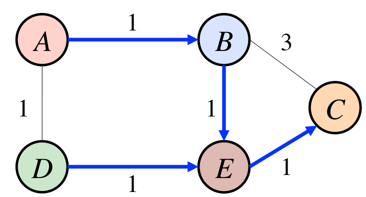
Distributed DVA Deployment History

- 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): supports 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 for intra-domain/AS routing
  - uses DUAL, supports CIDR
- 1994: BGPv4 for inter-domain routing
  - uses path vector, supports CIDR, runs on TCP

Jaffe-Moss Algorithm

Observations:

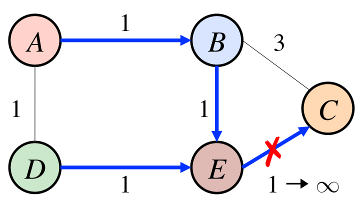
1. for each destination, nodes on the network form a directed spanning tree rooted at the destination
2. loops occur in BF algorithm only after link cost increases (incl. going to ∞)



Jaffe-Moss Algorithm

New idea: impose strong constraints on the ordering of route updates among nodes:

- when a link cost increase is detected, all nodes in the spanning tree rooted at the destination ($= C$) that uses the link must freeze their next-hop (successor) choice to that destination, i.e., they cannot pick an alternate next-hop

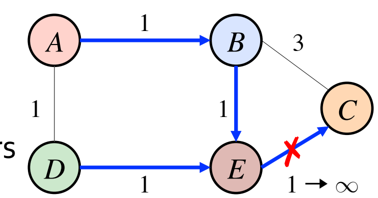


- instead, they run a diffusing computation

Diffusing Computation

Each node queries all its neighbors for a shorter path (queries carry increased link cost): first, E queries B and D

If query comes from shortest path next hop, a node freezes its path to the destination and recursively queries its neighbors for a shorter path: B freezes path to C and queries A, C, and E

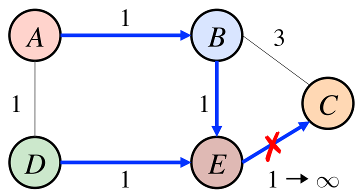


Otherwise, node replies with current cost: C replies to B

Diffusing Computation

A node doesn't unfreeze and reply to a query until it has received replies from all of its neighbors: B doesn't reply to E until it has heard from both C and A

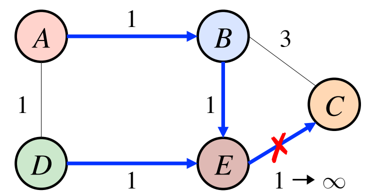
When a frozen node receives another query, it replies immediately with increased cost: D and B reply immediately to A with increased cost



Jaffe-Moss Algorithm

Short-coming: may freeze path unnecessarily

- it's right that E doesn't switch to B
- but there's no reason for B to wait for A's ACK before switching to its direct link to C



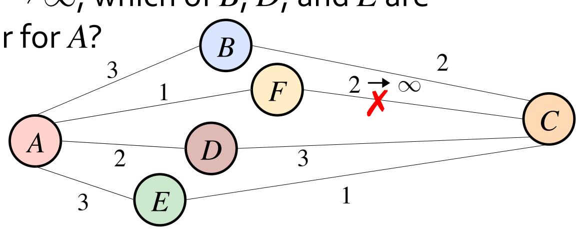
DUAL: Diffusing Update ALgorithm

Solves Jaffe-Moss' short-coming by not freezing a destination to which there is a viable alternate path

Successor: next hop node

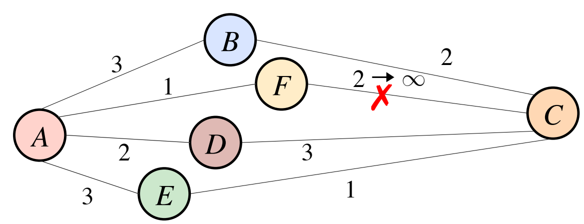
Feasible successor: an alternate successor whose cost towards destination is $\leq$ the cost of the current successor before the link cost increased

Cost from F to $C \rightarrow \infty$, which of B, D, and E are feasible successor for A?



DUAL: Diffusing Update ALgorithm

Why it works: a path through a feasible successor cannot contain a loop, i.e., it could not and cannot be using the link whose cost has gone up to get to destination



A node freezes a path to a particular destination and initiate diffusing computation only when it cannot find any feasible successor

DUAL: Diffusing Update ALgorithm

Worst-case complexity analysis: cost required for the network to converge after a resource failure:

- time complexity (TC): number of steps
- communication complexity (CC): number of messages

	DBF	ILS	JM	DUAL
TC	$O(N)$	$O(D)$	$O(x)$	$O(x)$
CC	$O(N^2)$	$O(2E)$	$O(E)$	$O(6dx)$

D: network diameter
d: max degree of a node ($\ll N$)
x: # nodes affected by failure;
worst case, e.g., on network partition, $x = N$

DBF: Distributed Bellman-Ford
ILS: Incremental Link-State
JM: Jaffe-Moss

Path Vector

Idea:

- instead of sending only the path cost to a destination in distance vector, send the full path to each destination
- a router adopts a neighbor as the next hop to a destination only if it is not 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

Example: A's path vector:

dest	A	B	C	D	E
metric	0	1	2	1	2
path	A	BA	CBA	DA	EDA