Chapter IV: Network Layer

UG3 Computer Communications & Networks (COMN)

Myungjin Lee
myungjin.lee@ed.ac.uk

Slides copyright of Kurose and Ross
IP addresses: how to get one?

Q: How does a host get IP address?

• hard-coded by system admin in a file
  – Windows: control-panel->network->configuration->tcp/ip-
    >properties
  – UNIX: /etc/rc.config

• **DHCP: Dynamic Host Configuration Protocol:**
  dynamically get address from as server
  – “plug-and-play”
DHCP: Dynamic Host Configuration Protocol

**goal:** allow host to *dynamically* obtain its IP address from network server when it joins network
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/“on”)
- support for mobile users who want to join network (more shortly)

**DHCP overview:**
- host broadcasts “DHCP discover” msg [optional]
- DHCP server responds with “DHCP offer” msg [optional]
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg
DHCP client-server scenario

223.1.1.0/24

223.1.1.1
223.1.1.2
223.1.1.3
223.1.1.4

223.1.2.0/24

223.1.2.1
223.1.2.2
223.1.2.9

DHCP server

arriving DHCP client needs address in this network

223.1.3.0/24

223.1.3.1
223.1.3.2
223.1.3.27
DHCP client-server scenario

DHCP server: 223.1.2.5

**DHCP discover**
- src: 0.0.0.0, 68
- dest.: 255.255.255.255,67
- yiaddr: 0.0.0.0
- transaction ID: 654

**DHCP offer**
- src: 223.1.2.5, 67
- dest: 255.255.255.255, 68
- yiaddr: 223.1.2.4
- transaction ID: 654
- lifetime: 3600 secs

**DHCP request**
- src: 0.0.0.0, 68
- dest:: 255.255.255.255, 67
- yiaddr: 223.1.2.4
- transaction ID: 655
- lifetime: 3600 secs

**DHCP ACK**
- src: 223.1.2.5, 67
- dest: 255.255.255.255, 68
- yiaddr: 223.1.2.4
- transaction ID: 655
- lifetime: 3600 secs
DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

– address of first-hop router for client (i.e., default gateway)
– name and IP address of DNS server
– network mask (indicating network versus host portion of address)
IP addresses: how to get one?

**Q:** how does network get subnet part of IP addr?

**A:** gets allocated portion of its provider ISP’s address space

<table>
<thead>
<tr>
<th>ISP's block</th>
<th>11001000 00010111 00010000 00000000</th>
<th>200.23.16.0/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization 0</td>
<td>11001000 00010111 00010000 00000000</td>
<td>200.23.16.0/23</td>
</tr>
<tr>
<td>Organization 1</td>
<td>11001000 00010111 00010010 00000000</td>
<td>200.23.18.0/23</td>
</tr>
<tr>
<td>Organization 2</td>
<td>11001000 00010111 00010100 00000000</td>
<td>200.23.20.0/23</td>
</tr>
<tr>
<td>...</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>Organization 7</td>
<td>11001000 00010111 00011110 00000000</td>
<td>200.23.30.0/23</td>
</tr>
</tbody>
</table>
Hierarchical addressing allows efficient advertisement of routing information:

“Send me anything with addresses beginning 200.23.16.0/20”

“Send me anything with addresses beginning 199.31.0.0/16”
Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1

“Send me anything with addresses beginning 200.23.16.0/20”

“Send me anything with addresses beginning 199.31.0.0/16 or 200.23.18.0/23”
IP addressing: the last word...

Q: how does an ISP get block of addresses?
A: ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/
  – allocates addresses
  – manages DNS
  – assigns domain names, resolves disputes
NAT: network address translation

rest of Internet

local network (e.g., home network) 10.0.0/24

138.76.29.7

10.0.0.1

10.0.0.2

10.0.0.3

all datagrams leaving local network have same single source NAT IP address: 138.76.29.7, different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)
NAT: network address translation

motivation: local network uses just one IP address as far as outside world is concerned:
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)
NAT: network address translation

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

NAT translation table

<table>
<thead>
<tr>
<th>WAN side addr</th>
<th>LAN side addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>138.76.29.7, 5001</td>
<td>10.0.0.1, 3345</td>
</tr>
<tr>
<td>……</td>
<td>……</td>
</tr>
</tbody>
</table>

3: reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345
NAT: network address translation

• 16-bit port-number field:
  – 60,000 simultaneous connections with a single LAN-side address!

• NAT is controversial:
  – routers should only process up to layer 3
  – violates end-to-end argument
    • NAT possibility must be taken into account by app designers, e.g., P2P applications
  – address shortage should instead be solved by IPv6
ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information
  - error reporting:
    - unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP msgs carried in IP datagrams
- **ICMP message:** type, code plus first 8 bytes of IP datagram causing error

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>echo reply (ping)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>dest. network unreachable</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>dest host unreachable</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>dest protocol unreachable</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>dest port unreachable</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>dest network unknown</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>dest host unknown</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>source quench (congestion control - not used)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>echo request (ping)</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>route advertisement</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>router discovery</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>TTL expired</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>bad IP header</td>
</tr>
</tbody>
</table>
Traceroute and ICMP

- Source sends series of UDP segments to dest
  - First set has TTL = 1
  - Second set has TTL = 2, etc.
  - Unlikely port number

- When nth set of datagrams arrives to nth router:
  - Router discards datagrams
  - And sends source ICMP messages (type 11, code 0)
  - ICMP messages includes name of router & IP address

- When ICMP messages arrives, source records RTTs

Stopping criteria:
- UDP segment eventually arrives at destination host
- Destination returns ICMP “port unreachable” message (type 3, code 3)
- Source stops
IPv6: motivation

• initial motivation: 32-bit address space soon to be completely allocated

• additional motivation:
  – header format helps speed processing/forwarding
  – header changes to facilitate QoS

IPv6 datagram format:

  – fixed-length 40 byte header
  – no fragmentation allowed
IPv6 datagram format

**priority:** identify priority among datagrams in flow

**flow Label:** identify datagrams in same “flow.”

(concept of “flow” not well defined)

**next header:** identify upper layer protocol for data

<table>
<thead>
<tr>
<th>ver</th>
<th>pri</th>
<th>flow label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>payload len</th>
<th>next hdr</th>
<th>hop limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source address (128 bits)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>destination address (128 bits)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>data</th>
</tr>
</thead>
</table>

32 bits
Other changes from IPv4

- **checksum**: removed entirely to reduce processing time at each hop
- **options**: allowed, but outside of header, indicated by “Next Header” field
- **ICMPv6**: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions
Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?

- **tunneling**: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers
Tunneling

**logical view:**

A IPv6 -- IPv4 tunnel connecting IPv6 routers -- IPv4 tunnel connecting IPv6 routers -- IPv6

**physical view:**

A IPv6 -- IPv4 -- IPv4 -- IPv4 -- IPv6
Tunneling

logical view:

physical view:

IPV4 tunnel connecting IPV6 routers
Interplay between routing, forwarding

Routing algorithm determines end-end-path through network.
Forwarding table determines local forwarding at this router.

IP destination address in arriving packet’s header.

<table>
<thead>
<tr>
<th>dest address</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>address-range 1</td>
<td>3</td>
</tr>
<tr>
<td>address-range 2</td>
<td>2</td>
</tr>
<tr>
<td>address-range 3</td>
<td>2</td>
</tr>
<tr>
<td>address-range 4</td>
<td>1</td>
</tr>
</tbody>
</table>
Graph abstraction

graph: $G = (N,E)$

$N$ = set of routers = \{ 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) \}
Graph abstraction: costs

c(x, x') = cost of link (x, x')
e.g., c(w, z) = 5

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

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

key question: what is the least-cost path between u and z?

routing algorithm: algorithm that finds that least cost path
Routing algorithm classification

**Q: global or decentralized information?**

*global:*
- all routers have complete topology, link cost info
- “link state” algorithms

*decentralized:*
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

**Q: static or dynamic?**

*static:*
- routes change slowly over time

*dynamic:*
- routes change more quickly
  - periodic update
  - in response to link cost changes
A Link-State Routing Algorithm

**Dijkstra’s algorithm**
- net topology, link costs known to all nodes
  - accomplished via “link state broadcast”
  - all nodes have same info
- computes least cost paths from one node (‘source’) to all other nodes
  - gives *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.’s

**notation:**
- \( c(x,y) \): link cost from node \( x \) to \( y \); \( = \infty \) if not direct neighbors
- \( D(v) \): current value of cost of path from source to dest. \( v \)
- \( p(v) \): predecessor node along path from source to \( v \)
- \( N' \): set of nodes whose least cost path definitively known
Dijkstra’s Algorithm

1 **Initialization:**
2 \( N' = \{u\} \)
3 for all nodes \( v \)
4 if \( v \) adjacent to \( u \)
5 then \( D(v) = c(u,v) \)
6 else \( D(v) = \infty \)
7
8 **Loop**
9 find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
10 add \( w \) to \( N' \)
11 update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \):
12 \[ D(v) = \min( D(v), D(w) + c(w,v) ) \]
13 /* new cost to \( v \) is either old cost to \( v \) or known shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
15 **until all nodes in \( N' \)**
Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v)</th>
<th>D(w)</th>
<th>D(x)</th>
<th>D(y)</th>
<th>D(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>p(v)</td>
<td>p(w)</td>
<td>p(x)</td>
<td>p(y)</td>
<td>p(z)</td>
</tr>
<tr>
<td>0</td>
<td>u</td>
<td>7,u</td>
<td>3,u</td>
<td>5,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>uw</td>
<td>6,w</td>
<td>5,u</td>
<td>11,w</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uwx</td>
<td>6,w</td>
<td></td>
<td>11,w</td>
<td>14,x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uwxv</td>
<td>6,w</td>
<td></td>
<td>10,v</td>
<td>14,x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uwxvy</td>
<td></td>
<td></td>
<td></td>
<td>12,y</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uwxvzy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

notes:
- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)
Dijkstra’s algorithm: another example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v),p(v)</th>
<th>D(w),p(w)</th>
<th>D(x),p(x)</th>
<th>D(y),p(y)</th>
<th>D(z),p(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u</td>
<td>2,u</td>
<td>5,u</td>
<td>1,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram illustrates the network with nodes and edges labeled with weights. The table shows the progress of the Dijkstra algorithm at each step, updating the distances (D) and predecessors (p) of each node.
Dijkstra’s algorithm: example (2)

resulting shortest-path tree from u:

resulting forwarding table in u:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>(u,v)</td>
</tr>
<tr>
<td>x</td>
<td>(u,x)</td>
</tr>
<tr>
<td>y</td>
<td>(u,x)</td>
</tr>
<tr>
<td>w</td>
<td>(u,x)</td>
</tr>
<tr>
<td>z</td>
<td>(u,x)</td>
</tr>
</tbody>
</table>
Dijkstra’s algorithm, discussion

**algorithm complexity:** $n$ nodes
- each iteration: need to check all nodes, $w$, not in $N'$
- $n(n+1)/2$ comparisons: $O(n^2)$
- more efficient implementations possible: $O(n \log n)$

**oscillations possible:**
- e.g., suppose link cost equals amount of carried traffic:

Initially:

![Initial Graph]

Given these costs, find new routing.... resulting in new costs

![Next Graph]

Given these costs, find new routing.... resulting in new costs

![Next Graph]

Given these costs, find new routing.... resulting in new costs

![Next Graph]
Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

\[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]

then

\[ d_x(y) = \min \{ c(x,v) + d_v(y) \} \]

- cost to neighbor \( v \)
- cost from neighbor \( v \) to destination \( y \)
- \( \min \) taken over all neighbors \( v \) of \( x \)
Bellman-Ford example

clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z) \}$$

$$= \min \{ 2 + 5, 1 + 3, 5 + 3 \} = 4$$

node achieving minimum is next hop in shortest path, used in forwarding table
Distance vector algorithm

• $D_x(y)$ = estimate of least cost from $x$ to $y$
  – $x$ maintains distance vector $D_x = [D_x(y): y \in N ]$

• node $x$:
  – knows cost to each neighbor $v$: $c(x,v)$
  – maintains its neighbors’ distance vectors. For each neighbor $v$, $x$ maintains $D_v = [D_v(y): y \in N ]$
Distance vector algorithm

key idea:

• from time-to-time, each node sends its own distance vector estimate to neighbors
• when $x$ receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$
Distance vector algorithm

iterative, asynchronous:
  each local iteration caused by:
  • local link cost change
  • DV update message from neighbor

distributed:
  • each node notifies neighbors only when its DV changes
    – neighbors then notify their neighbors if necessary

each node:

wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, notify neighbors
<table>
<thead>
<tr>
<th>from</th>
<th>node x table</th>
<th>cost to</th>
<th>node y table</th>
<th>cost to</th>
<th>node z table</th>
<th>cost to</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

The diagram shows a network with nodes labeled x, y, and z, connected by edges with labels 2, 1, and 7. The text displays tables for the cost from each node to every other node.
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \]
\[ = \min\{2+0, 7+1\} = 2 \]
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2 \]

**Node x Table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>from y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

**Node y Table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

**Node z Table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \]
\[ = \min\{2+0 , 7+1\} = 2 \]
\[
D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \\
= \min\{2+0, 7+1\} = 2
\]

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

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

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

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \]
\[ = \min\{2+1, 7+0\} = 3 \]
Distance vector: link cost changes

*link cost changes:*
- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors

```
"good news travels fast"
```

$t_0$: $y$ detects link-cost change, updates its DV, informs its neighbors.

$t_1$: $z$ receives update from $y$, updates its table, computes new least cost to $x$, sends its neighbors its DV.

$t_2$: $y$ receives $z$’s update, updates its distance table. $y$’s least costs do *not* change, so $y$ does *not* send a message to $z$. 
Distance vector: link cost changes

- node detects local link cost change
- *bad news travels slow* - “count to infinity” problem!
- 44 iterations before algorithm stabilizes

Can reach X with cost of 6 (via Z)
Can reach X with cost of 8 (via Z)
Can reach X with cost of 7 (via Y)
Can reach X with cost of 9 (via Y)

\[ \begin{align*}
\text{Can reach X with cost of} & \quad \text{6 (via Z)} \\
\text{Can reach X with cost of} & \quad \text{8 (via Z)} \\
\text{Can reach X with cost of} & \quad \text{7 (via Y)} \\
\text{Can reach X with cost of} & \quad \text{9 (via Y)}
\end{align*} \]
Distance vector: link cost changes

link cost changes:
- node detects local link cost change
- *bad news travels slow* - “count to infinity” problem!
- 44 iterations before algorithm stabilizes

poisoned 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)
- will this completely solve count to infinity problem?
Comparison of LS and DV algorithms

**message complexity**
- **LS:** with $n$ nodes, $E$ links, $O(nE)$ msgs sent
- **DV:** exchange between neighbors only
  - convergence time varies

**speed of convergence**
- **LS:** $O(n^2)$ algorithm requires $O(nE)$ msgs
  - may have oscillations
- **DV:** convergence time varies
  - may be routing loops
  - count-to-infinity problem

**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
  - error propagate thru network