The Network Layer: Part II

These slides are adapted from those provided by Jim Kurose and Keith Ross with their book “Computer Networking: A Top-Down Approach (6th edition).”
Outline

✓ Network layer functions, mainly forwarding and routing
✓ Network layer services
✓ Datagram vs. Virtual circuit networks
✓ Router architectures and design issues
✓ IPv4 (incl. fragmentation)
✓ Internet addressing, DHCP and NAT
  ❖ IPv6
  ❖ ICMP
  ❖ Routing algorithms (link state, distance vector, hierarchical)
  ❖ Routing in the Internet (OSPF, BGP)
IPv6 Motivations

- **initial motivation:** 32-bit IPv4 address space was getting used up quickly.

- additional motivations:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

- **IPv6:**
  - IP address size increased from 32 bits to 128 bits
  - fixed-length 40 byte header
  - fragmentation not allowed, no header checksum, options left out of the standard header
  - flow labels and priorities
**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>
<tr>
<td>payload len</td>
<td>next hdr</td>
<td>hop limit</td>
</tr>
<tr>
<td>source address</td>
<td></td>
<td>(128 bits)</td>
</tr>
<tr>
<td>destination address</td>
<td></td>
<td>(128 bits)</td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 bits
IPv4 Datagram Format

- **IP protocol version number**
- **Header length (bytes)**
- **“Type” of data**
- **16-bit identifier**
- **Time to live**
- **Upper layer header**
- **Upper layer protocol to deliver payload to**
- **32-bit source IP address**
- **32-bit destination IP address**
- **Options (if any)**
- **Data** (variable length, typically a TCP or UDP segment)
- **Checksum**
- **Fragment offset**
- **Flags**
- **Type of service**
- **Total datagram length (bytes)**
- **Max number of remaining hops** (decremented at each router)
- **Upper layer protocol for fragmentation/reassembly**
- **Example options**: timestamp, record route taken, specify list of routers to visit.
Transition from IPv4 to IPv6

- declaring a flag day to switch all routers from IPv4 to IPv6 impractical
- how will network operate with mixed IPv4 and IPv6 routers?

1. **dual-stack approach**: IPv6 capable routers also support IPv4
   - Shortcoming: protocol conversion between IPv6 and IPv4 packets causes loss of header fields

2. **tunneling approach**: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers
Tunneling Illustrated

logical view:
IPv6 IPv6
A ——— B
IPv6 IPv6
IPv4 tunnel connecting IPv6 routers
IPv6 IPv6
E ——— F
IPv6 IPv6

physical view:
IPv6 IPv6
A ——— B
IPv6 IPv6
IPv4
C ——— D
IPv4
IPv6 IPv6
E ——— F
IPv6 IPv6
Tunneling Illustrated

logical view:

physical view:

IPv4 tunnel connecting IPv6 routers

A-to-B: IPv6

B-to-C: IPv6 inside IPv4

E-to-F: IPv6
Outline

✓ Network layer functions, mainly forwarding and routing
✓ Network layer services
✓ Datagram vs. Virtual circuit networks
✓ Router architectures and design issues
✓ IPv4 (incl. fragmentation)
✓ Internet addressing, DHCP and NAT
✓ IPv6
   ◆ ICMP
   ◆ Routing algorithms (link state, distance vector, hierarchical)
   ◆ Routing in the Internet (OSPF, BGP)
Internet Control Message Protocol (ICMP)

- used by hosts & routers to communicate network-layer information
  - Typically for error reporting (e.g., unreachable host/network/port/protocol)
  - But other uses too (e.g., ping via echo request/reply)
- network-layer “above” IP:
  - ICMP messages 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>
ICMP Example: Traceroute

- source sends series of UDP segments to dest using an unlikely dest port number
  - first set has TTL=1
  - second set has TTL=2, and so on.
- when \( n \)th set of datagrams arrives at \( n \)th router:
  - router discards datagrams
  - sends back ICMP "TTL expired" messages (type 11, code 0) to source
  - ICMP messages include 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
ICMPv6

- New version of ICMP for IPv6
  - added new message types, e.g., “Packet Too Big”
  - includes multicast group management functions that were previously part of Internet Group Management Protocol (IGMP)
Outline

✓ Network layer functions, mainly forwarding and routing
✓ Network layer services
✓ Datagram vs. Virtual circuit networks
✓ Router architectures and design issues
✓ IPv4 (incl. fragmentation)
✓ Internet addressing, DHCP and NAT
✓ IPv6
✓ ICMP
  ❖ Routing algorithms (link state, distance vector, hierarchical)
  ❖ Routing in the Internet (OSPF, BGP)
Interplay between Routing & Forwarding

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

<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>

IP destination address in arriving packet’s header.
Graph Abstraction

graph: $G = (N, E)$

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

$E = \text{set of links} = \{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$
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  
or inversely related to congestion, or  
some other metric or combination thereof

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

Key question: what is the least-cost path between u and z  
(more generally, between a given pair of nodes/router)?
Routing algorithm: algorithm that finds that least cost path
Routing Algorithms: Various Classifications

Global vs. Decentralized state/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
- e.g., “distance vector” algorithms

Static vs. Dynamic

**static:**
- manual set routes, assume routes change at human timescales

**dynamic:**
- routes change more quickly in response to topology or load changes
  - periodic
  - event-driven, e.g., in response to link cost changes

Load sensitive vs. load insensitive
Link-State Routing Algorithms

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

**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
Dijsktra’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 \(D(v) = \infty\)

**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></td>
<td>10,v</td>
<td>14,x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uwxvy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12,y</td>
</tr>
<tr>
<td>5</td>
<td>uwxvz</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></td>
<td>3,y</td>
<td></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>4,y</td>
</tr>
<tr>
<td>5</td>
<td>uxyv wz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph](image)
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 (not a unique problem with LS/Dijkstra though):**
- e.g., suppose link cost equals amount of carried traffic:

  Initially:
  
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>e</td>
<td>1</td>
</tr>
</tbody>
</table>

  given these costs, find new routing…. resulting in new costs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+e</td>
<td>0</td>
<td>0</td>
<td>1+e</td>
<td>1</td>
</tr>
</tbody>
</table>

  given these costs, find new routing…. resulting in new costs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1+e</td>
<td>0</td>
</tr>
</tbody>
</table>

  given these costs, find new routing…. resulting in new costs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+e</td>
<td>0</td>
<td>0</td>
<td>1+e</td>
<td>0</td>
</tr>
</tbody>
</table>
Distance Vector Algorithms

A class of decentralised routing algorithms that are based on *Bellman-Ford equation (dynamic programming)*

let

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

then

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

\( c(x,v) \) cost to neighbor \( v \)
\( d_v(y) \) 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:
\[
d_u(z) = \min \{ \ c(u,v) + d_v(z), \ c(u,x) + d_x(z), \ c(u,w) + d_w(z) \ \}
\]
\[
= \min \{ 2 + 5, \ 1 + 3, \ 5 + 3 \} \ = 4
\]

neighbour providing minimum distance estimate is chosen as the next hop and used in forwarding table
**Distance Vector Algorithm**

- **node x:**
  - knows cost to each neighbor v: \( c(x,v) \)
  - maintains its neighbours’ distance vectors. For each neighbour v, x maintains  
    \[ D_v = [D_v(y): y \in N] \]

- \( D_x(y) = \) estimate of least cost from x to y
  - x computes distance vector \( D_x = [D_x(y): y \in N] \) based on \( c(x,v) \) and \( D_v \) from all neighbours v using the Bellman-Ford equation
key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbours
- when x receives new DV estimate from neighbour, 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 (3)

**iterative, asynchronous:** each local iteration caused by:
- local link cost change
- DV update message from neighbour

**distributed:**
- each node notifies neighbours only when its DV changes
  - neighbours then notify their neighbours if necessary

**each node:**
- wait for (change in local link cost or msg from neighbour)
- recompute estimates
- if DV to any dest has changed, notify neighbours
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2 \]

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

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \]
\[ = \min\{2+1 , 7+0\} = 3 \]
Distance Vector: Link Cost Changes

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

```
t_0: y detects link-cost change, updates its DV, informs its neighbours.
t_1: z receives update from y, updates its table, computes new least cost to x, sends its neighbours 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.
```

“good news travels fast”
**Distance Vector: Link Cost Changes**

**link cost changes:**
- node detects local link cost change
- *bad news travels slow* – “counting to infinity” problem!
- 44 iterations before algorithm stabilizes for the example on the right

**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 counting to infinity problem?
Comparison of LS and DV Algorithms

_message complexity_

- **LS**: with \( n \) nodes, \( E \) links, \( O(nE) \) msgs sent
- **DV**: exchange between neighbours only
  - convergence time varies

_speed of convergence_

- **LS**: \( O(n^2) \) algorithm requires \( O(nE) \) msgs
  - may have oscillations if metric load sensitive (same applies for vanilla DV too)
- **DV**: convergence time varies
  - routing loops possible, e.g., counting-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 \( \Rightarrow \) errors propagate through the network
Hierarchical Routing

our routing study thus far assumes:
- all routers identical
- network structure “flat”
  … not true in practice

scale: with 600 million destinations:
- can’t store all destinations in routing tables!
- routing table exchange would swamp links!

administrative autonomy
- internet = network of networks
- each network admin may want to control routing in its own network
Hierarchical Routing

- aggregate routers into regions, “autonomous systems” (AS)
- routers in same AS run same routing protocol
  - “intra-AS” routing protocol
  - routers in different AS can run different intra-AS routing protocols
- gateway router:
  - at “edge” of its own AS
  - has link to router in another AS
Interconnected ASes

- Forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS sets entries for internal dests
  - inter-AS & intra-AS sets entries for external dests
Inter-AS Tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

  **AS1 must:**
  1. learn which dests are reachable through AS2, which through AS3
  2. propagate this reachability info to all routers in AS1

  *job of inter-AS routing!*
Example: setting forwarding table in router 1d

- suppose AS1 learns (via inter-AS protocol) that subnet $x$ reachable via AS3 (gateway 1c), but not via AS2
  - inter-AS protocol propagates reachability info to all internal routers
- router 1d determines from intra-AS routing info that its interface $I$ is on the least cost path to 1c
  - installs forwarding table entry $(x,I)$
Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet $x$ is reachable from AS3 and from AS2.
- to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest $x$
  - this is also job of inter-AS routing protocol!
Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet $x$ is reachable from AS3 and from AS2.
- to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest $x$
  - this is also job of inter-AS routing protocol!
- **hot potato routing:** send packet towards closest of two routers.

![Diagram showing the process of choosing a gateway for forwarding packets](image)
Outline

✓ Network layer functions, mainly forwarding and routing
✓ Network layer services
✓ Datagram vs. Virtual circuit networks
✓ Router architectures and design issues
✓ IPv4 (incl. fragmentation)
✓ Internet addressing, DHCP and NAT
✓ IPv6
✓ ICMP
✓ Routing algorithms (link state, distance vector, hierarchical)
   ✷ Routing in the Internet (RIP, OSPF, BGP)
Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary)
RIP (Routing Information Protocol)

- included in BSD-UNIX distribution in 1982
- distance vector algorithm
  - distance metric: # hops (max = 15 hops), each link has cost 1
  - DVs exchanged with neighbors every 30 sec in response message (aka advertisement)
  - each advertisement: list of up to 25 destination subnets (in IP addressing sense)

From router A to destination subnets:

<table>
<thead>
<tr>
<th>subnet</th>
<th>hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>z</td>
<td>2</td>
</tr>
</tbody>
</table>
RIP: example

Routing table in router D

<table>
<thead>
<tr>
<th>destination subnet</th>
<th>next router</th>
<th># hops to dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>x</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
RIP: example

A-to-D advertisement

**dest** | **next** | **hops**
---|---|---
**w** | - | 1
**x** | - | 1
**z** | **C** | 4

routing table in router D

<table>
<thead>
<tr>
<th><strong>destination subnet</strong></th>
<th><strong>next router</strong></th>
<th><strong># hops to dest</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>w</strong></td>
<td><strong>A</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td><strong>B</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>z</strong></td>
<td><strong>B</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
RIP: link failure, recovery

if no advertisement heard after 180 sec --> neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly (?) propagates to entire net
- *poison reverse* used to prevent ping-pong loops (infinite distance = 16 hops)
RIP table processing

- RIP routing tables managed by *application-level* process called route-d (daemon)
- advertisements sent in UDP packets, periodically repeated