Two key network-layer functions

- **forwarding**: move packets from router’s input to appropriate router output

- **routing**: determine route taken by packets from source to dest.
  - **routing algorithms**

**analogy:**
- **routing**: process of planning trip from source to dest
- **forwarding**: process of getting through single interchange
Interplay between routing and forwarding

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

Value in arriving packet’s header:

Routing algorithm

<table>
<thead>
<tr>
<th>Header Value</th>
<th>Output Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram:

- Value in arriving packet’s header is read by the router.
- The routing algorithm uses this header value to determine the output link.
- The local forwarding table then directs the packet to the correct output link.
Connection setup

- 3rd important function in some network architectures:
  - ATM, frame relay, X.25
- before datagrams flow, two end hosts and intervening routers establish virtual connection
  - routers get involved
- network vs transport layer connection service:
  - network: between two hosts (may also involve intervening routers in case of VCs)
  - transport: between two processes
Network service model

Q: What service model for “channel” transporting datagrams from sender to receiver?

example services for individual datagrams:
- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a flow of datagrams:
- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing
## Network layer service models:

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>Service Model</th>
<th>Guarantees?</th>
<th>Bandwidth</th>
<th>Loss</th>
<th>Order</th>
<th>Timing</th>
<th>Congestion feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>best effort</td>
<td>none</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no (inferred via loss)</td>
</tr>
<tr>
<td>ATM</td>
<td>CBR</td>
<td>constant rate</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no congestion</td>
</tr>
<tr>
<td>ATM</td>
<td>VBR</td>
<td>guaranteed rate</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no congestion</td>
</tr>
<tr>
<td>ATM</td>
<td>ABR</td>
<td>guaranteed minimum</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>UBR</td>
<td>none</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</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)\}

aside: graph abstraction is useful in other network contexts, e.g., P2P, where $N$ is set of peers and $E$ is set of TCP connections
Routing: graph abstraction

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, ..., 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 ?

**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
**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 \( D(v) = \infty \)

7. **Loop**
   8. find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
   9. add \( w \) to \( N' \)
   10. update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \):
       \[ D(v) = \min( D(v), D(w) + c(w,v) ) \]
       /* new cost to \( v \) is either old cost to \( v \) or known shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
   11. until all nodes in \( N' \)
## Dijkstra’s algorithm: 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>4, y</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2, u</td>
<td>3, y</td>
<td>3, 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></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>
Distance vector algorithm

**Bellman-Ford equation (dynamic programming)**

let

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

then

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

\( c(x,v) \) is the cost to neighbor \( v \) of \( x \)

\( d_v(y) \) is the cost from neighbor \( v \) to destination \( y \)

\( \min \) taken over all neighbors \( v \) of \( x \)
clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

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

neighboring 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 \( y \): \( c(x,v) \)
  - 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:
  \[
  D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N
  \]
- under minor, natural conditions, the estimate $D_x(y)$ converge 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 of distance to all destinations, and neighbor on each shortest path
- if DV to any dest has changed, notify neighbors by sending new DV
\[
D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2
\]

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

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

**link cost changes:**
- node detects local link cost change
- *bad news travels slow* - “count to infinity” problem!
- 44 iterations before algorithm stabilizes: see text
Discussion of LS and DV algorithms