Graph Abstraction

Graph: \( G = \{N, E\} \)

\( N = \text{set of nodes} = \{u, v, w, x, y, z\} \)

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

\( c(u,v) = \text{cost of link} \ (u,v), \text{assume full-duplex (bidirectional)} \)

\( \cdot \text{ e.g., } c(w,z) = 5 \)

\( \cdot \text{ cost could always manually assigned, e.g., based on price} \)

\( \cdot \text{ could be hop count, or inversely related to bandwidth, reliability} \)

\( \cdot \text{ or could be dynamic, e.g., proportionally related to congestion} \)

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

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 algorithm?

Global or distributed, local information?

Global info:
\( \cdot \) all routers have complete topology, link cost info
\( \cdot \) “link state” algorithms

Local info:
\( \cdot \) routers know of only physically-connected neighbors, link costs to neighbors
\( \cdot \) iterative process of computation, exchange of info with neighbors
\( \cdot \) “distance vector” algorithms

Static or dynamic routing?

Static routing:
\( \Rightarrow \) routes change slowly over time

Dynamic routing:
\( \Rightarrow \) routes change more quickly
\( \Rightarrow \) periodic update in response to link cost changes
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

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: $D[x,y] := \text{cost of least-cost path from } x \text{ to } y$
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
Bellman’s Algorithm: Initial Values

<table>
<thead>
<tr>
<th>L</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>u</td>
<td>v</td>
<td>x</td>
<td>w</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>m</td>
<td>v</td>
<td>x</td>
<td>w</td>
<td>y</td>
<td>z</td>
<td>w</td>
</tr>
<tr>
<td>c(n,m)</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Initial values:

<table>
<thead>
<tr>
<th>D</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>w</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
<td>∞</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

l0: loopback

Bellman’s Algorithm: Example

```java
do {
    for each node 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:

<table>
<thead>
<tr>
<th>D</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>w</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
<td>∞</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>H</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>u</td>
<td>l0</td>
<td>l1</td>
<td>l2</td>
<td>l3</td>
<td>l4</td>
<td>l5</td>
</tr>
<tr>
<td>v</td>
<td>l1</td>
<td>l0</td>
<td>l4</td>
<td>l5</td>
<td>l6</td>
<td>l7</td>
</tr>
<tr>
<td>w</td>
<td>l2</td>
<td>l4</td>
<td>l0</td>
<td>l5</td>
<td>l6</td>
<td>l7</td>
</tr>
<tr>
<td>x</td>
<td>l2</td>
<td>l3</td>
<td>l6</td>
<td>l0</td>
<td>l5</td>
<td>l7</td>
</tr>
<tr>
<td>y</td>
<td>l2</td>
<td>l3</td>
<td>l4</td>
<td>l0</td>
<td>l5</td>
<td>l6</td>
</tr>
<tr>
<td>z</td>
<td>l0</td>
<td>l0</td>
<td>l0</td>
<td>l0</td>
<td>l0</td>
<td>l0</td>
</tr>
</tbody>
</table>

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

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

**Distributed DVA Implementation**

Each node \(i\):
- Knows the cost to each neighbor
  - Keeps entries of its table for local links
- Node \(i\) maintains \(D[i,*] = \{D[i,k]: k \in \mathbb{N}\}\)
- \(i\)'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 (col \(i\) of \(D\) and \(H\))

**Example:**
- \(u\)'s link table: \([l_i, l_z]\)
- \(u\)'s routing table:

<table>
<thead>
<tr>
<th>dest</th>
<th>(u)</th>
<th>(v)</th>
<th>(w)</th>
<th>(x)</th>
<th>(y)</th>
<th>(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(H)</td>
<td>(l_i)</td>
<td>(l_i)</td>
<td>(l_i)</td>
<td>(l_i)</td>
<td>(l_i)</td>
<td>(l_i)</td>
</tr>
</tbody>
</table>
- \(u\)'s route update/distance vector:

<table>
<thead>
<tr>
<th>dest</th>
<th>(u)</th>
<th>(v)</th>
<th>(w)</th>
<th>(x)</th>
<th>(y)</th>
<th>(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
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 route updates
- $t_2$: no further changes

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
**Propagating Link Breakage**

Scenario:
- At time $t_1$: link $AB$ breaks. $A$ and $B$ discover the failure, update their routing tables, and send out route updates.
- At time $t_2$: $C$, $D$, and $E$ receive $A$'s and $B$'s route updates and in turn update their own routing tables and send out their route updates.
- At time $t_3$: $A$, $B$, $D$, and $E$ receive route updates sent at time $t_2$, update their routing tables and send out their route updates.
- At time $t_4$: $A$, $B$, and $C$ receive route updates sent at time $t_3$, update their routing tables and send out their route updates.
- At time $t_5$: no more changes.

**Bouncing Effect**

One big problem with distributed DV algorithm:
- Bouncing Effect causes routing loop.

Example scenario (all examples are contrived, to illustrate problems):
- At time $t_1$: 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.
- At time $t_2$: $D$ gets an update from $E$, $A$ and $D$ both send out updates.
- At time $t_3$: link $BC$ breaks and $A$'s update arrives at $B$.
- At time $t_4$: $A$ and $E$ now goes through $B$ to $C$.
- Events at $t_3$ and $t_4$ 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.
Counting to ∞

As a means to resolve routing loop

Example scenario:
• \( t_3 \): 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 ∞

<table>
<thead>
<tr>
<th>time</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>( t_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

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 $\infty$

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

Split Horizon
Idea: don’t advertise reachability to next-hop neighbor

Example scenario:
• time $t_1$: link $DE$ breaks, $D$ sends out triggered update, $A$ sends out a periodic route update
• $t_2$: $D$ gets an update from $A$, all paths go through $A$. **Doesn’t send A reachability to B, C, and E because it uses A as next hop to get to these nodes**
• $t_3$: link $AB$ breaks and $D$’s update arrives at $A$, $A$ updates route table: only $D$ is now reachable
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

Split Horizon with Poisonous Reverse

Idea: advertise cost $\infty$ 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 $\infty$ because they’re using each other as next hop
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
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 $n = 2$

<table>
<thead>
<tr>
<th>Time</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$\infty$</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>10</td>
</tr>
<tr>
<td>$t_5$</td>
<td>11</td>
<td>$\infty$</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>$t_6$</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>$t_7$</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

Smaller hold-down period $n = 1$

<table>
<thead>
<tr>
<th>Time</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$\infty$</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$t_2$</td>
<td>3</td>
<td>$\infty$</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$\infty$</td>
<td>4</td>
<td>$\infty$</td>
<td>4</td>
</tr>
<tr>
<td>$t_4$</td>
<td>5</td>
<td>$\infty$</td>
<td>5</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>10</td>
</tr>
</tbody>
</table>

Route Poisoning

Idea: advertise cost $\infty$ 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 $\infty$ to detect loop, they differ only in convergence time
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:

<table>
<thead>
<tr>
<th>dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>path</td>
<td>A</td>
<td>AB</td>
<td>ABC</td>
<td>AD</td>
<td>ADE</td>
</tr>
</tbody>
</table>

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
### Comparison of LS and DV Routing

**Message complexity**
- **LS**: with n nodes, E links, $O(nE)$ messages sent
- **DV**: exchange between neighbors only

**Speed of Convergence**
- **LS**: relatively fast
- **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 propagates)

### Similarities of LS and DV Routing

**Shortest-path routing**
- Metric-based, using link weights
- Routers share a common view of how good a path is

As such, commonly used inside an organization
- RIP and OSPF are mostly used as intradomain protocols
- E.g., AT&T uses OSPF

But the Internet is a “network of networks”
- How to stitch the many networks together?
- When networks may not have common goals
- ... and may not want to share information