Graph Abstraction

Graph: $G = (N, E)$
$N = \text{set of nodes} = \{u, v, w, x, y, z\}$
$E = \text{set of links} = \{(u,v), (u,x), (v,w), (x,w), (w,y), (w,z), (y,z)\}$
$c(u,v) = \text{cost of link } (u,v), \text{ assume full-duplex (bidirectional)}$
  - e.g., $c(w,z) = 5$
  - cost could always manually assigned, e.g., based on price
  - could be hop count, or inversely related to bandwidth, reliability
  - or could be dynamic, e.g., proportionally related to congestion

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

Routing Algorithm Classification

Centralized or decentralized algorithm?
Global or distributed, local information?
Global info:
  - all routers have complete topology, link cost info
  - “link state” algorithms
Local info:
  - routers know of only physically-connected neighbors, link costs to neighbors
  - iterative process of computation, exchange of info with neighbors
  - “distance vector” algorithms

Static or dynamic routing?
Static routing:
  ⇒ routes change slowly over time
Dynamic routing:
  ⇒ routes change more quickly
  ⇒ 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 a 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 $D[\cdot]$ encodes shortest path

Define: $D[x,y] := \text{cost of least-cost path from } x \text{ to } y$
Then: $D[x,y] = \min \{c(v,x) + D[v,y]\}$, where $v$ is a neighbor of $x$ and min is taken over all neighbors of $x$

Define two other tables:
  = $L[\cdot]$: link table
  = $H[\cdot]$: next hop table
Bellman's Algorithm: Initial Values

Initial values:

<table>
<thead>
<tr>
<th></th>
<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>u</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>v</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>w</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>x</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>y</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>z</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
</tbody>
</table>

do: loopback

Bellman's Algorithm: Example

do {
    for each node i in graph do {
        for all node k not i in graph do {
            m = c(i,j)+D[j,k];
            if (m < D[i,k]) {
                D[i,k]= m;
                H[i,k]= index of k where m=i,m;
            }
        }
    }
} while there has been a change in D[];

Initial values:

<table>
<thead>
<tr>
<th></th>
<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>u</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>v</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>w</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>x</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>y</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>z</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
</tbody>
</table>

Bellman's Algorithm: Example

do {
    for each node i in graph do {
        for each j neighbor of i do {
            H[i,k]= index of k where m=i,m;
        }
    }
} while there has been a change in D[];

Initial values:

<table>
<thead>
<tr>
<th></th>
<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>u</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>v</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>w</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>x</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>y</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>z</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</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
Distributed DVA Implementation

Each node $i$:
- knows the cost to each neighbor
- keeps entries of $i$-table for local links
- Node $i$ maintains $D[i,j] = \{D[i,j], k + N\}$
- $i$'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:
- $i$'s link table: $[u,v]$.
- $i$'s routing table:

<table>
<thead>
<tr>
<th>dest</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$H$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

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 mag 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
**Scenario:**

- Time 1: Link A/B breaks; A and B discover failure, update table, and send out route updates
- Time 2: C, D, and E receive A’s an B’s route updates and in turn update their own routing table and send out their route updates
- Time 3: A, B, D, and E receive route updates sent at time 2, update their routing tables and send out their route updates
- Time 4: A, B, and C receive route updates sent at time 3, update their routing tables and send out their route updates
- Time 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):

- Time 1: Cost of link CE 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
- Time 2: D gets an update from E, A, and D both send out updates
- Time 3: Link BC breaks and A’s update arrives at B
- Time 4: A and E now go through B to C
- Events at 1, 3, and 5 iterate until cost of B to C through D = 10, at which time, E switches to the EC link
- Meanwhile, datagrams from A or B to C loop until TTL expires

**RIP (Routing Information Protocol)**

An implementation of distance vector algorithm
- Distributed with BSD-UNIX in 1982
- Distance metric: # of hops (max 15 hops, $\infty$ = 16 hops)
- Distance vectors sent/advertised once every 30 secs
  - Takes 8 minutes to count to $\infty$ 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 $\Rightarrow$ neighbor/links 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

Why Split Horizon is not a Solution

Example scenario:
- time \( t_2 \): 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 \( E \)
- \( t_2: D \) and \( E \) 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

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

Example scenario:
- time \( t_2 \): 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 \( A \). \( D \) doesn’t send \( A \) reachability to \( B, C \), and \( E \) because it uses \( A \) as next hop to get to these nodes

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
- \( t_2: D \) and \( E \) break simultaneously, right before updates from \( t_2 \) arrive, now \( A \) and \( D \) adopt each other as next hop to \( B, C \), and \( E \)
- \( t_2: D \) 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
Smaller hold-down period n = 1

Route Poisoning
Idea: advertise cost ∞ 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 ∞ 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>

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)

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

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