How to Build Routing Tables

- Consider network as a graph

- Every edge (link) has a weight
  - Weight indicates desirability of sending traffic over link
    - Cost
    - Congestion
    - Speed

- The basic problem is to find the shortest path between every pair of nodes

- The length of path = sum of weights of its edges

Shortest Path Routing

- Each node determines the next node on its path to some source node S

- Upon receiving a packet destined to S, the node sends the packet to the next node of the path (next-hop)

- For now assume one source for simplicity
**Basic Idea**

- **Start with a guess** \( d(s,v) = v's\) guess of its distance from source
- **Initially**
  \[
  d(s,s) = 0 \\
  d(s,v) = \infty
  \]
- **Relaxation**: Pick an edge \((u,v)\)
  
  If \( d(s,u) + w(u,v) < d(s,v) \)
  
  Then \( d(s,v) \leftarrow d(s,u) + w(u,v) \)
  
  (Get better estimate)
- **Repeat until no better estimates can be obtained**

**Convergence**

- Convergence is not always guaranteed
  - e.g. Consider a negative weight cycle

![Negative Weight Cycle Diagram]

Every time we go through cycle we get better estimate \( \Rightarrow \) solution does not exist.

- If no negative weight cycles \( \Rightarrow \) convergence
- If no zero-weight cycles \( \Rightarrow \) shortest path tree can be obtained if every node keeps track of where it heard best information
EXAMPLE

BUT WHEN CAN WE STOP PERFORMING RELAXATIONS?
IS THERE AN UPPER BOUND?

BELLMAN-FORD

GIVEN GRAPH $G = (V, E)$

INIT $d(s,v) = \infty \quad \forall \; v \neq s$
$d(s,s) = 0$

ALG.
for $i \leftarrow 1$ to $|V| - 1$
for each edge $(u,v) \in E$
do
if $d(s,u) + w(u,v) < d(s,v)$
then $d(s,v) \leftarrow d(s,u) + w(u,v)$
$f(s,v) \leftarrow u$

THIS CONVERGES IF NO NEGATIVE WEIGHT CYCLES
CONVERGENCE ARGUMENT

• Consider the shortest path from $S$ to $V$

• By property of shortest path, the path from $S$ to $V$ is also shortest path.

• Proof by induction: Distance to $V_1$ will stabilize after first round; distance to $V_2$ will stabilize after second round, etc...

• Since each path can have at most $|V|-1$ edges (no negative weight cycles), all distances will stabilize after $|V|-1$ rounds.

DISTRIBUTED BELLMAN-FORD

• Neighbor nodes exchange messages of the form $d(s,y)$

  "I am $y$ and I am at distance $d$ from source"

NODE $X$

INIT

- if $(x = S)$
  then $d(S,X) = 0$
- else $d(S,X) = \infty$

ALG.

- if $d(S,Y)$ received (from $Y$)
  then Relay $(x,y)$
- if $d(S,X)$ changes
  then Send $d(S,X)$ to neighbors
ALL PAIRS

• RUN BELLMAN-FORD FOR EVERY SOURCE IN PARALLEL
  • EVERY NODE IS A SOURCE

NODE \( x \)

**INIT.**
- \( d(x,x) = 0 \)
- \( d(v,x) = \infty \) (UNDEFINED)

**ALG.**
- If \( d(v,y) \) RECEIVED (FROM Y)
  - then \( \if d(v,y) + w(x,y) < d(v,x) \)
    - then \( d(v,x) \leftarrow d(v,y) + w(x,y) \)
    - \( f(v,x) \leftarrow y \)

  - if \( d(v,x) \) CHANGES
    - then SEND \( d(v,x) \) to ALL NEIGHBORS

LINK OR NODE FAILURE

• ASSUME NODE \( x \) CAN DISCOVER THAT LINK \( (x,y) \) IS DOWN
  - LINK DOWN
  - \( y \) DOWN (NO PERIODIC UPDATES)
  - NODE \( x \) CAN SEND A CONTROL PACKET & WAIT FOR ACK

• IN GENERAL \( w(x,y) \) CHANGES by \( \Delta \)

NODE \( x \)

for each \( v \) s.t. \( f(v,x) = y \) (PATH THROUGH \( y \))

- do \( d(v,x) \leftarrow d(v,x) + \Delta \)

- SEND \( d(v,x) \) TO ALL NEIGHBORS

NODE \( z \)

if \( d(v,x) \) RECEIVED & \( f(v,z) = x \)
then \( d(v,z) = d(v,x) + w(x,z) \)
**Algorithm**

**NODEX**

**INIT**
\[ d(x,x) = 0 \]

**Alg.**
if \( d(v,y) \) RECEIVED (from \( y \))
then if \( f(v,x) = y \)
then \( d(v,x) = d(v,y) + w(x,y) \)
else REBOX \((x,y,y)\)

if \( w(x,y) \) CHANGED BY \( \Delta \)
then for each \( v \) s.t. \( f(v,x) = y \)
do \( d(v,x) \leftarrow d(v,x) + \Delta \)

if \( d(v,x) \) CHANGES
then SEND \( d(v,x) \) to ALL NEIGHBORS

(*) AN ALTERNATIVE IS FOR NODEX TO KEEP THE SMALLEST DISTANCE HEARD SO FAR FOR EVERY \( v \) & WHERE IT HEARD IT FROM

**Counting to Infinity Problem**

\[ \begin{align*}
 & A \quad 1 \quad B \quad 1 \quad C \\
\end{align*} \]

- After stabilization

\[ \begin{array}{c|c}
 A & B \\
 \hline
 d(C,A) = 2 & d(C,B) = 2 \\
 f(C,A) = B & f(C,A) = C \\
\end{array} \]

- Assume link B-C FAILS & B DETECTS THAT & SET \( w(B,C) = \infty \) & INFORMS ITS NEIGHBORS (i.e. A)

- Depending on the timing of messages, A WEIRD BEHAVIOR CAN HAPPEN
### Counting to Infinity

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(c,A)$</td>
<td>2</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$f(c,A)$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(c,A)$</td>
<td>$\infty$</td>
<td>3</td>
</tr>
<tr>
<td>$f(c,A)$</td>
<td>$B$</td>
<td>$A$</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(c,A)$</td>
<td>4</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$f(c,A)$</td>
<td>$B$</td>
<td>$A$</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(c,A)$</td>
<td>$\infty$</td>
<td>5</td>
</tr>
<tr>
<td>$f(c,A)$</td>
<td>$B$</td>
<td>$A$</td>
</tr>
</tbody>
</table>

... 

No one really knows that C is not reachable.

### Solutions

- **Consider some large enough integer to be \( \infty \)**

- **Split horizon**: Don't send information to neighbor you learned it from
  - Unfortunately does not work in general for larger loops, works for above example (loop size 2)

- **Upon increase in weight of an edge), stop accepting messages for some time**
  - Makes it slower
**LINK STATE**

- EACH NODE KNOWS STATE OF ITS LINKS
- FLOOD THAT INFO SO THAT EACH NODE HAS A KNOWLEDGE OF WHOLE GRAPH
- EACH NODE COMPUTES THE SHORTEST PATH TREE ASSUMING IT'S SOURCE

**PHASE 1: FLOODING**

- EACH NODE ITS NEIGHBOR LIST & LINK STATES TO ALL ITS NEIGHBORS IN A MESSAGE CALLED LSP (LINK STATE PACKET)
  - NODE ID
  - LIST OF NEIGHBORS & LINK WEIGHTS TO REACH THEM
  - SEQUENCE # (AVOID STATE INFO)
  - TTL (TIME TO LIVE)
- IF NODE X RECEIVES MOST RECENT LSP FROM NODE Y IT UPDATES ITS KNOWLEDGE & FORWARDS A COPY ON ALL LINKS EXCEPT (X,Y) (TO STOP FLOODING EVENTUALLY) & OTHERWISE DICKARDS LSP

**DIJKSTRA'S ALGORITHM**

**PHASE 2:** COMPUTE SHORTEST PATH USING DIJKSTRA'S ALGORITHM

(FASTER THAN BELLMAN FORD, BUT ASSUMES ALL WEIGHTS > 0)

**INIT**

\[ d(s,s) = 0 \]
\[ d(s,u) = \infty \quad u \notin S \]

**ALL**

\[ S \leftarrow \emptyset \]

while \( (S \neq V) \)

choose \( u \notin S \) with \( \min \) \( d(s,u) \)

\[ S \leftarrow S \cup \{u\} \]

for each \( v \in S \) at \( (u,v) \in E \)

relax \( (u,v) \)
EXAMPLE

\[
\begin{array}{ccccccc}
 & A & B & C & D & E & F \\
\emptyset & 0 & \infty & \infty & \infty & \infty & \infty \\
A & 0 & 2 & \infty & 1 & \infty & \infty \\
A, D & 0 & 2 & 4 & 1 & 2 & \infty \\
A, D, B & 0 & 2 & 4 & 1 & 2 & \infty \\
A, D, B, E & 0 & 2 & 3 & 1 & 2 & 4 \\
A, D, B, E, C & 0 & 2 & 3 & 1 & 2 & 4 \\
A, D, B, E, C, D & 0 & 2 & 3 & 1 & 2 & 4 \\
\end{array}
\]

WHY POSITIVE WEIGHTS?

COUNTER EXAMPLE

\[
\begin{array}{cccc}
 & A & B & C \\
\emptyset & 0 & \infty & \infty \\
A & 0 & 2 & 1 \\
A, C & 0 & -1 & 1 \\
A, C, B & 0 & -1 & 1 \\
\end{array}
\]

d[A] = 0

d[B] = -1

d[C] = 1 (WRONG)
WHY IT WORKS?

- WHEN A VERTEX U IS CHOSEN TO GO INTO S, LET'S CALL THIS FREEZING THE VERTEX.

- CLAIM: WHEN U FREEZED \( \Rightarrow \) COMPUTED \( d(s,u) = \delta(s,u) \)
  WHERE \( \delta(s,u) \) IS CORRECT SHORTEST DISTANCE FROM S

PROOF: CONSIDER 1ST TIME VIOLATION OF CLAIM

\[ d(s,u) > \delta(s,u) \]

Shortest path from S to U MUST CROSS SET S
JUST BEFORE FREEZING U

CONTINUE

\[ d(s,x) = \delta(s,x) \quad (u \text{ is first violation}) \]

\[ \Rightarrow d(s,y) = \delta(s,y) \]

\[ s \rightarrow x \rightarrow y \text{ is shortest path from } S \text{ to } Y \text{ by property of a shortest path & edge } (x,y) \text{ is relaxed already} \]

Now if

\[ d(s,u) > \delta(s,u) \Rightarrow d(s,u) > \delta(s,y) + \delta(y,u) \]

\[ = d(s,y) \geq 0 \]

\[ d(s,u) > d(s,y) + \epsilon \Rightarrow \]

\[ d(s,u) > d(s,y) \]

SO U WOULD NOT BE CHOSEN FOR FREEZING.