Considerations in implementing routing algorithms

Problems with Distance Vector

- At the root of the problems is slow convergence (slower than LS)
- The major problem is the “count-to-infinity” problem
DV “count to infinity” problem

Initial State

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & 1 \text{to z} \\
\end{array}
\]

On next cycle, y discovers link is down

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & \times & \text{z} \\
\end{array}
\]

When y’s DV changes, y sends it to x

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & 4 \text{to z} & 3 \text{to z} \\
\end{array}
\]

y-z link goes down

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & \times & \text{z} \\
\end{array}
\]

At this point, y will re-calculate its distance to z

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & 3 \text{to z} \\
\end{array}
\]

From x’s DV, y will think x has another path to z, will add 1 and use it

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & 3 \text{to z} \\
\end{array}
\]

x recalculates dist to z as y’s dist +1

\[
\begin{array}{c}
\text{x} & \text{y} & \text{z} \\
2 \text{to z} & \times & \text{z} \\
\end{array}
\]

x and y will continue to increment their distances to “infinity”

Note that both nodes think the best path is through the other one. Packets will bounce between them until TTL expires.

Some ways to speed up convergence

- Set a small “infinity”
- Report entire path
- Split horizon
- Split horizon with poison reverse
- Hold Down
Set a small “infinity”

- The idea is to set a reasonably-sized number that you will consider to be “infinity” (say, the diameter of your network +1)
- This will bound the time lost in counting to infinity
- Problem: what happens when the network grows?

Report the entire path

- Instead of just advertising distances, routers advertise the entire path to the destination.
- If router A sees itself on router B’s path, it knows not to use that path.
- This fixes the problem, but it’s very expensive in terms of routing table storage and network bandwidth
Split Horizon

- Assume router A sends traffic to destination D through neighbor router B
- Under Split Horizon, when A sends its DV to B, it will not report its distance to D
- This cures some count-to-infinity problems, but not all. For example:

\[ X \rightarrow Y \rightarrow W \rightarrow Z \]

Split Horizon with Poison Reverse

- Instead of just not advertising distances to the neighbor node they came from, advertise $\infty$
Note: a similar-sounding thing

• “Route poisoning”: when a link fails, advertise its cost as $\infty$

Hold down

• When a link goes down, neighboring routers advertise its cost as $\infty$ for some period of time (The "hold down interval") before switching routes
• The idea is that the infinite cost will spread through the network so that the old distance will be dropped before the new one is advertised
• Problem: How well this works depends on the interval selected. AND, it slows down convergence instead of speeding it up.
DV algorithm: the bottom line

- Simple, easy to build, but slow convergence and count-to-infinity make it less favored than LS

Considerations in implementing the Link State algorithm
When is an LSP generated?

- When
  - Some refresh time has elapsed
  - The router detects a new neighbor
  - The router detects that a cost to a neighbor has changed
  - The router detects that a link has gone down

Some potential problems with LSP distribution (1)

- Different routers have different LSPs
  - This is acceptable for transient states, but not for long-term states
- To avoid this, the distribution strategy must ensure that LSPs reach all routers, even if:
  - Links or routers are broken
  - A router needs this LSP to know how to route it
Some potential problems with LSP distribution (2)

- LSP distribution can overwhelm the network with LSPs to the point that nothing else can be processed.

Basic flooding

- Each router copies each received LSPs and forwards it on every link except the one the LSP was received on
- This can lead to exponential growth of LSPs:
Improved flooding

- Each router keeps a copy of all received LSPs.
- When a router receives a duplicate LSP, it does not forward it.

A problem with improved Flooding

- Since LSPs can take different routes to get to another router, they can arrive out-of-order.
- How does a router know that the most recent LSP it received is the latest one?
LSP timestamping

- We could timestamp LSPs to show which order to put them in

- A problem:
  - An error (or an intruder) could cause a timestamp to show a time that is a long time in the future – all succeeding LSPs would be ignored
  - We could do a sanity check of received timestamps if each router’s clock was globally synchronized (or near-synchronized), but that might be harder than distributing LSPs

Sequence numbering

- The idea:
  - Each router gives a sequence number to the LSPs it generates. Numbers are assigned sequentially at each router
  - Receiving routers can detect outdated LSPs by comparing SN against the SN of the last-received LSP from that router
Some problems with sequence numbering

- Error can cause large SN
- Sequence number wrap-around can make newer LSP have smaller SN
- Router crash can make router forget next SN to use

- Need a fall-back method in case any of these problems happen

Sequence + Age schemes

- In addition to sequence number, add an “age” field to LSP
- When router generates an LSP, it sets age to some max value
- As LSP sits in a receiving router’s memory, the age field is continuously decremented
- An LSP with age=0 is replaced, regardless of sequence number
- LSPs with age=0 are not forwarded
That one has problems, too

- Due to wraparound, if a router malfunctions, you can have:
  \[ SN1 < SN2 < SN3 < SN1 \]
  When this happens, every LSP will be replaced (and the new one will be propagated)

- If that happens and the network is flooded with LSPs (this is likely in the above case), LSPs may be replaced before they can time out

- This happened in the ARPANET and crashed the network

The fix

- SNs do not wrap around, they are reset when they hit the max. Succeeding LSPs will be ignored by other routers until the previous LSP times out

- LSPs to be forwarded are buffered before queuing.
  - If an LSP is updated while it is in the buffer, it is overwritten – queues cannot fill with LSPs from one source

- LSPs are ACKed

- This method widely used (OSPF, PNNI, IS-IS)
Comparing DV and LS: memory

- Assume n neighbors and d destinations. Each router must store:
  - DV
    - Must keep a DV (length d) for each of n neighbors => $O(n^d)$
  - LS
    - Must keep an LSP (length n+) for each of d destinations (keep in mind that routers are addressable: "destinations" include routers) => $O(n^d)$
Comparing DV and LS: bandwidth

- Bandwidth usage is highly dependent on network topology
- Not a significant factor unless you are considering extreme situations

Comparing DV and LS: processing

- DV
  - All n DVs must be scanned => $O(n*d)$
- LS
  - Dijkstra’s algorithm dominates
    - $O(\text{number_links} \times \log d) \Rightarrow O(n \times d \log d)$
- Both types can be sped up for cases where only a few states have changed since last calculation
Comparing DV and LS: robustness

- Both DV and LS are vulnerable to some extent to problems and attacks
  - Router claims a link that doesn’t exist
  - Router claims no link where one exists
  - Oddball sequence numbering
  - Incorrect or omitted LSP forwarding
  - Incorrect age handling
  - Failure to ACK LSPs
  - Incorrect path calculation

Comparing DV and LS: convergence

- The principal performance difference
- When network situation changes, how long does it take for the information to be reflected everywhere?
- LS converges faster:
  - DV has looping problem – fixes are slower
  - DV must re-calculate distances before passing along data (LS forwards immediately)