EECS 122, Lecture 15

Kevin Fall

kfall@cs.berkeley.edu

Routing

- Algorithm to establish routing table to make widely distributed endpoints appear to be directly connected
- Key questions:
  - how to choose a best path?
  - How to scale to millions of users?
  - How to adapt to failures or changes?

Global and Local Knowledge

- Forwarding is a local decision, requiring only next-hop information
- Computation of “best route” requires global knowledge
- But global knowledge is challenging:
  - hard to collect, often out of date, and big
  - how to summarize in a locally-relevant way?

General Needs for Routing

- Compute optimal paths for each destination (need notion of optimal)
- Be robust in the case of failures/changes
- Minimize control message exchanges
- Minimize routing table space

Routing Pitfalls

- **Loops**: should local forwarding information be inconsistent with global topology, can form loops (black holes)
- **Oscillations**: dynamically adapting to load can shift load, lead to congestion, and repeat
- Unusual under normal operation, often due to user mis-configuration

Fundamental Choices

- Centralized or Distributed Routing
- Source-based versus Hop-by-hop
- Stochastic versus Deterministic paths
- Single or Multi-path
- Dynamic versus Static route selection

Routing for Packet Networks

- Internet doesn't have very predictable traffic flow, may have unreliable links, and not terribly much redundancy (compare vs phone network)
- Routers are assumed to know:
  - address of each neighbor
  - cost of reaching each neighbor

Choices in Internet Routing

- Centralized or Distributed Routing
- Source-based versus Hop-by-hop
- Stochastic versus deterministic paths
- Single or multi-path
- Dynamic versus static route selection

Distance Vector & Link-State Routing

- Distance-Vector
  - tell neighbors about distances to all destinations
  - node's computation depends on neighbors
- Link-State
  - tell all routers distance to each neighbor
  - each router computes its best paths
- Both are distributed algorithms

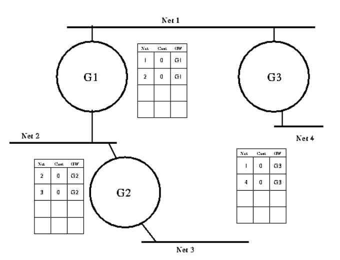
Distance Vector Operation

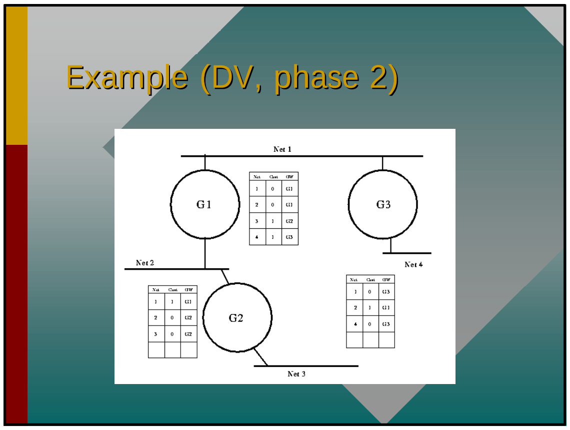
- Each router maintains a *distance vector* :
  - (dest, cost) tuple; one per destination
  - initialize with lowest costs to attached neighbors, and highest cost (infinity) to all non-neighbors
- Periodically send copy of distance vector to all neighbors

DV Route Computation

- Upon receiving a distance vector, compare current cost to destination with calculated cost using the sending router to reach the destination
- If neighbor path results in lower cost, switch
- Assuming no changes, eventually converges to proper shortest paths

Example (DV, phase 1)



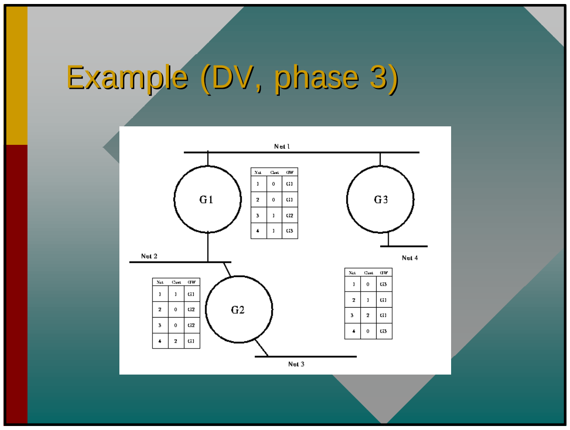


Example (DV, phase 2)

Net	Cost	GW
1	0	G1
2	0	G1
3	1	G2
4	1	G3

Net	Cost	GW
1	1	G1
2	0	G2
3	0	G2

Net	Cost	GW
1	0	G3
2	1	G1
4	0	G3

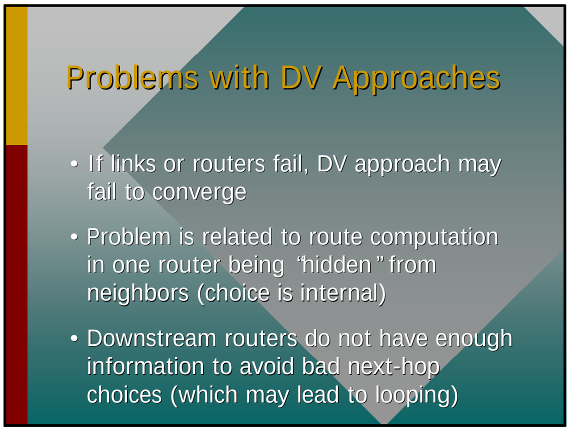


Example (DV, phase 3)

Net	Cost	GW
1	0	G1
2	0	G1
3	1	G2
4	1	G3

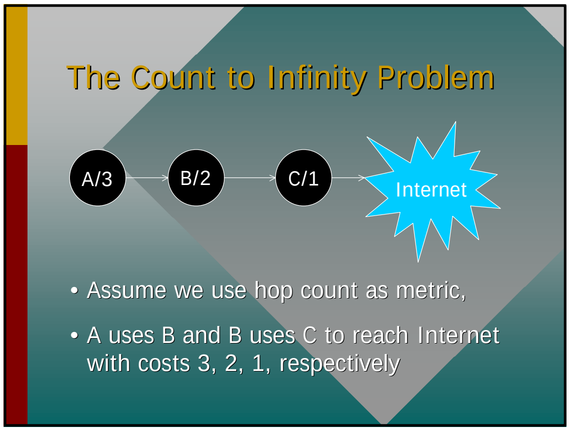
Net	Cost	GW
1	1	G1
2	0	G2
3	0	G2
4	2	G1

Net	Cost	GW
1	0	G3
2	1	G1
3	2	G1
4	0	G3



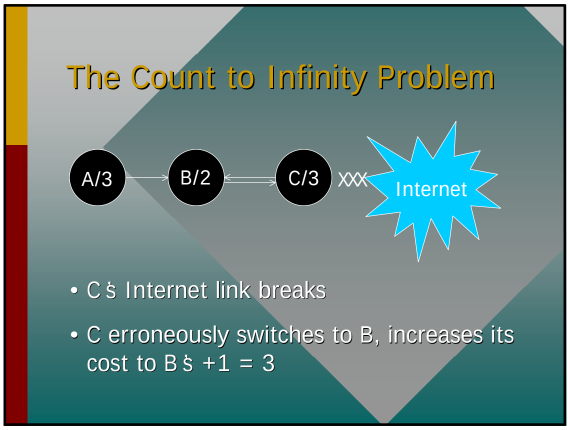
Problems with DV Approaches

- If links or routers fail, DV approach may fail to converge
- Problem is related to route computation in one router being ‘hidden” from neighbors (choice is internal)
- Downstream routers do not have enough information to avoid bad next-hop choices (which may lead to looping)



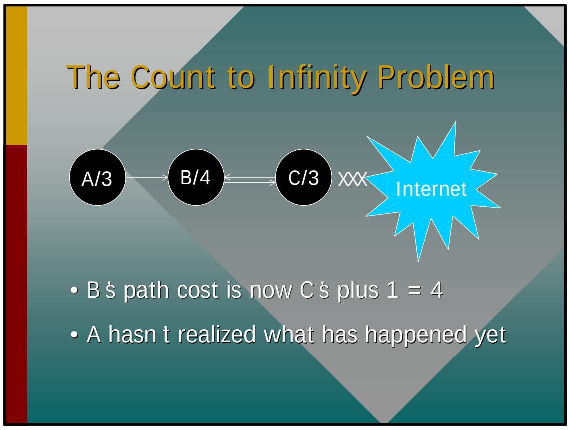
The Count to Infinity Problem

- Assume we use hop count as metric,
- A uses B and B uses C to reach Internet with costs 3, 2, 1, respectively



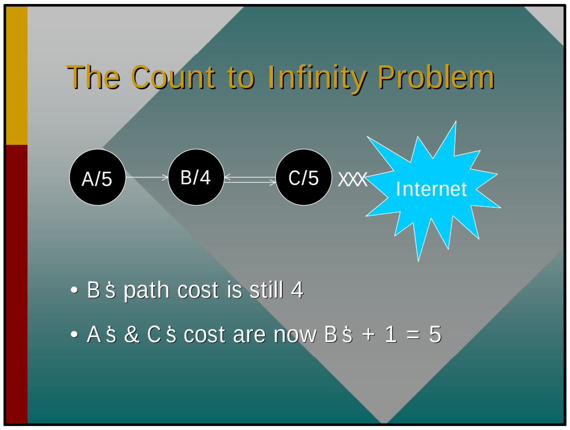
The Count to Infinity Problem

- C’s Internet link breaks
- C erroneously switches to B, increases its cost to B’s +1 = 3



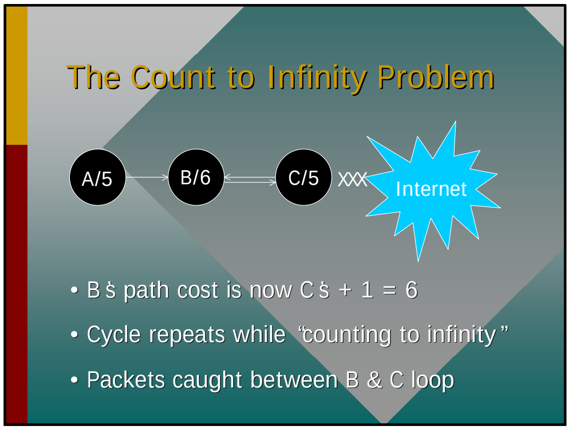
The Count to Infinity Problem

- B’s path cost is now C’s plus 1 = 4
- A hasn’t realized what has happened yet



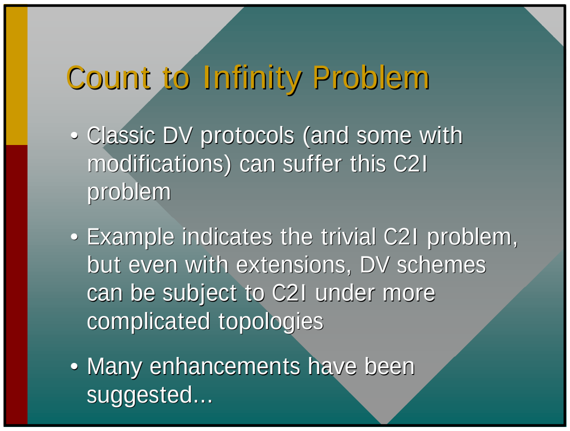
The Count to Infinity Problem

- B's path cost is still 4
- A's & C's cost are now B's + 1 = 5



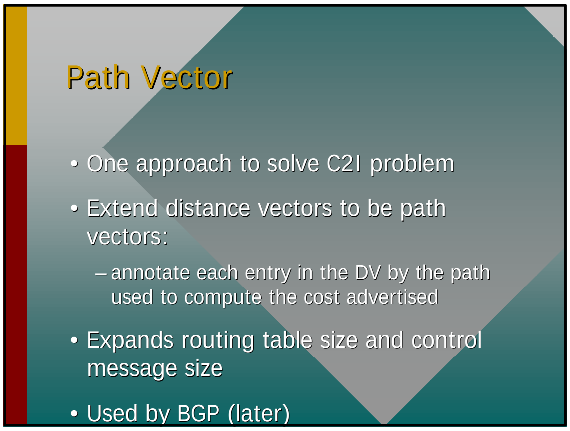
The Count to Infinity Problem

- B's path cost is now C's + 1 = 6
- Cycle repeats while "counting to infinity"
- Packets caught between B & C loop



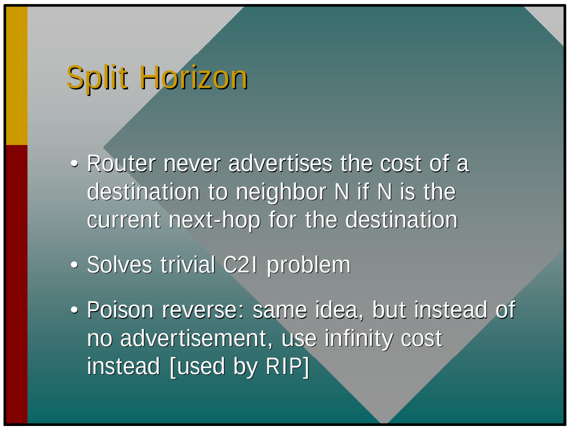
Count to Infinity Problem

- Classic DV protocols (and some with modifications) can suffer this C2I problem
- Example indicates the trivial C2I problem, but even with extensions, DV schemes can be subject to C2I under more complicated topologies
- Many enhancements have been suggested…



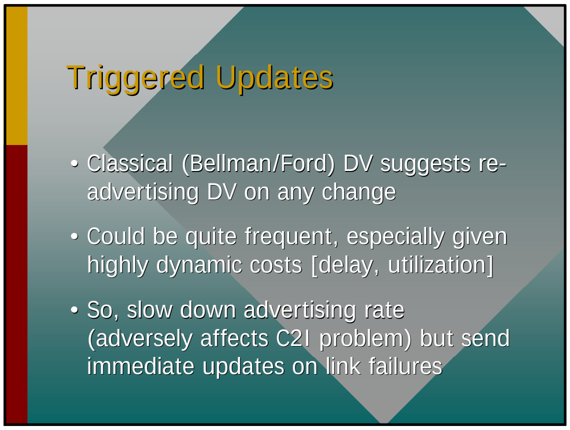
Path Vector

- One approach to solve C2I problem
- Extend distance vectors to be path vectors:
  - annotate each entry in the DV by the path used to compute the cost advertised
- Expands routing table size and control message size
- Used by BGP (later)



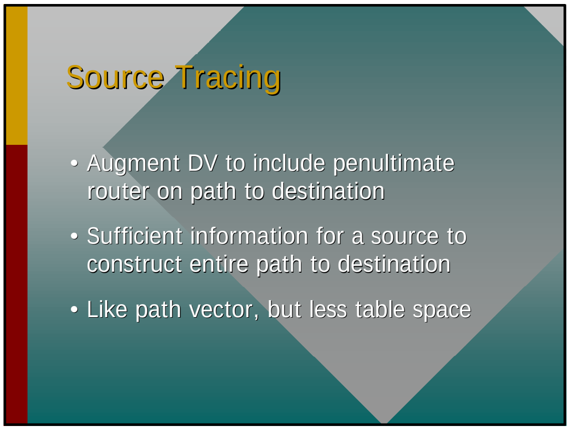
Split Horizon

- Router never advertises the cost of a destination to neighbor N if N is the current next-hop for the destination
- Solves trivial C2I problem
- Poison reverse: same idea, but instead of no advertisement, use infinity cost instead [used by RIP]



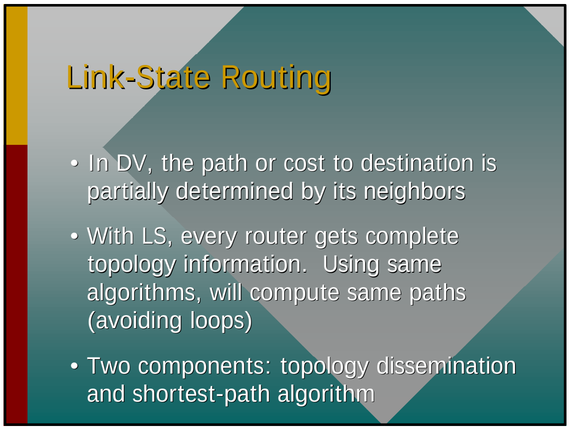
Triggered Updates

- Classical (Bellman/Ford) DV suggests re-advertising DV on any change
- Could be quite frequent, especially given highly dynamic costs [delay, utilization]
- So, slow down advertising rate (adversely affects C2I problem) but send immediate updates on link failures



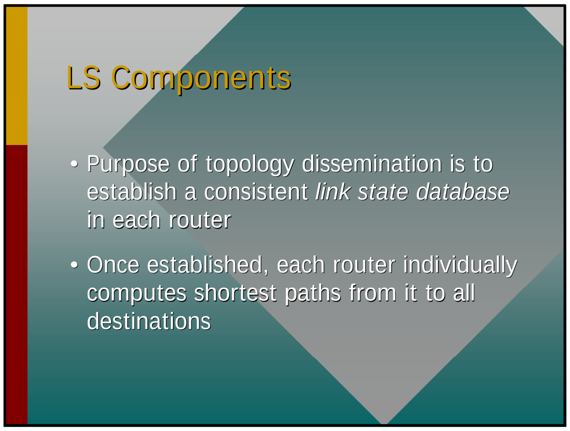
Source Tracing

- Augment DV to include penultimate router on path to destination
- Sufficient information for a source to construct entire path to destination
- Like path vector, but less table space



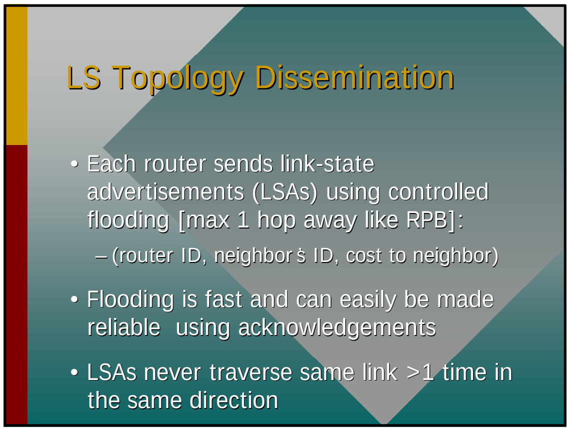
Link-State Routing

- In DV, the path or cost to destination is partially determined by its neighbors
- With LS, every router gets complete topology information. Using same algorithms, will compute same paths (avoiding loops)
- Two components: topology dissemination and shortest-path algorithm



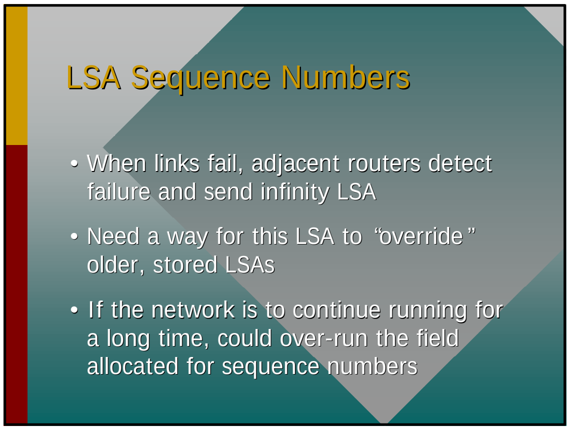
LS Components

- Purpose of topology dissemination is to establish a consistent *link state database* in each router
- Once established, each router individually computes shortest paths from it to all destinations



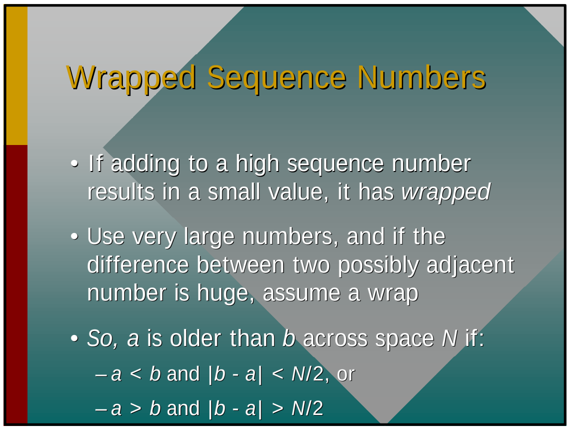
LS Topology Dissemination

- Each router sends link-state advertisements (LSAs) using controlled flooding [max 1 hop away like RPB]:
  - (router ID, neighbor's ID, cost to neighbor)
- Flooding is fast and can easily be made reliable using acknowledgements
- LSAs never traverse same link >1 time in the same direction



LSA Sequence Numbers

- When links fail, adjacent routers detect failure and send infinity LSA
- Need a way for this LSA to "override" older, stored LSAs
- If the network is to continue running for a long time, could over-run the field allocated for sequence numbers



Wrapped Sequence Numbers

- If adding to a high sequence number results in a small value, it has *wrapped*
- Use very large numbers, and if the difference between two possibly adjacent number is huge, assume a wrap
- *So,* a is older than b across space N if:
  - $a < b$ and $|b - a| < N/2$, or
  - $a > b$ and $|b - a| > N/2$

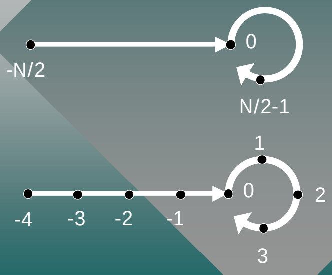
Bootstrapping Seq. Numbers

- What sequence number should a booting router use for its LSAs?
- Might risk flooding messages which are always ignored
- Clever Solution: *Lollipop Sequence Space [Perlman83]*

Lollipop Sequence Space

- better solution for newly-booted routers
- scheme where new seq number is unique from all others it could have used
- Partition space of size N into three parts:
  - [-N/2..0], 0, [N/2..N/2-1]
  - start with -N/2, then -N/2+1, etc...
  - once 0 reached, stay in circular part of space

Why Called Lollipop?



Lollipop Sequence Space Comparisons

- a is older than b on span N if:
  - a < 0 and a < b, or
  - a > 0, a < b, and (b - a) < N/4, or
  - a > 0, b > 0, and (a - b) > N/4
  - [makes -N/2 the oldest seq number]
- When receiving an old number, must inform sender of latest seq number

Partitioning

- A network partition (division) causes LS databases on the sides of the partition to diverge
- If the partition is repaired, simple exchange of updates are not sufficient; need to resynchronize entire LS database
- Version numbers used to identify which entries need to be exchanged

Database Description Records

- contain (link ID, version) pairs
- like LSAs but with less info, so cheaper to exchange
- allows routers to determine the set of records they lack or are out-of-date
- routers then request the records they need using request/response protocol

Link vs Router Failure

- link failures detected by routers which can flood this info directly
- most LS protocols use HELLO messages to detect router failure
- failure to respond to some number of the HELLO queries indicates failed router and causes flooding of corresponding info

Computing Shortest Paths

- once LSAs are reliably flooded, need to execute shortest path for all destinations
- Dijkstra's shortest path algorithm
  - computes shortest paths from root (local router) to all possible destinations
  - greedy algorithm which adds the least cost path to next candidate node to current shortest path set of nodes

Dijkstra's Shortest Path

- Two sets P (permanent) & T (temporary)
- P: in current SP set, T: not yet in set
  - P: initially current node, T: initially NULL
- Every node in T must be reachable by a path from a node in P
- Find every way to reach the T node from a P node; add min cost one to P, repeat

Dijkstra's Shortest Path

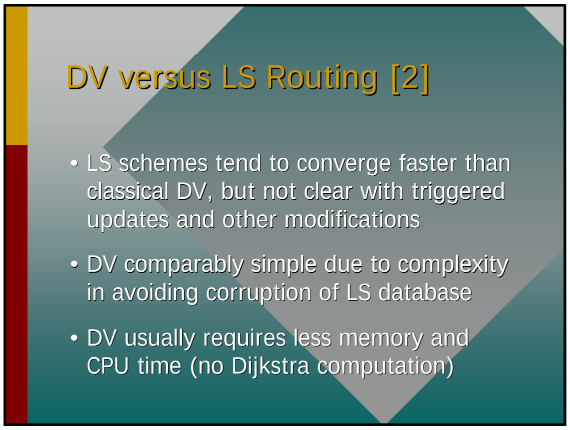
- More precisely:
  - For the node *p* just added to P, add each of its neighbors *n* to T such that:
    - if *n* is not in T, add it, annotating it with *p*'s ID and the cost to reach it through *p*
    - if *n* already in T and path to *n* through *p* has lower cost, remove earlier instance of *n* and add new instance annotated with *p*'s ID and cost to reach it through *p*
  - Pick the node *n* that has the smallest cost in T and, if not already in P, add it to P. Use its annotation to determine the router *p* to use to reach *n*. If T is empty, done.

Shortest Path

- Algorithm performs O((E +N) logN) [E: link, N: router]; if E remains relatively constant with increased N, just O(NlogN)
- At completion of algorithm, each router knows the router on the shortest path to reach it
- Can use technique like source-tracing to compute next hops for every destination

DV versus LS Routing

- Conventional wisdom is that LS is more stable and avoids loops better, but loops may form during topology changes
- Modifications to basic DV scheme takes care of loops
- LSAs might carry data using multiple metrics [also recall easy multicast]



DV versus LS Routing [2]

- LS schemes tend to converge faster than classical DV, but not clear with triggered updates and other modifications
- DV comparably simple due to complexity in avoiding corruption of LS database
- DV usually requires less memory and CPU time (no Dijkstra computation)