Network Routing I
(The Simple Case Without Failures)

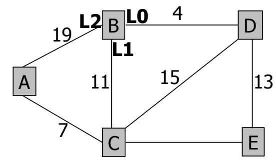
Lecture 20

6.02 Spring 2009

April 22, 2009

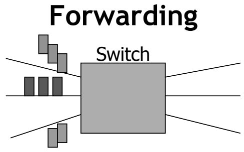
- Forwarding and routing
- Distance-vector protocol with Bellman-Ford step
- Link-state protocol with Dijkstra's shortest-paths

The Problem: Finding Paths



- How to find a good path (or paths) between any two nodes?
- Addressing (naming nodes)
- Forwarding (what a switch does when packet arrives)
- Routing (building and updating data structures to ensure that forwarding works)

Forwarding



- Core function is conceptually simple
  - lookup(dst_addr) in routing table returns *route* (i.e., *outgoing link*) for packet
  - enqueue(packet, link_queue)
  - send(packet) along outgoing link
- And do some book-keeping before enqueue
  - Decrement hop limit (TTL); if 0, discard packet
  - Recalculate checksum (in IP, header checksum)

Routing Table Structure

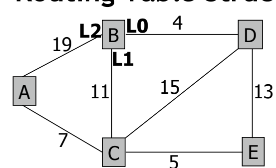


Table @ B

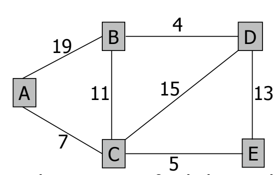
ROUTE

Destination	Link (next-hop)	Cost
A	L1	18
B	'Self'	0
C	L1	11
D	L0	4
E	L1	16

Why is Network Routing Hard?

- Inherently distributed problem
  - Information about links and neighbors is local to each node, but we want global reach
- Efficiency: want reasonably good paths, and must find them without huge overhead
- Handling failures and "churn" (next lecture)
  - Must tolerate link, switch, and network faults
  - Failures and recovery could be arbitrarily timed, messages could be lost, etc.
- Scaling to large size very hard (later courses)
  - And on the Internet, many independent, competing organizations must cooperate
  - Mobility makes the problem harder

Shortest Path Routing

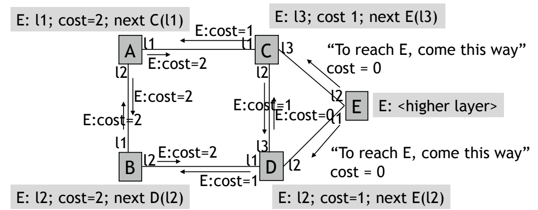


- Each node wants to find the path with *minimum total cost* to other nodes
  - We use the term "shortest path" even though we're interested in min cost (and not min #hops)
- Several possible approaches
  - *Vector protocols*
  - *Link-state protocols*

Distributed Routing: A Common Plan

- **Determining live neighbors**
  - HELLO protocol (periodic)
- **Advertisement step (periodic)**
  - Send some information to all neighbors
- **Integration step**
  - Compute routing table using info from advertisements

Distance Vector Routing



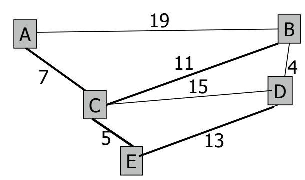
- Advertisement: Each node periodically announces a vector of <destination:pathcost> tuples to all its neighbors
- Integration: On hearing advertisement, run Bellman-Ford step:
  if (current cost to dest > cost in advertisement) then
  update cost, nexthop
- **More details in notes**

Link-State Routing

- HELLO protocol for neighbor liveness
- Advertisement step:
  - Information about its links to its neighbors
  - Neighbors re-send on *their* links → **flooding**
  - Result: Each node discovers map of the network
- Integration: Each node runs the same shortest path algorithm over its map
  - If each node implements computation correctly and each node has the same map, then routing tables will be correct

Integration Step: Dijkstra's alg

- Many algorithms: We'll study Dijkstra's

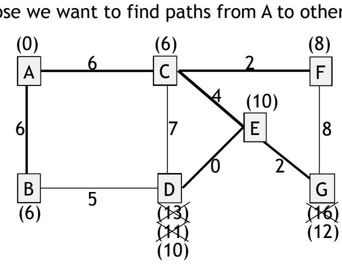


E	C	A	D	B
0	5	12	13	16
A	C	E	B	D
0	7	12	18	22
B	D	C	E	A
0	4	11	16	18
C	E	A	B	D
0	5	7	11	15
D	B	E	C	A
0	4	13	15	22

- Key property of shortest paths:
  Suppose shortest path from X to Y goes through Z. Then, the sub-path from X to Z must be a shortest path. [Why?]

Dijkstra's Algorithm Example

Suppose we want to find paths from A to other nodes



Link-State Advertisements and Flooding

- Periodically send LSA (Link-State Advertisement) [seq#, [(nbhr1, linkcost1), (nbhr2, linkcost2), …] to all neighbors
- If seq > last_heard:
  - save seq, LSA; rebroadcast LSA to neighbors
- LSAs aren't reliable messages, so periodic
- Periodic messages help handle dynamism: state in each node is "soft" and *times out if not refreshed*

[seq, (G, 8), (C, 2)]