λIntroduction to Networking

λMarcel Flores

λ

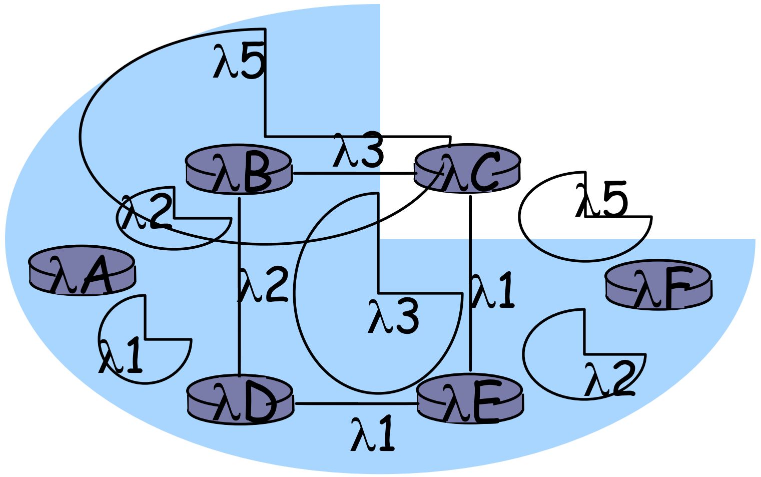
λRouting

λRouting protocol

λGoal: determine "good" path
λ(sequence of routers) thru
λnetwork from source to
dest.

λGraph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level



- "good" path:
  - typically means minimum cost path
  - other def's possible

λRouting Algorithm classification

- Global or decentralized information?
  - Global:
    - all routers have complete topology, link cost info
    - “link state” algorithms
  - Decentralized:
    - router knows physically-connected neighbors, link costs to neighbors
    - iterative process of computation, exchange of info with neighbors
    - “distance vector” algorithms

- Static or dynamic?
  - Static:
    - routes change slowly over time
  - Dynamic:
    - routes change more quickly
      - periodic update
      - in response to link cost changes

λA Link-State Routing Algorithm

λDijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives routing table for that node
- iterative: after k iterations, know least cost path to k dest.'s

λNotation:

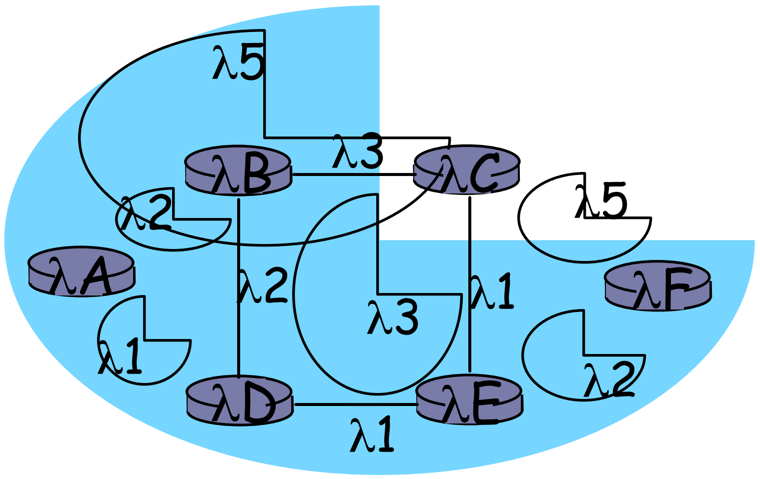
- c(i,j): link cost from node i to j. cost infinite if not direct neighbors
- D(v): current value of cost of path from source to dest. V
- p(v): predecessor node along path from source to v, that is next v
- N: set of nodes whose least cost path definitively known

λDijsktra's Algorithm

```
λ1  Initialization:
λ2    N = {A}
λ3    for all nodes v
λ4      if v adjacent to A
λ5        then D(v) = c(A,v)
λ6        else D(v) = infinity
λ7
λ8  Loop
λ9    find w not in N such that D(w) is a minimum
λ10   add w to N
λ11   update D(v) for all v adjacent to w and not in N:
λ12       D(v) = min( D(v), D(w) + c(w,v) )
λ13   /* new cost to v is either old cost to v or known
λ14    shortest path cost to w plus cost from w to v */
λ15 until all nodes in N
```

λDijkstra's algorithm: example

λStep	λstart N	λD(B),p(B)	λD(C),p(C)	λD(D),p(D)	λD(E),p(E)	λD(F),p(F)
λ0	λA	λ2,A	λ5,A	λ1,A	λinfinity	λinfinity
λ1	λAD	λ2,A	λ4,D		λ2,D	λinfinity
λ2	λADE	λ2,A	λ3,E			λ4,E
λ3	λADEB		λ3,E			λ4,E
λ4	λADEBC					λ4,E
λ5	λADEBCF					



λDistance Vector Routing Algorithm

λiterative:

- continues until no nodes exchange info.
- *self-terminating*: no “signal” to stop

λasynchronous:

- nodes need *not* exchange info/iterate in lock step!

λdistributed:

- each node communicates *only* with directly-attached neighbors

λDistance Table data structure

- each node has its own
- row for each possible destination
- column for each directly-attached neighbor to node
- example: in node X, for dest. Y via neighbor Z:

$\lambda D^{\lambda X}(Y,Z)$ λ= λdistance *from* X *to* λY, *via* Z as next hop

$$\lambda= \lambda c(X,Z) + \min_{\lambda w}\{D^{\lambda Z}(Y,w)\}$$

λDistance Vector Routing Algorithm

λiterative:

.continues until no nodes exchange info.

.*self-terminating*: no “signal” to stop

λasynchronous:

.nodes need *not* exchange info/iterate in lock step!

λdistributed:

.each node communicates *only* with directly-attached neighbors

λEach node:

λ*wait* for (change in local link cost of msg from neighbor)

λ*recompute* distance table

λif least cost path to any dest has changed, *notify* neighbors

- **Routing Lab**
- **Project 3**
  - A distance-vector algorithm and a link-state algorithm in the context of a simple routing simulator
  - Event-driven Simulation
  - main loop repeatedly pulls the earliest event from a queue and passes it to a handler until there are no more events in the queue.

- make TYPE=GENERIC" will build a single executable "routesim", which contains no routing algorithm.
- You will do TYPE=DISTANCEVECTOR and TYPE=LINKSTATE
- To run: ./routesim topologyfile eventfile [singlestep]

- Events in routesim come from the topology file, the event file, and from handlers that are executed in response to events.
- The topology file generally only contains events that construct the network topology (the graph)
  - arrival_time ADD_NODE node_num latency bandwidth
  - arrival_time DELETE_NODE node_num latency bandwidth
  - arrival_time ADD_LINK src_node_num dest_node_num latency bandwidth
  - arrival_time DELETE_LINK src_node_num dest_node_num latency bandwidth

- The event file generally only contains events that modify link characteristics in the graph, or draw the graph, a path and etc.
  - arrival_time CHANGE_NODE node_num latency bandwidth
  - arrival_time CHANGE_LINK src_node_num dest_node_num latency bandwidth
  - arrival_time DRAW_TOPOLOGY
  - arrival_time DRAW_TREE src_node_num

- Note that although each link event contains both bandwidth and latency numbers, your algorithms will determine shortest paths using only the link latencies.