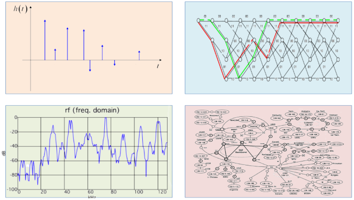
Routing Algorithms

6.02 Fall 2013 Lecture 20



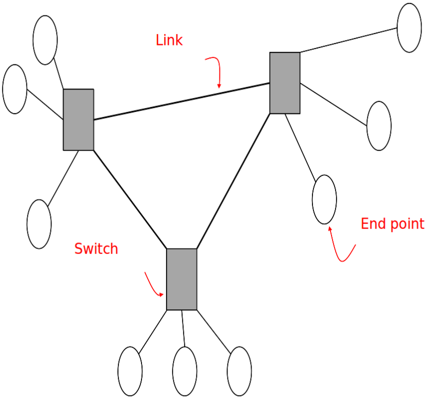
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

Today's Plan

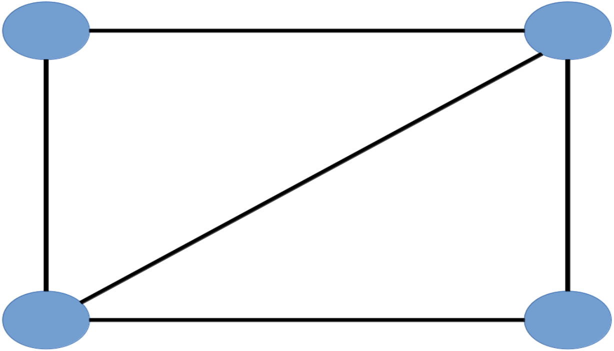
Routing Algorithms:

- Addressing, Forwarding, Routing
- Distance-Vector Algorithms
- Link-State Algorithms



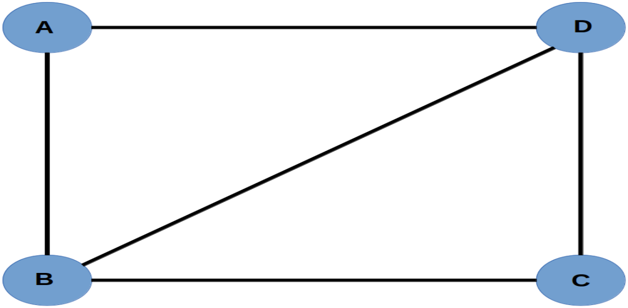
Network Model

Some nodes, some links . . .



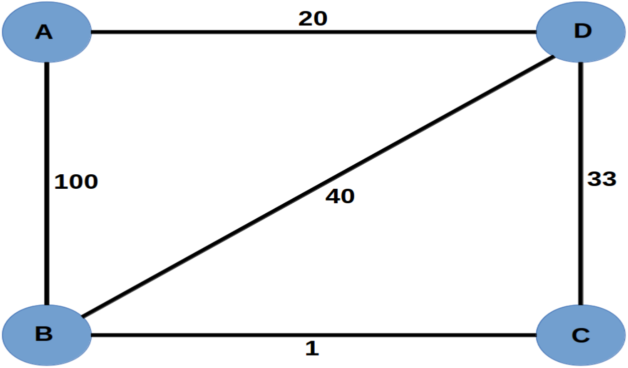
Network Model

- node names, i.e. **addresses**: A, B, C, D
- link names: ->A (from B or D), ->B (from A, D, or C), etc.



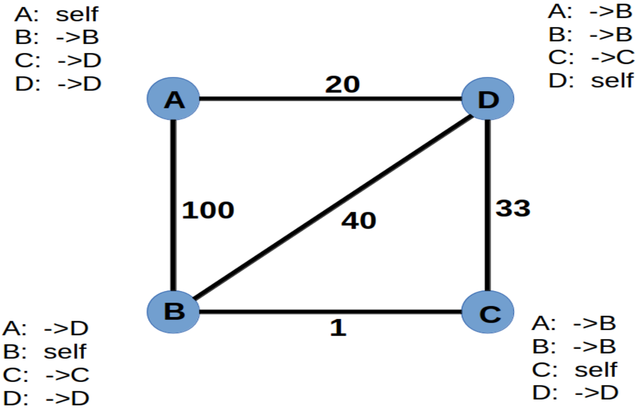
Network Model

Link costs: A->B is 100, D->B is 40, etc.



Network Model

Forwarding table: **incorrect**



A: self
B: ->B
C: ->D
D: ->D

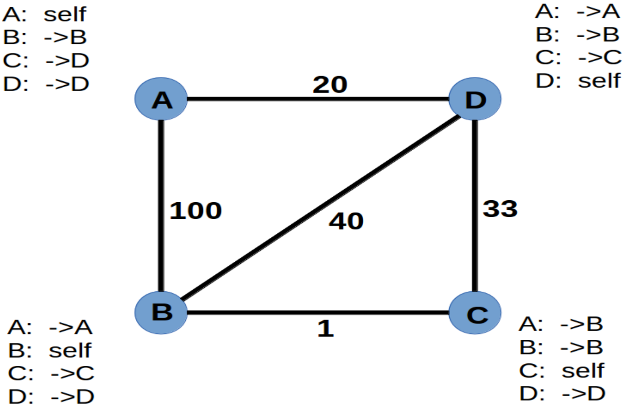
A: ->B
B: ->B
C: ->C
D: self

A: ->D
B: self
C: ->C
D: ->D

A: ->B
B: ->B
C: self
D: ->D

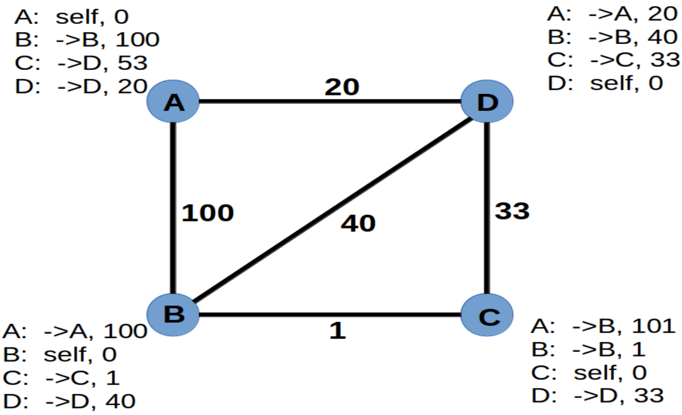
Network Model

Forwarding table: **correct**



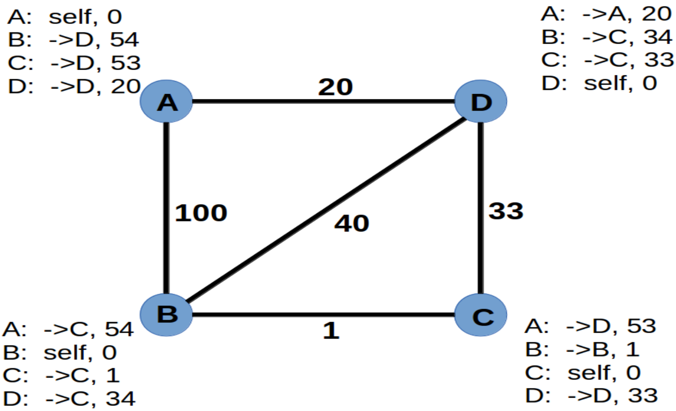
Network Model

Routing table: correct forwarding table with total costs



Network Model

Forwarding table: **optimal**



A: self, 0
B: ->D, 54
C: ->D, 53
D: ->D, 20

A: ->A, 20
B: ->C, 34
C: ->C, 33
D: self, 0

A: ->C, 54
B: self, 0
C: ->C, 1
D: ->C, 34

A: ->D, 53
B: ->B, 1
C: self, 0
D: ->D, 33

Routing Algorithms

Nodes

- know (current) neighbor links and costs (HELLO protocol)
- exchange simple info, repeatedly
- aim to to establish optimal routing ASAP

Basic ideas:

- distance-vector algorithms:
  maintain/exchange w/neighbors your routing distance tables
- link-state algorithms:
  memorize and forward neighbor/cost info for every node

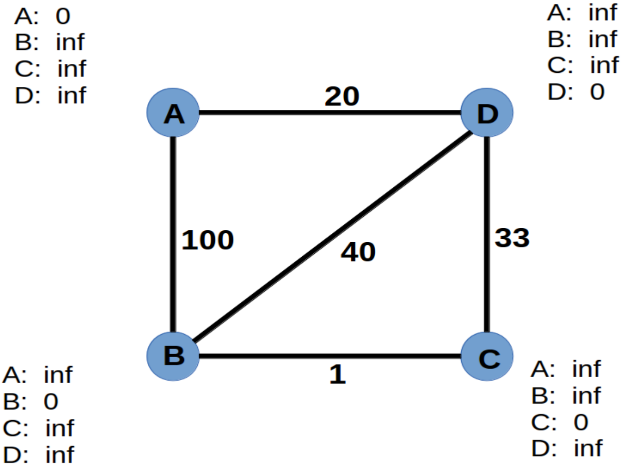
Distance-Vector Algorithms

- each node maintains its own **distance vector**, i.e. a list of best available upper bounds for the distance to every **known** node on the network, as well as its neighbors' distance vectors
- distance vector (or just its **updates**) are sent to all neighbors regularly
- distance vector updates from neighbors may cause an update of node's own distance vector

Bellman-Ford integration:

if node A is linked directly to node B at cost x, and the cost from node B to node C is not larger than y then the minimal cost from A to C is not larger than . . .

Distance-Vector: Initialization



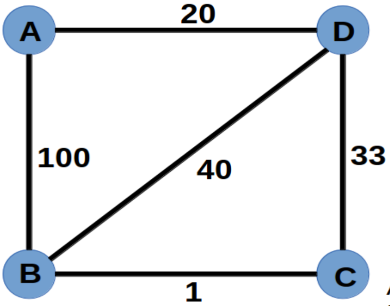
A: 0
B: inf
C: inf
D: inf

A: inf
B: inf
C: inf
D: 0

A: inf
B: 0
C: inf
D: inf

A: inf
B: inf
C: 0
D: inf

Distance-Vector: After First Step



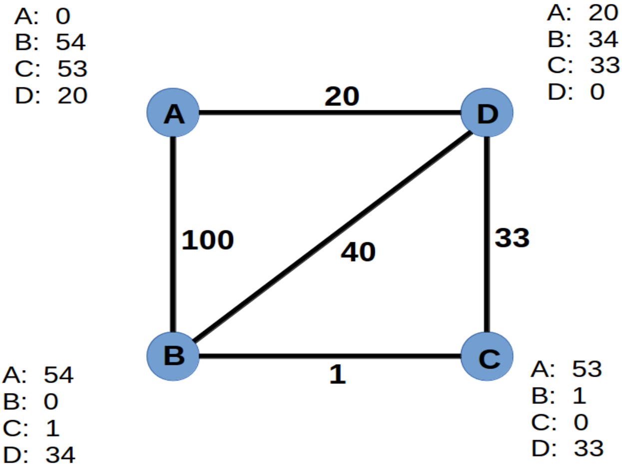
A: 0
B: 60
C: 53
D: 20

A: 20
B: 34
C: 33
D: 0

A: 60
B: 0
C: 1
D: 34

A: 53
B: 1
C: 0
D: 33

Distance-Vector: After Second Step, or Steady-State



Distance-Vector Algorithms: Formally

$$\mathrm{cost}^{+}(a, c) = \min_{\text{neighbor } b} \{\mathrm{cost}(a, b) + \mathrm{cost}(b, c)\}$$

- convergence guarantee: if network configuration remain constant, upper bounds of the cost eventually stop increasing (after a time equal to the maximal number of hops in an optimal path)
- optimality guarantee: if network configuration remain constant, upper bounds of the cost eventually beome equal to the true minimal cost
- correctness guarantee: once the upper bounds are exact, forwarding to the neighbor with the minimal cost to the destination is correct and optimal (correctness relies on assuming that all link costs are positive)

Link-State Algorithms

- nodes send out info about their links to neighbors, with costs (LSA – link-state advertising)
- nodes forward all LSAs they receive (only once)
- in time equal to the **diameter** of the network (measured in **hops**) all nodes will have all LSAs
- once a node has all LSAs from all nodes, it can optimize routing on its own (e.g., using the Dijkstra algorithm)

LSA: [node, (neighbor1,cost1), . . . ,(neighborN,costN)]

Dijkstra's Shortest Path Algorithm

Initialize:

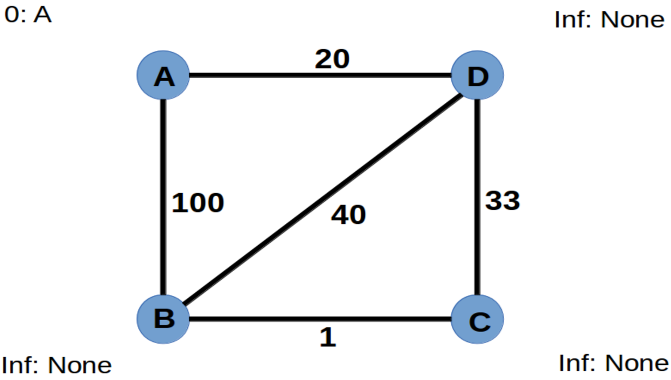
- **nodeset**: `[all nodes]` (the set of nodes to be processed)
- **costs**: `costs[me]=0, costs[other]=inf` (costs to node, upper bounds)
- **routes**: `routes[me]=me, routes[other]=None` (forwarding table)

Step (while **nodeset** isnt empty):

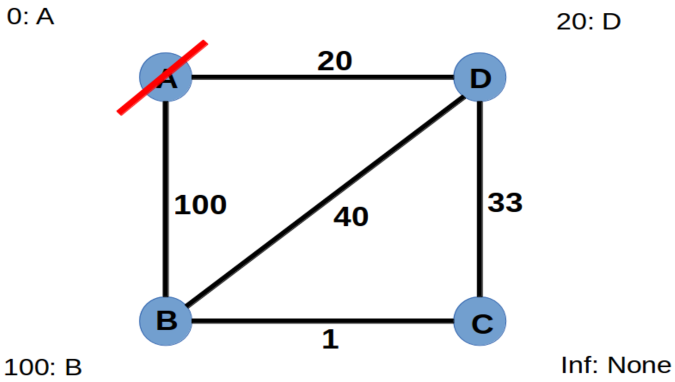
- find **u**, the node in nodeset with smallest **cost[u]**
- remove **u** from nodeset
- for all neighbors **v** of **u**:

```
if cost[v]>cost[u]+cost(u,v):
   cost[v]=cost[u]+cost(u,v)
   routes[v]=routes[u] if u not me, v if u=me
```

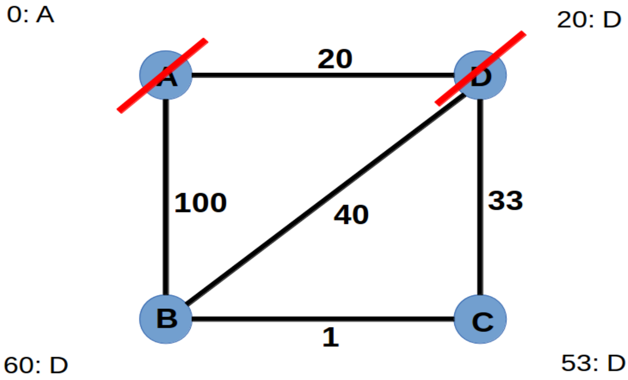
Dijkstra′s Algorithm: Initialization



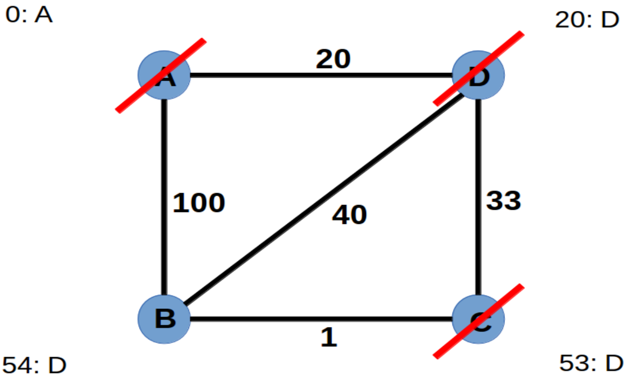
Dijkstra's Algorithm: First Step



Dijkstra's Algorithm: Second Step



Dijkstra′s Algorithm: Third Step



Dijkstra's Shortest Path Algorithm: Complexity

Parameters:

- **N**: number of nodes
- **L**: number of links

Complexity:

- finding **u**: **N** times: **O(log N)** each time, total **O(N log N)**
- updating **costs**: **O(L)**, since each link appears twice