Network Layer:
intro;
Distance Vector Protocols

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

11/27/2018

Outline

- Admin and recap
- Network overview
- Network control-plane
    - Routing

Admin

- Assignment four meeting
  - Today:
    - 2:30-3:30 pm
    - 5:00-6:30 pm
  - Wednesday

- Exam 2 date?

Recap: BW Allocation Framework

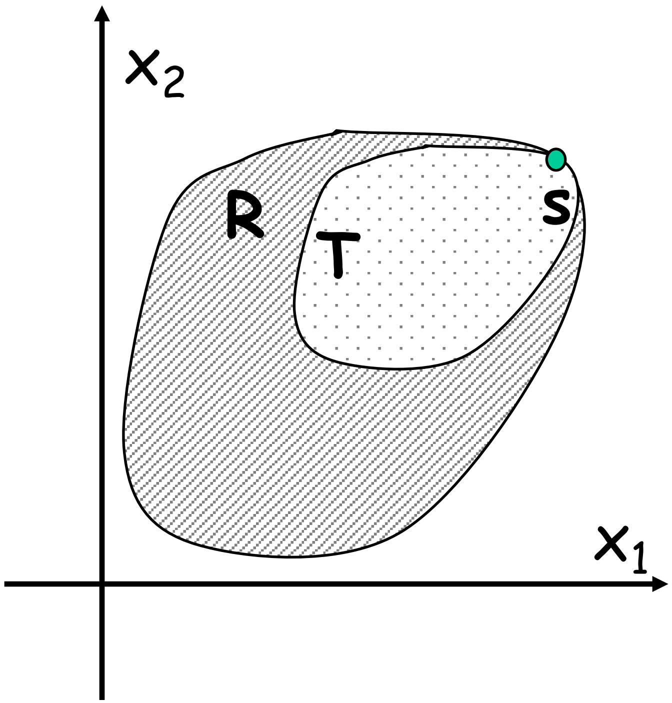
$$
\begin{array}{ll}
\text{max} & \sum_{f \in F} U_f(x_f) \\
\text{subject to} & \sum_{f: f \text{ uses link } l} x_f \le c_l \quad \text{for any link } l \\
\text{over} & x \ge 0
\end{array}
$$

- Forward engineering: systematically design of
  - objective function
  - distributed alg to achieve objective
- Science/reverse engineering: what do TCP/Reno, TCP/Vegas achieve?

Objective	Allocation (x1, x2, x3)		
TCP/Reno	0.26	0.74	0.74
TCP/Vegas	1/3	2/3	2/3
Max throughput	0	1	1
Max-min	½	½	½
Max sum log(x)	1/3	2/3	2/3
Max sum of $-1/(RTT^2\ x)$	0.26	0.74	0.74

Recap: Systematic Derivation of Objective Function

- NBS axioms
  - Pareto optimality
  - symmetry
  - invariance of linear transformation
  - independence of irrelevant alternatives



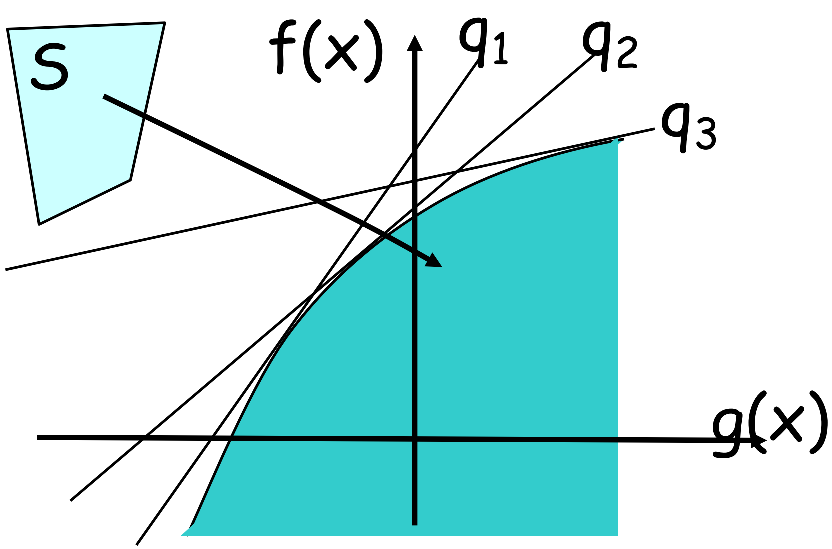
- NBS solution
  - the rate allocation point is the feasible point which maximizes

$$x_1 x_2 \cdots x_F$$

Recap: Systematic Derivation of Alg: Foundation (Strong Dual Theorem)

$$\begin{array}{ll} \max & f(x) \\ \text{subject to} & g(x) \le 0 \\ \text{over} & x \in S \end{array}$$

f(x) concave
g(x) linear
S is a convex set



$$D(q) = \max_{x \in S} \left(f(x) - q g(x) \right)$$

-D(q) is called the dual;
q (>= 0) are called prices in economics

Recap: Primal-Dual Decomposition of Network-Wide Resource Allocation

- SYSTEM(U):

$$\begin{array}{ll} \max & \sum_{f \in F} U_f(x_f) \\ \text{subject to} & \sum_{f: f \text{ uses link } l} x_f \le c_l \text{ for any link } l \\ \text{over} & x \ge 0 \end{array}$$

- USER$_f$:

$$\begin{array}{ll} \max_{x_f} & U_f(x_f) - x_f p_f \\ \text{over} & x_f \ge 0 \end{array}$$

- NETWORK:

$$\min_{q \ge 0} \tilde{D}(q) = \sum_l q_l (c_l - \sum_{f: \text{f uses } l} x_f)$$

TCP/Reno Dynamics

$$\boxed{\Delta x_f \propto U'_f\left(x_f\right) - p_f}$$

$$\Delta x = \frac{RTT}{2} x^2 \boxed{\left(\frac{2}{x^2 RTT^2} - p\right)}$$

$$U'_f\left(x_f\right) - p_f$$

$$\Rightarrow U'_f(x_f) = \left(\frac{\sqrt{2}}{x_f RTT}\right)^2 \qquad \Rightarrow U_f(x_f) = -\frac{2}{RTT^2 x_f}$$

TCP/Vegas Dynamics

$$\boxed{\Delta x_f \propto U'_f\left(x_f\right) - p_f}$$

$$\Delta x = \frac{x}{RTT}\left(\boxed{\frac{\alpha}{x} - (RTT - \mathrm{RTTmin}}\right))$$

$$U'_f\left(x_f\right) - p_f$$

$$\Rightarrow U'_f(x_f) = \tfrac{\alpha}{x} \qquad \Rightarrow U_f(x_f) = \alpha \log(x_f)$$

Outline

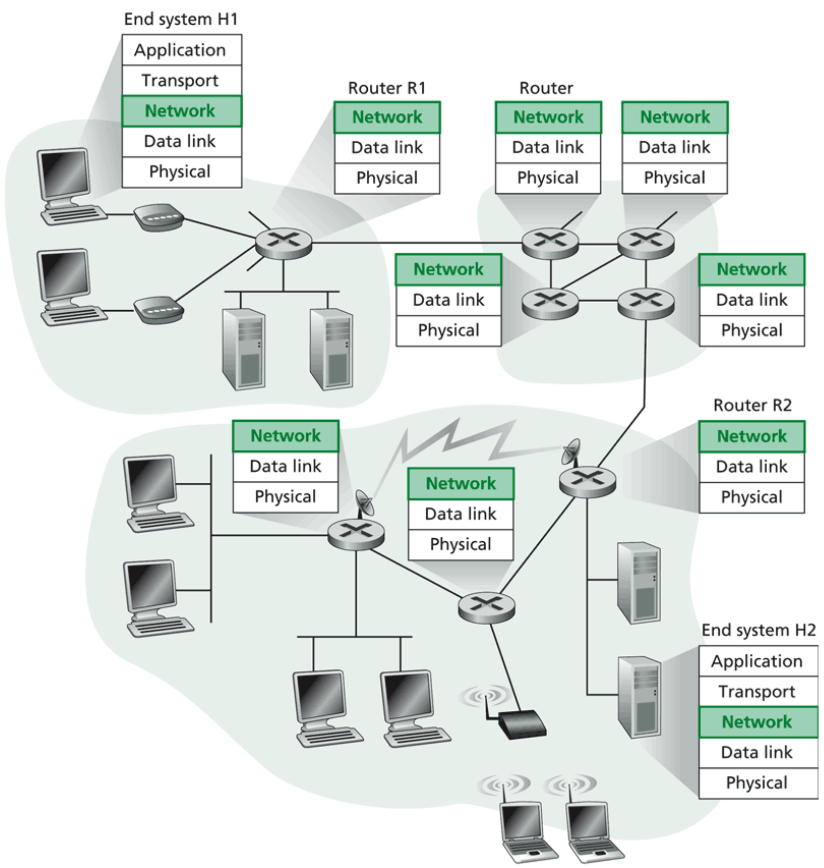
- Admin and recap
- *Network overview*

Network Layer

- Transport packets from source to destination
- Network layer in *every* host, router

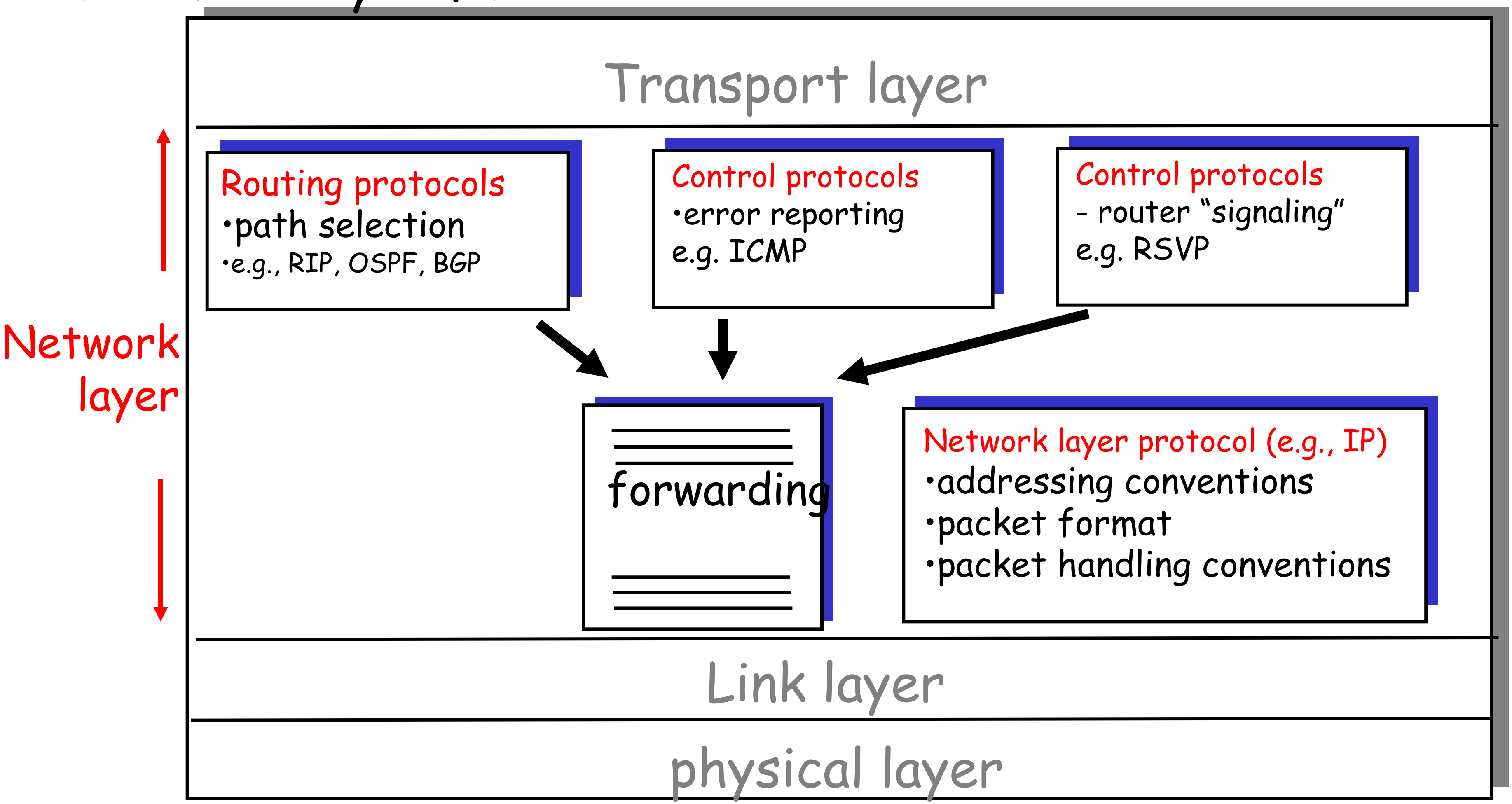
Basic functions:

- inter-networking (e.g., fragmentation/assembly)
- routing (determine route(s) taken by packets of a flow), and forwarding (move the packets along the route(s))



Current Internet Network Layer

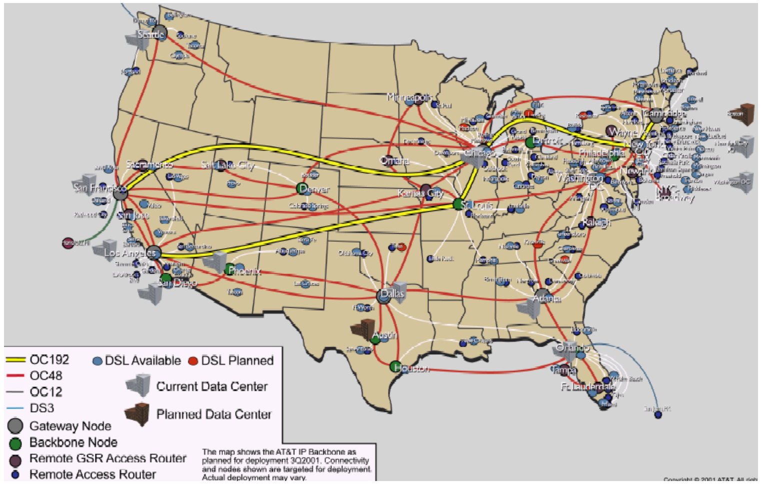
Network layer functions:



Routing: Overview

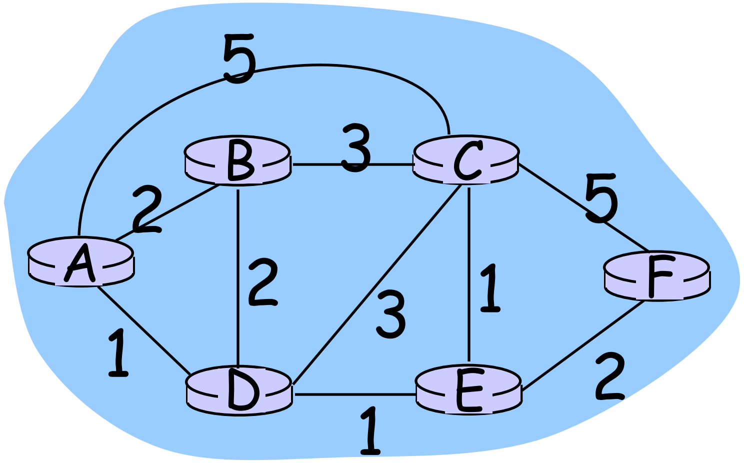
Routing

Goal: determine "good" paths (sequences of routers) thru networks from source to dest.



Graph abstraction for the routing problem:

- graph nodes are routers
- graph edges are physical links
  - links have properties: delay, capacity, $ cost
- compute path on graph



Network Layer: Complexity Factors/Objectives

- For network providers
  - efficiency of routes
  - policy control on routes
  - scalability

- For users
  - quality of services, e.g.,
    - guaranteed bandwidth?
    - preservation of inter-packet timing (no jitter)?
    - loss-free delivery?
    - in-order delivery?

- Users and network may interact

Routing Design Space

- Robustness
- Optimality
- Simplicity

- Routing has a large design space
  - who decides routing?
    - source routing: end hosts make decision
    - network routing: networks make decision
  - how many paths from source s to destination d?
    - multi-path routing
    - single path routing
  - what does routing compute?
    - network cost minimization
    - QoS aware
  - will routing adapt to network traffic demand?
    - adaptive routing
    - static routing
  - …

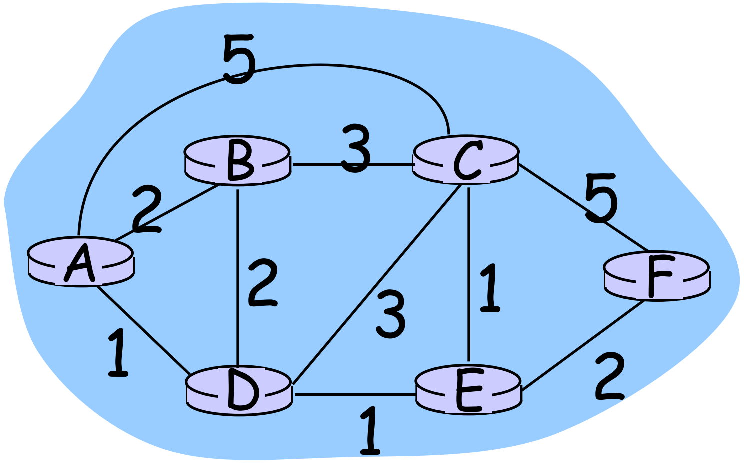
Routing Design Space: Internet

- Robustness
- Optimality
- Simplicity

- Routing has a large design space
  - who decides routing?
    - source routing: end hosts make decision
    - network routing: networks make decision
      - (applications such as overlay and p2p are trying to bypass it)
  - what does routing compute?
    - network cost minimization (shortest path)
    - QoS aware
  - how many paths from source s to destination d?
    - multi-path routing
    - single path routing (with small amount of multipath)
  - will routing adapt to network traffic demand?
    - adaptive routing
    - static routing (mostly static; adjust in larger timescale)
  - ...

Basic Formulation

- Assign link weights
- Compute shortest path



Example: Cisco Proprietary Recommendation on Assigning Link Costs

- Link metric:
  - metric = [K1 * bandwidth^{-1} + (K2 * bandwidth^{-1}) / (256 - load) + K3 * delay] * [K5 / (reliability + K4)]

By default, k1=k3=1 and k2=k4=k5=0. The default composite metric for EIGRP, adjusted for scaling factors, is as follows:

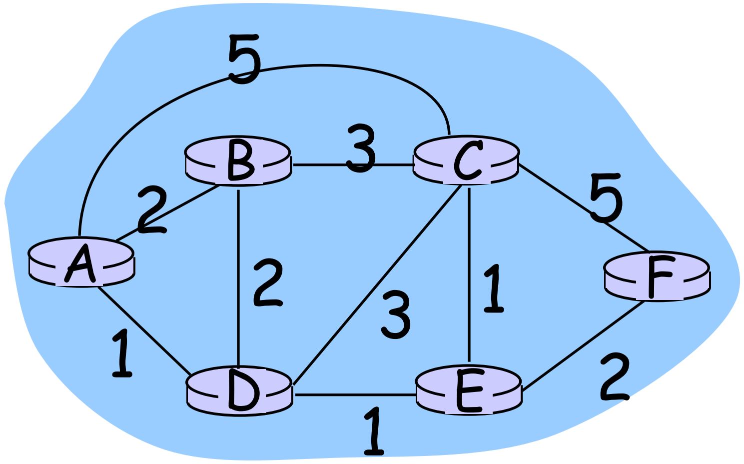
$$\mathrm{EIGRP}_{metric} = 256 \times \{ [10^7/\mathrm{BW}_{min}] + [\mathrm{sum\_of\_delays}] \}$$

BW_{min} is in kbps and the sum of delays are in 10s of microseconds.

EIGRP : Enhanced Interior Gateway Routing Protocol

Example: EIGRP Link Cost

- The bandwidth and delay for an Ethernet interface are 10 Mbps and 1 ms, respectively.
- The calculated EIGRP metric is as follows:
  - $256 \times [10^7/\text{BWks} + \text{delayin10us}]$
  - $= 256 \times [10^7/10{,}000 + 100]$
  - $= 256 \times [1000 + 100]$
  - $= 256{,}000 + 25{,}600$
  - $= 281{,}600$



Outline

- Admin and recap
- Network overview
- Network control plane
  - Routing
    - Link weights assignment
    - Distributed routing computation

Why Study?

- Just as Dijkstra' Shortest Path algorithm is among the most classical algorithms in algorithm design, distributed shortest path protocols provide many insights in distributed protocol design.

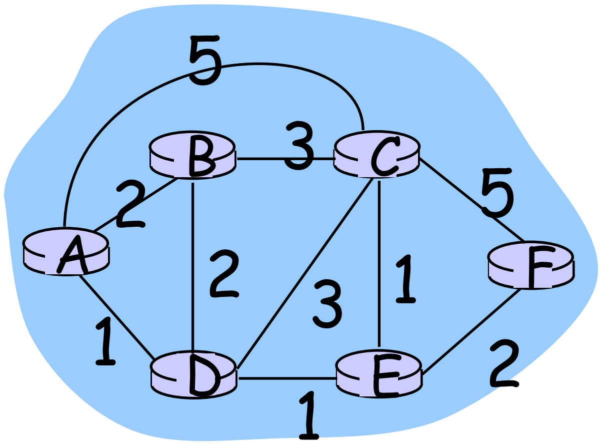
- Please learn not only the protocols, but also the techniques (convergence, global invariants, …)

Outline

- Admin and recap
- Network overview
- Network control plane
    - Routing
        - Link weights assignment
        - Routing computation

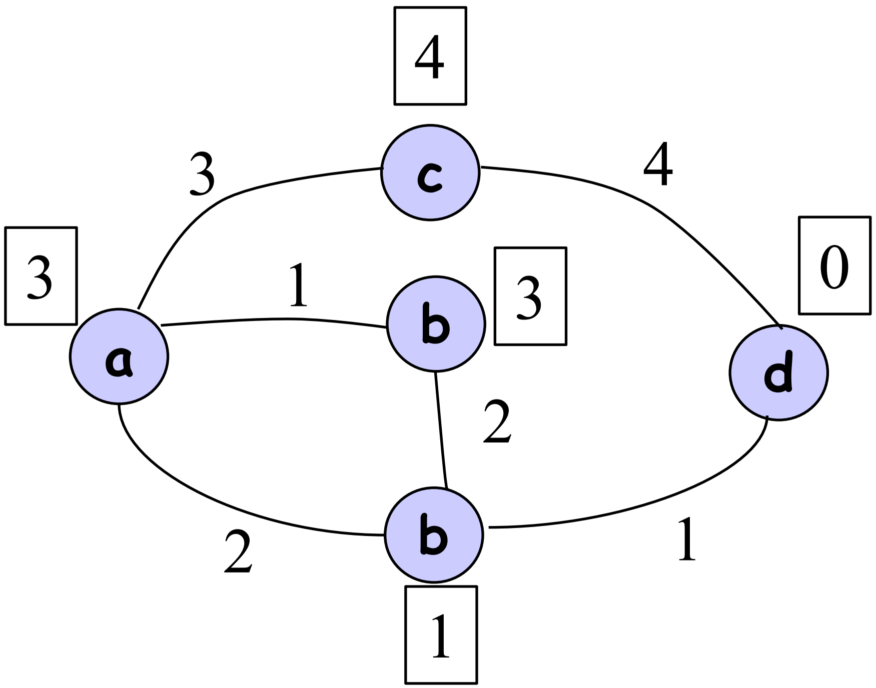
Basic Routing Computation Setting

- Setting: static (positive) costs assigned to network links
  - The static link costs may be adjusted in a longer time scale: this is called traffic engineering



- Goal: distributed computing to compute the shortest path from a source to a destination
  - Conceptually, runs for each destination separately

Intuition



$d_i \leq d_j + d_{ij}$, for each neighbor j

$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

Understanding Shortest Path and an Exercise of Primal-Dual

$$\max d_s - d_D$$

$$for\ any\ edge\ i \rightarrow j\!:\ d_i \leq dj + d_{\mathrm{ij}}$$

$$d_i \geq 0$$

$$\text{Dual:}\, D(x) = \max(d_s - dD - \sum x_{ij}(\mathrm{d_i} - d_{\mathrm{j}} - d_{\mathrm{ij}}))$$

$$= \sum x_{ij} d_{ij}$$

x_{ij} is a flow from s to D

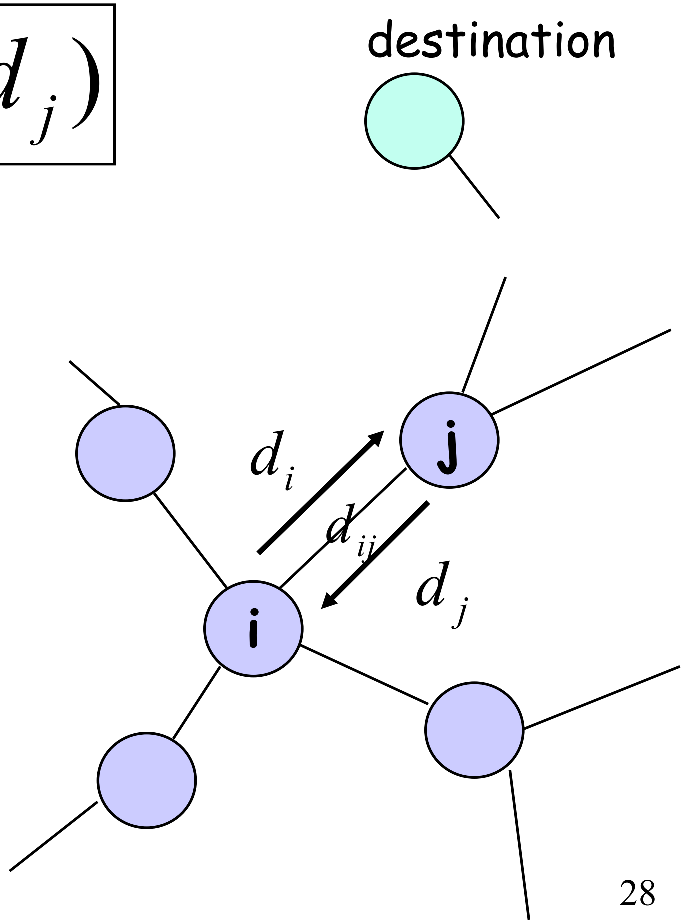
Outline

- Admin and recap
- Network overview
- Network control plane
    - Routing
        - Link weights assignment
        - Routing computation
            - *Distributed distance vector protocols*

Distance Vector Routing: Basic Idea

- Based on Bellman-Ford equation: At node i, the basic update rule

$$\boxed{d_i = \min_{j \in N(i)} (d_{ij} + d_j)}$$



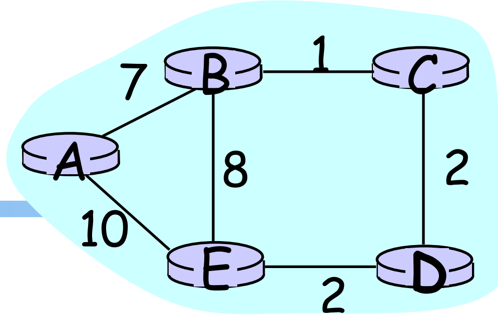
where

- d_i denotes the distance estimation from i to the destination,
- N(i) is set of neighbors of node i, and
- d_{ij} is the distance of the direct link from i to j

Outline

- Admin and recap
- Network overview
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - *distributed distance vector protocols*
        - *synchronous Bellman-Ford (SBF)*

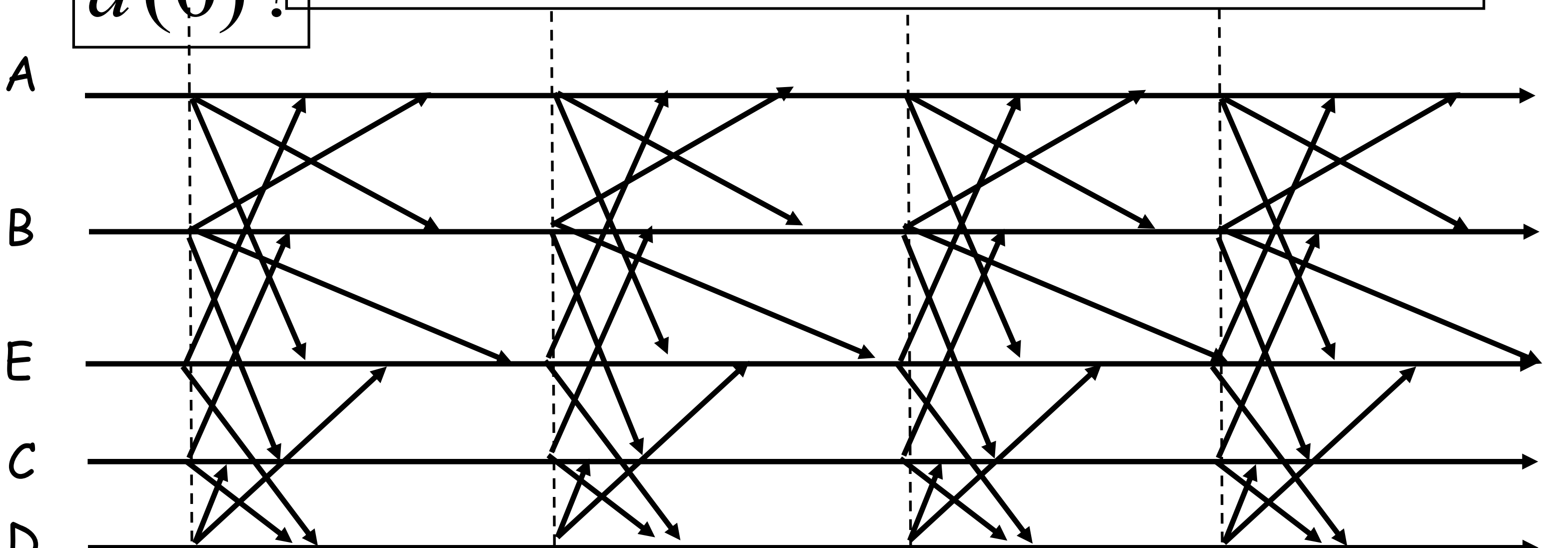
Synchronous Bellman-Ford (SBF)



- Nodes update in rounds:
  - there is a global clock;
  - at the beginning of each round, each node sends its estimate to all of its neighbors;
  - at the end of the round, updates its estimation

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

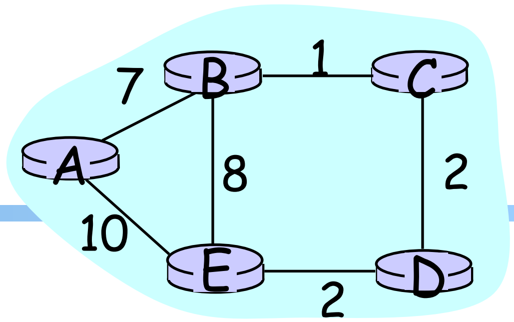
$$d(0)?$$



Outline

- Admin and recap
- Network overview
- Network control-plane path
  - Routing
    - Link weights assignment
    - Routing computation
      - *distributed distance vector protocols*
        - *synchronous Bellman-Ford (SBF)*
          - SBF/∞

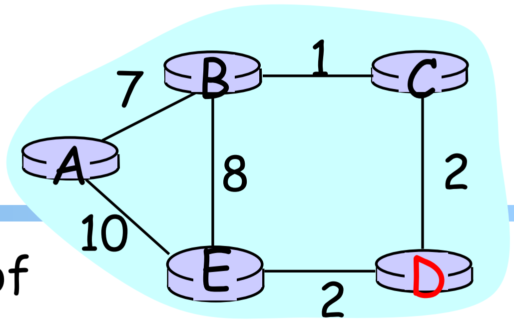
SBF/∞



- Initialization (time 0):

$$d_i(0) = \begin{cases} 0 & \mathrm{i = dest} \\ \infty & \mathrm{otherwise} \end{cases}$$

Example



Consider D as destination; d(t) is a vector consisting of estimation of each node at round t

	A	B	C	E	D
d(0)	∞	∞	∞	∞	0
d(1)	∞	∞	2	2	0
d(2)	12	3	2	2	0
d(3)	10	3	2	2	0
d(4)	10	3	2	2	0

Observation: $d(0) \geq d(1) \geq d(2) \geq d(3) \geq d(4) = d^*$

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

A Nice Property of SBF: Monotonicity

- Consider two configurations d(t) and d'(t)
- If $d(t) \geq d'(t)$
  - i.e., each node has a higher estimate in one scenario (d) than in another scenario (d'),
- then $d(t+1) \geq d'(t+1)$
  - i.e., each node has a higher estimate in d than in d' after one round of synchronous update.

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

Correctness of SBF/∞

- Claim: $d_i(h)$ is the length $L_i(h)$ of a shortest path from i to the destination using $\leq h$ hops
  - base case: h = 0 is trivially true
  - assume true for $\leq$ h,
    i.e., $L_i(h) = d_i(h)$, $L_i(h-1) = d_i(h-1)$, …

$$d_i(h+1) = \min_{j \in N(i)}(d_{ij} + d_j(h))$$

Correctness of SBF/∞

- consider $\leq$ h+1 hops:

$$L_i(h+1) = \min(L_i(h), \min_{j \in N(i)}(d_{ij} + L_j(h)))$$

$$= \min(d_i(h), \min_{j \in N(i)}(d_{ij} + d_j(h)))$$

$$= \min(d_i(h), d_i(h+1))$$

since $d_i(h) \leq d_i(h-1)$

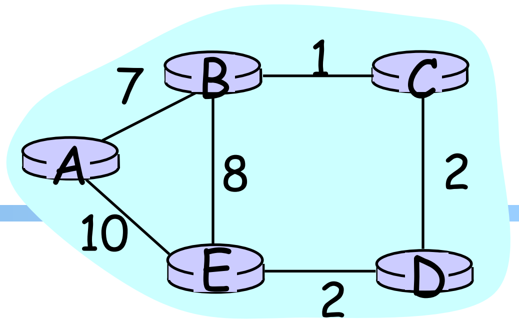
$$\boxed{d_i(h+1) = \min_{j \in N(i)}(d_{ij} + d_j(h)) \leq \min_{j \in N(i)}(d_{ij} + d_j(h-1)) = d_i(h)}$$

$$L_i(h+1) = d_i(h+1)$$

Outline

- Admin and recap
- Network overview
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - *Distributed distance vector protocols*
        - *synchronous Bellman-Ford (SBF)*
          - SBF/∞
          - SBF/-1

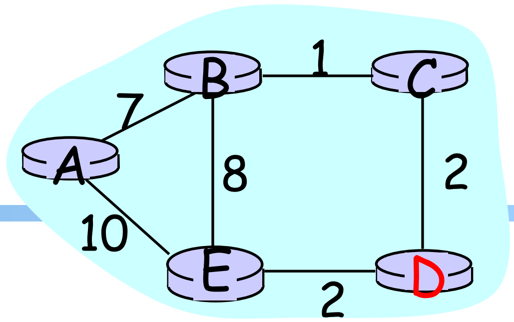
SBF at another Initial Configuration: SBF/-1



- Initialization (time 0):

$$d_i(0) = \begin{cases} 0 & \mathrm{i} = \mathrm{dest} \\ -1 & \mathrm{otherwise} \end{cases}$$

Example



Consider D as destination

	A	B	C	E	D
d(0)	-1	-1	-1	-1	0
d(1)	6	0	0	2	0
d(2)	7	1	1	2	0
d(3)	8	2	2	2	0
d(4)	9	3	3	2	0
d(5)	10	3	3	2	0
d(6)	10	3	3	2	0

Observation: $d(0) \leq d(1) \leq d(2) \leq d(3) \leq d(4) \leq d(5) = d(6) = d^*$

Correctness of SBF/-1

- SBF/-1 converges due to monotonicity

- Remaining question:
  - Can we guarantee that SBF/-1 converges to shortest path?

Correctness of SBF/-1

- Common between SBF/∞ and SBF/-1: they solve the Bellman equation

$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

where $d_D = 0$.

- We have proven SBF/∞ is the shortest path solution.
- SBF/-1 computes shortest path if Bellman equation has a unique solution.

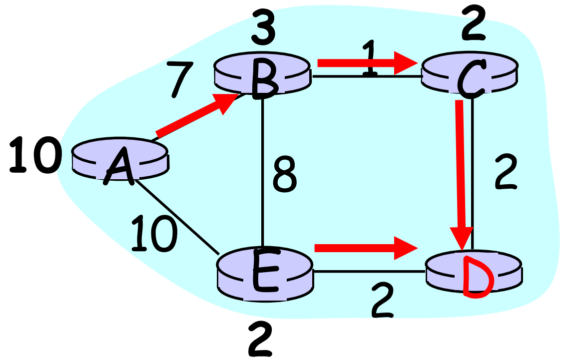
$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

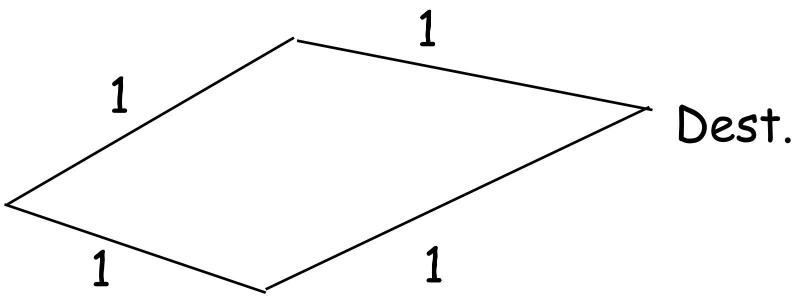
Uniqueness of Solution to BE

- Assume another solution d, we will show that d = d*

case 1: we show d $\geq$ d*

Since d is a solution to BE, we can construct paths as follows: for each i, pick a j which satisfies the equation; since d* is shortest, d $\geq$ d*





$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

Uniqueness of Solution to BE

Case 2: we show d ≤ d*

assume we run SBF with two initial configurations:

- one is d
- another is SBF/∞ (d^{∞}),

-> monotonicity and convergence of SBF/∞ imply that d ≤ d*

Summary: “Extreme” SBF Initial States

$$d_i(0) = \begin{cases} 0 & \text{i} = \text{dest} \\ \infty & \text{otherwise} \end{cases}$$

$$d_i(0) = \begin{cases} 0 & \text{i} = \text{dest} \\ -1 & \text{otherwise} \end{cases}$$

- Nice properties of both cases
  - Monotonicity
  - Convergence

Discussion

- Will SBF converge under any non-negative initial conditions?

Outline

- Admin and recap
- Network overview
- Network control plane
    - Routing
        - Link weights assignment
        - Routing computation
            - *Distributed distance vector protocols*
                - synchronous Bellman-Ford (SBF)
                - *asynchronous Bellman-Ford (ABF)*

Asynchronous Bellman-Ford (ABF)

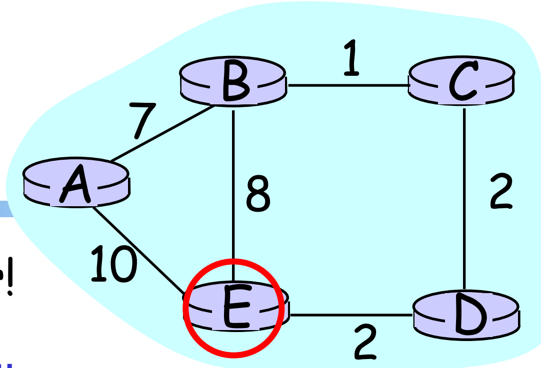
- No notion of global iterations
  - each node updates at its own pace
- Asynchronously each node i computes

$$d_i = \min_{j \in N(i)} (d_{ij} + d_j^i)$$

using last received value d^i_j from neighbor j.

- Asynchronously node j sends its estimate to its neighbor i:
  - We assume that there is an upper bound on the delay of estimate packet

ABF: Example



Below is just one step! The protocol repeats forever!

destinations

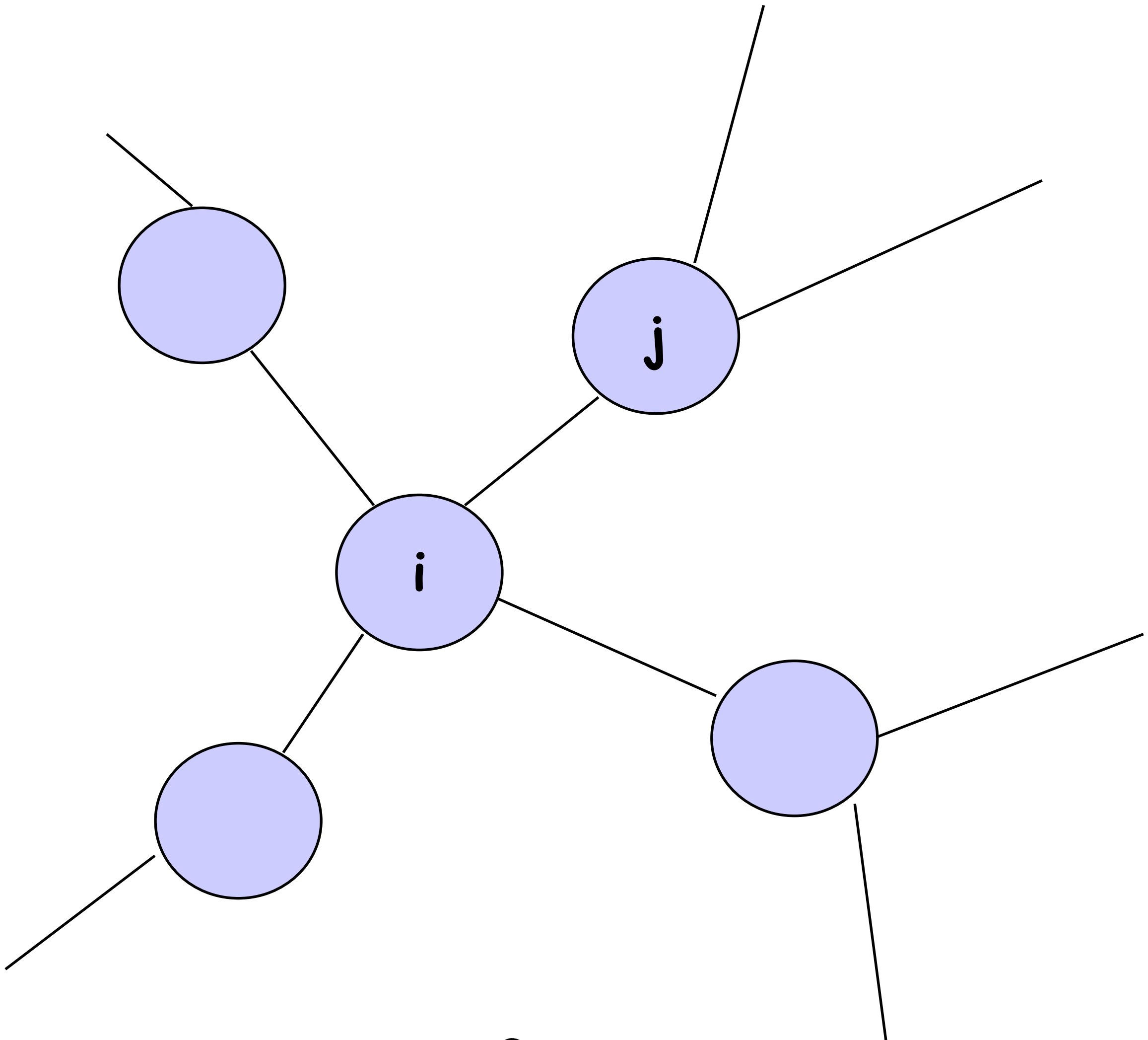
$d_E()$	distance tables from neighbors			computation			E's routing table	distance table E sends to its neighbors
	A	B	D	A	B	D		
A	0	7	∞	10	15	∞	A: 10	A: 10
B	7	0	∞	17	8	∞	B: 8	B: 8
C	∞	1	2	∞	9	4	D: 4	C: 4
D	∞	∞	0	∞	∞	2	D: 2	D: 2
	10	8	2					E: 0

next hop distance

Asynchronous Bellman-Ford (ABF)

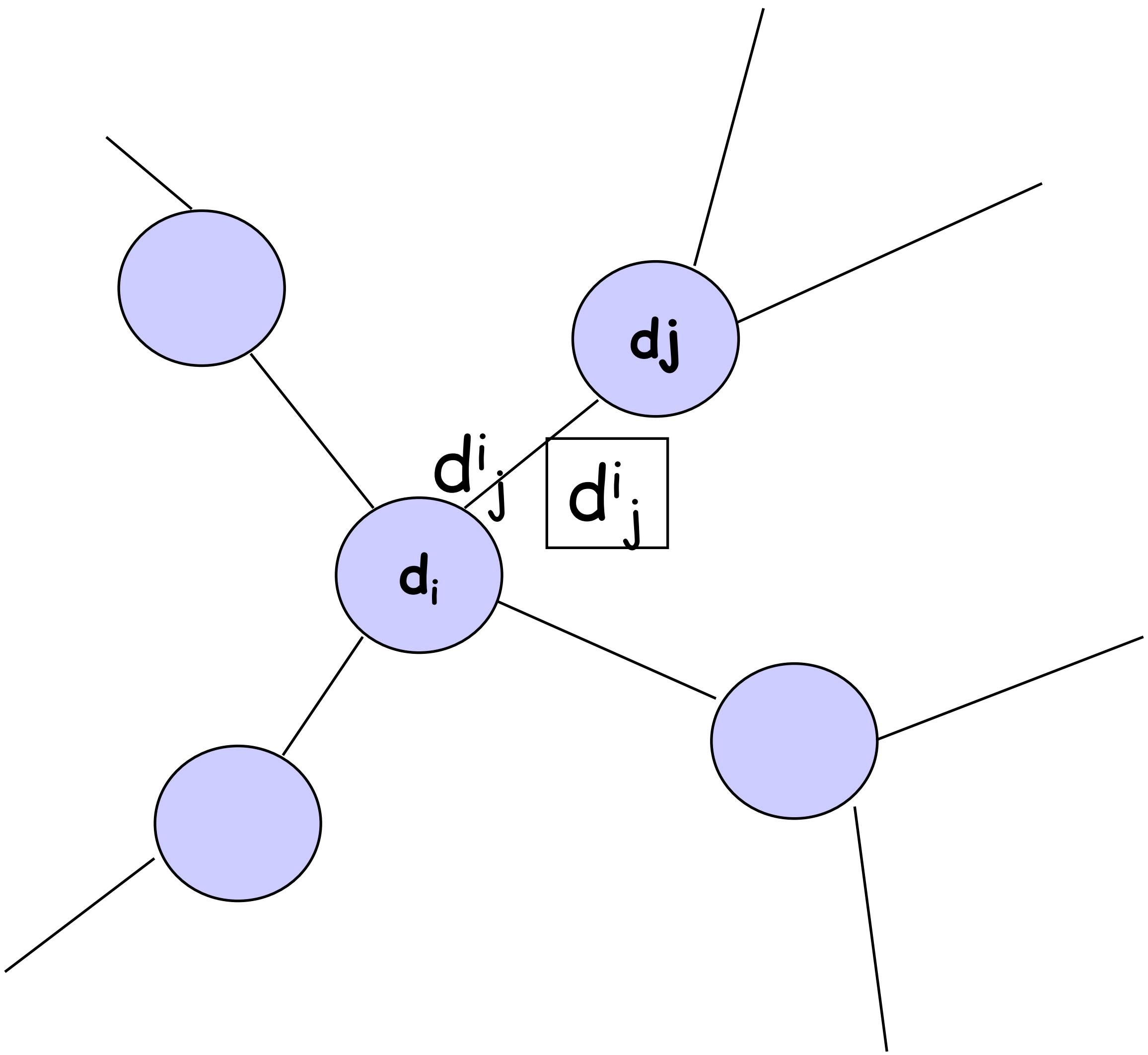
- ABF will eventually converge to the shortest path
  - links can go down and come up – but if topology is stabilized after some time t and connected, ABF will eventually converge to the shortest path !

ABF Convergence Proof Complexity: Complex System State



What is system state?

System State



three types of distance state from node j:

- d_j: current distance estimate state at node j

- d^i_j: last d_j that neighbor i received

- d^i_j: those d_j that are still in transit to neighbor i

ABF Convergence Proof: The Sandwich Technique

- Basic idea:
  - bound system state using extreme states
- Extreme states:
  - SBF/∞; call the sequence U()
  - SBF/-1; call the sequence L()

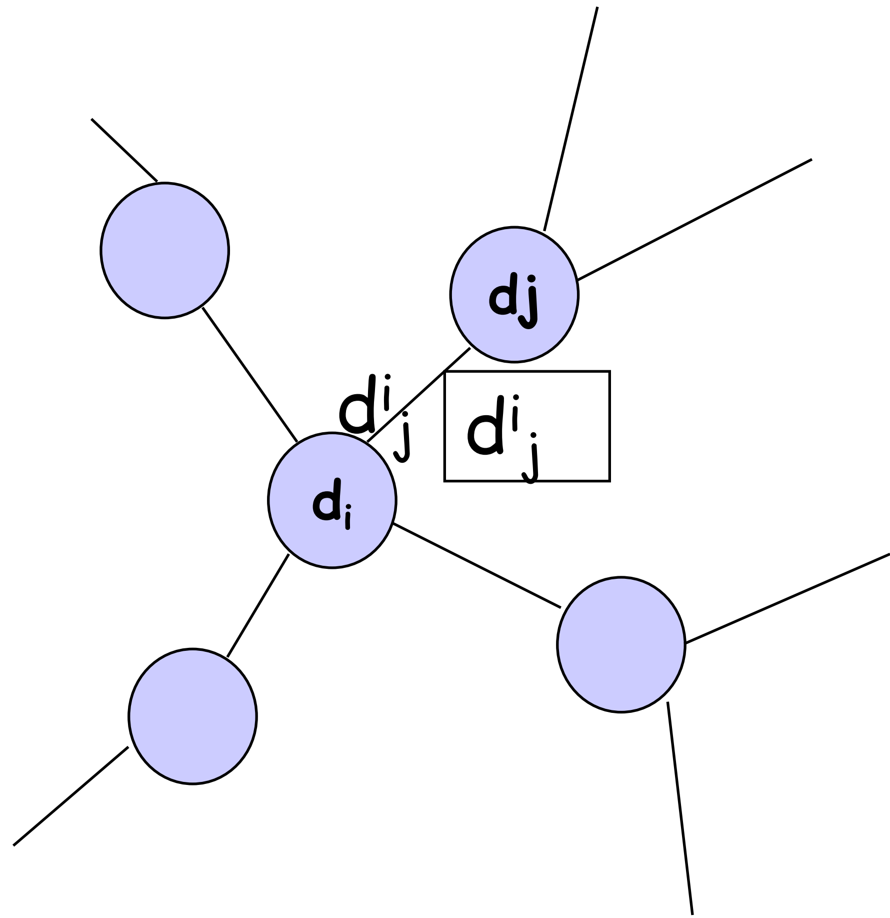
ABF Convergence

- Consider the time when the topology is stabilized at time 0

- U(0) and L(0) provide upper and lower bounds at time 0 on all corresponding elements of states
  - $L_j (0) \leq d_j \leq U_j (0)$ for all d_j state at node j
  - $L_j (0) \leq d^i_j \leq U_j (0)$
  - $L_j (0) \leq$ `update messages` $d^i_j \leq U_j (0)$

ABF Convergence

- d_j
  - after at least one update at node j: d_j falls between $L_j(1) \leq d_j \leq U_j(1)$
- d^i_j :
  - eventually all d^i_j that are only bounded by $L_j(0)$ and $U_j(0)$ are replaced with in $L_j(1)$ and $U_j(1)$



Distributed, Asynchronous, Routing Protocol: Summary of Features

- Distributed
  - each node communicates its routing table to its directly-attached neighbors
- Iterative
  - continues periodically or when link changes, e.g. detects a link failure
- Asynchronous
  - nodes need *not* exchange info/iterate in lock step!
- Convergence
  - in finite steps, independent of initial condition if network is connected

Distributed, Asynchronous, Routing Protocol: Summary of Analytical Technqiue

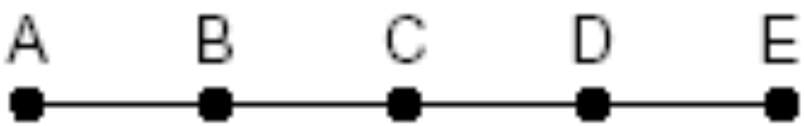
- Tool box: a key technique for analyzing convergence (liveness) of distributed protocols: monotonicity and the bounding-box (sandwich) theorem
  - Consider two configurations d(t) and d'(t):
    - if d(t) <= d'(t), then d(t+1) <= d'(t+1)
  - Identify two extreme configurations to sandwich any real configurations

Outline

- Admin and recap
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - Distance vector protocols (distributed computing)
        - synchronous Bellman-Ford (SBF)
        - asynchronous Bellman-Ford (ABF)
        - *properties of DV*

Properties of Distance-Vector Algorithms

- Good news propagate fast



∞	∞	∞	∞	Initially
1	∞	∞	∞	After 1 exchange
1	2	∞	∞	After 2 exchanges
1	2	3	∞	After 3 exchanges
1	2	3	4	After 4 exchanges

Properties of Distance-Vector Algorithms

- Bad news propagate slowly

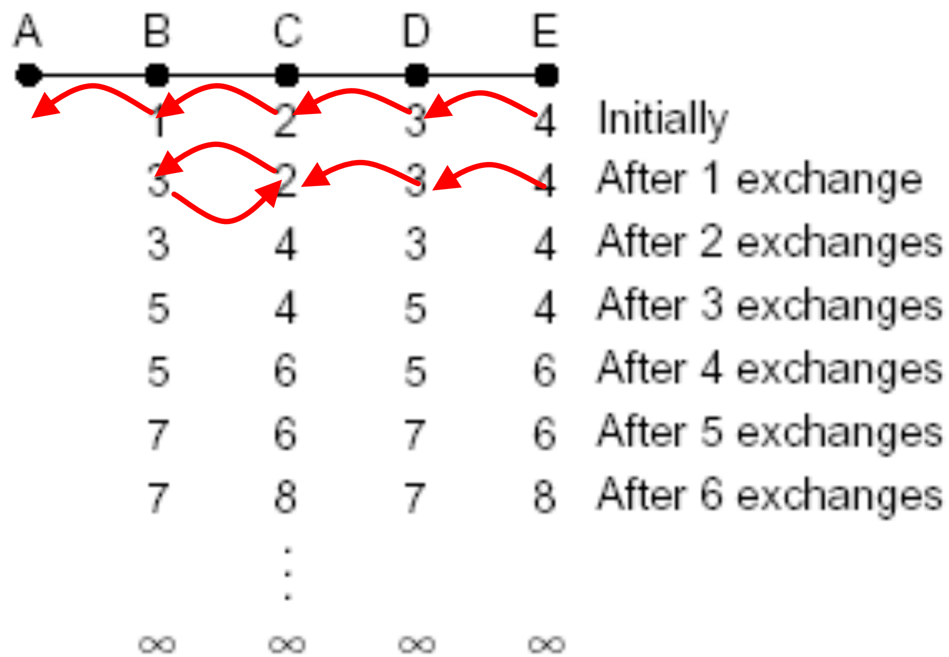
A-B link down

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

- This is called the *counting-to-infinity* problem
- Q: what causes counting-to-infinity?

Counting-To-Infinity is Because of Routing Loop

- Counting-to-infinity is caused by a routing loop, which is a global state (consisting of the nodes' local states) at a global moment (observed by an oracle) such that there exist nodes A, B, C, … E such that A (locally) thinks B as next hop, B thinks C as next hop, … E thinks A as next hop



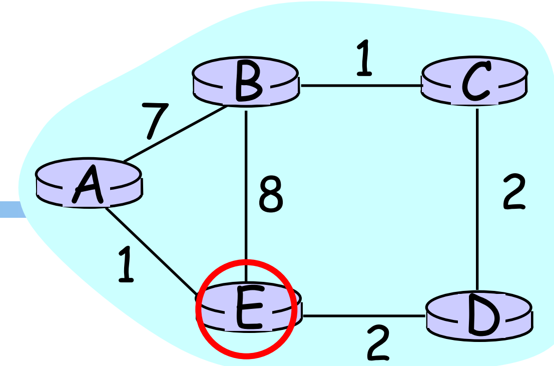
Discussion

- Why avoid routing loops is hard?
- Any proposals to avoid distributed routing loops?

Outline

- Admin and recap
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - Distance vector protocols (distributed computing)
        - synchronous Bellman-Ford (SBF)
        - asynchronous Bellman-Ford (ABF)
        - properties of DV
        - distributed protocols w/ safety (loop prevention)
          - *reverse poison/split horizon*

The Reverse-Poison (Split-horizon) Hack



If the path to dest is through neighbor h, report ∞ to neighbor h for dest.

$D^E()$	distance tables from neighbors: A	B	D	computation: A	B	D	E's distance table
A	0	7	∞	1	15	∞	1, A
B	7	0	∞	8	8	∞	8, B
C	∞	1	2	∞	9	4	4, D
D	∞	∞	0	∞	∞	2	2, D
	1	8	2				
	c(E,A)	c(E,B)	c(E,D)				

destinations

distance

through neighbor

distance table E sends to its neighbors

To A	To B	To D
A: ∞	A: 1	A: 1
B: 8	B: ∞	B: 8
C: 4	C: 4	C: ∞
D: 2	D: 2	D: ∞
E: 0	E: 0	E: 0

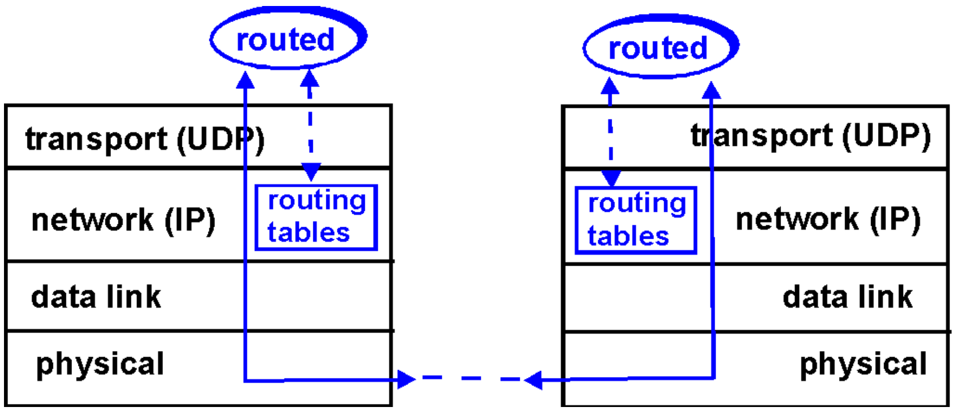
Reverse-Poison Example

- Exercise: Can Reverse-poison guarantee no loop for this network?

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

DV+RP => RIP
(Routing Information Protocol)

- Included in BSD-UNIX Distribution in 1982
- Link cost: 1
- Distance metric: # of hops
- Distance vectors
  - exchanged every 30 sec via Response Message (also called **advertisement**) using UDP
  - each advertisement: route to up to 25 destination nets



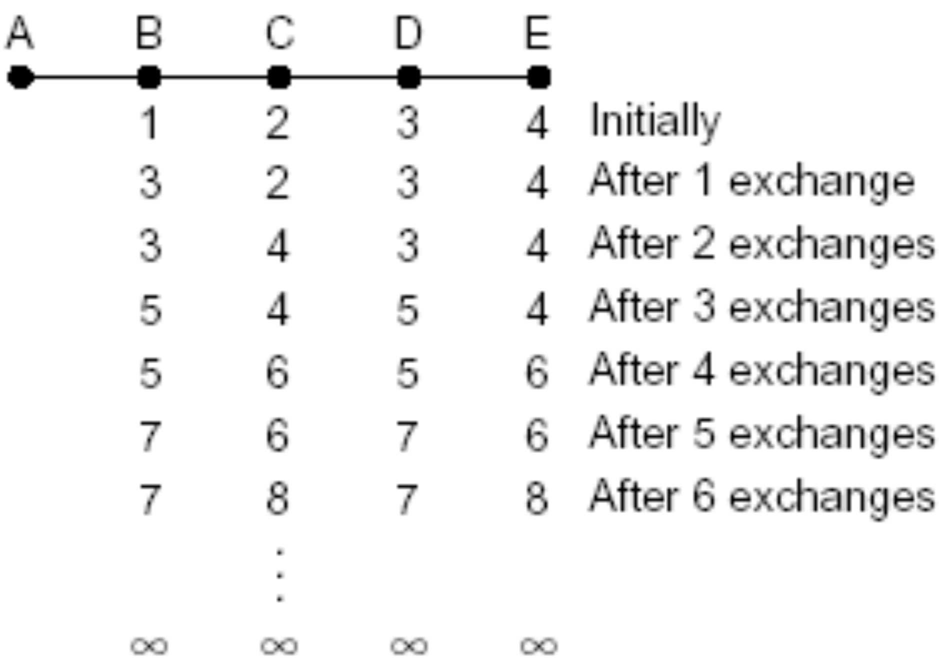
RIP: Link Failure and Recovery

If no advertisement heard after 180 sec --> neighbor/link declared dead

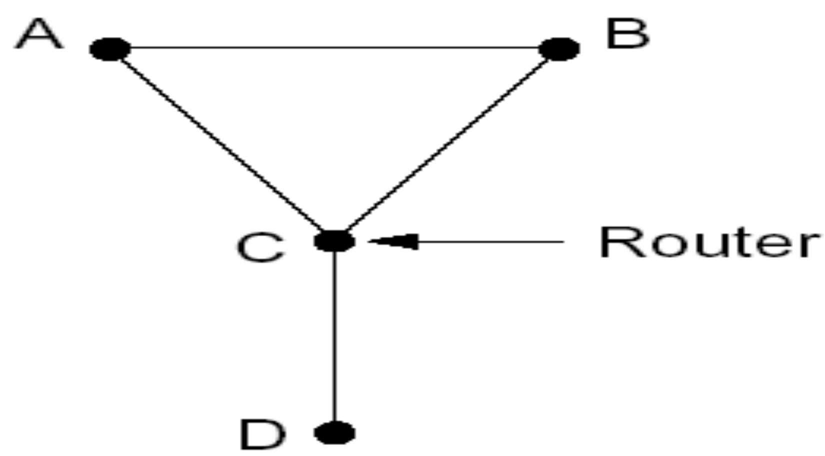
- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly propagates to entire net
- reverse-poison used to prevent ping-pong loops
- set infinite distance = 16 hops (why?)

General Routing Loops and Reverse-poison

- Exercise: Can Reverse-poison guarantee no loop for this network?



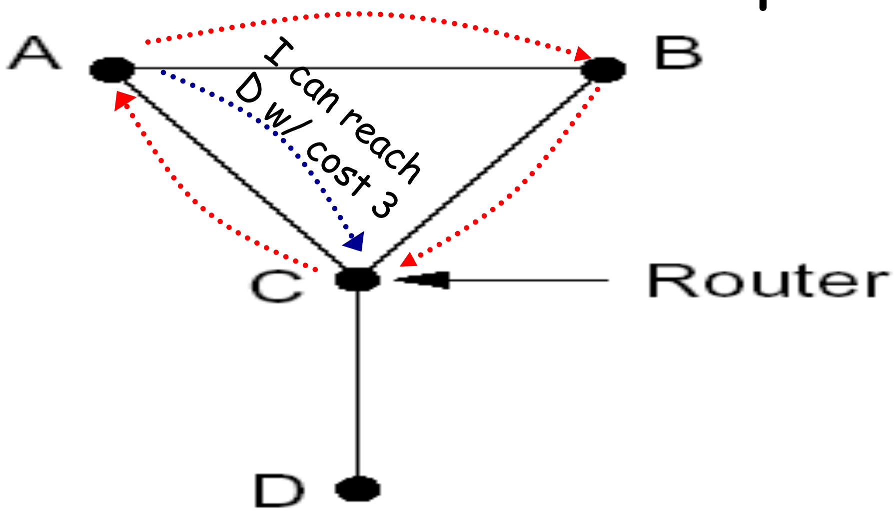
✔



?

General Routing Loops and Reverse-poison

- Reverse-poison removes two-node loops but may not remove more-node loops



- Unfortunate timing can lead to a loop
  - When the link between C and D fails, C will set its distance to D as ∞
  - A receives the bad news (∞) from C, A will use B to go to D
  - A sends the news to C
  - C sends the news to B

Backup Slides

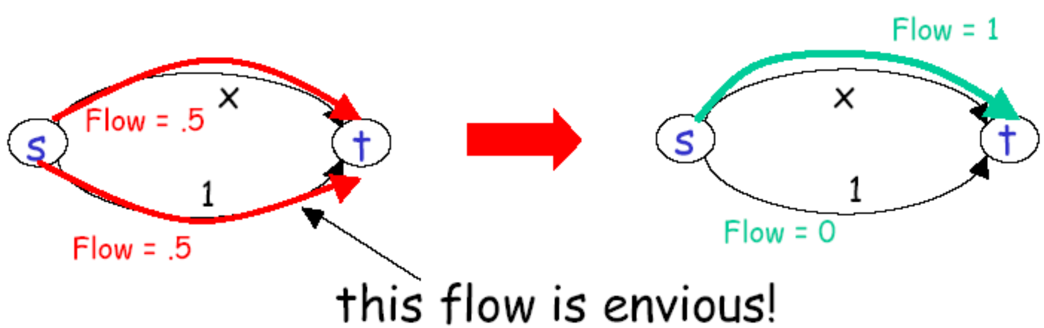
Routing Design Space: User-based, Multipath, Adaptive

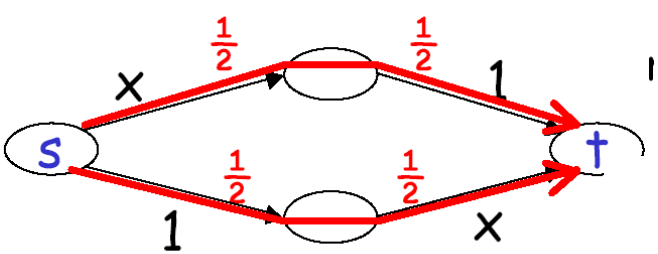
- Robustness
- Optimality
- Simplicity

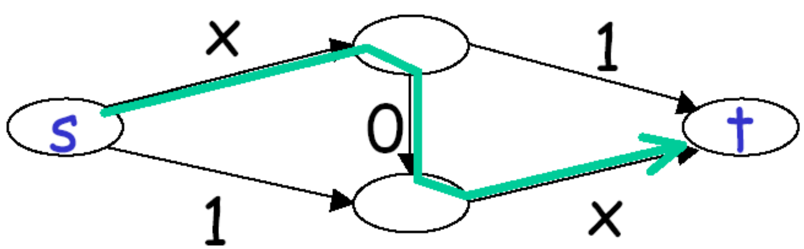
- Routing has a large design space
  - who decides routing?
    - source routing: end hosts make decision
    - network routing: networks make decision
  - how many paths from source s to destination d?
    - multi-path routing
    - single path routing
  - what does routing compute?
    - network cost minimization
    - QoS aware
  - will routing adapt to network traffic demand?
    - adaptive routing
    - static routing
  - ...

User Optimal, Multipath, Adaptive

- User optimal: users pick the shortest routes (selfish routing)







Braess's paradox

Price of Anarchy

For a network with linear latency functions

→

total latency of user (selfish) routing for given traffic demand

≤ 4/3

total latency of network optimal routing for the traffic demand

Price of Anarchy

- For any network with continuous, non-decreasing latency functions →

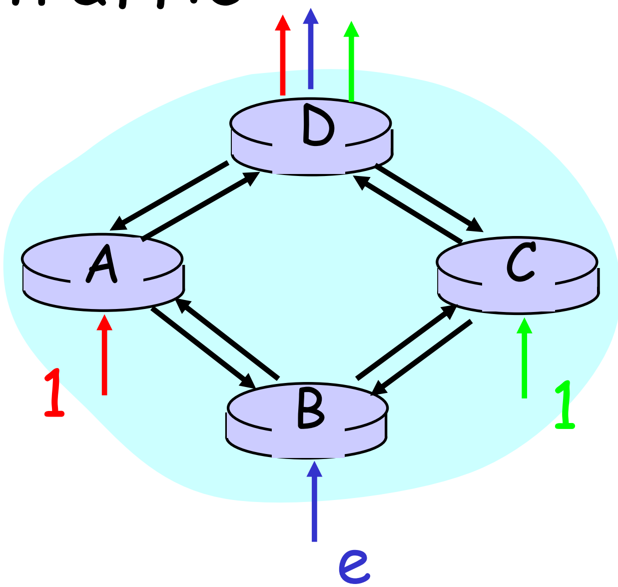
  total latency of user (selfish) routing for given traffic demand

  ≤

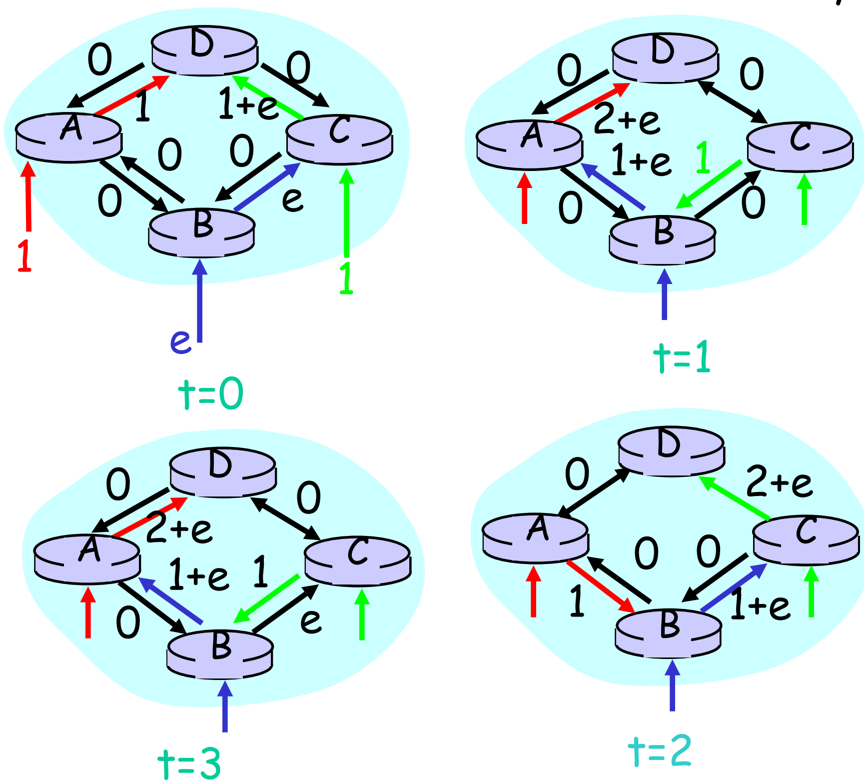
  total latency of network optimal routing for twice traffic demand

Assigning Link Weight: Dynamic Link Costs

- Assign link costs to reflect current traffic



Link costs reflect current traffic intensity



Solution: Link costs are a combination of current traffic intensity (dynamic) and topology (static). To improve stability, the static topology part should be large. Thus less sensitive to traffic; thus non-adaptive.