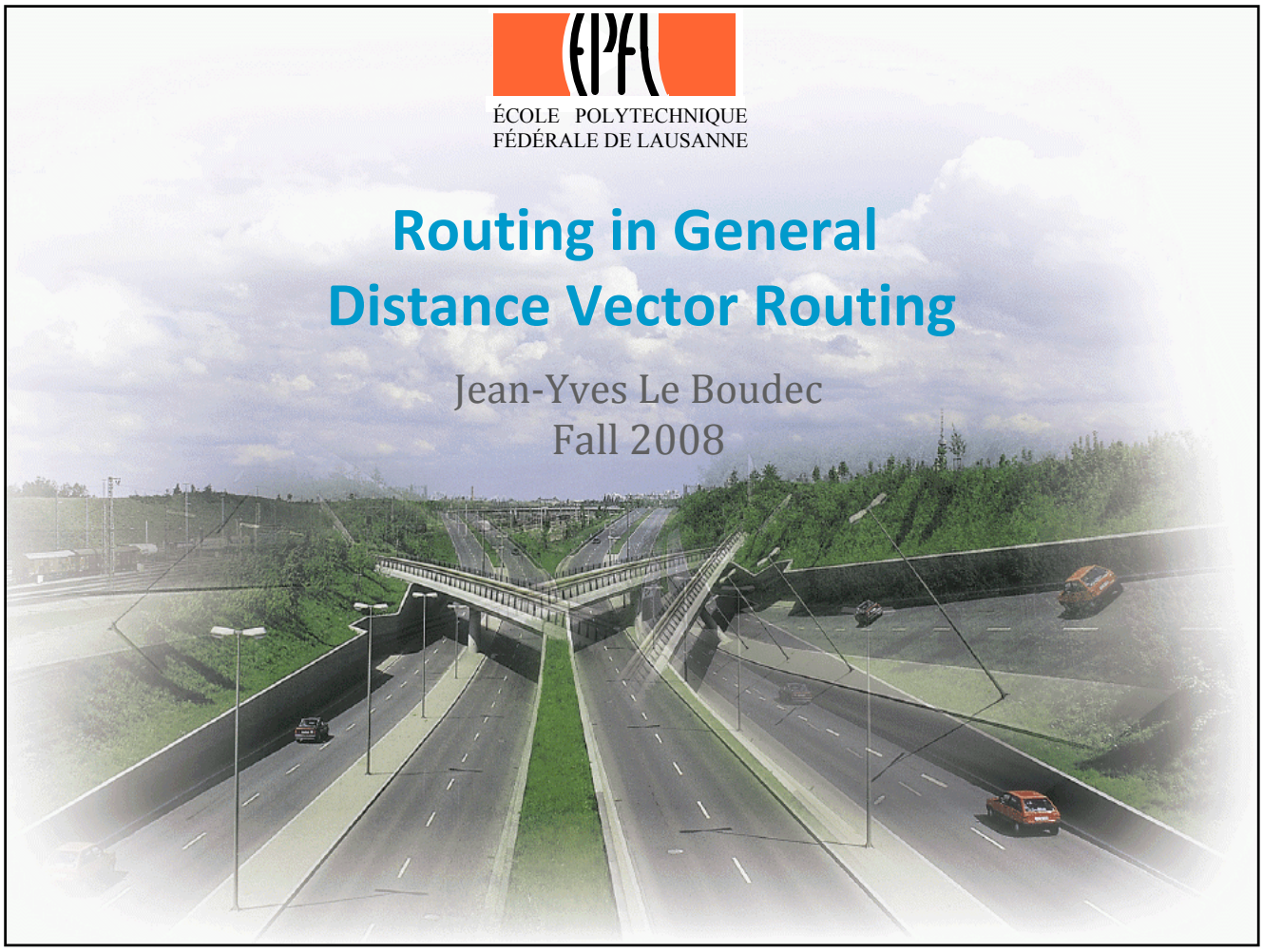

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Routing in General
Distance Vector Routing
Jean-Yves Le Boudec
Fall 2008

Contents

1. Introduction

Why were routing protocols invented

- IP assumes routing tables are maintained at hosts and routers used by **Packet Forwarding**
- Routing = control method
  - maintain routing tables automatically
  - in routers
- At host routing tables are usually maintained by
  - default rules
  - plus ICMP redirect
  - in old times: was done also by a routing protocol (RIP). Today, a host usually does *not* run any routing protocol
- Compare to: LANs connected by bridges operate at layer 2 like connectionless packet forwarders
  - **Q.** How do they maintain routing information ?

solution

Routing vs Packet Forwarding

- Packet Forwarding
  - for every packet
  - done in real time
- Routing
  - computation of routing tables or data structures for unicast and multicast
  - normally only between routers
  - non-real time: latency up to 2 minutes
  - uses dedicated protocols (RIP, OSPF, EIGRP (Cisco) for unicast and DVMRP, M-OSPF, PIM)
  - ICMP-redirect may alter routing tables, but only in hosts

Interior Routing

- Routing methods are of two types
  - Inside an administrative domain = Interior Routing
  - Between domains = Exterior Routing

- **Problem** solved by a routing protocol
  What a routing protocol does
  - find reachable destinations
  - find best paths towards destinations
    - best in the sense of some metric
    - in this module, best means along shortest path, for some additive metric (number of hops, delay)

Metrics

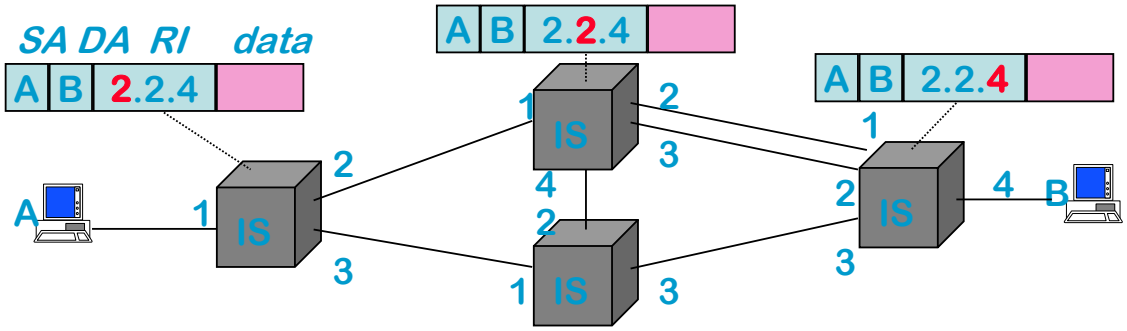
- Distance vector and link state find paths that minimize a **metric**
  - Static metric - does not depend on the network state; for example:
    - number of hops
    - link capacity and static delay
    - cost
  - Dynamic metric- depend on the network state
    - link load
    - current delay
    - see end of section

Simple Routing Methods

How routing protocols work

- **static configuration**
  - for toy networks only
- **flooding**
  - each packet duplicated on each outgoing link; loops prevented by packet id or other mechanism ; duplicate packets may be received at destination
  - simple and robust
    - no need for routing tables
    - robust - tolerates link or router failures
    - optimal in some sense
      - the first packet has found the shortest path to the destination
  - costly
    - many duplicated packets – little useful traffic
  - used as an ingredient by mobile ad-hoc routing methods (AODV, OLSR)
- **source routing**
  - source writes route into packet header
  - router reads next hop from packet header, moves pointer
  - route discovered by flooding

Source Routing

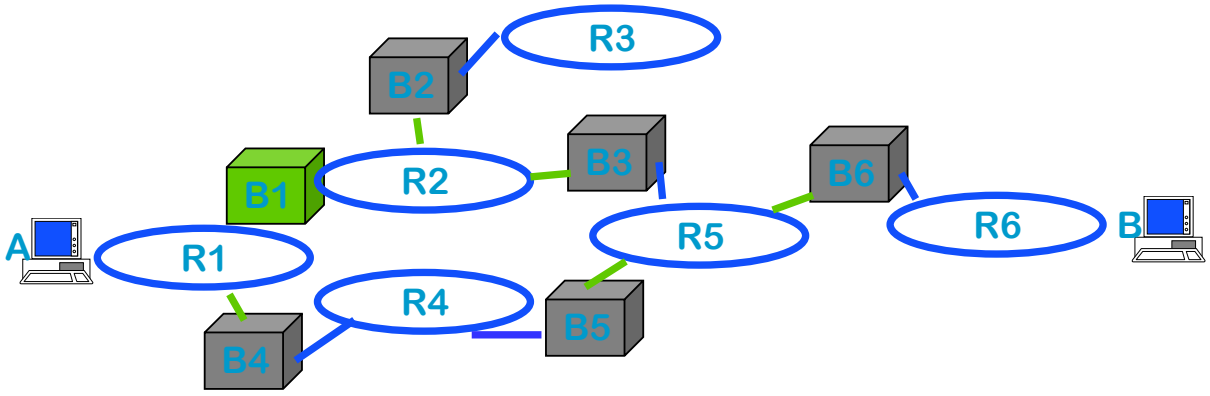


Q. What are the routes that can be used from A to B ?

solution

A route is described by a sequence of port numbers

Route Discovery in Token Rings



One “All Route Broadcast” packet is generated by A. This creates 5 different packets.

1. A-R1-B1-R2-B2-R3
2. A-R1-B1-R2-B3-R5-B6-R6
3. A-R1-B1-R2-B3-R5-B5-R4
4. A-R1-B4-R4-B5-R5-B3-R2-B2-R3
5. A-R1-B4-R4-B5-R5-B6-R6

2 of them reach B (numbers 2 and 5)

route_1 = R1.B1.R2.B3.R5.B6.R6

route_2 = R1.B4.R4.B5.R5.B6.R6

In the 1980's, the token was invented as a competitor to Ethernet. Bridging is in theory independent of whether we use token ring or ethrnet, however in practice token ring LANs used source routing bridges instead of spanning tree bridges.

Source routing bridges work as illustrated on the figure:

- Bridges and token rings have numbers. Think of a token ring as functionally the same as an Ethernet collision domain
- Assume A has a packet to send to B (here A and B are MAC addresses, but this works equally well with IP addresses). A needs to find a description of a route to B. To this end, A floods the network with an "all-route-broadcast" packet. The packet is generated by A and sent over ring R1. This packet has a special destination address that means "all-route-broadcast".
- All bridges listen to all rings that they are attached to (this is their job as forwarding devices). When they see a packet with destination address "all-route-broadcast", they forward the packet to all other rings they are attached to, except if the packet has already visited this ring (the packet contains in its header the list of rings and bridges that it has already visited).

  For example, the packet created by A is seen by B1 [resp. B4] who forwards a copy on R2 [resp. R4]. B2 and B3 see the packet on R2 and forward it to R5 and R3. Etc.
- At some point in time, B4 sees a packet on R4 put by B5, which contains as list of visits: "A-R1-B1-R2-B3-R5-B5-R4" (packet number 3). This packet contains R1 in its list, therefore B4 does *not* forward it.
- This generates 5 packets in total (numbered 1 to 5 on the figure), 2 of them reach ring R6. When B sees any of them, it sends an acknowledgement to A. This ack is source routed, along the reverse route.
- A then receives two acks, each of them contains source route information that can be inverted by A. A now has two routes to B and can choose for example the shortest (in number of hops).

DSR (Dynamic Source Routing) is a protocol for routing in ad-hoc networks that uses the same mechanism, but with IP addresses instead of MAC addresses.

Other Methods

- **Distance vector** (Bellman-Ford)
  - routers only know their local state
    - link metric and neighbor estimates
  - interior routing protocols (RIP, IGRP)
- **Link state**
  - knowledge of the global state
    - topology database
    - global optimization (Shortest Path First - Dijkstra)
  - interior routing protocols (OSPF, PNNI (ATM))
- **Path vector**
  - no knowledge of the global state
    - path: sequence of AS with attributes
    - global optimization and policy routing
  - exterior routing protocols (BGP)

2. Distance Vector

- *What* it does:
  - Computes best paths to all destinations
  - Fully distributed
  - Using as only information the distances from self to all destinations
- *How* it works
  - uses distributed Bellman-Ford – see next slides

Note: individual link cost is setup by network management

We first describe the *centralized* Bellman-Ford algorithm.

The Centralized Bellman-Ford Algorithm

- **What:** Given a directed graph with links costs A(i,j), computes the best path from i to j for any couple (i,j).
  - We assume A(i, j) > 0 and A(i,j) = ∞ when i and j are not connected.
- **How:** Take for example j=1 and let p(i) be the cost of the best path from i to 1.
  - Define $p^k(i)$ as the cost of the best path from i to 1 in at most k hops. Let $p^0(1) = 0$, $p^0(i) = \infty$ for $i \neq 1$.
  - (Bellman Ford, BF1)

$$p^0(1) = 0,\ p^0(i) = \infty \text{ for } i \neq 1$$

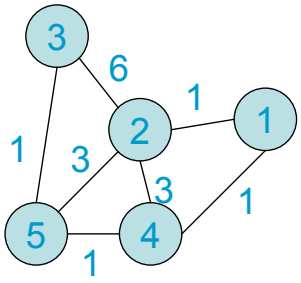
for $k = 1, 2, \ldots$ do
$$p^k(i) = \min_{j \neq i} \left[A(i,j) + p^{k-1}(j) \right] \text{ for } i \neq 1$$
$$p^k(1) = 0$$
until $p^k = p^{k-1}$

- **Theorem**
  1. If the network is fully connected, the algorithm stops at the latest for k=n and then $p^k(i)=p(i)$ for all i
  2. The shortest path from $i \neq 1$ to 1 is defined by pred(i) = $\text{Argmin}_{j \neq i}$ [A(i,j) + p(j)].

Idea of Proof: $p^k(i)$ is the distance from i to 1 in at most k hops.
Comment: recursion is equivalent to : $p^k(i) = \min\{ \min_{j \neq i,\ j \neq 1} [A(i,j) + p^{k-1}(j)] , A(i,1) \}$

Example

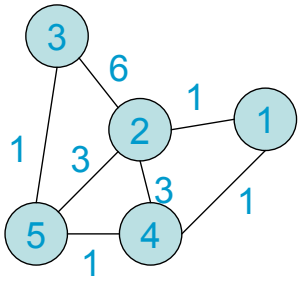
- Apply the theorem: write $p^k(i)$, pred(i) and draw the shortest paths to node 1.



solution

Impact of Initial Conditions

- Example: **Q.** does the algorithm converge to the shortest path with initial condition as shown ?



k \ *i*	*1*	*2*	*3*	*4*	*5*
0	0	0	0	0	0
1					
2					
3					
4					

k \ *i*	*1*	*2*	*3*	*4*	*5*
0	0	6	1	1	0
1					
2					
3					
4					

solution

Impact of Initial Condition

Theorem

- The algorithm converges in a finite number of steps to the correct values for all initial conditions such that $p^0(1)=0$ and for every node i that is connected to 1
- If there is no path from i to 1, the algorithm lets $p^k(i)$ converge to ∞

Proof

We do the proof assuming all nodes are connected.

1. Let p^k be the vector $p^k[i]$, i=2,…. Let B be the mapping that transforms an array $x[i]_{i=2\ldots}$ into the array Bx defined for $i \neq 1$ by
   $Bx[i]=\min_{j \neq i,\, j \neq 1}[A(i,j) + x(j)]$
   Let b be the array defined for $i \neq 1$ by
   $b[i]= A(i,1)$
   The algorithm can be rewritten in vector form as
   (1) $p^k = B\, p^{k-1} \wedge b$
   where $\wedge$ is the pointwise minimum
2. Eq (1) is a min-plus linear equation and the operator B satisfies $B(x \wedge y)= Bx \wedge By$. Thus, Eq(1) can be solved using min-plus algebra into
   (2) $p^k = B^k p^0 \wedge B^{k-1} b \wedge \ldots \wedge Bb \wedge b$
3. Define the array e for $i \neq 1$ by $e[i]= \infty$. Let $p^0=e$. Eq (2) becomes
   (3) $p^k = B^{k-1} b \wedge \ldots \wedge Bb \wedge b$. Now we have the Bellman Ford algorithm with classical initial conditions, thus, by Theorem 1:
   (4) for $k \geq n-1$: $B^{k-1} b \wedge \ldots \wedge Bb \wedge b = q$
   where q[i] is the distance from i to 1.
4. We can rewrite Eq(2) for $k \geq n-1$ as
   (5) $p^k = B^k p^0 \wedge q$
5. $B^k p^0[i]$ can be written as $A[i,i_1]+ A[i_1,i_2]+ \ldots + A[i_{k-1},i_k]+ p[i_k]$ thus
   (6) $B^k p^0[i] \geq k\, a$, where a is the minimum of all A[i,j]. Thus $B^k p^0[i]$ tends to ∞ when k grows. Thus for k large enough, $B^k p^0$ is larger than q and can be ignored in Eq(5). ⊳

Distributed Bellman Ford

- BF1 can be used in a centralized algorithm to compute p(i) i.e. find the shortest path. However, this is not its main interest, because there is a better algorithm (Dijkstra) that can be used in a centralized method
- But: it can be distributed, as follows.

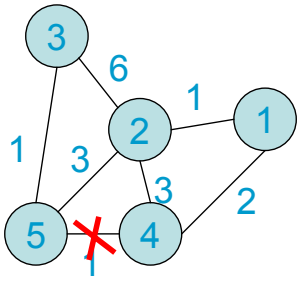
Distributed Bellman-Ford Algorithm v1, BFD1
every node, say i, maintains an estimate q(i) of the distance p(i) to some fixed node 1; initial conditions are arbitrary but q(1)=0 at all steps
from time to time, i sends the new value q(i) to all its neighbours
when node i receives a value $q(j_0)$ from *any neighbour* j_0, it sets $q(j_0)$ to the received value and updates q(i) by recomputing

eq (1) $q(i) := \min_{j \text{ neighbour}} (A(i,j)+q(j))$

if eq (1) causes q(i) to be modified, pred(i) is set to a value of j that achieves the min

- **Theorem**: if the time to reliably send a message is bounded by T, the algo converges to the same result as the centralized version in at most nT time units (if the network is fully connected)

Distributed Bellman-Ford v1



Q: give a possible scenario after link 4—5 breaks
solution

A possible run of algorithm v1. The table shows the successive values of q(i)

i	1	2	3	4	5
	0	∞	∞	∞	∞
1 -> 2	0	1	∞	∞	∞
2 -> 5	0	1	∞	∞	4
2 -> 3	0	1	7	∞	4
5 -> 4	0	1	7	5	4
2 -> 4	0	1	7	4	4
1 -> 4	0	1	7	2	4
4 -> 5	0	1	7	2	3
5 -> 2	0	1	7	2	3
5 -> 3	0	1	4	2	3

link breaks

Naive Distributed Bellman-Ford

- The previous distributed version requires a node to remember all previously received estimates q(j) for all neighbours, even if they are not the best ones
- In practice this is a problem if we need to compute the shortest paths to not just one destination, but to a large number.

- A naive distributed Bellman-Ford would be as v1 except we replace eq(1) by:

Distributed Bellman-Ford Algorithm v1a, BFD1a
when node i receives new value q(j) from node j do

eq (1a) q(i) := min { A(i,j) + q(j), q(i) }

- **Q.** does this work ? why or why not ?

solution

Distributed Bellman-Ford, cont'd

- There is an alternative algorithm, that requires only to remember the best neighbour (pred(i))

Distributed Bellman-Ford Algorithm, version 2 BFD2

every node, say i, maintains an estimate q(i) of the distance p(i) to some fixed node 1; initial conditions are arbitrary but q(1)=0 at all steps

from time to time, i sends its value q(i) to all its neighbours

when node i receives a value $q(j_0)$ from *any neighbour* j_0, it sets $q(j_0)$ to the received value and updates q(i) by recomputing

eq (2) if j_0 == pred(i)
then $q(i) := A(i,j_0)+q(j_0)$
else $q(i) := \min \{ A(i,j_0) + q(j_0), q(i) \}$

if eq (2) causes q(i) to be modified, pred(i) is set to j_0

Distributed Bellman-Ford v2

- **Theorem:** If the time to reliably send a message to all neighbours and perform local computations is bounded by T', then the algorithm BFD2 converges to the correct values in at most m (T+T') time units, where m is the number of steps of convergence of the centralized algorithm with same initial conditions

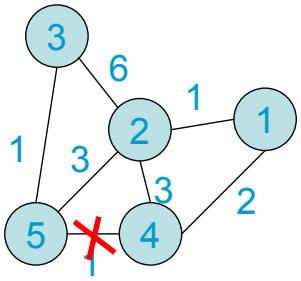
- **Comment:** The main difference with version 1 is that eq(2) replaces eq(1). Assume we use v2, and we start from a condition such that q(i) is indeed equal to the minimum given by eq (1) (which is what, intuitively, is true most of the time).

  When j is not equal to pred(i), both eq(1) and eq(2) have the same effect: the new value of q(i) is the same in both cases. In contrast, if j == pred(i), then eq (2) sets q(i) to the new value A(i,j)+q(j), whereas eq(1) sets it to $\min_{j \text{ neighbour}} (A(i,j)+q(j))$. Eq(2) provides an upper bound on eq(1), in this case. It turns out that the algorithm still works, by the same mechanism that makes the algorithm work even when the initial conditions are arbitrary. Indeed, node i will send its new value to all remaining neighbours, who will in turn do an update and eventually, node i will receive values of q(j) that will correct the problem. In other words, if the new value of q(i) is too high (compared to what would be obtained with eq (1)), this is repaired in one round of exchanges with the neighbours.

Distributed Bellman-Ford v2

A possible run of algorithm v1:

i	1	2	3	4	5
	0	∞	∞	∞	∞
1 -> 2	0	1	∞	∞	∞
2 -> 5	0	1	∞	∞	4
2 -> 3	0	1	7	∞	4
5 -> 4	0	1	7	5	4
2 -> 4	0	1	7	4	4
1 -> 4	0	1	7	2	4
4 -> 5	0	1	7	2	3
5 -> 2	0	1	7	2	3
5 -> 3	0	1	4	2	3
link breaks					

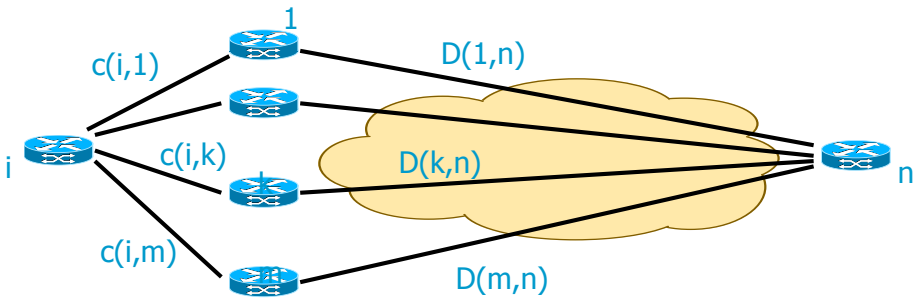


Q: give a possible scenario after link 4—5 breaks

solution

How it is used in practice

- Node *i* computes shortest path and next hop for all network prefixes *n* that it heard of.
- Initially: *D(i,n)* = 0 if *i* directly connected to *n* and *D(i,n)* = +∞ for any *n* that was never heard of.
- Node *i* receives from neighbour *k* latest values of *D(k,n)* for all *n* (this is the distance vector). Node *i* computes the best estimates according to algorithm BFD2
- This converges if network is stable
  - hello mechanism to reset computation after changes
  - if neighbour *k* is no longer present, node *i* will no longer receive hello messages, and after a timeout, this has the same effect as if node *i* would receive the message from *k*: *D*(*k,n*)=∞ for all *n.* Then algorithm BFD2 is run



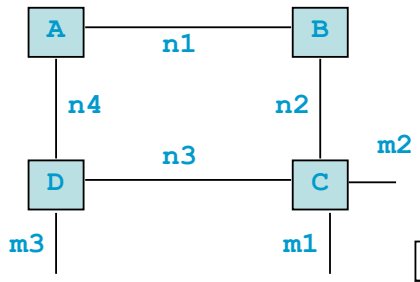
Example 1

A

net	dist	nxt
n1	0	n1,A
n4	0	n4,A

B

net	dist	nxt
n1	0	n1,B
n2	0	n2,B



D

net	dist	nxt
n3	0	n3,D
n4	0	n4,D
m3	0	m3,D

C

net	dist	nxt
n2	0	n2,C
n3	0	n3,C
m1	0	m1,C
m2	0	m2,C

Example 1

A

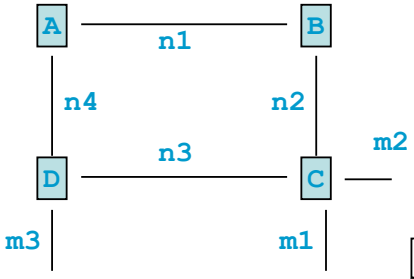
net	dist	nxt
n1	0	n1,A
n4	0	n4,A

from A

n1	0
n4	0

B

net	dist	nxt
n1	0	n1,B
n2	0	n2,B
n4	1	n1,A



D

net	dist	nxt
n3	0	n3,D
n4	0	n4,D
m3	0	m3,D

from D

n3	0
n4	0
m3	0

C

net	dist	nxt
n2	0	n2,C
n3	0	n3,C
m1	0	m1,C
m2	0	m2,C
n4	1	n3,D
m3	1	n3,D

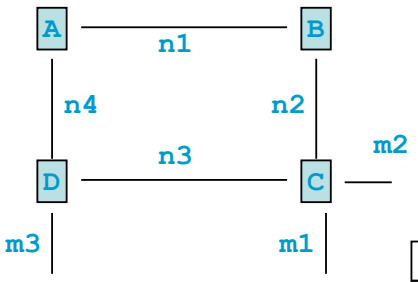
Example 1

A

net	dist	nxt
n1	0	n1,A
n4	0	n4,A

B

net	dist	nxt
n1	0	n1,B
n2	0	n2,B
n3	1	n2,C
n4	1	n1,A
m1	1	n2,C
m2	1	n2,C
m3	2	n2,C



from C

n2	0
n3	0
m1	0
m2	0
n4	1
m3	1

D

net	dist	nxt
n3	0	n3,D
n4	0	n4,D
m3	0	m3,D

C

net	dist	nxt
n2	0	n2,C
n3	0	n3,C
m1	0	m1,C
m2	0	m2,C
n4	1	n3,D
m3	1	n3,D

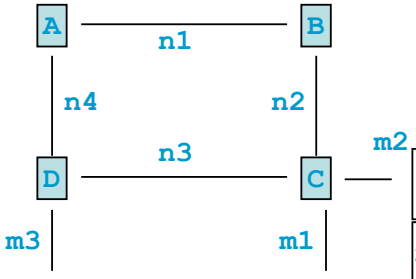
Example 1 - Final

A

net	dist	nxt
n1	0	n1,A
n2	1	n1,B
n3	1	n4,D
n4	0	n4,A
m1	2	n4,D
m2	2	n4,D
m3	1	n4,D

B

net	dist	nxt
n1	0	n1,B
n2	0	n2,B
n3	1	n2,C
n4	1	n1,A
m1	1	n2,C
m2	1	n2,C
m3	2	n2,C



D

net	dist	nxt
n1	1	n4,A
n2	1	n3,C
n3	0	n3,D
n4	0	n4,D
m1	1	n3,C
m2	1	n3,C
m3	0	m3,D

C

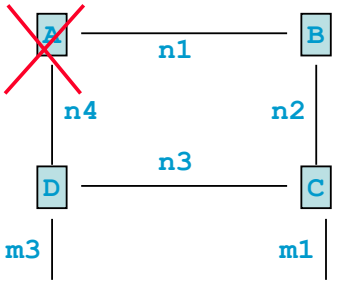
net	dist	nxt
n1	1	n2,B
n2	0	n2,C
n3	0	n3,C
m1	0	m1,C
m2	0	m2,C
n4	1	n3,D
m3	1	n3,D

Example 1 - Failure

We show only the router in the next hop field

B

net	dist	nxt
n1	0	B
n2	0	B
n3	1	C
n4	1	A
m1	1	C
m2	1	C
m3	2	C



D

net	dist	nxt
n1	1	A
n2	1	C
n3	0	D
n4	0	D
m1	1	C
m2	1	C
m3	0	D

C

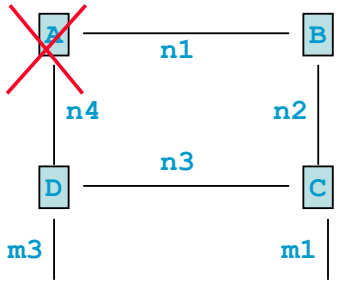
net	dist	nxt
n1	1	B
n2	0	C
n3	0	C
m1	0	C
m2	0	C
n4	1	D
m3	1	D

Example 1 - Failure

B

net	dist	nxt
n1	0	B
n2	0	B
n3	1	C
n4	1	A
m1	1	C
m2	1	C
m3	2	C

timeout



D

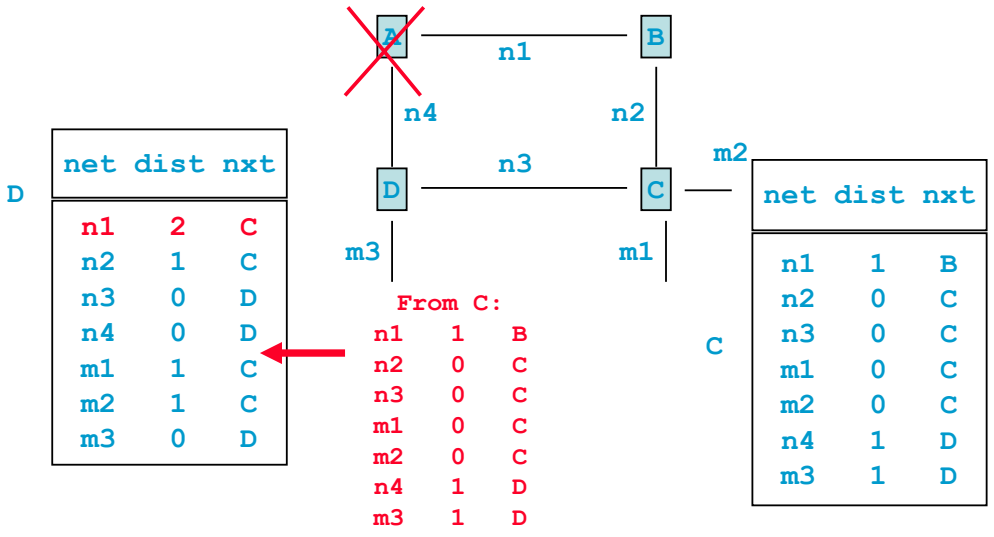
timeout

net	dist	nxt
n1	1	A
n2	1	C
n3	0	D
n4	0	D
m1	1	C
m2	1	C
m3	0	D

C

net	dist	nxt
n1	1	B
n2	0	C
n3	0	C
m1	0	C
m2	0	C
n4	1	D
m3	1	D

Example 1 - Failure



B

net	dist	nxt
n1	0	B
n2	0	B
n3	1	C
m1	1	C
m2	1	C
m3	2	C

D

net	dist	nxt
n1	2	C
n2	1	C
n3	0	D
n4	0	D
m1	1	C
m2	1	C
m3	0	D

From C:

n1	1	B
n2	0	C
n3	0	C
m1	0	C
m2	0	C
n4	1	D
m3	1	D

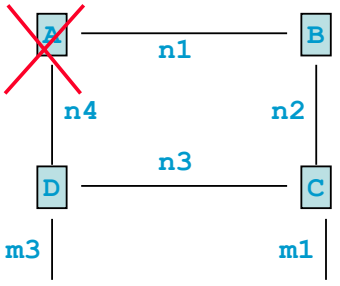
C

net	dist	nxt
n1	1	B
n2	0	C
n3	0	C
m1	0	C
m2	0	C
n4	1	D
m3	1	D

Example 1 - After Failure

B

net	dist	nxt
n1	0	B
n2	0	B
n3	1	C
n4	2	C
m1	1	C
m2	1	C
m3	2	C



D

net	dist	nxt
n1	2	C
n2	1	C
n3	0	D
n4	0	D
m1	1	C
m2	1	C
m3	0	D

C

net	dist	nxt
n1	1	B
n2	0	C
n3	0	C
m1	0	C
m2	0	C
n4	1	D
m3	1	D

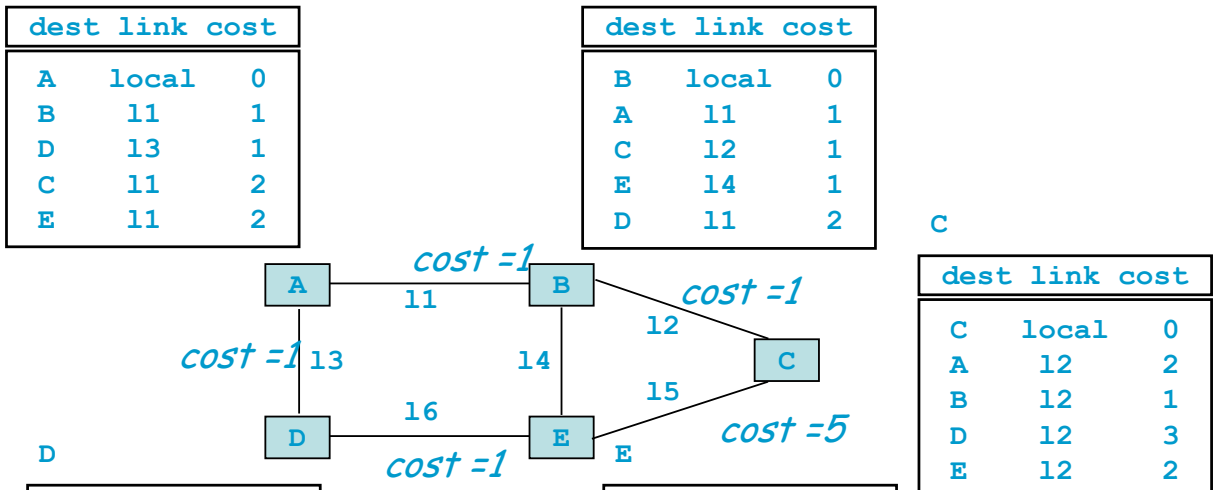
Example 1: conclusions

- Example 1 illustrates
  - how Bellman Ford is mapped to the network concepts
  - how topology changes are taken into account
    - most recent announcement replaces previous ones
    - non refreshed announcements become obsolete
  - how distance vector carries reachability information

Example 2

To simplify, we identify destination with router

Assume algorithm has converged

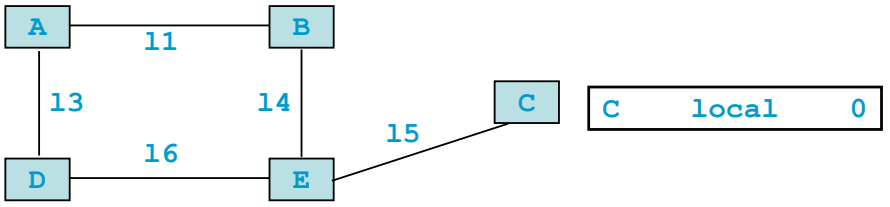


Example 2

- we now show only table entries: to C
- link 2 fails
- B updates its table

C	11	2

C	12	∞



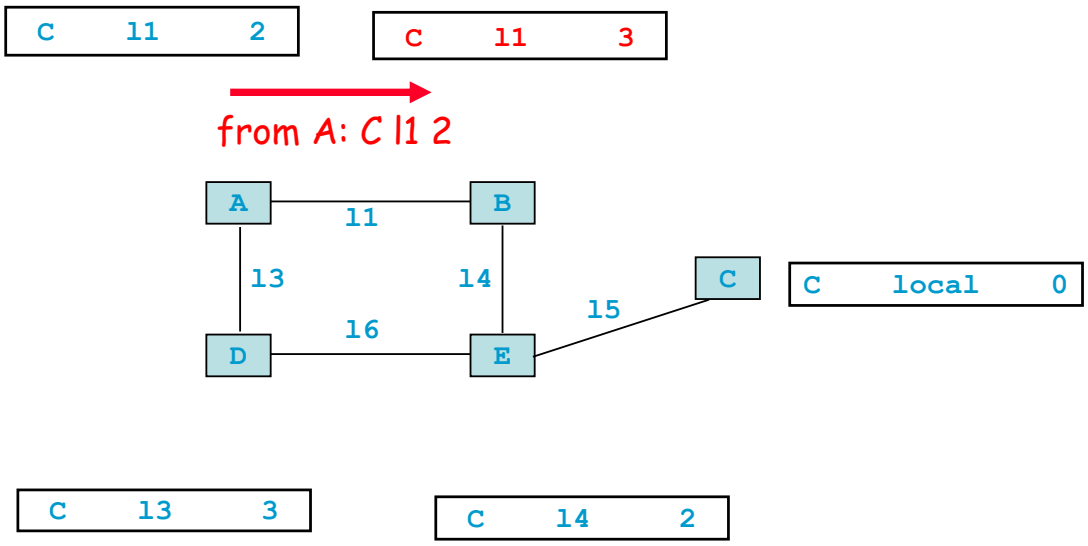
C	local	0

C	13	3

C	14	2

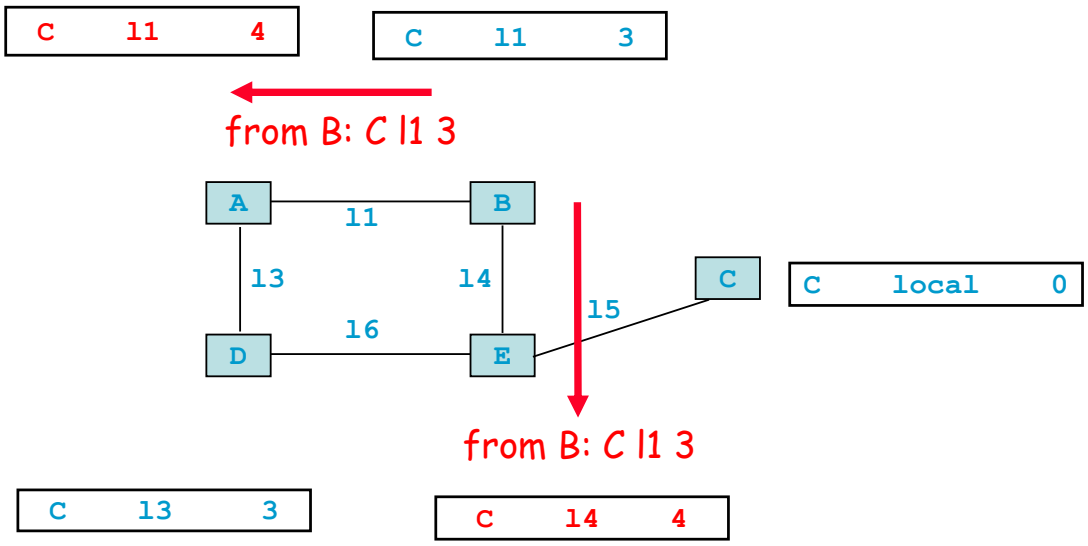
Example 2: Link failure

- Just before B updates its table, A broadcasts its table with cost 2 to C
- B updates



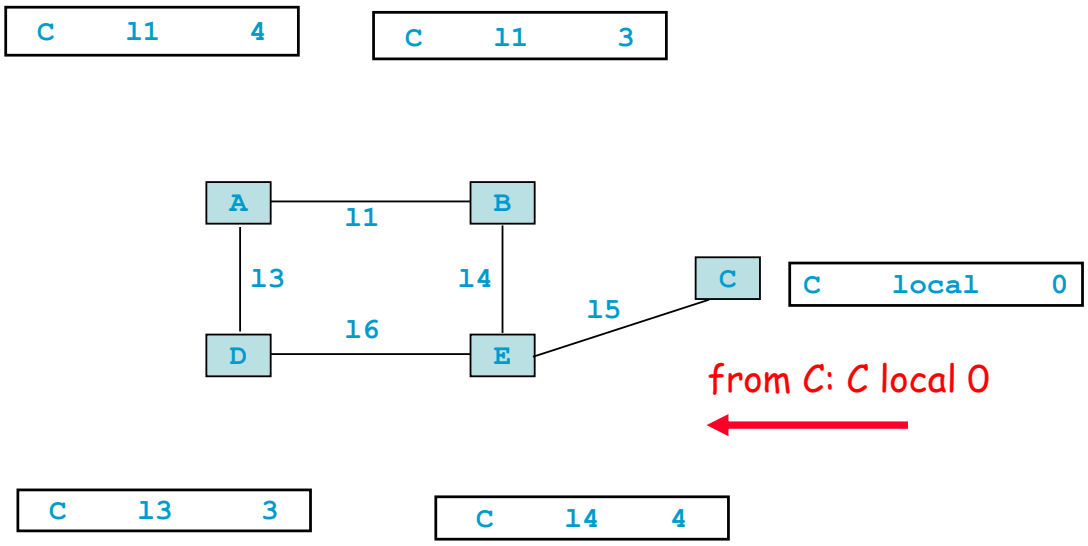
Example 2: Link failure

- B sends update to A and E
- A and E update



Example 2: Link failure

- C sends update
- it is ignored by E because it it less good



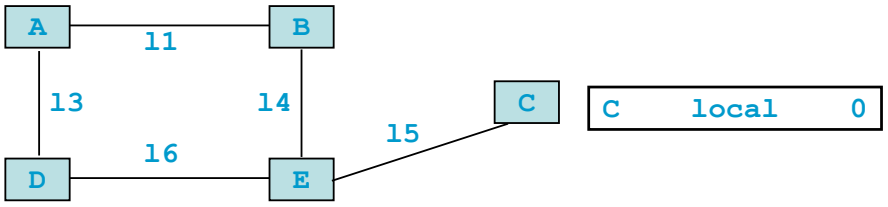
Example 2: Link failure

- A broadcasts its table with cost 4 to C
- B updates … we have a loop between A and C
- cost is increase by 2 at every iteration

C	11	4

C	11	5

from A: C l1 4



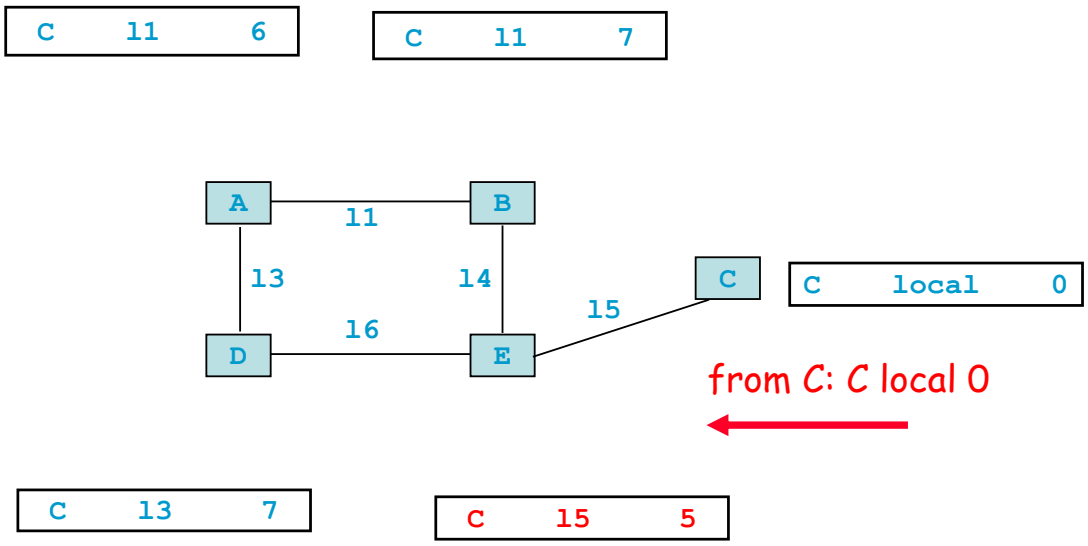
C	local	0

C	13	3

C	14	4

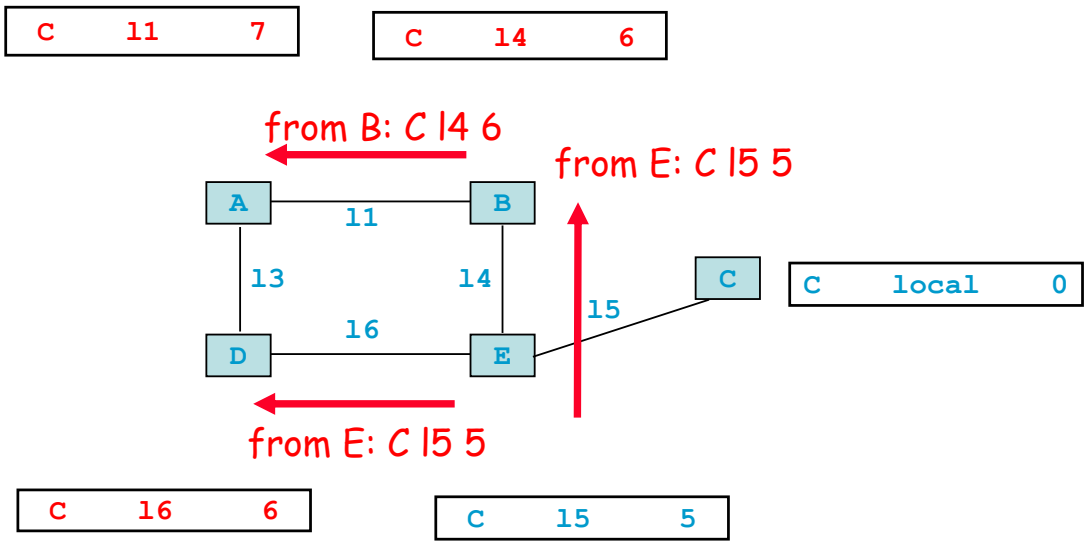
Example 2: Link failure

- E now accepts announcement from C



Example 2: Link failure

- E sends announcements to D and B
- B and D send announcements to A
- the algorithm has converged – stable state



Conclusions from Example 2

- the algorithm converges after modification of the topology, but the convergence may be very slow
  - bounce effect
- Q: during convergence time, how are routing tables ?
  solution

Example 3

Assume now all link costs are equal to 1
Links l1 and l6 fail
D detects failure and sets costs to ∞

A

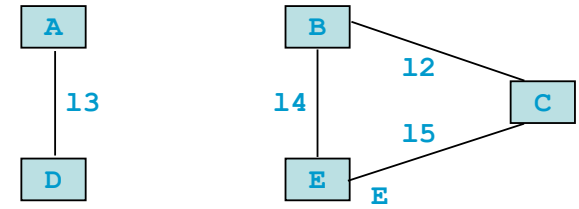
dest	link	cost
A	local	0
B	l3	3
D	l3	1
C	l3	3
E	l3	2

B

dest	link	cost
B	local	0
A	l4	3
C	l2	1
E	l4	1
D	l4	2

C

dest	link	cost
C	local	0
A	l5	3
B	l2	1
D	l5	2
E	l5	1



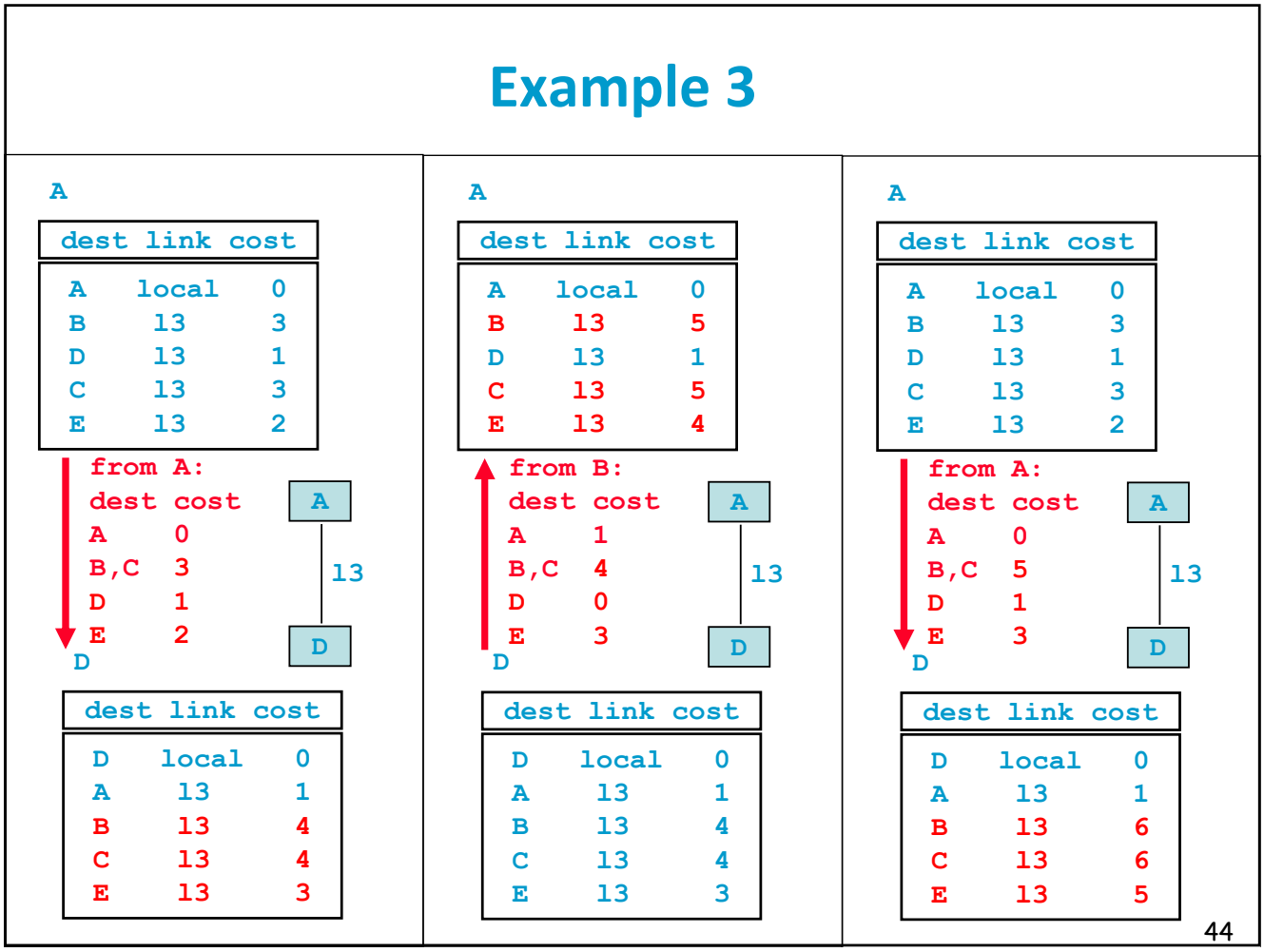
D

dest	link	cost
D	local	0
A	l3	1
B	l3	∞
C	l6	∞
E	l6	∞

E

dest	link	cost
E	local	0
A	l6	2
B	l4	1
D	l6	1
C	l5	1

Example 3


A
dest link cost
A local 0
B 13 3
D 13 1
C 13 3
E 13 2
from A:
dest cost
A 0
B,C 3
D 1
E 2
A
13
D
D
dest link cost
D local 0
A 13 1
B 13 4
C 13 4
E 13 3
A
dest link cost
A local 0
B 13 5
D 13 1
C 13 5
E 13 4
from B:
dest cost
A 1
B,C 4
D 0
E 3
A
13
D
D
dest link cost
D local 0
A 13 1
B 13 4
C 13 4
E 13 3
A
dest link cost
A local 0
B 13 3
D 13 1
C 13 3
E 13 2
from A:
dest cost
A 0
B,C 5
D 1
E 3
A
13
D
D
dest link cost
D local 0
A 13 1
B 13 6
C 13 6
E 13 5

Conclusion from Example 3

- The costs to C, B, E grow unbounded "Count to Infinity"
  - the true costs are infinite
- Convergence to a stable state if we set
  - ∞ = large number
  - e.g. RIP: ∞ = 16
- "Split Horizon"
  - a heuristic to prevent this
  - if A routes packets to X via B, it does not announce this route to B

Example 3: with Split Horizon

A

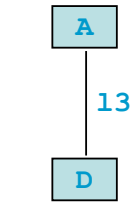
dest	link	cost
A	local	0
B	13	3
D	13	1
C	13	3
E	13	2

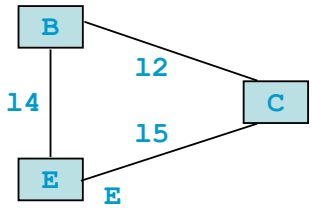
B

dest	link	cost
B	local	0
A	14	3
C	12	1
E	14	1
D	14	2

C

dest	link	cost
C	local	0
A	15	3
B	12	1
D	15	2
E	15	1





D

dest	link	cost
D	local	0
A	13	1
B	13	∞
C	16	∞
E	16	∞

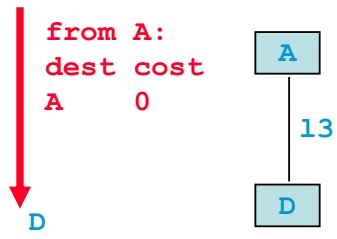
E

dest	link	cost
E	local	0
A	16	2
B	14	1
D	16	1
C	15	1

Example 3: with Split Horizon

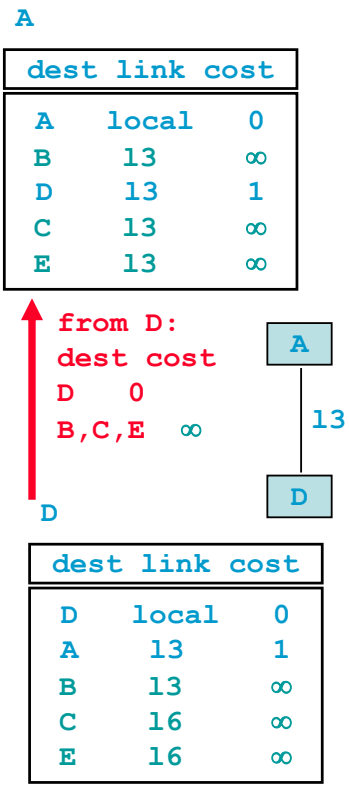
A

dest	link	cost
A	local	0
B	13	3
D	13	1
C	13	3
E	13	2



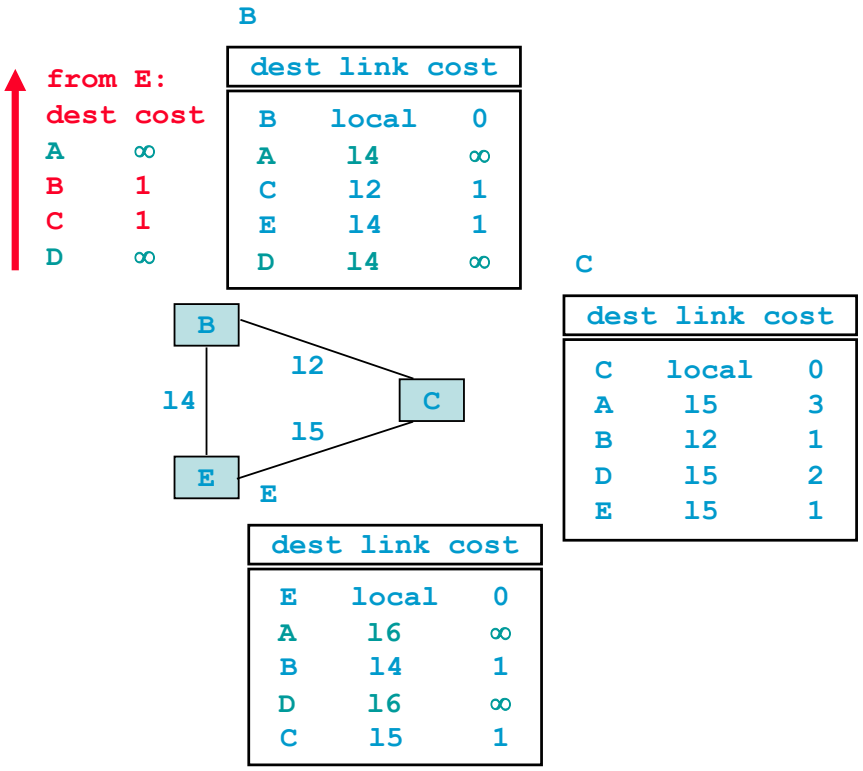
dest	link	cost
D	local	0
A	13	1
B	13	∞
C	16	∞
E	16	∞

Split horizon

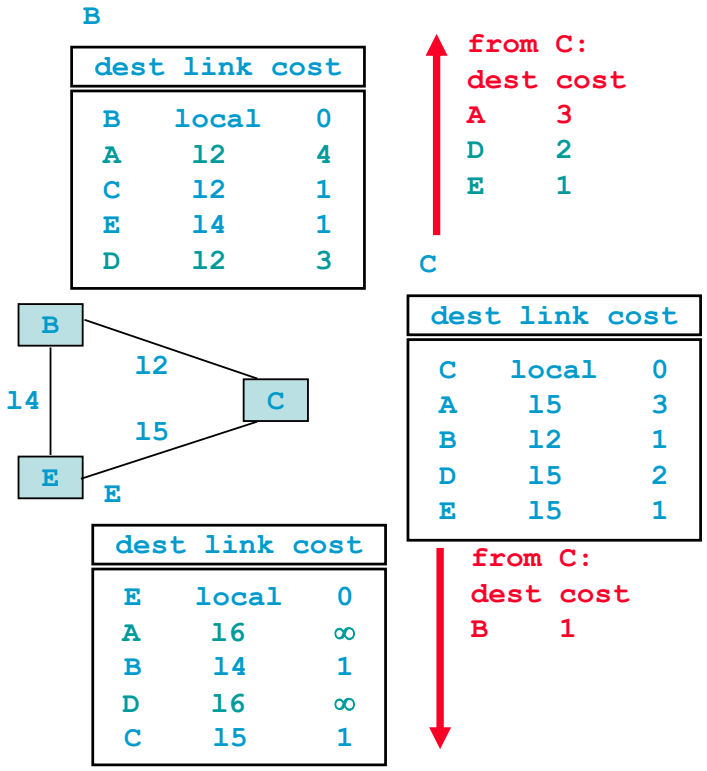


- Split horizon cuts the process of counting to infinity

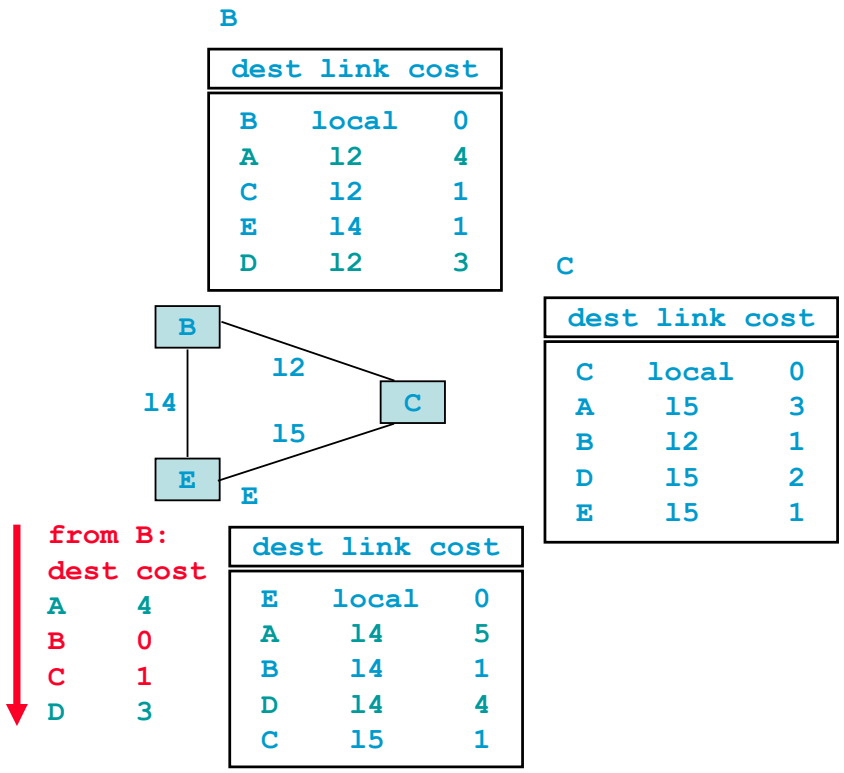
Split horizon may fail



Split horizon may fail



Split horizon may fail



Conclusion: Distance Vector

- convergence to stable state may be slow after changes
- count to infinity must be prevented by setting a maximum distance

3. Distance Vector Protocols
RIP

- Distance vector protocol
- Metric - hops
- Network span limited to 15
  - ∞ = 16
- Split horizon
- Destination network identified by IP address
  - Netmasks in RIPv2
- Encapsulated as UDP packets, port 520
- Largely implemented (**`routed`** on Unix)
- Broadcast every 30 seconds or when update detected
- Route not announced during 3 minutes
  - cost becomes ∞
- Authentication in RIPv2 by MD5 (shared secret)

IGRP (Interior Gateway Routing Protocol)

- Proprietary protocol by CISCO
- Metric that estimates the global delay
- Maintains several routes of similar cost
  - load sharing
- Takes into account netmasks
- No limit of 15
  - number of routers included in messages
- Broadcast every 90 sec

Metric example

- Metric
  - Trans = 10000000/Bandwidth (time to send 10 Kb)
  - delay = (sum of Delay)/10
  - m = [K_1*Trans + (K_2*Trans)/(256-load) + K_3*delay]
  - default: K1=1, K2=0, K3=1, K4=0, K5=0
  - if K5 ≠ 0, m = m * [K_5/(Reliability + K_4)]
- Bandwidth in Kb/s, Delay in μs
  - At Venus: Route for 172.17/16: Metric = 10000000/784 + (20000+1000)/10 = 14855
  - At Saturn: Route for 12./8: Metric = 10000000/224 + (20000 + 1000)/10 = 46742

3. Load Dependent Routing

- We come back in this section to ***what*** routing protocols do.
- Instead of maximizing a "path quality" metric (nb hops, delay) assume we want to maximize the ***total network utility***
  - for example: total transported flows
  - see congestion control chapter for other definitions
- how should routing be done ?
- Q1: show an example where shortest path routing does not provide the optimal total flow (where path cost is static)

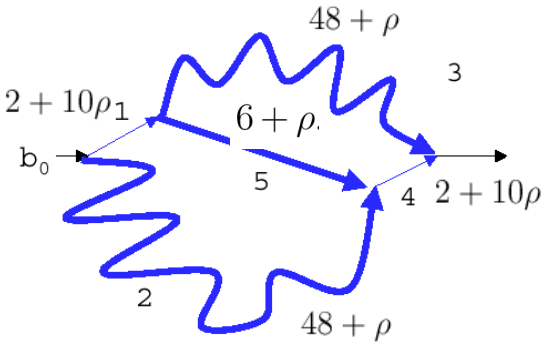
  solution

- One solution might be to take delay as the path cost
  - high load on a link => high cost => link is less used
  - however, this does not solve the problem: there is the Braess paradox

Braess Paradox (1)

- Assume all flows pick the route with shortest delay
- Assume parallel paths exist and flows can make use of them
- Delay is function of load as given below; link 5 is (temporarily) closed
- Total offered load is b0 = 6 Gb/s
- For example,
  - if we split traffic into : route 1-3: b = 1, route 2-4 b = 5
  - the delay along route 1-3 is 61, along route 2-4 is 105
  - thus the link costs will change and routing decisions will change also
- Eventually, there will be an equilibrium (called "Wardrop Equilibrium")
  - delay is equal on all competing routes
- Q: compute the equilibrium traffic flow on every link

Solution



Braess Paradox (2)

- Q: same question when we open link 5 with delay function:

$$f_5(\rho) = 6 + \rho.$$

Solution

Braess Paradox

- With shortest delay routing, adding a new link may decrease overall throughput
  - Thus shortest delay routing is not either a global optimum

Optimal Routing

- One can change the objective of routing: instead of computing shortest paths,one could solve a global optimization problem:
  - minimize total delay subject to flow constraints
  - this is a well posed optimization problem
  - the optimal solution depends on all flows
  - but it can be implemented in a distributed algorithm similar to TCP congestion control ; see [BertsekasGallager92]
- **Q.** Can you imagine a way to use classical routing (like distance vector, which finds shortest paths) and still find the optimum network utility ?

solution

Conclusion

- Distance vector is smart
  - Fully distributed, little information stored
- Largely deployed (Unix BSD **`routed`**)
- Simplicity
- But: slow convergence
  - Not suited for large and complex networks
  - Link State protocols should be used instead

Review Questions

Explain the following terms:

- distance vector
- bounce effect
- count to infinity
- split horizon
- Bellman Ford
- RIP, IGMP
- source routing

- Explain why shortest path routing is not necessarily a globally optimum
- What is the Braess paradox ?

Solutions

1. Introduction

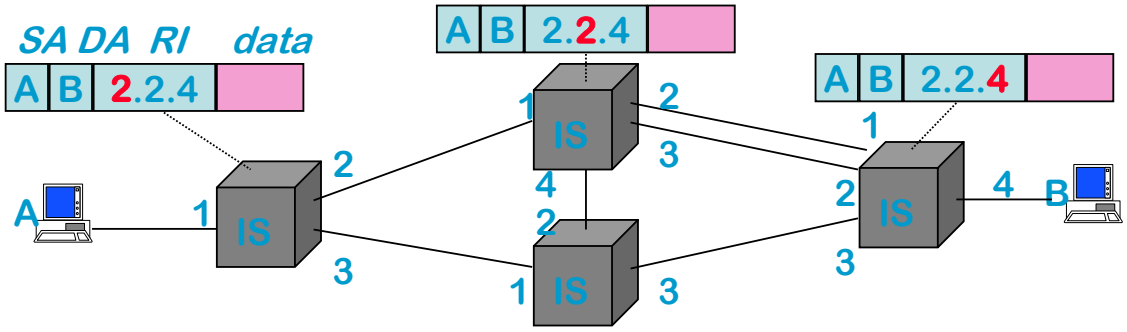
Why were routing protocols invented

- Connectionless Network Layer assumes routing tables are maintained at hosts and routers
  - used by **Packet Forwarding**
- Routing = control method
  - maintain routing tables automatically
  - in routers
- At host
  - normally done by default rules
  - plus ICMP redirect
  - in old times: was done also by a routing protocol (RIP)
- Compare to: LANs connected by bridges operate at layer 2 like connectionless packet forwarders
  - **Q.** How do they maintain routing information ?

    A. By learning from the packets they observe; broadcast is used to bootstrap

back

Source Routing



Q. What are the routes that can be used from A to B ?

A.

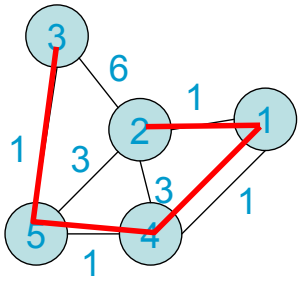
A 2 2 4
A 2 3 4
A 2 4 3 4
A 3 3 4
A 3 2 2 4
A 3 2 3 4

back

A route is described by a sequence of port numbers

Example

- Apply the theorem: write $p^k(i,1)$, pred(i) and draw the shortest paths to node 1.



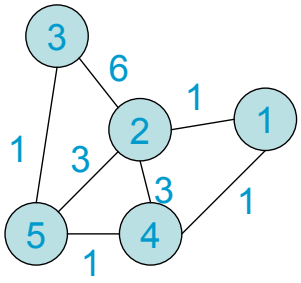
k\i	1	2	3	4	5
0	0	∝	∝	∝	∝
1	0	1	∝	1	∝
2	0	1	7	1	2
3	0	1	3	1	2

i	1	2	3	4	5
pred(i)	1	1	5	1	4

back

Impact of Initial Conditions

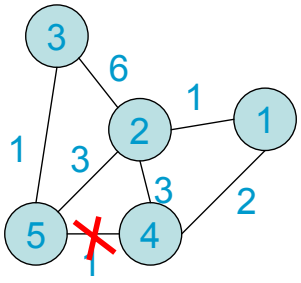
- Example: **Q.** does the algorithm converge to the shortest path with initial condition as shown ? **A.** yes



k \ *i*	*1*	*2*	*3*	*4*	*5*
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	2	1	2
3	0	1	3	1	2
4	0	1	3	1	2

k \ *i*	*1*	*2*	*3*	*4*	*5*
0	0	6	1	1	0
1	0	1	1	1	2
2	0	1	3	1	2

Distributed Bellman-Ford v1



Q: give a possible scenario after link 4—5 breaks

back

A possible run of algorithm v1:

i	1	2	3	4	5
	0	∞	∞	∞	∞
1 -> 2	0	1	∞	∞	∞
2 -> 5	0	1	∞	∞	4
2 -> 3	0	1	7	∞	4
5 -> 4	0	1	7	5	4
2 -> 4	0	1	7	4	4
1 -> 4	0	1	7	2	4
4 -> 5	0	1	7	2	3
5 -> 2	0	1	7	2	3
5 -> 3	0	1	4	2	3
link breaks	4 does as if received ∝ from 5 5 does as if received ∝ from 4 and continue computations from there				
	0	1	4	2	4
5 -> 3	0	1	5	2	4

Naive Distributed Bellman-Ford

- The previous distributed version requires a node to remember all previously received estimates q(j) for all neighbours, even if they are not the best ones
- In practice this is a problem if we need to compute the shortest paths to not just one destination, but to a large number.

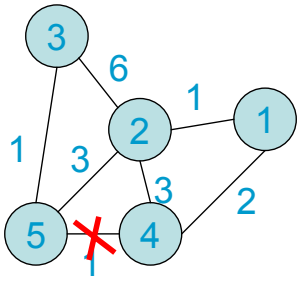
- A naive distributed Bellman-Ford would be as v1 except we replace eq(1) by:

Distributed Bellman-Ford Algorithm v1a, BFD1a
when node i receives new value q(j) from node j do

eq (1a) q(i) := min { A(i,j) + q(j), q(i) }

- **Q.** does this work ? why or why not ?
  A. no. q(i) can only decrease. So if we start from initial conditions as in example « Impact of Initial Conditions », the algorithm will not converge to the right value. It gets « stuck » with a low value. It is possible to show that it works if all initial conditions are above the final values, for example $q(j)=\infty$ initially. But even then, it will not work if there is a topology change, since this is equivalent to starting from different initial conditions
- back

Distributed Bellman-Ford v2



Q: give a possible scenario after link 4—5 breaks

back

A possible run of algorithm v1:

i	1	2	3	4	5
	0	∞	∞	∞	∞
1 -> 2	0	1	∞	∞	∞
2 -> 5	0	1	∞	∞	4
2 -> 3	0	1	7	∞	4
5 -> 4	0	1	7	5	4
2 -> 4	0	1	7	4	4
1 -> 4	0	1	7	2	4
4 -> 5	0	1	7	2	3
5 -> 2	0	1	7	2	3
5 -> 3	0	1	4	2	3
link breaks	4 does as if received ∞ from 5				
	5 ≠ pred(4)				
	0	1	4	2	3
	5 does as if received ∞ from 4				
	4 == pred(5)				
	0	1	4	2	∞
5 -> 3	0	1	∞	2	∞
2 -> 3	0	1	7	2	∞
2 -> 5	0	1	7	2	4
5 -> 3	0	1	5	2	4

Conclusions from Example 2

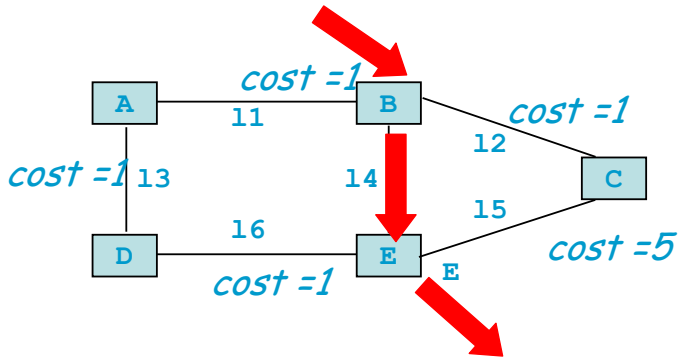
- Q: during convergence time, how are routing tables ?
  A:
  - they are incorrect
  - there are loops – packets are discarded (TTL expires)

  back

3. Load Dependent Routing

Q. show an example where shortest path routing does not provide the optimal total flow (where path cost is static)

A. assume all data flow goes from B to E: Static shortest path routing will pick the direct link BE only instead of distributing the load also on some of the longer links (BADE and BCE)



back

Braess Paradox (1)

- A. there are two paths
1: links 1, 3; 2: links 2,4
let bi be the traffic on path I

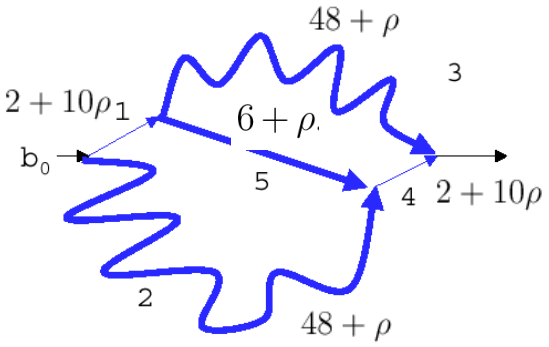
Delay equations:

$50 + 11b1 = 50 + 11b2$

Total flow

$b1 + b2 = b0$

equilibrium is for b1 = b2 = 3
delay is 83



back

Braess Paradox (2)

- Q: same question when we open link 5 with delay function:

$$f_5(\rho) = 6 + \rho.$$

- A: there are three paths
  1: links 1, 3;
  2: links 2,4;
  3: links 1, 5, 4

delay equations

50 + 11b1 + 10b3 = 50 + 11b2 + 10b3 = 10 + 10b1 + 10 b2 + 21 b3

total flow

b1 + b2 + b3 = b0

We find b1= b2 = b3 = 2 Gb/s

The total delay on all paths is the same, equal to 92 : larger than before!

back

Optimal Routing

- One can change the objective of routing: instead of computing shortest paths,one could solve a global optimization problem:
  - minimize total delay subject to flow constraints
  - this is a well posed optimization problem
  - the optimal solution depends on all flows
  - but it can be implemented in a distributed algorithm similar to TCP congestion control ; see [BertsekasGallager92]
- **Q.** Can you imagine a way to use classical routing (like distance vector, which finds shortest paths) and still find the optimum network utility ?

  A.
  - Let a centralized network management procedure update the link costs (used by distance vector routing).
  - given link costs c_i and traffic matrix compute total throughput or average delay (a hard optimization problem, solved with heuristics)
  - every few minutes, update the link costs in all routers – let the routing algorithm compute new paths

back