Exercise Session 9: Network Layer - Solutions

COM-208: Computer Networks

Before we start

A few basic rules:

- An IP prefix A/M is the range of IP addresses whose M most significant bits are the same as A's M most significant bits. E.g., 10.0.0.16 belongs to IP prefix 10.0.0.0/24 , because the 24 most significant bits of 10.0.0.16 are the same as the 24 most significant bits of 10.0.0.0.
- This implies that a /M IP prefix contains 2^{32-M} IP addresses. E.g., a /24 IP prefix contains $2^{32-24} = 2^8 = 256$ IP addresses. In other words, we don't have the freedom to create an IP prefixes that contains an arbitrary number of IP addresses, it must always contain a number that is a power of 2.
- /M is called the "subnet mask", representing the bitmask consisting of M consecutive 1's followed by enough 0's to reach 32 bits. The subnet mask is applied to any IP address using bitwise AND to obtain the IP range the address belongs to. E.g. 100.52.12.18 belongs to the prefix 100.52.12.16/28 because: (1) the /28 mask represents the mask **1111 1111.1111 1111.1111 1111.1111 0000**, (2) applying the mask to 100.52.12.18 (**0110 0100.0011 0100.0000 1100.0001 0010** in binary) results in 100.52.12.16 (**0110 0100.0011 0100.0000 1100.0001 0000** in binary).
- Each IP subnet must have its own IP prefix. Hence, IP prefixes allocated to different IP subnets must not overlap.

When assigning IP addresses to network interfaces in an IP subnet, assume that the following IP addresses cannot be assigned to any network interface:

- **The first IP address in the subnet's IP prefix (called "network address").** E.g., the first IP address in 10.0.0.0/24 is 10.0.0.0. This address is sometimes reserved for special uses, e.g., a discovery service provided by the subnet.
- **The last IP address in the subnet's prefix (called "broadcast address").** E.g., the last IP address in 10.0.0.0/24 is 10.0.0.255. This address is sometimes

reserved to be used as the subnet's broadcast address, i.e., as the destination IP address for packets that should be received by all end-systems in a subnet.

IP prefix allocation

Basic

IP subnets A, B and C contain 10, 5, and 3 network interfaces, respectively. Allocate an IP prefix to each subnet, and assign an IP address to each network interface, from IP prefix 1.2.3.0/27.

Consider two cases for allocating prefixes to subnets. In each case, follow the given order:

(a) A, B, C

(b) B, A, C

In the context of this exercise, when we say that allocation "follows a given order," we mean that, if Subnet X comes before Subnet Y in that order, the IP addresses for Subnet X should be arithmetically smaller than the IP addresses for Subnet Y (in the sense that IP address 1.2.3.4 is arithmetically smaller than IP address 1.2.3.5).

Note: Allocating addresses might be infeasible in some cases.

(a) Network prefix 1.2.3.0/27 contains $2^{32-27} = 32$ addresses. We need to pick three subnets from the address space 1.2.3.0 - 1.2.3.31, while satisfying the requirements from above.

Subnet A must have at least 12 IP addresses (10 for end-systems, 1 for network address, and 1 for broadcast address). However, since we need to round to to a power of 2 we will allocate 16 addresses: from 1.2.3.0 to 1.2.3.15. Let's represent some of them in binary format:

0000 0001.0000 0010.0000 0011.0000 0000 = 1.2.3.0

0000 0001.0000 0010.0000 0011.0000 0001 = 1.2.3.1

0000 0001.0000 0010.0000 0011.0000 0010 = 1.2.3.2

0000 0001.0000 0010.0000 0011.0000 0011 = 1.2.3.3

...

The prefix is 0000 0001.0000 0010.0000 0011.0000 = 1.2.3.0 with length 28, thus Subnet A is 1.2.3.0/28, or 1.2.3.0 - 1.2.3.15.

Note: to obtain the CIDR notation (that is 1.2.3.0/28), we:

1 Take the binary prefix of the subnet: 0000 0001.0000 0010.0000 0011.0000 0;

2 Append zeros until we obtain 32 bits: 0000 0001.0000 0010.0000 0011.0000 0000;

3 Transform the value into dotted IP notation: 1.2.3.0;

4 Append "/" and the length of the binary prefix: 1.2.3.0/29.

Subnet B needs 7 addresses, so we need to allocate 8: from 1.2.3.16 to 1.2.3.23. We expect that the mask has length 29. Let's represent some of the addresses in binary:

0000 0001.0000 0010.0000 0011.0001 0000 = 1.2.3.16

0000 0001.0000 0010.0000 0011.0001 0001 = 1.2.3.17

0000 0001.0000 0010.0000 0011.0001 0010 = 1.2.3.18

...

The interval is 1.2.3.16 - 1.2.3.23, and the prefix is 0000 0001.0000 0010.0000 0011.0001 0 with length 29, which corresponds to the block 1.2.3.16/29.

For Subnet C, we need to allocate 5 addresses. Thus we need a block with 8 addresses. We try to allocate starting from 1.2.3.23:

0000 0001.0000 0010.0000 0011.0001 1 000 = 1.2.3.24

0000 0001.0000 0010.0000 0011.0001 1 001 = 1.2.3.25

0000 0001.0000 0010.0000 0011.0001 1 010 = 1.2.3.26

0000 0001.0000 0010.0000 0011.0001 1 011 = 1.2.3.27

...

The interval is 1.2.3.24 - 1.2.3.31, the prefix is 0000 0001.0000 0010.00000 0011.0001 1 with length 29. Thus the CIDR notation is 1.2.3.24/29.

(b) It is not possible to allocate the addresses in this order.

The reason is that you have only two options in how to allocate addresses for Subnet A (either prefix 1.2.3.0/28 or prefix 1.2.3.16/28).

- If you allocate prefix 1.2.3.0/28 to Subnet A, then there will be no room to allocate smaller addresses for Subnet B.
- Instead, if you allocate prefix 1.2.3.16/28, there will be no room to allocate bigger addresses to Subnet C.

Network configuration

Consider the topology shown in Figure 1. There are three IP subnets (A, B and C) that contain some end-systems, and two IP subnets (D and E) that contain no end-systems. The green boxes (a, b, c, ... g) denote network interfaces for routers R1, R2 and R3.

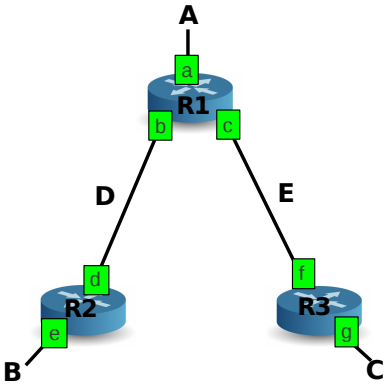

Figure 1: Problem topology.

- (*Basic*) Allocate an IP prefix to each subnets. Your allocation must respect the following constraints:
  - All prefixes must be allocated from 214.97.254.0/23.
  - Subnet A should have enough addresses to support 250 interfaces.
  - Subnet B should have enough addresses to support 120 interfaces.
  - Subnet C should have enough addresses to support 60 interfaces.
  - Each of subnets D and E should have enough addresses to support 2 interfaces.

> First, you need to round the number of required addresses (number of interfaces + 1 for network address + 1 for broadcast address) up to the nearest power of two.
>
> Then, a good strategy for allocating subnets on the available address space is the following:
>
> - Sort the subnets in descending order according to the (rounded) number of addresses they require
> - Start allocating addresses from one end of the address space (e.g. the beginning)
> - For every remaining subnet (following the order in which you sorted them), allocate the address space after the previously

allocated prefix. (i.e., do not leave gaps between consecutive subnets)

If you follow this strategy, you can guarantee that every subnet is well-aligned, and that there will be no gaps between allocated address blocks.

Nevertheless, there are multiple possible solutions for this exercise. Here is a possible allocation:

Subnet A: 214.97.254/24 (256 addresses)
Subnet B: 214.97.255.0/25 (128 addresses)
Subnet C: 214.97.255.128/26 (64 addresses)

Subnet D: 214.97.255.192/30 (4 addresses)
Subnet E: 214.97.255.196/30 (4 addresses)

- (*Basic*) Using your previous answer, provide the forwarding tables for each of the three routers (R1, R2, R3). Each table should contain two columns which show (i) the destination IP prefix, and (ii) the corresponding output link.

The simplest way to create a forwarding table is to create a separate rule for every subnet that we need to access.

If we take into account that the address ranges for different subnets do not overlap, we don't have to worry about longest-prefix-length matching.

Router 1:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 110000	Port b
11010110 01100001 11111111 110001	Port c
11010110 01100001 11111111 10	Port c
11010110 01100001 11111111 0	Port b
11010110 01100001 11111110	Port a

Router 2:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 110000	Port d
11010110 01100001 11111111 110001	Port d
11010110 01100001 11111111 10	Port d
11010110 01100001 11111111 0	Port e
11010110 01100001 11111110	Port d

Router 3:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 110000	Port f
11010110 01100001 11111111 110001	Port f
11010110 01100001 11111111 10	Port g
11010110 01100001 11111111 0	Port f
11010110 01100001 11111110	Port f

- (*Advanced*) Can you reduce the number of entries of each forwarding table, i.e., for each table create an equivalent one, which has the same outcome but consists of fewer entries?

We can reduce the number of entries by grouping the Longest prefix matches for the same outgoing interface, starting from the longest matches (top-bottom). This will generate new entries with first column being the longest prefix between all (or some) of those entries and second column being the output port.

However, we need to be careful not to create conflict with another outgoing interface. For example, in the forwarding table of R1, we would need to summarize the entries for Port c with "11010110 01100001 11111111 1" (longest prefix match between "11010110 01100001 11111111 110001" and "11010110 01100001 11111111 10"). Then, if we wanted to summarize the entries for Port b we would need to add an entry with prefix "11010110 01100001 11111111" (longest prefix match between "11010110 01100001 11111111 110000" and "11010110 01100001 11111111 0"). But if we use this prefix and place it in the forwarding table, it would look like this:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 1	Port c
11010110 01100001 11111111	Port b
11010110 01100001 11111110	Port a

Recall that the router scans the forwarding table in order and stops on the first prefix match. In that case, a packet matching "11010110 01100001 11111111 10 " will be forwarded to Port b instead of Port c (which is wrong, see the third line of the original forwarding table). In this situation, the prefixes of Port b can not be grouped.

Therefore, the tables with summarized entries are shown below.

Router 1:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 110000	Port b
11010110 01100001 11111111 1	Port c
11010110 01100001 11111111 0	Port b
11010110 01100001 11111110	Port a

Router 2:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 0	Port e
11010110 01100001 1111111	Port d

Router 3:

Longest Prefix Match	Outgoing Interface
11010110 01100001 11111111 10	Port g
11010110 01100001 1111111	Port f

Network Address Translation

Intermediate

Figure 2 illustrates how Network Address Translation (NAT) works. Consider a similar topology, but suppose that the NAT gateway has external IP address 24.34.112.235, while the private IP address space is 192.168.1.0/24.

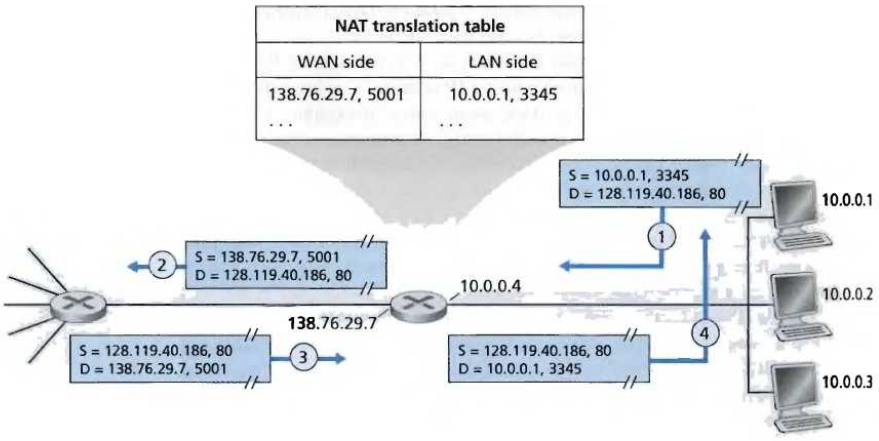

Figure 2: Network Address Translation (NAT) Process

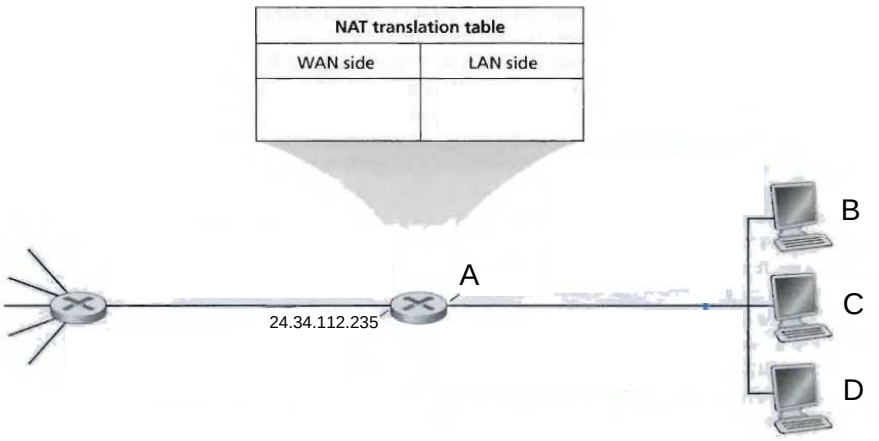

Figure 3: Problem topology.

- Complete Figure 3 by assigning IP addresses to all interfaces (labels A, B, C, and D) in the internal (private) network.

> Router interface: 192.168.1.1
> Home device addresses: 192.168.1.2, 192.168.1.3, 192.168.1.4

- Suppose that each end-system has two ongoing TCP connections, all to IP address 128.119.40.86, port 80. Provide the six corresponding entries in the NAT translation table.

Here is an example of a NAT translation table. Multiple solutions exist.

WAN Side	LAN Side
24.34.112.235, 4000	192.168.1.2, 3345
24.34.112.235, 4001	192.168.1.2, 3346
24.34.112.235, 4002	192.168.1.3, 3445
24.34.112.235, 4003	192.168.1.3, 3446
24.34.112.235, 4004	192.168.1.4, 3545
24.34.112.235, 4005	192.168.1.4, 3546

Link-state routing

Basic

Consider the network in Figure 4. Execute the link-state (Dijkstra's) algorithm we saw in class to compute the least-cost path from each of x, v, and t to all the other routers.

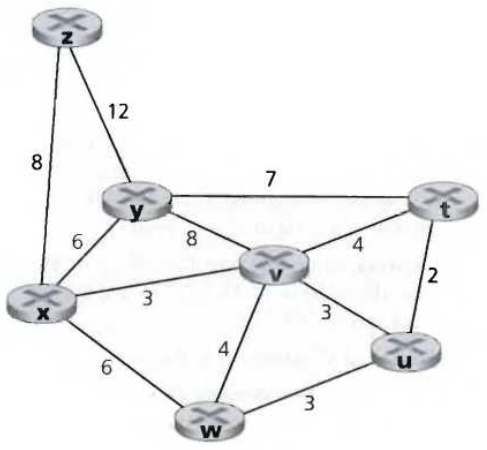

Figure 4: Network topology.

The least cost path from routers x, v, and t to all the other routers is displayed in the next table along with the execution steps of the link-state

algorithm.

For each router i, $C(i)$ stands for cost to i, and $p(i)$ stands for predecessor to i.

1. Least-cost path from x to all network nodes.

step	nodes visited	C(t),p(t)	C(u),p(u)	C(v),p(v)	C(w),p(w)	C(y),p(y)	C(z),p(z)
0	x	∞	∞	3,x	6,x	6,x	8,x
1	x,v	7,v	6,v	3,x	6,x	6,x	8,x
2	x,v,u	7,v	6,v	3,x	6,x	6,x	8,x
3	x,v,u,w	7,v	6,v	3,x	6,x	6,x	8,x
4	x,v,u,w,y	7,v	6,v	3,x	6,x	6,x	8,x
5	x,v,u,w,y,t	7,v	6,v	3,x	6,x	6,x	8,x
6	x,v,u,w,y,t,z	7,v	6,v	3,x	6,x	6,x	8,x

2. Least-cost path from v to all network nodes.

step	nodes visited	C(t),p(t)	C(u),p(u)	C(w),p(w)	C(x),p(x)	C(y),p(y)	C(z),p(z)
0	v	4,v	3,v	4,v	3,v	8,v	∞
1	v,x	4,v	3,v	4,v	3,v	8,v	11,x
2	v,x,u	4,v	3,v	4,v	3,v	8,v	11,x
3	v,x,u,t	4,v	3,v	4,v	3,v	8,v	11,x
4	v,x,u,t,w	4,v	3,v	4,v	3,v	8,v	11,x
5	v,x,u,t,w,y	4,v	3,v	4,v	3,v	8,v	11,x
6	v,x,u,t,w,y,z	4,v	3,v	4,v	3,v	8,v	11,x

3. Least-cost path from t to all network nodes.

step	nodes visited	C(u),p(u)	C(v),p(v)	C(w),p(w)	C(x),p(x)	C(y),p(y)	C(z),p(z)
0	t	2,t	4,t	∞	∞	7,t	∞
1	t,u	2,t	4,t	5,u	∞	7,t	∞
2	t,u,v	2,t	4,t	5,u	7,v	7,t	∞
3	t,u,v,w	2,t	4,t	5,u	7,v	7,t	∞
4	t,u,v,w,x	2,t	4,t	5,u	7,v	7,t	15,x
5	t,u,v,w,x,y	2,t	4,t	5,u	7,v	7,t	15,x
6	t,u,v,w,x,y,z	2,t	4,t	5,u	7,v	7,t	15,x

Distance-vector routing

Basic

Consider the network in Figure 5. Execute the distance-vector (Bellman-Ford) algorithm we saw in class and show the information that router z knows after each iteration.

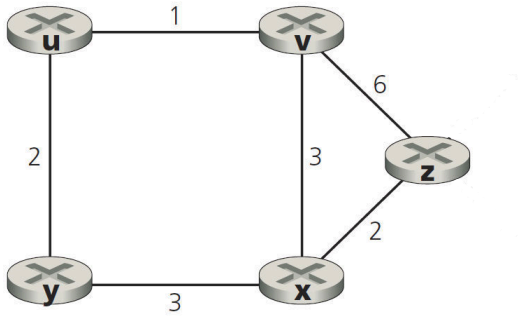

Figure 5: Network topology.

Each node in the topology has its own view of the network, which is updated independently from other nodes at the end of each step. Therefore, for every step of the algorithm, you also need to update each of the other cost tables. Otherwise, your solution may be incorrect.

In our solution we only show the cost table for node z, as required by the question. The cost table at node z consists of 5 columns (all possible destinations) and 3 rows (all possible sources—one row for node z and one row for each neighbor). Each entry of the table denotes the cost between the associated source-destination nodes.

Initially (at step 0), node z has the following view of the network:

		To				
		u	v	x	y	z
From	v	∞	∞	∞	∞	∞
	x	∞	∞	∞	∞	∞
	z	∞	6	2	∞	0

At step 1:

		To				
		u	v	x	y	z
From	v	1	0	3	∞	6
	x	∞	3	0	3	2
	z	7	5	2	5	0

At step 2:

		To				
		u	v	x	y	z
From	v	1	0	3	3	5
	x	4	3	0	3	2
	z	6	5	2	5	0

At step 3:

		To				
		u	v	x	y	z
From	v	1	0	3	3	5
	x	4	3	0	3	2
	z	6	5	2	5	0

We see that from step 2 to step 3 the cost tables did not change, indicating that the algorithm has converged.

Convergence

Intermediate

What is the maximum number of iterations required for the distance-vector (Bellman-Ford) algorithm that we saw in class to converge (i.e., to finish, assuming no change occurs in the network graph and link costs)? Justify your answer.

At each iteration, a node exchanges cost tables with its neighbors. Thus, if you are node A, and your neighbor is B, all of B's neighbors (which are all one or two hops from you) will know the least-cost path of one or two hops to you after one iteration (i.e., after B tells them its cost to you).

Let d be the "diameter" of the network, which is computed as follows:

- Find the least-cost path between each pair of nodes.
- The diameter is equal to the greatest length (in number of links) of any of those least-cost paths.

Using the reasoning above, after $d - 1$ iterations, all nodes will know the least-cost path of length $\leq d$ hops to all other nodes. Since any path of length $> d$ hops is costlier than any of the least-cost paths, the cost tables of the nodes will not change after that point. Hence, the algorithm converges in at most $d - 1$ iterations.

Poisoned reverse

Advanced

Consider the network in Figure 6. The routers in the network run the distance-vector (Bellman-Ford) algorithm we saw in class, with poisoned reverse enabled. Suppose the algorithm has run for some time and has converged to the correct least-cost paths.

Table 7 contains the reachability information from each router to router A (e.g., from router D we can reach router A through router C with cost 3).

Now suppose that link $A - B$ goes down.

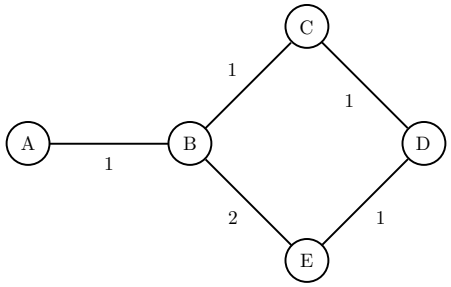

Figure 6: Network topology.

- Describe the next 6 steps of the algorithm. For each step, show the reachability information from each router to router A (cost of the least-cost path and next hop).

See Figures 7 and 8.

To fill in the first table we use the second, auxiliary one: Figure 8 shows the routing announcements exchanged between routers and the computations done to update the routing tables. Only the information relevant to routes to router A is shown. At each step, each router sends to its neighbors the best route to A known in the previous step, except if the route is via that neighbor, in which case because of poisoned reverse, ∞ is sent; we mark such messages with (**pr**). Then, each router uses all the announcements it received from the neighbors to update its routing table.

Note that poisoned reverse does not help the algorithm converge.

route to router A		from router B	from router C	from router D	from router E
	step 0	1 via A	2 via B	3 via C	3 via B
	step 1	∞	2 via B	3 via C	3 via B
	step 2	∞	∞	3 via C	4 via D
	step 3	6 via E	∞	∞	4 via D
	step 4	6 via E	7 via B	∞	∞
	step 5	∞	7 via B	8 via C	∞
	step 6	∞	∞	8 via C	9 via D

Figure 7: Reachability information from each router to router A.

		router B	router C	router D	router E
step 0	initial table	1 via A	2 via B	3 via C	3 via B
step 1	announcements	$C \to B : \infty$ (**pr**) $E \to B : \infty$ (**pr**) link to A down	$B \to C : 1$ $D \to C : \infty$ (**pr**)	$C \to D : 2$ $E \to D : 3$	$B \to E : 1$ $D \to E : 3$
	table update	∞	$1 + 1 = 2$ via B	$2 + 1 = 3$ via C	$1 + 2 = 3$ via B
step 2	announcements	$C \to B : \infty$ (**pr**) $E \to B : \infty$ (**pr**) link to A down	$B \to C : \infty$ $D \to C : \infty$ (**pr**)	$C \to D : 2$ $E \to D : 3$	$B \to E : \infty$ $D \to E : 3$
	table update	∞	∞	$2 + 1 = 3$ via C	$3 + 1 = 4$ via D
step 3	announcements	$C \to B : \infty$ $E \to B : 4$ link to A down	$B \to C : \infty$ $D \to C : \infty$ (**pr**)	$C \to D : \infty$ $E \to D : \infty$ (**pr**)	$B \to E : \infty$ $D \to E : 3$
	table update	$4 + 2 = 6$ via E	∞	∞	$3 + 1 = 4$ via D
step 4	announcements	$C \to B : \infty$ $E \to B : 4$ link to A down	$B \to C : 6$ $D \to C : \infty$	$C \to D : \infty$ $E \to D : \infty$ (**pr**)	$B \to E : \infty$ (**pr**) $D \to E : \infty$
	table update	$4 + 2 = 6$ via E	$6 + 1 = 7$ via B	∞	∞
step 5	announcements	$C \to B : \infty$ (**pr**) $E \to B : \infty$ link to A down	$B \to C : 6$ $D \to C : \infty$	$C \to D : 7$ $E \to D : \infty$	$B \to E : \infty$ (**pr**) $D \to E : \infty$
	table update	∞	$6 + 1 = 7$ via B	$7 + 1 = 8$ via C	∞
step 6	announcements	$C \to B : \infty$ (**pr**) $E \to B : \infty$ link to A down	$B \to C : \infty$ $D \to C : \infty$ (**pr**)	$C \to D : 7$ $E \to D : \infty$	$B \to E : \infty$ $D \to E : 8$
	table update	∞	∞	$7 + 1 = 8$ via C	$8 + 1 = 9$ via D

Figure 8: Routing announcements exchanged between routers with the routing table updates w.r.t. A.

- Will the algorithm converge to the correct least-cost path values? If yes, in how many steps?

  No, the algorithm does not converge (the algorithm takes an infinite number of steps to converge to the correct least-cost path values). This is because A is now disconnected from the network and thus the other routers are unable to know that is unreachable.

- Propose a simple way to make the algorithm converge faster.

  Potential solutions:

  – Routers keep track of the entire route-path, not just the next hop (e.g., BGP)

  – Routers use an upper limit for the cost of a path (e.g., 30). The algorithm converges slowly, but will eventually converge.