Project 3

Distance Vector Simulator

CS305 Computer Networks
May 7th, 2017

Authors
Darren Norton, Yizhong Chen

1. Introduction

It is a huge convenience that when we direct our web-browser to google.com, the browser quickly opens up google's web-page and allows us to explore the collective knowledge in the internet. There are a great many routines operating behind the scenes to make this happen, one of the more critical being the filling of forwarding tables. When our devices finally have the IP address of google.com, we would want to send a packet to it, but we aren't directly connected to google.com, so who do we send it to? More specifically, we are only worried about how does our router know the next router to send the packet to, and how does our router know this is the shortest path? This problem is partly solved by routing protocols.

When a packet arrives to a router, the router indexes its forwarding table and determines the link interface to which the packet is to be directed. This is the ultimate goal of forwarding tables, as they are simply a map of IP address ranges to link interfaces. Routing protocols, operating in network routers, exchange and compute the information that is used to configure these forwarding tables, typically in a way that achieves the shortest path. The interplay between routing algorithms and forwarding tables was shown in Figure 1.1.

Some of the goals of routing are correctness, simplicity, robustness, stability, fairness and optimality. There are a few routing protocol categories that balance these goals, including Link-State (LS) and Distance-Vector (DV). For our project, the domain under investigation is the Distance-Vector approach taken on by the Routing Information Protocol, RIP.

In this project, our goals are to: 1. Implement a distance-vector routing algorithm and check how it works in a distributed environment, 2. learn to program with datagram sockets (UDP), and 3. learn to program with multiple threads.

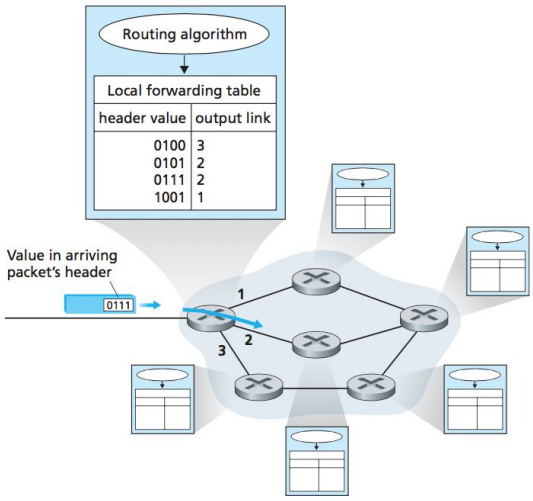


Figure 1.1 - Routing algorithms determine values in forwarding tables

2. Distance Vector Algorithm

2.1 Motivation

Distance vector routing is a simple distributed routing protocol. Distance vector routing allows routers to automatically discover the destinations reachable inside the network as well as the shortest path to reach each of these destinations. The shortest path is computed based on metrics or costs that are associated to each link. One of the more impressive features in Distance Vector algorithms is the lack of knowledge of every component in the network. Distance Vector doesn't have to know that nodes X,Y, Z, exist in the network, instead it only requires knowledge of neighboring nodes. This allows the algorithm to scale well as the network grows in size.

The algorithm is comes naturally by definition. Any time the router asks itself, "What is the shortest path to X?", the answer comes in the form of "Either directly from myself, or the distance to my neighbors plus their shortest path to the node, which they have already told me". Distance Vector is a dynamic programming based algorithm, and therefore has cheap computation of shortest paths. Most overhead comes from waiting for distance vectors from its neighbors.

2.2 How It Works

A router that uses distance vector routing regularly sends its distance vector over all its interfaces. The distance vector is a summary of the router's routing table that indicates the distance towards each known destination. This distance vector can be computed from the routing table by using the pseudo-code below (Kurose):

At each node, x:

```
1   Initialization:
2      for all destinations y in N:
3         Dx(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4      for each neighbor w
5         Dw(y) = ? for all destinations y in N
6      for each neighbor w
7         send distance vector Dx = [Dx(y): y in N] to w
8
9   loop
10     wait (until I see a link cost change to some neighbor w or
11           until I receive a distance vector from some neighbor w)
12
13     for each y in N:
14        Dx(y) = minv{c(x,v) + Dv(y)}
15
16     if Dx(y) changed for any destination y
17        send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19  forever
```

In general, whenever a router's distance vector changes, it has to update its neighbors with the new vector. This can cause the neighboring router to update its own distance vector and propagate the change back to the current router. In a way, this is how routers learn about other routers. Once a router receives a distance vector containing nodes that are not its neighbors, it can start to handle incoming traffic for that router. Eventually, the current router will know the shortest path to that router via all of its neighbors.

2.2.1 Good News Travels Fast

One of the advantages of the Distance Vector algorithm is that good news tends to travel quickly. When a router updates its distance vector, it will also tell its neighbors about this update. If the update is caused by a reduction in cost between two routers,

then this propagates to other routers relatively quickly. As soon as the neighboring router receives the update, it updates its own distance vector, where it is more likely to change due to the update being brought with good news. For example, consider the network in Figure 2.1, which will be analyzed at greater detail later in the report.

Figure 2.1 - Sample Network

Imagine that the cost between each link is 5, and that the network has reached a state of convergence, where all nodes have the shortest paths to each node in their forwarding tables. If we were to change the cost from B to C to 1, we would see that B sends its update to A and C, which both send their updates to B. C also sends to D, which sends to C and E. E would then update D and the process is over. For simplicity in explanation, let's assume that B is the only node updated upon the node change. This implementation is different in our code, but the idea is the same. This went smoothly because we calculate for the minimum.

(Node, Distance)	A	B	C	D	E
(A, 5)	0	5	10	15	20
(B, 0)	5	0	1	6	11
(C, 5)	10	5	0	5	10

Figure 2.2 - Calculating B's Distance Vector shaded in yellow, after updating B to C = 1

(Node, Distance)	A	B	C	D	E
(A, 5)	0	5	6	11	16
(B, 0)	5	0	1	6	11
(C, 1)	6	1	0	5	10

Figure 2.3 - B's Distance Vector table after convergence

2.2.2 Bad News, Not So Much

Imagine, that in the same Figure, we changed the value from B to C up to 25. B would update A and C again, but what would they put in their tables? Using Figure 2.2 as the base table, let's update B's vector.

(Node, Distance)	A	B	C	D	E
(A, 5)	0	5	6	11	16
(B, 0)	5	0	11	16	21
(C, 25)	6	1	0	5	10

Figure 2.4 - Calculating B's Distance Vector shaded in yellow, after updating B to C = 25

Notice the red fields in Figure 2.4 This vector will get to A and C and they will update their vectors and send them back to B as shown in Figure 2.5. As you may see, it will take A quite some time to get a convergent distance vector. This trouble is caused by B sending a misinformed vector. This would take a few iterations before reaching the true value of 30, essentially on a larger number this would count to infinity. The algorithm can get around this by introducing a variant called poison reverse.

(Node, Distance)	A	B	C	D	E
(A, 5)	0	5	16	21	26

Figure 2.5 - B receives distance vector from A, expected A to C to be 30, but is 16

2.2.3 Poison Reverse

As we can see from the scenario above, increasing the cost to links, or even dropping links can cause a group of routers to count to infinity. There are a few steps we can take in routing algorithms to prevent this behavior. The first being, we can limit the maximum value for the metric between two nodes. RIP handles this by establishing 15 hops as the maximum, and therefore any node within 16 or more hops is not reachable by the given router. Another prevention typically present in RIP would be to add poison reverse.

In essence, bad news would cause the count to infinity due to the algorithm not being smart about tracking paths to nodes. For example, in Figure 2.6 router D can tell router C that it's shortest path to B is 2 (assuming unit metric cost). If the link between B and C increases, the effect takes quite a while due to C believing that D has a shortest path to B that is of length 2. C doesn't know how D got there, for all it knows, there could be a direct link from D to B of length 2. The issue is that the shortest path from D to B goes through C, and C isn't being told this information. Poison reverse is exactly what solves this issue. With poison reverse, D would tell C that it's shortest path to B is infinite, since it goes through C. To generalize this, given any arbitrary nodes X, Y, and Z, if the shortest path from X to Z includes Y, X would tell Y that it's shortest path to Z is infinity (or in RIP's case, 16).

Figure 2.6 - Sample Network

3. Our Code

Our project implements the Distance Vector Algorithm using code similar to that shown in Section 2.2. We accomplish this using three additional threads to handle asynchronous behavior, and by representing each object in the algorithm's transactions in some sort of data structure. We also adhere to the project requirements established by Lafayette College's Computer Science Department. We have listed them verbatim here:

Each router will act similarly to what a real router in using the RIP routing protocol. That is:

- A router should update its distance vector and forwarding table every time it receives a new distance vector from his neighbors or the weight to its neighbor changes.
- A router will send an update every *n* seconds or every time its own distance vector changes.
- If a router does not receive an update after 3 time periods of *n* seconds, it should drop the neighbor.
- A router will forward a message with a specific destination according to its forwarding table.
- A router will accept changes in its weights to its neighbors and advertise it to them
- A router will support poisoned reverse as an option.

3.1 Data Structures

Socket Address

To start off, we decided to represent the addresses of known routers in what we call in a SocketAddress. A SocketAddress is simple, in that it contains an IP address and a Port, which designate the location and port of another router running our program. The known values for SocketAddresses are provided in text files which are passed in upon program startup.

DistanceVector

We represent Distance Vectors as HashMaps mapping SocketAddresses to costs. Each Distance Vector also contains the SocketAddress of the node it originated from. The DistanceVector class is meant to be concise and serializable as it is supposed to be sent over the network via UDP. It's serialization is as follows:

```
SOURCE: sourceIP:sourcePort
destIP:destPort | distance
...
...
```

DistanceVectorCalculation

To take some of the weight off of DistanceVector, we decided to create a class called DistanceVectorCalculation which contains both the DistanceVector, and a HashMap mapping SocketAddresses to ArrayList of paths. The HashMap can be

imagined as the path from the current node to each destination node. We could have just kept the next hop in the path, reducing the ArrayList to a SocketAddress, but having an ArrayList allows for a deeper print-out if necessary for debugging.

ForwardingTable

We created a class called ForwardingTable to represent the Router's forwarding table. The ForwardingTable class is what you might expect the DistanceVectorCalculation to hold, that is, a mapping between SocketAddress to SocketAddress as the next hop. Having this data structure allows the Router to easily handle messaging traffic, where if the traffic is not destined for itself, it can look for the next hop.

Router

Finally, we encase all of these components in a Router class. As expected, the Router is the most crucial component in this algorithm and is complex as such. There are quite a few key additions. First there is the HashMap mapping neighbors' SocketAddresses to weights. This would represent the physical connections to a Router's neighbors. There is a HashMap mapping SocketAddresses to the DistanceVectors the router has received from them. There is a list of known nodes, and finally a HashMap to help keep count of the recent communications from its neighbors to help drop neighbors after 3 time intervals.

3.2 Threads

Our project uses multiple threads to handle asynchronous behavior. In general we split the work between three different classes that implement Runnable: DVUpdateThread, DVCommandThread, and RouterUDPReceiver. In general, we have to be careful about race-conditions while threading. We added a Lock to the Router and each thread will wait until the Router is available to do its work.

DVUpdateThread

As per the project requirements, the Router must update its neighbors every n seconds. We schedule these updates by using a ScheduledThreadPoolExecutor. This thread handles updating behavior by invoking broadcasting methods in the Router.

DVCommandThread

We have four supported commands, **PRINT, MSG, CHANGE, and !help.** These commands adhere to the project requirements, again copied here for convenience.

- PRINT - print the current node's distance vector, and the distance vectors received from the neighbors.
- MSG <dst-ip> <dst-port> <msg> - send message msg to a destination with the specified address.

- CHANGE <dst-ip> <dst-port> <new-weight> - change the weight between the current node and the specified node to new-weight and update the specified node about the change.

RouterUDPReceiver

The RouterUDPReceiver is the most complex class, as it deals with decoding the message type and invoking the correct function in the Router class. In general, we support three message types:

1. Distance Vector Update
2. New Weight Assignments
3. Messages

In general, our router distinguishes these messages by a 1-byte identifier at the beginning of each message, where a byte with value 0 is a distance vector update, 1 is a new weight, and 2 is a message. We chose this approach for simplicity, although it may require some additional engineering in a practical networking application.

Distance Vector Update Layout

The distance vector update layout is quite simple, in that it only contains the serialization of the DistanceVector.

0	Distance Vector Serialization

New Weight Layout

The new weight layout is surprisingly simple, in that we expect an incoming weight assignment to start with a byte with value 1, and look like IP:Port, Weight. For example, to assign a new weight to neighbor A at node B, we send a message to B in the form of

1	A's IP Address	:	A's Port	New Weight

Message Byte Layout

We designed the protocol for Messaging to start with a byte with value 2. We then provide the destination the message is being sent to, where if A were sending to D through B, D would be the destination. To make sure we have the destination correct, we place a delimiter byte with value of 15 after the destination address. Finally, after this delimiter we place the actual message.Here is the format:

2	Dest's IP Address	:	Dest's Port	15	Message

3.3 Operation

We implemented the Distance Vector algorithm using the project requirements as a template. Upon invocation of the method to update the distance vector, we do so, storing a DistanceVectorCalculation object in the Router. The DistanceVector is calculated similarly to the pseudo-code in Section 2.2, with few constraits. Most notably, we set the maximum number of hops to be 15, where a 16 would represent infinity. If a

minimum path is found to have length 16, the path is not recognized in the forwarding table.

When the Router starts up, it will calculate its own distance vector, update its forwarding table, broadcast its neighbors' weights to other nodes, and then send its own distance vector to its neighbors. After this initial state, it will send its distance vectors at n-time-unit intervals. At any point, the Router can receive distance vectors, messages, or neighbor updates from any of its neighbors.

When the Router receives a distance vector, it will calculate a new distance vector, but only send an update to its neighbors if the update caused a change in its current distance vector. If there is no change to the current distance vector, then this is old news, and the Router has no need to send new traffic to its neighbors. We retransmit this vector at n-time-unit intervals in order to tell other Routers that this Router is still alive.

In the event that a Router has not heard from its neighbors for 3 n-time-unit intervals, the Router is assumed to have dropped and a Router that is a neighbor to it will have removed it from its forwarding table and current distance vector. We know that this Router will be recognized if it comes back up, due to its capability to send weight updates to its neighbors on startup. A weight update to a neighbor will also cause a recalculation of distance vectors in its neighbors, and possibly retransmission if a new distance vector is produced.

The last important functionality to talk about is messaging, as this is the ultimate goal of the Distance Vector algorithm. We have to be able to send messages across the

network, even if we aren't directly connected with them. When the MSG command is invoked from the command line, the Router will attach its own SocketAddress to the message and send it on its way. If the ForwardingTable doesn't know how to forward the packet it will be dropped. If all is well in the network, the message will arrive to the receiver with a list of Router SocketAddresses that it was passed through. From that list, we can determine if the path was correct or not.

4. Correctness Results

For this section, we will calculate what we expect to see in two ways. First we will provide the final convergent vector for the node, and second we will provide a brief qualitative analysis of what we expect to happen in the network. This will then be compared to the measured values from our simulation.

Each experiment presented was set-up with three conditions in mind. Firstly, all of the neighbors for each node in the initialization file were correct according to the diagrams of the networks provided. Secondly, the routers would resend their distance vectors at 30s time intervals. Lastly, the routers were started alphabetically 1 second apart. We feel that it is important to do this in order to keep experimental data consistent across multiple trials. For example, A is started at time 0, B at time 1, and so on.

One final note, is that we assigned addresses and ports according to the following table:

Node	Address	Port
A	127.0.0.1	9876
B	127.0.0.1	9877
C	127.0.0.1	9878
D	127.0.0.1	9879
E	127.0.0.1	9880
F	127.0.0.1	9881

4.1 First Network

4.1.1 Node A Convergence

Calculations

We expect the communications along the network to go like this: Since A is the first to send its distance vector [A = 0, B = 1, C = ∞, D = ∞, E = ∞], B (neighbor of A) processes the received vector and update its distance vector with a route to A. B sends its distance vector [A = 1, B = 0, C = 1, D = ∞, E = ∞] to C. C processes the receive vector and update its distance vector and then send it to D. D will also process C's distance vector and update itself, then send its distance vector to E. E will process D's distance vector and update itself. Then, the process will go again in opposite direction: E -> D -> C -> B -> A.

From this process, we expect that A's Distance Vector Table (obtained from the PRINT command) looks like this:

(Node, Dist)	**A**	**B**	**C**	**D**	**E**
(A, 0)	0	1	2	3	4
(B, 1)	1	0	1	2	3

Alongside this, we expect that if we send a message from A to E, that it be forwarded through B, C, D. When E receives the message, we should see that the message has the format of:

message | A's address | B's address | C's address | D's address

That is, when we send Hello from A to E, we should see

Hello | 127.0.0.1 9876 | 127.0.0.1 9877 | 127.0.0.1 9878 | 127.0.0.1 9879

Measured Values

Here is the distance vector table at node A taken from our printouts located in Section 8.1 of this report:

(Node, Dist)	**A**	**B**	**C**	**D**	**E**
(A, 0)	0	1	2	3	4
(B, 1)	1	0	1	2	3

The table perfectly matches our expectations, along with the manner in which it was derived.

Here is the printed message from node E, after it was sent from A:

RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9876 | 127.0.0.1:9877 | 127.0.0.1:9878 | 127.0.0.1:9879

The message perfectly matches our expectations, and we thus conclude that the first experiment is correct.

4.1.2 Changing Weight After Convergence

Calculations

Assuming the network converges as in Experiment 4.1.1, we will change the cost from B to C to 10. This is where we expect some counting to infinity. B will tell C of this new cost, and C will calculate a new distance vector: [A = 3, B = 2, C = 0, D = 1, E = 2]. Remember that C was told from D that D's distance to B was 2 (it previously calculated this with the path D -> C -> B, cost = 2). When C receives an updated weight, it will look to D to get to B. This will cause the path from C to B to increase by 1. C will broadcast its new vector, and D will recalculate the distance from D to B to increase by 1, causing C again to increase by 1. This occurs until infinity or the value from B to C plus 1.In our case, this reaches the value from B to C plus 1 after several iterations.

From this process, we expect that A's Distance Vector Table (obtained from the PRINT command) looks like this:

(Node, Dist)	**A**	**B**	**C**	**D**	**E**
(A, 0)	0	1	11	12	13
(B, 1)	1	0	10	11	12

Alongside this, we expect that if we send a message from A to E, that it be forwarded through B, C, D. When E receives the message, we should see that the message has the format of:

message | A's address | B's address | C's address | D's address

That is, when we send Hello from A to E, we should see

Hello | 127.0.0.1 9876 | 127.0.0.1 9877 | 127.0.0.1 9878 | 127.0.0.1 9879

Measured Values

Here is the distance vector table at node A taken from our printouts located in Section 8.2 of this report:

(Node, Dist)	**A**	**B**	**C**	**D**	**E**
(A, 0)	0	1	11	12	13
(B, 1)	1	0	10	11	12

The table perfectly matches our expectations, along with the manner in which it was derived.

Here is the printed message from node E, after it was sent from A:

RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9876 | 127.0.0.1:9877 | 127.0.0.1:9878 | 127.0.0.1:9879

The message perfectly matches our expectations.

Finally, in the printouts it is quite visible (highlighted in light red) that the Routers are experiencing a slow convergence by counting, which is as expected.

Our experiment perfectly matches our expectations, thus we conclude that the second experiment is correct.

4.1.3 Changing Weight After Convergence with Poison Reverse

Calculations

The passing strategy will be same as 4.1.2, however, since this is poison reversed, in A's table, the path from B to A passes through A, the cost from B to A will be infinity, which is 16 in this case. Remember that we chose to use 15 hops for the maximum cost. Another factor to pay attention to is that we do not expect the count to infinity problem. Now that the router is using poison reverse, we should see the inferred knowledge from D to C that D can not reach B through C.

(Node, Dist)	**A**	**B**	**C**	**D**	**E**
(A, 0)	0	1	11	12	13
(B, 1)	16	0	10	11	12

Alongside this, we expect that if we send a message from A to E, that it be forwarded through B, C, D. When E receives the message, we should see that the message has the format of:

message | A's address | B's address | C's address | D's address

That is, when we send Hello from A to E, we should see

Hello | 127.0.0.1 9876 | 127.0.0.1 9877 | 127.0.0.1 9878 | 127.0.0.1 9879

Measured Values

Here is the distance vector table at node A taken from our printouts located in Section 8.3 of this report:

(Node, Dist)	**A**	**B**	**C**	**D**	**E**
(A, 0)	0	1	11	12	13
(B, 1)	16	0	10	11	12

The table perfectly matches our expectations, along with the manner in which it was derived.

Here is the printed message from node E, after it was sent from A:

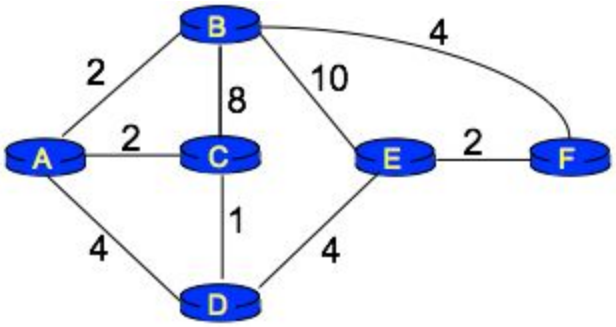
RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9876 | 127.0.0.1:9877 | 127.0.0.1:9878 | 127.0.0.1:9879

Finally, in the printouts we see that the poison reverse has countered the count to infinity problem as expected.

Our experiment perfectly matches our expectations, thus we conclude that the third experiment is correct.

4.2 Second Network

4.2.1 Node E Convergence



Calculations

We expect the communications along the network to go like this: A is the first to send its distance vector [A = 0, B = 2, C = 2, D = 4, E = ∞, F = ∞] to its neighbors, B, C and D. When B receives the distance vector of A, it will update its own distance vector [A = 2, B = 0, C = 4, D = 6, E = 10, F = 4], and then send the distance vector to its neighbors, A, C, E, F. At the same time, C and D will also update their distance vectors and send to neighbors. The algorithm will run recursively until no changes in distance vectors. The final table of E will be shown as below.

(Node, Dist)	**A**	**B**	**C**	**D**	**E**	**F**
(B, 10)	2	0	4	5	6	4
(D, 4)	3	5	1	0	4	6
(E, 0)	7	6	5	4	0	2
(F, 2)	6	4	7	6	2	0

Alongside this, we expect that if we send a message from E to A, that it be forwarded through D and C. When A receives the message, we should see that the message has the format of:

message | E's address | D's address | C's address

That is, when we send Hello from E to A, we should see

Hello | 127.0.0.1 9880 | 127.0.0.1 9879 | 127.0.0.1 9878

Measured Values

Here is the distance vector table at node A taken from our printouts located in Section 8.4 of this report:

(Node, Dist)	**A**	**B**	**C**	**D**	**E**	**F**
(B, 10)	2	0	4	5	6	4
(D, 4)	3	5	1	0	4	6
(E, 0)	7	6	5	4	0	2
(F, 2)	6	4	7	6	2	0

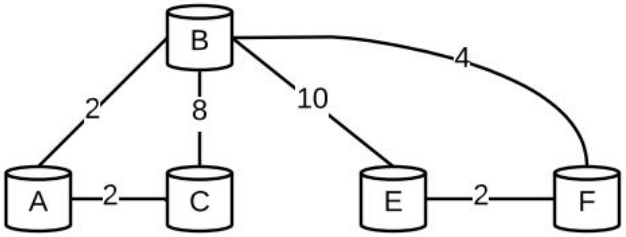
The table perfectly matches our expectations, along with the manner in which it was derived.

Here is the printed message from node E, after it was sent from A:

RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9880 | 127.0.0.1:9879 | 127.0.0.1:9878

Our experiment perfectly matches our expectations, thus we conclude that the fourth experiment is correct.

4.2.2 Removing a Node After Convergence



Calculations

Assuming that the network in 4.2.1 reaches convergence, we will remove D from the network resulting in the network above. We expect to see that after 3 complete time intervals of 30 seconds, that D is dropped from the network. Once this happens, new routes should be calculated, one for example being E to A, switching from E -> D -> C -> A, to E -> F -> B -> A, with respective costs 7 to 8. This will be reflected in the final table of E shown below.

(Node, Dist)	**A**	**B**	**C**	**E**	**F**
(B, 10)	2	0	4	6	4
(E, 0)	8	6	10	0	2
(F, 2)	6	4	8	2	0

Alongside this, we expect that if we send a message from E to A, that it be forwarded through F and B as opposed to D and C. When A receives the message, we should see that the message has the format of:

message | E’s address | F’s address | B’s address

That is, when we send Hello from E to A, we should see

Hello | 127.0.0.1 9880 | 127.0.0.1 9881 | 127.0.0.1 9877

Measured Values

Here is the distance vector table at node A taken from our printouts located in Section 8.5 of this report:

(Node, Dist)	**A**	**B**	**C**	**E**	**F**
(B, 10)	2	0	4	6	4
(E, 0)	8	6	10	0	2
(F, 2)	6	4	8	2	0

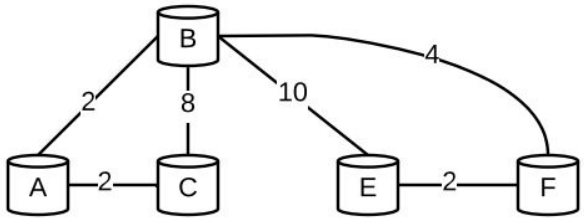
The table perfectly matches our expectations, along with the manner in which it was derived. After 3 complete intervals, D was dropped from the network.

Here is the printed message from node A, after it was sent from E:

RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9880 | 127.0.0.1:9881 | 127.0.0.1:9877

Our experiment perfectly matches our expectations, thus we conclude that the fifth experiment is correct.

4.2.3 Removing a Node After Convergence with Poison Reverse



Calculations

This experiment is the same as experiment 4.2.3 except this one will use poison reverse. We expect to see that after 3 complete time intervals of 30 seconds, that D is dropped from the network. Even though we use poison reverse, we don't expect the number of intervals to be significantly less as the most significant differences are only by cost 1. This experiment will be reflected in the final table of E shown below.

(Node, Dist)	**A**	**B**	**C**	**E**	**F**
(B, 10)	2	0	4	16	4
(E, 0)	8	6	10	0	2
(F, 2)	6	4	8	16	0

Alongside this, we expect that if we send a message from E to A, that it be forwarded through F and B as opposed to D and C. When A receives the message, we should see that the message has the format of:

message | E's address | F's address | B's address

That is, when we send Hello from E to A, we should see

Hello | 127.0.0.1 9880 | 127.0.0.1 9881 | 127.0.0.1 9877

Measured Values

Here is the distance vector table at node A taken from our printouts located in Section 8.6 of this report:

(Node, Dist)	**A**	**B**	**C**	**E**	**F**
(B, 10)	2	0	4	16	4
(E, 0)	8	6	10	0	2
(F, 2)	6	4	8	16	0

The table perfectly matches our expectations, along with the manner in which it was derived. After 3 complete intervals, D was dropped from the network, and we didn't observe noticeable change in the number of messages sent.

Here is the printed message from node A, after it was sent from E:

RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9880 | 127.0.0.1:9881 | 127.0.0.1:9877

Our experiment perfectly matches our expectations, thus we conclude that the sixth and final experiment is correct.

5. Conclusions

After coding the Distance Vector algorithm, we feel that we have a moderately well understanding of why this vector is used. The power of being able to compute the shortest path from any two routers without knowing the entire network globally, is what makes Distance Vector a viable routing algorithm.

We also wanted to discuss the cost of increasing the maximum hop count for the algorithm in a qualitative way. Given a large enough network, the maximum hop count may need to be increased. Without poison reversal, this can cause a large increase of traffic per count increment. Thanks to poison reversal though, the traffic increase can largely be avoided.

One last observation we wanted to make is that it seems like poison reversal was a bug fix for the original Distance Vector algorithm. We feel that the original design of the algorithm should have included the technique in the first place, and that the subsequent implementation was merely a patch. In any circumstances, we find that it is highly important to use poison reverse in order to cut down unnecessary network traffic.

As a result of completing the project, we feel more comfortable with the topics of UDP programming, multithreading, and the Distance Vector algorithm.

6. References

1. Computer Networking : Principles, Protocols and Practice - Distance Vector Routing; http://cnp3book.info.ucl.ac.be/principles/dv.html
2. Kurose, James and Ross, Keith; Computer Networking: A Top-Down Approach, 6th Edition
3. Sadovnik, Amir; “Project 3: Simulating the distance vector algorithm”
4. Class ReenTrantLock, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html
5. Class Scanner, Java Platform SE8; https://docs.google.com/document/d/1sOsSnPDXb2qP1mSZAg5VBpiYRoYWsLt6bRYryqpICDk/edit?ts=590e26e5#
6. Class DatagramPacket, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html
7. Class DatagramSocket, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html
8. Class InetAddress, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html
9. Class BufferedReader, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html
10. Package Java.net, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html
11. Package Java.util.concurrent, Java Platform SE7; https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html

7. Division of Labor

	Yizhong	Darren
Project Report	☑	☑
Java Documentation	☑	☑
Test and Analysis	☑	☑
DistanceVector	☑	☑
DistanceVectorCalculation	☑	☑
DistanceVectorFactory	☑	☑
DVCommandThread		☑
DVUpdateThread		☑
ForwardingTable		☑
Router	☑	☑
RouterFactory	☑	☑
RouterUDPReceiver		☑
RouterUDPSender		☑
SocketAddress	☑	☑

8. Appendix A Code Printouts

Using the project description as a guide, we have attached print-outs with some annotation to describe points a,b,c,d, and e. We used the following mapping between names and address:

Name	Address
A	127.0.0.1 9876
B	127.0.0.1 9877
C	127.0.0.1 9878
D	127.0.0.1 9879
E	127.0.0.1 9880
F	127.0.0.1 9881

8.1 a) Network at node A (each link has weight 1):

//Started
New weight to Neighbor 127.0.0.1:9877 of 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9879 2

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9879 3 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 3

127.0.0.1:9879 2

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 4 127.0.0.1:9877
127.0.0.1:9879 3 127.0.0.1:9877

Update sent to all neighbors at time 13:40:27
127.0.0.1:9877 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 3
127.0.0.1:9879 2

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9876
127.0.0.1:9878 | 2
127.0.0.1:9877 | 1
127.0.0.1:9876 | 0
127.0.0.1:9880 | 4
127.0.0.1:9879 | 3
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 1
127.0.0.1:9877 | 0
127.0.0.1:9876 | 1
127.0.0.1:9880 | 3

127.0.0.1:9879 | 2

//SEND MESSAGE TO E
MSG 127.0.0.1 9880 hello

//PRINTED FROM E
RECEIVED MESSAGE FINALLY: hello | 127.0.0.1:9876 | 127.0.0.1:9877 | 127.0.0.1:9878 | 127.0.0.1:9879

//Message was sent from A -> B -> C -> D -> E

8.2 b) After convergence, change the weight between B and C to 10… (no poison)

//Started
New weight to Neighbor 127.0.0.1:9877 of 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9879 2

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9879 3 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:

127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 3
127.0.0.1:9879 2

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 4 127.0.0.1:9877
127.0.0.1:9879 3 127.0.0.1:9877

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9876
127.0.0.1:9878 | 2
127.0.0.1:9877 | 1
127.0.0.1:9876 | 0
127.0.0.1:9880 | 4
127.0.0.1:9879 | 3
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 1
127.0.0.1:9877 | 0
127.0.0.1:9876 | 1
127.0.0.1:9880 | 3
127.0.0.1:9879 | 2

//CHANGED B-C TO 10, OBSERVE COUNTING PROBLEM

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 3

127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 5
127.0.0.1:9879 4

New dv calculated:
127.0.0.1:9878 4 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 6 127.0.0.1:9877
127.0.0.1:9879 5 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 5
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 7
127.0.0.1:9879 6

New dv calculated:
127.0.0.1:9878 6 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 8 127.0.0.1:9877
127.0.0.1:9879 7 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 9
127.0.0.1:9879 8

New dv calculated:

127.0.0.1:9878 8 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 10 127.0.0.1:9877
127.0.0.1:9879 9 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 9
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 11
127.0.0.1:9879 10
New dv calculated:

127.0.0.1:9878 10 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 12 127.0.0.1:9877
127.0.0.1:9879 11 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 10
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 12
127.0.0.1:9879 11

New dv calculated:
127.0.0.1:9878 11 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 13 127.0.0.1:9877
127.0.0.1:9879 12 127.0.0.1:9877

Update sent to all neighbors at time 13:43:26
127.0.0.1:9877 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 10
127.0.0.1:9877 0
127.0.0.1:9876 1
127.0.0.1:9880 12
127.0.0.1:9879 11

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9876
127.0.0.1:9878 | 11
127.0.0.1:9877 | 1
127.0.0.1:9876 | 0
127.0.0.1:9880 | 13
127.0.0.1:9879 | 12
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 10
127.0.0.1:9877 | 0
127.0.0.1:9876 | 1
127.0.0.1:9880 | 12
127.0.0.1:9879 | 11

//SEND MESSAGE TO E
MSG 127.0.0.1 9880 Hello

//PRINTED FROM E
RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9876 | 127.0.0.1:9877 | 127.0.0.1:9878 | 127.0.0.1:9879

//Message was sent from A -> B -> C -> D -> E

8.3 c) The same situation as b with poisoned reverse.

//Started
New weight to Neighbor 127.0.0.1:9877 of 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 16

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9879 2

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9879 3 127.0.0.1:9877

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9880 3
127.0.0.1:9879 2

New dv calculated:
127.0.0.1:9878 2 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 4 127.0.0.1:9877
127.0.0.1:9879 3 127.0.0.1:9877

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9876
127.0.0.1:9878 | 2
127.0.0.1:9877 | 1
127.0.0.1:9876 | 0
127.0.0.1:9880 | 4
127.0.0.1:9879 | 3
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 1
127.0.0.1:9877 | 0
127.0.0.1:9876 | 16
127.0.0.1:9880 | 3
127.0.0.1:9879 | 2

//CHANGED B-C TO 10, OBSERVE QUICK CONVERGENCE
New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 1

127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9880 3
127.0.0.1:9879 2

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 10
127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9880 12
127.0.0.1:9879 11

New dv calculated:
127.0.0.1:9878 11 127.0.0.1:9877
127.0.0.1:9877 1 127.0.0.1:9877
127.0.0.1:9876 0 ~
127.0.0.1:9880 13 127.0.0.1:9877
127.0.0.1:9879 12 127.0.0.1:9877

Update sent to all neighbors at time 13:47:20
127.0.0.1:9877 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 10
127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9880 12
127.0.0.1:9879 11

Update sent to all neighbors at time 13:47:50
127.0.0.1:9877 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 10

127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9880 12
127.0.0.1:9879 11

Update sent to all neighbors at time 13:48:20
127.0.0.1:9877 1

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 10
127.0.0.1:9877 0
127.0.0.1:9876 16
127.0.0.1:9880 12
127.0.0.1:9879 11

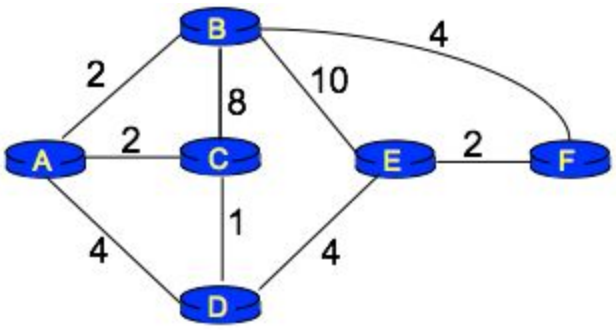
//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9876
127.0.0.1:9878 | 11
127.0.0.1:9877 | 1
127.0.0.1:9876 | 0
127.0.0.1:9880 | 13
127.0.0.1:9879 | 12
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 10
127.0.0.1:9877 | 0
127.0.0.1:9876 | 16
127.0.0.1:9880 | 12
127.0.0.1:9879 | 11

//SEND MESSAGE TO E
MSG 127.0.0.1 9880 Hello

//PRINTED FROM E
RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9876 | 127.0.0.1:9877 | 127.0.0.1:9878 | 127.0.0.1:9879

//Message was sent from A -> B -> C -> D -> E

8.4 d) More complicated network at E:



//Started
Update sent to all neighbors at time 14:00:34
127.0.0.1:9877 10
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9879 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 5
127.0.0.1:9876 3
127.0.0.1:9881 6
127.0.0.1:9880 4
127.0.0.1:9879 0

New dv calculated:
127.0.0.1:9878 5 127.0.0.1:9879
127.0.0.1:9877 9 127.0.0.1:9879
127.0.0.1:9876 7 127.0.0.1:9879
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~
127.0.0.1:9879 4 127.0.0.1:9879

New weight to Neighbor 127.0.0.1:9881 of 2

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9877 4
127.0.0.1:9881 0
127.0.0.1:9880 2

New dv calculated:
127.0.0.1:9878 5 127.0.0.1:9879
127.0.0.1:9877 6 127.0.0.1:9881
127.0.0.1:9876 7 127.0.0.1:9879
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~
127.0.0.1:9879 4 127.0.0.1:9879

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 9
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 6

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0

127.0.0.1:9880 2
127.0.0.1:9879 6

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9880
127.0.0.1:9878 | 5
127.0.0.1:9877 | 6
127.0.0.1:9876 | 7
127.0.0.1:9881 | 2
127.0.0.1:9880 | 0
127.0.0.1:9879 | 4
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 4
127.0.0.1:9877 | 0
127.0.0.1:9876 | 2
127.0.0.1:9881 | 4
127.0.0.1:9880 | 6
127.0.0.1:9879 | 5
Distance vector from : 127.0.0.1:9881
SOURCE: 127.0.0.1:9881
127.0.0.1:9878 | 7
127.0.0.1:9877 | 4
127.0.0.1:9876 | 6
127.0.0.1:9881 | 0
127.0.0.1:9880 | 2
127.0.0.1:9879 | 6
Distance vector from : 127.0.0.1:9879
SOURCE: 127.0.0.1:9879
127.0.0.1:9878 | 1
127.0.0.1:9877 | 5
127.0.0.1:9876 | 3

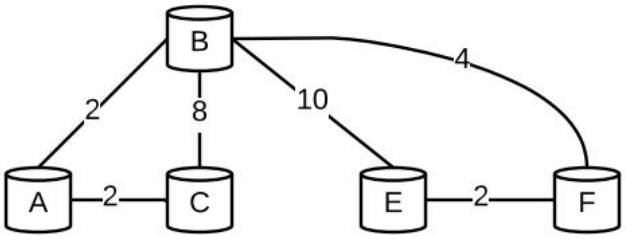
127.0.0.1:9881 | 6
127.0.0.1:9880 | 4
127.0.0.1:9879 | 0

//SEND MESSAGE TO A
MSG 127.0.0.1 9876 Hello

//PRINTED FROM A
RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9880 | 127.0.0.1:9879 | 127.0.0.1:9878

//Message was sent from E -> D -> C -> A

8.5 e) The same as in d, but remove node D after convergence.



//Started
Update sent to all neighbors at time 01:49:01
127.0.0.1:9877 10
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9879 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 5
127.0.0.1:9876 3
127.0.0.1:9881 6
127.0.0.1:9880 4
127.0.0.1:9879 0

New dv calculated:
127.0.0.1:9878 5 127.0.0.1:9879
127.0.0.1:9877 9 127.0.0.1:9879
127.0.0.1:9876 7 127.0.0.1:9879
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~
127.0.0.1:9879 4 127.0.0.1:9879

New weight to Neighbor 127.0.0.1:9881 of 2

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9877 4

127.0.0.1:9881 0
127.0.0.1:9880 2

New dv calculated:
127.0.0.1:9878 5 127.0.0.1:9879
127.0.0.1:9877 6 127.0.0.1:9881
127.0.0.1:9876 7 127.0.0.1:9879
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~
127.0.0.1:9879 4 127.0.0.1:9879

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 9
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 6

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 6

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9880
127.0.0.1:9878 | 5
127.0.0.1:9877 | 6
127.0.0.1:9876 | 7
127.0.0.1:9881 | 2
127.0.0.1:9880 | 0
127.0.0.1:9879 | 4
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 4
127.0.0.1:9877 | 0
127.0.0.1:9876 | 2
127.0.0.1:9881 | 4
127.0.0.1:9880 | 6
127.0.0.1:9879 | 5
Distance vector from : 127.0.0.1:9881
SOURCE: 127.0.0.1:9881
127.0.0.1:9878 | 7
127.0.0.1:9877 | 4
127.0.0.1:9876 | 6
127.0.0.1:9881 | 0
127.0.0.1:9880 | 2
127.0.0.1:9879 | 6
Distance vector from : 127.0.0.1:9879
SOURCE: 127.0.0.1:9879
127.0.0.1:9878 | 1
127.0.0.1:9877 | 5
127.0.0.1:9876 | 3
127.0.0.1:9881 | 6
127.0.0.1:9880 | 4

127.0.0.1:9879 | 0

//D WAS SHUT DOWN, WAIT 3 INTERVALS
New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

//ONE
Update sent to all neighbors at time 01:49:31
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 6

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

//TWO

Update sent to all neighbors at time 01:50:01
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 6

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

//THREE
Update sent to all neighbors at time 01:50:31
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 7
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 6

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 9

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 5

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 9

New dv received from 127.0.0.1:9877 with the following distances:

127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 10

//FOUR
Update sent to all neighbors at time 01:51:01
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

//DROP D SINCE 3 COMPLETE INTERVALS HAVE GONE
Neighbor 127.0.0.1:9879 dropped

New dv calculated:
127.0.0.1:9878 9 127.0.0.1:9881
127.0.0.1:9877 6 127.0.0.1:9881
127.0.0.1:9876 8 127.0.0.1:9881
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 8
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2
127.0.0.1:9879 14

New dv calculated:
127.0.0.1:9878 10 127.0.0.1:9881
127.0.0.1:9877 6 127.0.0.1:9881

127.0.0.1:9876 8 127.0.0.1:9881
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6
127.0.0.1:9879 14

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 6

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 8
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 8
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 2

//PRINT VECTOR TO SHOW CONVERGENCE

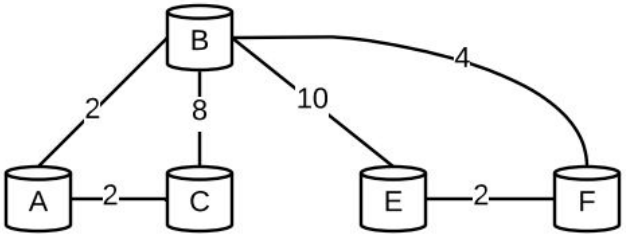
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9880
127.0.0.1:9878 | 10
127.0.0.1:9877 | 6
127.0.0.1:9876 | 8
127.0.0.1:9881 | 2
127.0.0.1:9880 | 0
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 4
127.0.0.1:9877 | 0
127.0.0.1:9876 | 2
127.0.0.1:9881 | 4
127.0.0.1:9880 | 6
Distance vector from : 127.0.0.1:9881
SOURCE: 127.0.0.1:9881
127.0.0.1:9878 | 8
127.0.0.1:9877 | 4
127.0.0.1:9876 | 6
127.0.0.1:9881 | 0
127.0.0.1:9880 | 2

//SEND MESSAGE TO A
MSG 127.0.0.1 9876 Hello

//PRINTED FROM A
RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9880 | 127.0.0.1:9881 | 127.0.0.1:9877

//Message was sent from E -> F -> B -> A

8.6 e+) The same as in d, but remove node D after convergence, with poison reverse.



//Started
Update sent to all neighbors at time 17:18:14
127.0.0.1:9877 10
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9879 with the following distances:
127.0.0.1:9878 1
127.0.0.1:9877 5
127.0.0.1:9876 3
127.0.0.1:9881 16
127.0.0.1:9880 16
127.0.0.1:9879 0

New dv calculated:
127.0.0.1:9878 5 127.0.0.1:9879
127.0.0.1:9877 9 127.0.0.1:9879
127.0.0.1:9876 7 127.0.0.1:9879
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~
127.0.0.1:9879 4 127.0.0.1:9879

New weight to Neighbor 127.0.0.1:9881 of 2

New dv received from 127.0.0.1:9881 with the following distances:

127.0.0.1:9877 4
127.0.0.1:9881 0
127.0.0.1:9880 16

New dv calculated:
127.0.0.1:9878 5 127.0.0.1:9879
127.0.0.1:9877 6 127.0.0.1:9881
127.0.0.1:9876 7 127.0.0.1:9879
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~
127.0.0.1:9879 4 127.0.0.1:9879

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 5

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 8
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16
127.0.0.1:9879 9

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 16
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16

127.0.0.1:9879 16

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9880
127.0.0.1:9878 | 5
127.0.0.1:9877 | 6
127.0.0.1:9876 | 7
127.0.0.1:9881 | 2
127.0.0.1:9880 | 0
127.0.0.1:9879 | 4
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 4
127.0.0.1:9877 | 0
127.0.0.1:9876 | 2
127.0.0.1:9881 | 4
127.0.0.1:9880 | 16
127.0.0.1:9879 | 5
Distance vector from : 127.0.0.1:9881
SOURCE: 127.0.0.1:9881
127.0.0.1:9878 | 16
127.0.0.1:9877 | 4
127.0.0.1:9876 | 6
127.0.0.1:9881 | 0
127.0.0.1:9880 | 16
127.0.0.1:9879 | 16
Distance vector from : 127.0.0.1:9879
SOURCE: 127.0.0.1:9879
127.0.0.1:9878 | 1
127.0.0.1:9877 | 5
127.0.0.1:9876 | 3
127.0.0.1:9881 | 16

127.0.0.1:9880 | 16
127.0.0.1:9879 | 0

//D WAS SHUT DOWN, WAIT 3 INTERVALS
New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 5

//ONE
Update sent to all neighbors at time 17:18:44
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 16
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16
127.0.0.1:9879 16

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 5

//TWO
Update sent to all neighbors at time 17:19:14
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 16
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16
127.0.0.1:9879 16

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 5

//THREE
Update sent to all neighbors at time 17:19:44
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 16
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16

127.0.0.1:9879 16

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 9

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 5

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 5

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2
127.0.0.1:9881 4
127.0.0.1:9880 16
127.0.0.1:9879 10

//FOUR
Update sent to all neighbors at time 17:20:14
127.0.0.1:9877 6
127.0.0.1:9881 2
127.0.0.1:9879 4

//DROP D SINCE 3 COMPLETE INTERVALS HAVE GONE BY
Neighbor 127.0.0.1:9879 dropped

New dv calculated:
127.0.0.1:9878 14 127.0.0.1:9877
127.0.0.1:9877 6 127.0.0.1:9881
127.0.0.1:9876 8 127.0.0.1:9881
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 8
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16

New dv calculated:
127.0.0.1:9878 10 127.0.0.1:9881
127.0.0.1:9877 6 127.0.0.1:9881
127.0.0.1:9876 8 127.0.0.1:9881
127.0.0.1:9881 2 127.0.0.1:9881
127.0.0.1:9880 0 ~

New dv received from 127.0.0.1:9877 with the following distances:
127.0.0.1:9878 4
127.0.0.1:9877 0
127.0.0.1:9876 2

127.0.0.1:9881 4
127.0.0.1:9880 16

New dv received from 127.0.0.1:9881 with the following distances:
127.0.0.1:9878 8
127.0.0.1:9877 4
127.0.0.1:9876 6
127.0.0.1:9881 0
127.0.0.1:9880 16

//PRINT VECTOR TO SHOW CONVERGENCE
PRINT
Current distance vector:
SOURCE: 127.0.0.1:9880
127.0.0.1:9878 | 10
127.0.0.1:9877 | 6
127.0.0.1:9876 | 8
127.0.0.1:9881 | 2
127.0.0.1:9880 | 0
Distance vector from : 127.0.0.1:9877
SOURCE: 127.0.0.1:9877
127.0.0.1:9878 | 4
127.0.0.1:9877 | 0
127.0.0.1:9876 | 2
127.0.0.1:9881 | 4
127.0.0.1:9880 | 16
Distance vector from : 127.0.0.1:9881
SOURCE: 127.0.0.1:9881
127.0.0.1:9878 | 8
127.0.0.1:9877 | 4
127.0.0.1:9876 | 6
127.0.0.1:9881 | 0
127.0.0.1:9880 | 16

//SEND MESSAGE TO A
MSG 127.0.0.1 9876 Hello

//PRINTED FROM A
RECEIVED MESSAGE FINALLY: Hello | 127.0.0.1:9880 | 127.0.0.1:9881 | 127.0.0.1:9877

//Message was sent from E -> F -> B -> A