Wassim Gharbi
Erik Laucks
Zurabi Mestiashvili

May 4th, 2018
CS 305: Computer Networks
Submitted to Prof. Amir Sadovnik

Project #3: Distance Vector Simulation

I. Introduction

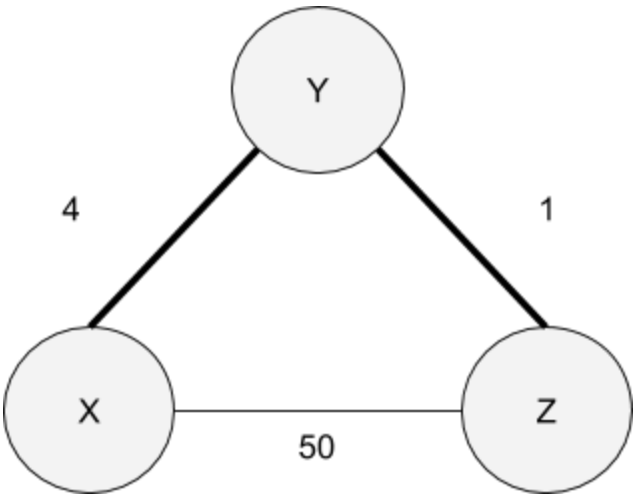
The goals of this project revolved around gaining a deeper understanding of the distance vector routing algorithm in a distributed setting, getting familiar with socket programming, and acquiring experience with multi-threading. The simulation is a multi-threaded program that examines how a network of routers interacts with each other in order to simplify, and better understand the RIP protocol. Our implementation relies on an abstracted network layer that uses Java UDP sockets to transfer information over a localhost network. In order to study the behavior of the RIP protocol across a network, we have several different options as to the way to run the software: One version that uses the base distance vector algorithm and one version that utilizes poisoned reverse, as discussed in the book and in class. As an extension to the project, we also added a broadcast function that uses reverse path forwarding to send a message to all of the nodes in the network. In this report, we demonstrate the accuracy and correctness of our implementation as well as how we achieved our solution.

II. Simulated Protocol

We have implemented our protocol to be analogous to the real-world RIP protocol with a few modifications. Our protocol has routers update their distance vectors whenever they receive vectors from other routers and rebroadcast their vectors whenever they're updated. Our routers will also drop neighbors if they don't receive information from them after a certain amount of time. Lastly, our routers can use poisoned reverse and will forward messages using their determined forwarding tables. However, our protocol allows links to have weights, as opposed to RIP, which assumes all links have weight 1.

Expected Exchanges

Consider a network with the following topology and the following scenarios:



Received distance vector w/ update

If node Z receives the distance vector {4, 0, 1} from Y:

Before

Node z	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	50	1	0

After

Node z	x	y	z
x	∞	∞	∞
y	4	0	1
z	5	1	0

Received distance vector w/o update

If node Y receives the distance vector {0, 4, 5} from X:

Before

Node y	x	y	z
x	∞	∞	∞
y	4	0	1
z	∞	∞	∞

After

Node y	x	y	z
x	0	4	5
y	4	0	1
z	∞	∞	∞

Received distance vector w/ poisoned reverse

If node Y receives the distance vector {0, ∞, ∞} from X:

Before

Node y	x	y	z
x	∞	∞	∞
y	4	0	1
z	∞	∞	∞

After

Node y	x	y	z
x	0	∞	∞
y	4	0	1
z	∞	∞	∞

Drops neighbor after timeout

If node Z doesn't receive communication from Y after 3n seconds:

Before

Node z	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

After

Node z	x	y	z
x	0	4	5
y	∞	∞	∞
z	50	54	0

Received message destined for other router

If X wants to send a message to Z, it will use its forwarding table to send the packet to Y. When Y receives the packet, it will use its forwarding table to send the packet to Z.

Received broadcast message

The bolded paths in the network diagram represent the spanning tree that will used for the reverse path forwarding in the broadcast function. If a router receives a broadcast from a node on the spanning tree, it forwards the packet, otherwise it discards the packet.

III. Code Design & Data Structures

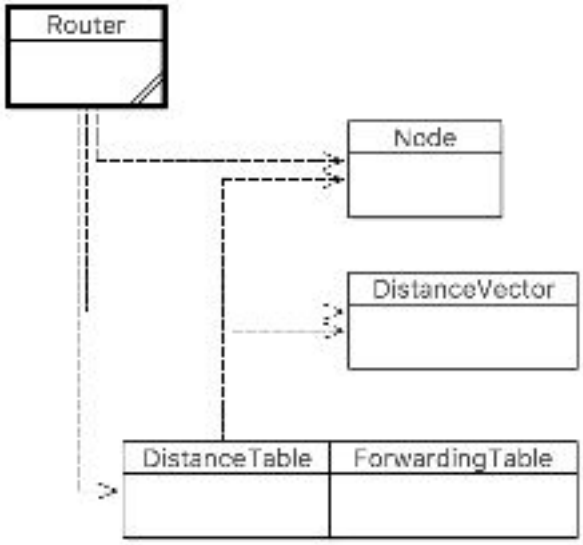


Figure 3. Project Structure

In order to implement all the features discussed in the previous section, we designed multiple abstractions of the various components of the network layer and routers as follows:

- The **`Router`** class takes the parameters for the simulation, sets up the network layer and threads, and runs the simulation based on the given parameters. This class manages the input control, broadcasting packets, controlling the distance and forwarding tables, and sending out timed updates of its own distance vector.
- The **`DistanceTable`** class models the distance table of a given router. This class makes sure that the columns of the table are kept consistent amongst the distance vectors and it responsible for calculating the forwarding tables.

- The **`DistanceVector`** class represents a single distance vector. This class provides encoding methods to display the information in the distance vector in various ways.

IV. Simulator Usage & Parameters

Building the simulation

Build the project using the included Makefile (do not use BlueJ) as follows :

```
$ make cc
```

Running the simulation

The simulation runs in different debugging modes, and takes multiple parameters that control the simulator's behaviors. **Do not run the simulation inside BlueJ** since we have added ANSI color codes which are not compatible with BlueJ's terminal and will therefore render the output illegible. In order to run the simulation, use the command

```
$ make run
```

to run a predefined test or use

```
$ java Router [-reverse] configFile
```

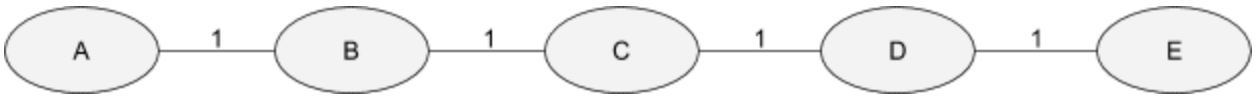
to customize the parameters of the simulation. The parameters are as follows:

Parameter	Usage
`String reverse`	Flag to use poisoned reverse in the router
`String configFile`	Input file containing the network topology for the neighbors of the current router

V. Correctness Results

Result A

The following simple network at node A (each link has weight 1)



Theoretical

Node A	A	B	C	D	E
A	0	1	2	3	4
B	1	0	1	2	3

Experimental

Node A	A	B	C	D	E
A	0	1	2	3	4
B	1	0	1	2	3

Result B

The following simple network at node A (each link has weight 1)

after convergence then change the weight between B and C to 10… (no poisoned reverse)

Theoretical

Node A	A	B	C	D	E
A	0	1	11	12	13
B	1	0	10	11	12

Experimental

Node A	A	B	C	D	E
A	0	1	11	12	13
B	1	0	10	11	12

Result C

The following simple network at node A (each link has weight 1)

after convergence then change the weight between B and C to 10… (poisoned reverse)

Theoretical

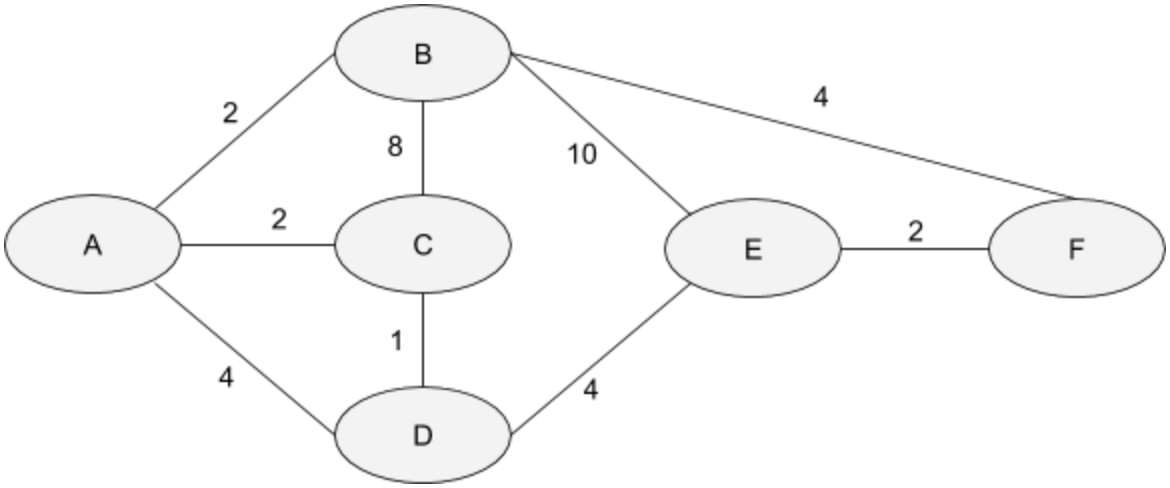
Node A	A	B	C	D	E
A	0	1	11	12	13
B	1	0	10	11	12

Experimental

Node A	A	B	C	D	E
A	0	1	11	12	13
B	1	0	10	11	12

Result D

The following more complicated network at E



Theoretical

Node E	A	B	C	D	E	F
B	2	0	4	5	6	4
D	3	5	1	0	4	6
E	7	6	5	4	0	2
F	6	4	7	6	2	0

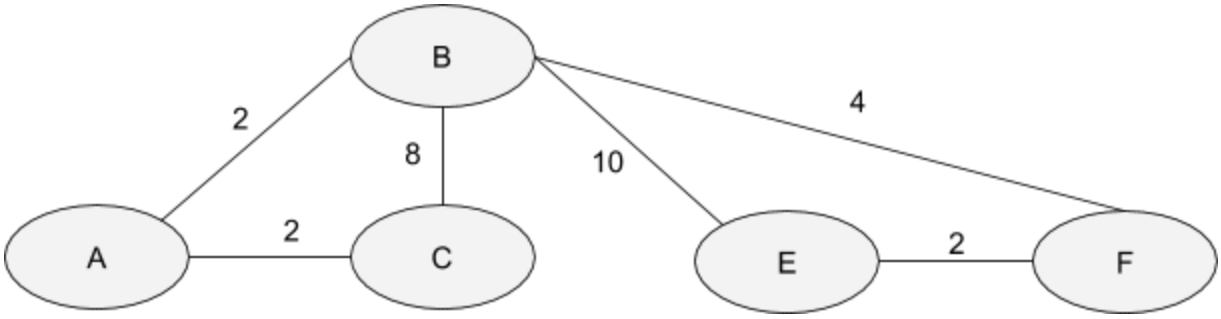
Experimental

Node E	A	B	C	D	E	F
B	2	0	4	5	6	4
D	3	5	1	0	4	6
E	7	6	5	4	0	2
F	6	4	7	6	2	0

```
⟳ Updated distance table
        /127.0.0.1:3002 (Cost 10)       |       2       0       4       5       6       4
        /127.0.0.1:3004 (Cost 4)        |       3       5       1       0       4       6
        /127.0.0.1:3005 (Cost 0)        |       7       6       5       4       0       2
        /127.0.0.1:3006 (Cost 2)        |       6       4       7       6       2       0
```

Result E

The following more complicated network at E, but remove node D after convergence



Theoretical

Node E	A	B	C	E	F
B	2	0	4	6	4
E	8	6	10	0	2
F	6	4	8	2	0

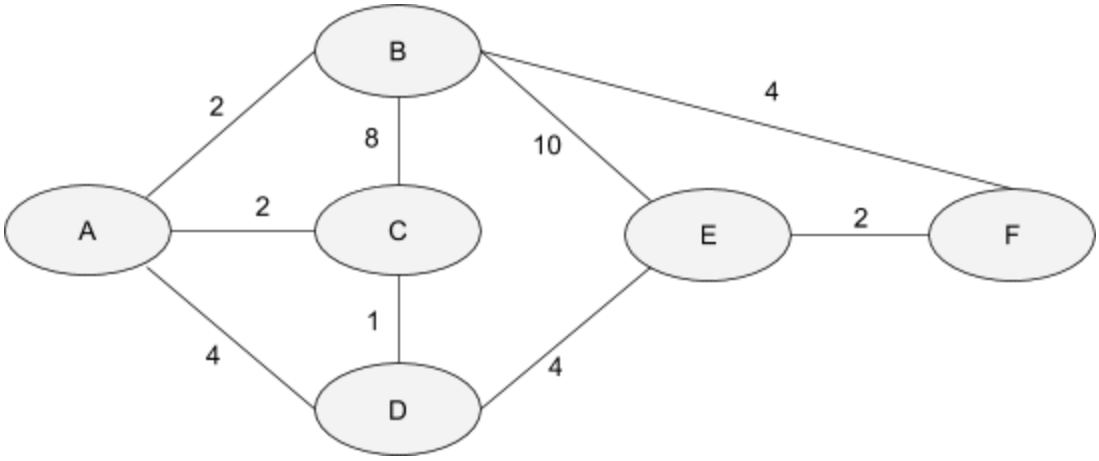
Experimental

Node E	A	B	C	E	F
B	2	0	4	6	4
E	8	6	10	0	2
F	6	4	8	2	0

```
Updated distance table
        /127.0.0.1:3002 (Cost 10)       |       2       0       4       6       4
        /127.0.0.1:3005 (Cost 0)        |       8       6       10      0       2
        /127.0.0.1:3006 (Cost 2)        |       6       4       8       2       0
```

Result F

The following more complicated network, **broadcasting** from A (reverse path forwarding).



Theoretical

Node	Receives Messages From	Forwards Message From	Forwards Message To
A	D	A	B, C, D
B	A, C, E	A	C, E, F
C	A, B, D	A	B, D
D	C, E	C	A, E
E	B, D, F	D	B, F
F	B, E	B	E

Experimental

Node	Receives Messages From	Forwards Message From	Forwards Message To	Experimental Result
A 3001	D	A	B, C, D	➡ Broadcast message "bc: aloha /127.0.0.1:3001" forwarded to 3002 ➡ Broadcast message "bc: aloha /127.0.0.1:3001" forwarded to 3003 ➡ Broadcast message "bc: aloha /127.0.0.1:3001" forwarded to 3004 ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004 ✖ Broadcast message rejected (already received)

B 3002	A, C, E	A	C, E, F	⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3002" forwarded to 3003 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3002" forwarded to 3005 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3002" forwarded to 3006 ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3003 ✖ Broadcast message rejected (reverse path forwarding) ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004 /127.0.0.1:3005 ✖ Broadcast message rejected (reverse path forwarding)
C 3003	A, B, D	A	B, D	⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3003" forwarded to 3002 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3003" forwarded to 3004 ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3002 ✖ Broadcast message rejected (reverse path forwarding)
D 3004	C, E	C	A, E	⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 ✖ Broadcast message rejected (reverse path forwarding) ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3003 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004" forwarded to 3001 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004" forwarded to 3005
E 3005	B, D, F	D	B, F	⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3002 ✖ Broadcast message rejected (reverse path forwarding) ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004 /127.0.0.1:3005" forwarded to 3002 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004 /127.0.0.1:3005" forwarded to 3006 ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3002 /127.0.0.1:3006 ✖ Broadcast message rejected (reverse path forwarding)
F 3006	B, E	B	E	⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3002 ➡ Broadcast message "bc: aloha /127.0.0.1:3001 /127.0.0.1:3002 /127.0.0.1:3006" forwarded to 3005 ⇅ RECEIVED BROADCAST : aloha /127.0.0.1:3001 /127.0.0.1:3003 /127.0.0.1:3004 /127.0.0.1:3005 ✖ Broadcast message rejected (reverse path forwarding)

VI. Conclusion

After examining our correctness results, we have found that it is optimal to use poisoned reverse when configuring routers. This makes the routers much more resilient to changes in the network, such as new routers entering or old routers leaving. We have also noted that the more connected a network of routers is, the longer it will take for the algorithm to run, as it will take longer for changes to propagate through the network and for all the updates to end. We also saw that broadcasting can be rather inefficient in terms of how many packets are sent even when using reverse path forwarding. However, our simulation does provide a very close to real-life implementation of RIP.

VII. Member Contributions

Erik

Contributed to various parts of the project, such as the **`DistanceTable`** and **`DistanceVector`** classes, the basic implementation of the algorithm, and the calculation of forwarding tables. Wrote the report and verified the correctness results.

Wassim

Formalized the software design and documentation. Implemented the low-level aspects of the program, including **`Router`** class, the UDP socket sending and receiving, the I/O management, multi-threading and timers. Implemented the extra part of the project (broadcasting and reverse path forwarding).

Zura

Implemented the advanced features of the distance vector algorithm, including poisoned reverse, printing, sending packets, changing path weights, and forwarding/dropping packets. Added all of the debugging and terminal output.

VIII. References

Kurose, James F., and Keith W. Ross. Computer Networking: A Top-Down Approach. Pearson, 2013.