Project 3 Report:
Distance Vector Routing Simulator

Ha Vu & Thanh Vu
Professor Amir Sadovnik
CS 305: Networks
May 6, 2017

I. Introduction

1. Goals

In this project, we aim to implement the distance vector algorithm in the routing task of a network of routers, based upon the existing Routing Information Protocol (RIP). Our project simulate a router using a multithreading process that can communicate with one other using UDP. In particular, a router simulation should be able to handle the following tasks [1]:

1. 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.
2. Advertise its distance vector every *n* seconds or every time its own distance vector changes.
3. Drop the neighbor if it does not receive an update after 3 time periods of *n* seconds.
4. Forward a message with a specific destination to a neighbor router according to its forwarding table.
5. Accept changes in its weights to its neighbors and advertise it to them.
6. Support poisoned reverse as an option.

Section II will go into more details on the algorithm, data structures and class design of the simulator. In Section III, we will verify whether our simulation achieves all of these goals by a series of correctness testings in five scenarios. Discussion on success and limitations will be in the last section.

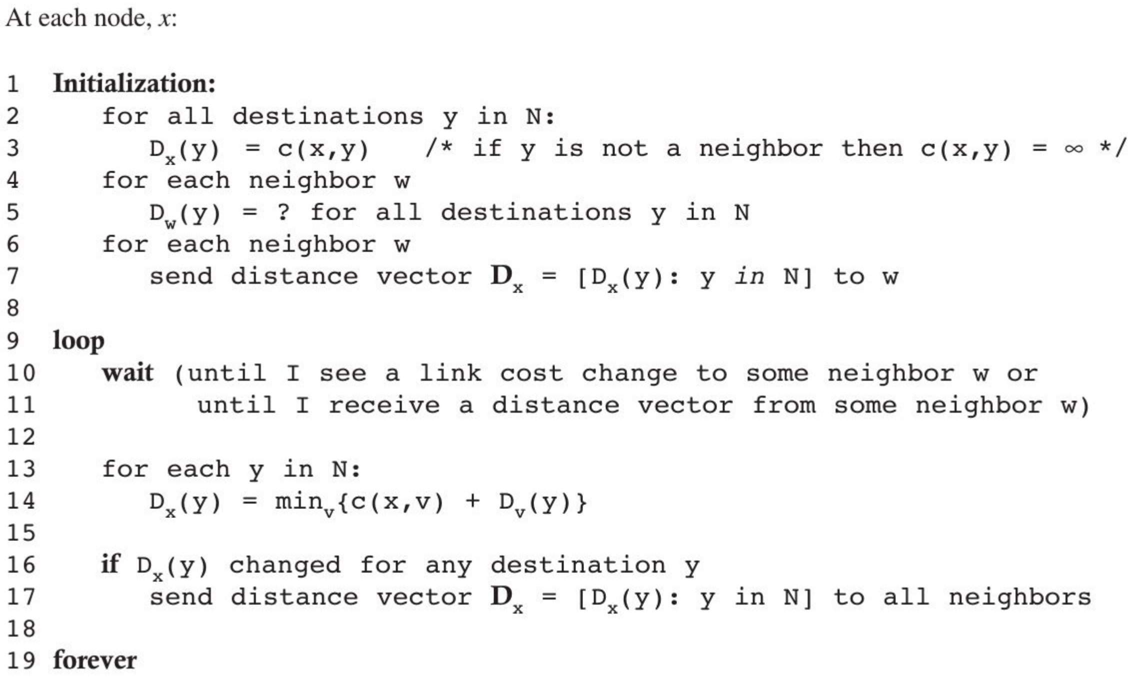
2. Contributions

Ha	**Thanh**
Distance vector algorithm Data structures for distance vector, forward table, link weights (in Neighbor class) Router listener thread	Messaging protocols between routers, including 3 types of messages Auto updater and console reader threads Correctness testing
Overall design Report	

II. Design

1. The Distance Vector Algorithm

Based on the Bellman-Ford equation, the distance vector algorithm is a localized, iterative, asynchronous and distributed algorithm to find the shortest path locally at each router. It has been summarized in pseudocode by Kurose and Ross as below [2, p. 373].



At each node, x:

```
Initialization:
   for all destinations y in N:
      $D_x(y) = c(x,y)$   /* if y is not a neighbor then $c(x,y) = \infty$ */
   for each neighbor w
      $D_w(y)$ = ? for all destinations y in N
   for each neighbor w
      send distance vector $\mathbf{D}_x = [D_x(y)$: y in N] to w

loop
   wait (until I see a link cost change to some neighbor w or
         until I receive a distance vector from some neighbor w)

   for each y in N:
      $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$

   if $D_x(y)$ changed for any destination y
      send distance vector $\mathbf{D}_x = [D_x(y)$: y in N] to all neighbors

forever
```

Figure 1: Distance Vector algorithm in pseudocode [2, p. 373]

On top of this idealized algorithm, the distance vector algorithm implemented in our program is also based upon the Routing Information Protocol, a distance-vector Intra-AS routing protocol which has additional checks and built-in fields to prevent real life problems such as router dropping in and out of network, weight changes leading to routing loops, etc. The implementation of this algorithm is discussed below.

Implementation

In our implementation, as an infinity distance means that the destination is unreachable, any router that is set at infinity distance from the current router is not included in the distance vector of the router. In addition, the router itself is not included in the distance vector to simplify the code and remove redundancy in the advertised vectors; we compensate for this by setting the initial distance to neighbors as their link weights every time the algorithm is run.

The pseudocode for the algorithm is as below:

```
In current router x:
   1. Empty the distance vector
   2. For each neighbor n,
         a. Set the distance to neighbor $D_x(n)$ as its link weight
         b. Set the neighbor address as the nexthop to itself in the
            forwarding table
   3. Then, for each neighbor n's distance vector:
      if destination address y of an entry is not this router address
         a. Update vector: $D_x(y) = \min(c_x(n) + D_n(y), D_x(y))$
         b. If vector is changed, set the n as the nexthop to y in
            forwarding table
```

The algorithm is implemented under the function `runDVAlgorithm()` in the `Router` class, and is invoked whenever there is a new change in distance vector update or a link weight change.

Poisoned Reverse

Poisoned reverse is implemented to counteract the count-to-infinity problem in a small router loop. Poisoned reverse is the solution in which if a router always routes through a neighbor to get to another neighbor, it will advertise to the transition neighbor that its distance to the destination is infinity [2, p. 377].

We implement this option under the function `advertiseDV()` in the `Router` class. The pseudocode is as below:

```
For each neighbor n of current router:
   1. Create a copy dvToSend of current distance vector
   2. Preprocess dvToSend:
      For each destination address y in the forward table:
            if next hop to y is n && n is not y:
                  remove entry to y in the forward table
   3. Advertise this dvToSend to n
```

This option is enabled when `-reverse` is included in the command line arguments.

2. Design and Data Structures

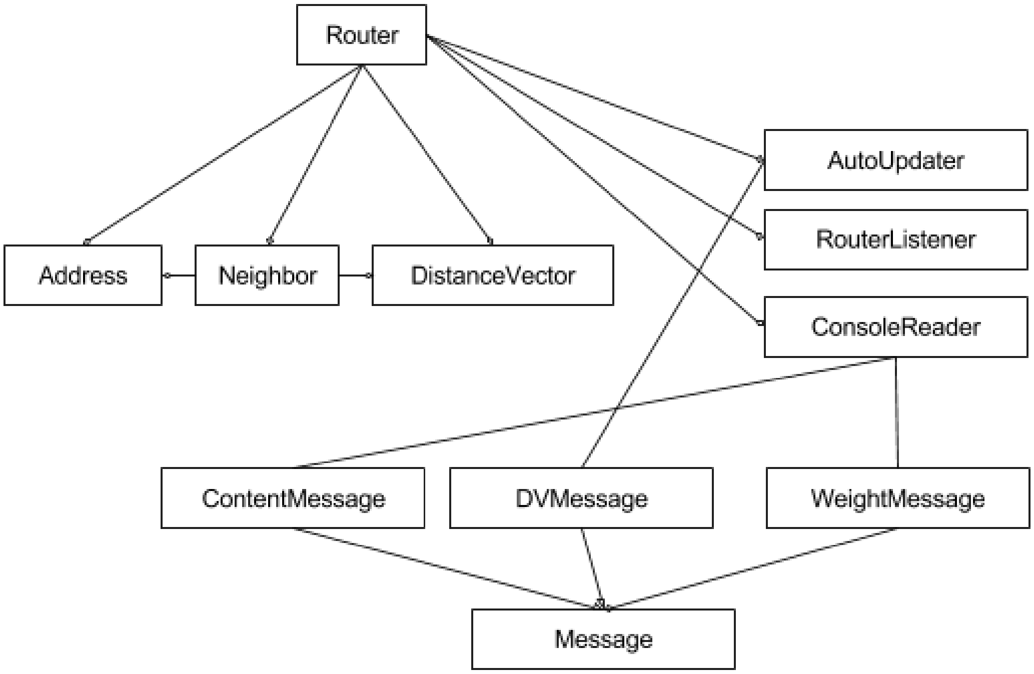


Figure 2: A class diagram of the program

The program's class design is show in Figure 2. The `Router` class is the overall class which contains all the functionality of a router simulated. A `Router` contains an `Address` (includes ip and port number), a `DistanceVector`, and a container of `Neighbor` routers. A `Neighbor` contains all information about a neighboring router, including its `Address`, its own `DistanceVector`, the link weight between the router and itself, and the `Timer` that keeps track of when to drop itself.

In addition, a router contains 3 threads, each contains an object implementing Runnable interface. `AutoUpdater` automatically advertise a distance vector update every *T* seconds to its neighbors. `RouterListener` handles the incoming stream of packets from other neighbors, and handle them appropriately. `ConsoleReader` reads the input from the user and carries out actions such as sending messages, update weight, and print the current distance vector.

Last but not least, the communications between routers are facilitated through 3 types of `Message` (base class). `ContentMessage` is sent from user and forwarded to a router destination. `DVMessage` advertises the router's distance vector to its neighbors. `WeightMessage` informs a neighbor router when there is a link weight change between them.

Data Structures

As the program uses multiple threads, all data containers are threadsafe containers using the `java.util.concurrent` package. Additionally, due to the wide usage of lookup tables, we implement all of our containers with hashing (`HashMap` or `HashSet`) to achieve an O(1) lookup time complexity.

In many cases, `Address` is used as a key for hashing, and thus `equals()` and `hashCode()` methods in `Address` are overriden to enable its usage as key.

`DistanceVector` is implemented as a `ConcurrentHashMap<Address, Integer>`, which is a HashMap data structure that is synchronized and allows for threadsafe operation. It maps a destination address to its distance after running distance vector algorithm. A `Router` stores a `DistanceVector` for itself, and one for each `Neighbor`.

Forward table is implemented in the Router as a `ConcurrentHashMap<Address, Neighbor>`. It maps a destination address to a `Neighbor` router to forward to. When a `Neighbor` is dropped, mappings that contain that `Neighbor` are also removed.

In the router, `Neighbor` objects are stored in a `ConcurrentHashMap<Address, Neighbor>` object named `neighborCache`, which maps a neighbor address to its information. This cache never drops any `Neighbor`: information about dropped `Neighbors` are still kept. The "live" neighbors are thus kept track of through an addtional `Set<Address>` named `liveNeighborAdds`, which contains all the addresses of neighbors that are considered live. This implementation allows for continuous use of neighbor information when a neighbor is dropped and goes live again.

Communication protocols

Routers in the network communicate using three different types of messaging protocols: DVMessage, WeightMessage, and ContentMessage. The first two protocols are used for routers to communicate and maintain the integrity of the network; i.e. they are control protocols. The last protocol, ContentMessage, is used to fulfil the network's main job - delivering messages from one host to another. The details of these protocols are provided as follows

a) DV Message:

- + Functionality: Used for sending distance vector
- + Format:

```
[type][D][src-ip][D][src-port][D][dst-ip][D]
     [dest-port][D][distance-vector]
```

  with

```
[D] = some delimiter
[distance-vector] = [ip]:[port] [dist] [ip]:[port]
     [dist] [ip]:[port] [dist] ...
```

- + Example: with [D] = " "

```
DV 127.0.0.1 10032 127.0.0.1 10033 127.0.0.1:10033 7
     127.0.0.1:10034 1 127.0.0.1:10035 11
```

b) Weight Message:

- + Functionality: Used for announcing weight change to neighbor routers

+ Format:

  ```
  [type][D][src-ip][D][src-port][D][dst-ip][D]
       [dest-port][D][weight]
  ```

  with

  ```
  [D] = some delimiter
  ```

+ Example: with [D] = “ ”

  ```
  DV 127.0.0.1 10032 127.0.0.1 10033 7
  ```

c) Content Message:

+ Functionality: Used for delivering content messages through the network
+ Format:

  ```
  [type][D][src-ip][D][src-port][D][dst-ip][D]
       [dest-port][D][msg][timetolive][path]
  ```

  with

  ```
  [D] = some delimiter
  [path] = [ip]-[port] [ip]-[port] [ip]-[port] ...
  ```

+ Example: with [D] = “ ”

  ```
  DV 127.0.0.1 10032 127.0.0.1 10033 hello network 15
       127.0.0.1-10033 127.0.0.1-10034 1 ...
  ```

Threads

All of our thread implementation uses the Runnable interface. As each of the thread is continuous until the `Router` stops, all actions are encapsulated in a `while(running)` loop, in which running is a `volatile boolean`. To stop a thread, `running` just needs to be set to `false`.

AutoUpdater

To keep the neighbor routers up-to-date and detect dropped neighbors, we implemented AutoUpdater to automatically advertise distance vector. At runtime, the AutoUpdater thread starts a scheduled task for sending the distance vector update of its router to the neighboring routers every *T* seconds. The implementation of this thread utilizes Java Timer class.

RouterListener

At its runtime, the `RouterListener` thread opens a `DatagramSocket` (UDP socket), listening at the specified port in the address. In the `while` loop, the socket constant listens for packets, translate it into the appropriate type of message. In particular:

- `ContentMessage`: `RouterListener` would forward it if it's destined to another router, otherwise it would print out the message.
- `DVMessage`: `RouterListener` would restart the timer for the neighbor that the message comes from, and then update the new distance vector, run the distance vector algorithm, and advertise if there is any change in the vector.

- `WeightMessage`: `RouterListener` would update the link weight to the source neighbor, run distance vector algorithm, and advertise if there is any change in vector.

ConsoleReader

The ConsoleReader thread listens to the terminal, reads user's commands, and perform the necessary operations. The thread accepts the following four commands:

a) **PRINT** -- print out the current router's distance vector and the distance vectors received from its neighbors.
b) **MSG <dst-ip> <dst-port> <msg>** -- send the message *msg* to a destination router with the specified address(dst-ip and dst-port).
c) **CHANGE <dst-ip> <dst-port> <new-weight>** -- change the weight between the current router and the destination router to *new-weight* and inform the destination router about the change.
d) **STOP** -- stop this ConsoleReader thread

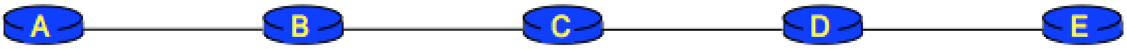
Timer in Neighbor

To keep track of whether a `Neighbor` is still alive, each `Neighbor` has a `Timer` object. The timer schedules a task (which essentially is a new thread) at *T*n* seconds from the scheduling time to drop the `Neighbor` from the `Router`. Whenever a `DVMessage` is received from the `Neighbor`, the previously scheduled task is canceled, and the `Timer` is restarted.

III. Correctness Results

(Full output results can be found in submission/output_results/ folder)

1. Simple network

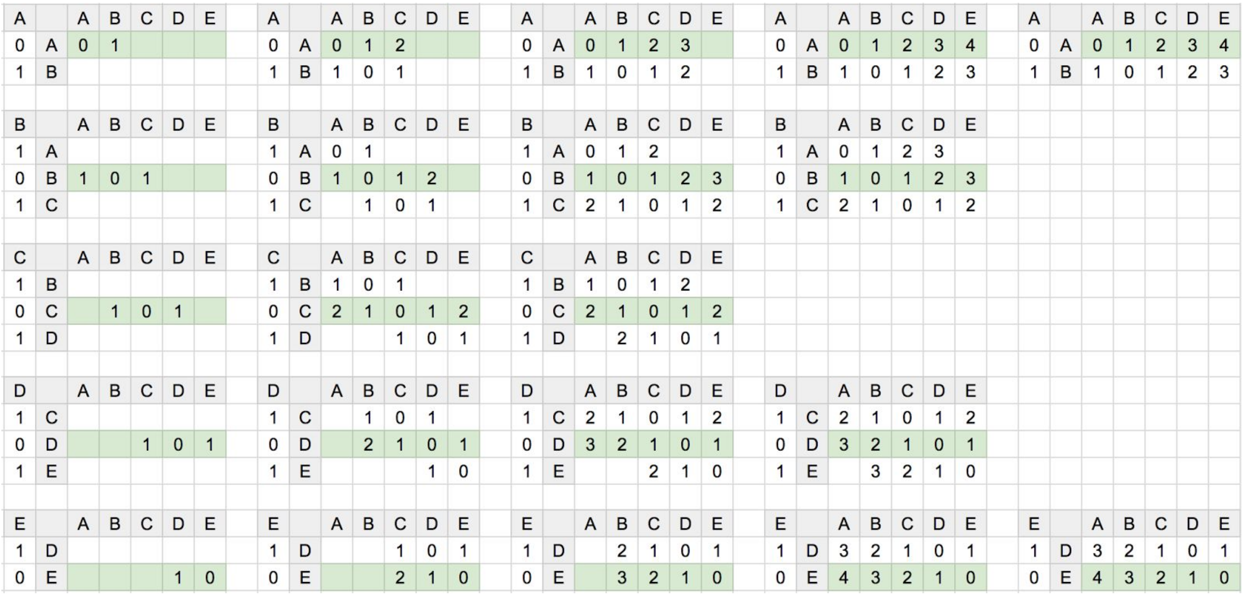


a. Creation and Stabilization

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

Expectation

** blank space = unreachable = infinite distance*



A		A	B	C	D	E
0	A	0	1			
1	B					

A		A	B	C	D	E
0	A	0	1	2		
1	B	1	0	1		

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

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

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

B		A	B	C	D	E
1	A					
0	B	1	0	1		
1	C					

B		A	B	C	D	E
1	A	0	1			
0	B	1	0	1	2	
1	C		1	0	1	

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

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

C		A	B	C	D	E
1	B					
0	C		1	0	1	
1	D					

C		A	B	C	D	E
1	B	1	0	1		
0	C	2	1	0	1	2
1	D			1	0	1

C		A	B	C	D	E
1	B	1	0	1	2	
0	C	2	1	0	1	2
1	D		2	1	0	1

D		A	B	C	D	E
1	C					
0	D			1	0	1
1	E					

D		A	B	C	D	E
1	C		1	0	1	
0	D		2	1	0	1
1	E				1	0

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

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

E		A	B	C	D	E
1	D					
0	E				1	0

E		A	B	C	D	E
1	D			1	0	1
0	E			2	1	0

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

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

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

Results

Router A's output

```
run:
new dv calculated:
127.0.0.1:10022 1 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10022 1
Update sent to all neighbors at time 20
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10021 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 2
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
```

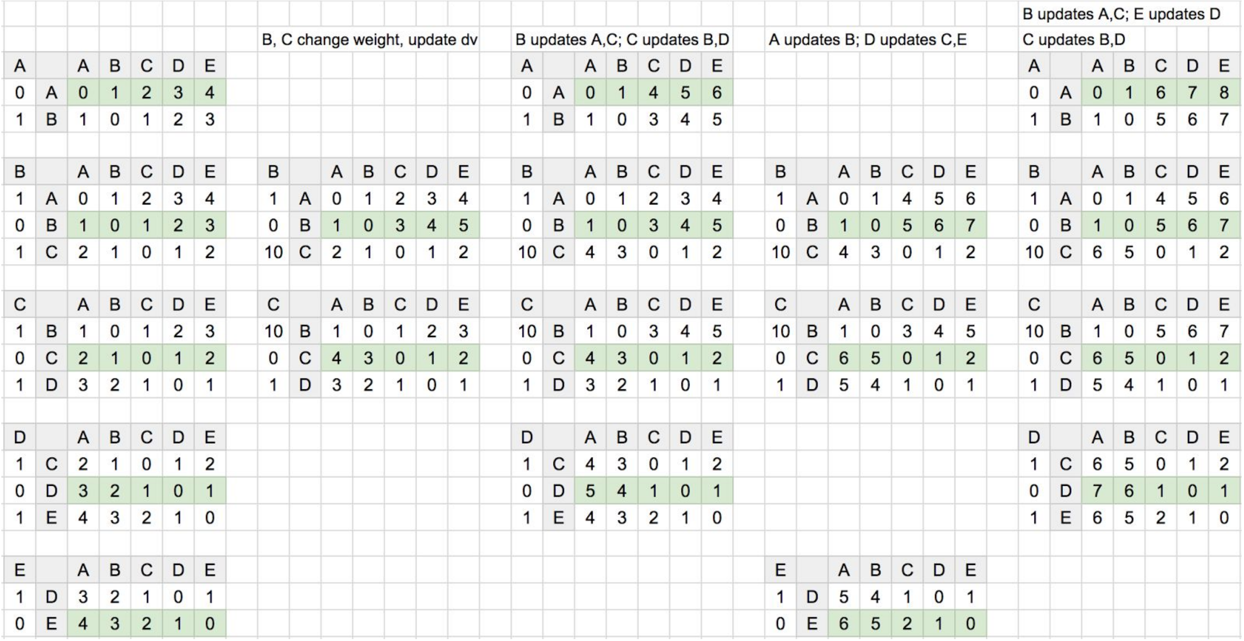
```
127.0.0.1:10024 2
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10024 3 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 4 127.0.0.1:10022
127.0.0.1:10024 3 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
Update sent to all neighbors at time 40
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
Update sent to all neighbors at time 60
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
```

127.0.0.1:10024 2
Update sent to all neighbors at time 80
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3

b. Weight change

The same network as (a), but after convergence then change the weight between B and C to 10, no poisoned reverse.

Expectation



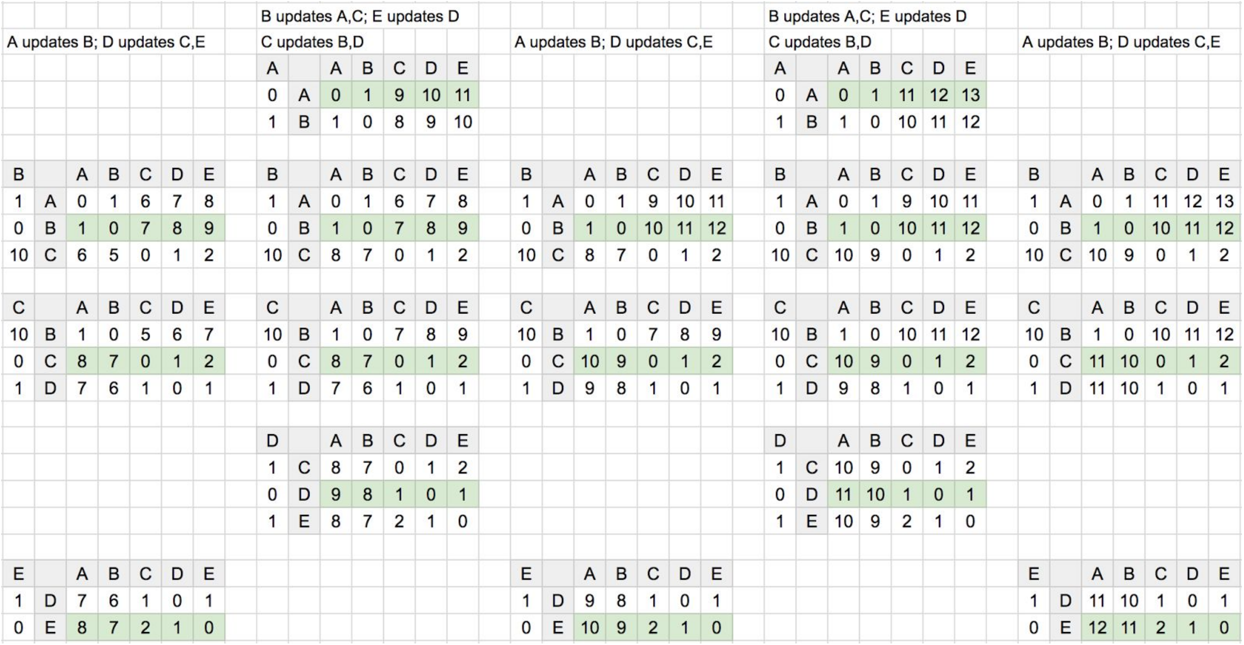
A		A	B	C	D	E
0	A	0	1	2	3	4
1	B	1	0	1	2	3
B		A	B	C	D	E
1	A	0	1	2	3	4
0	B	1	0	1	2	3
1	C	2	1	0	1	2
C		A	B	C	D	E
1	B	1	0	1	2	3
0	C	2	1	0	1	2
1	D	3	2	1	0	1
D		A	B	C	D	E
1	C	2	1	0	1	2
0	D	3	2	1	0	1
1	E	4	3	2	1	0
E		A	B	C	D	E
1	D	3	2	1	0	1
0	E	4	3	2	1	0

B, C change weight, update dv						
B		A	B	C	D	E
1	A	0	1	2	3	4
0	B	1	0	3	4	5
10	C	2	1	0	1	2
C		A	B	C	D	E
10	B	1	0	1	2	3
0	C	4	3	0	1	2
1	D	3	2	1	0	1

B updates A,C; C updates B,D						
A		A	B	C	D	E
0	A	0	1	4	5	6
1	B	1	0	3	4	5
B		A	B	C	D	E
1	A	0	1	2	3	4
0	B	1	0	3	4	5
10	C	4	3	0	1	2
C		A	B	C	D	E
10	B	1	0	3	4	5
0	C	4	3	0	1	2
1	D	3	2	1	0	1
D		A	B	C	D	E
1	C	4	3	0	1	2
0	D	5	4	1	0	1
1	E	4	3	2	1	0

A updates B; D updates C,E						
B		A	B	C	D	E
1	A	0	1	4	5	6
0	B	1	0	5	6	7
10	C	4	3	0	1	2
C		A	B	C	D	E
10	B	1	0	3	4	5
0	C	6	5	0	1	2
1	D	5	4	1	0	1
E		A	B	C	D	E
1	D	5	4	1	0	1
0	E	6	5	2	1	0

B updates A,C; E updates D						
C updates B,D						
A		A	B	C	D	E
0	A	0	1	6	7	8
1	B	1	0	5	6	7
B		A	B	C	D	E
1	A	0	1	4	5	6
0	B	1	0	5	6	7
10	C	6	5	0	1	2
C		A	B	C	D	E
10	B	1	0	5	6	7
0	C	6	5	0	1	2
1	D	5	4	1	0	1
D		A	B	C	D	E
1	C	6	5	0	1	2
0	D	7	6	1	0	1
1	E	6	5	2	1	0



A updates B; D updates C,E

B		A	B	C	D	E
1	A	0	1	6	7	8
0	B	1	0	7	8	9
10	C	6	5	0	1	2

C		A	B	C	D	E
10	B	1	0	5	6	7
0	C	8	7	0	1	2
1	D	7	6	1	0	1

E		A	B	C	D	E
1	D	7	6	1	0	1
0	E	8	7	2	1	0

B updates A,C; E updates D
C updates B,D

A		A	B	C	D	E
0	A	0	1	9	10	11
1	B	1	0	8	9	10

B		A	B	C	D	E
1	A	0	1	6	7	8
0	B	1	0	7	8	9
10	C	8	7	0	1	2

C		A	B	C	D	E
10	B	1	0	7	8	9
0	C	8	7	0	1	2
1	D	7	6	1	0	1

D		A	B	C	D	E
1	C	8	7	0	1	2
0	D	9	8	1	0	1
1	E	8	7	2	1	0

A updates B; D updates C,E

B		A	B	C	D	E
1	A	0	1	9	10	11
0	B	1	0	10	11	12
10	C	8	7	0	1	2

C		A	B	C	D	E
10	B	1	0	7	8	9
0	C	10	9	0	1	2
1	D	9	8	1	0	1

E		A	B	C	D	E
1	D	9	8	1	0	1
0	E	10	9	2	1	0

B updates A,C; E updates D
C updates B,D

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

B		A	B	C	D	E
1	A	0	1	9	10	11
0	B	1	0	10	11	12
10	C	10	9	0	1	2

C		A	B	C	D	E
10	B	1	0	10	11	12
0	C	10	9	0	1	2
1	D	9	8	1	0	1

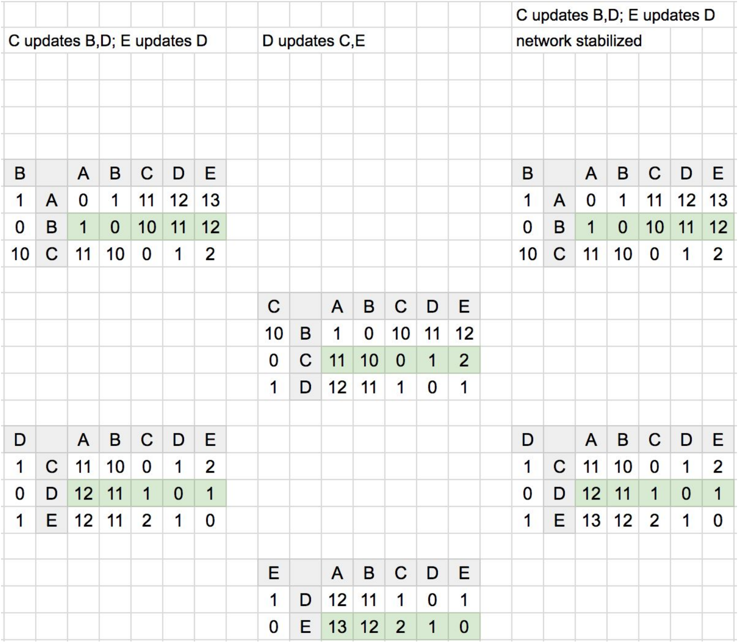
D		A	B	C	D	E
1	C	10	9	0	1	2
0	D	11	10	1	0	1
1	E	10	9	2	1	0

A updates B; D updates C,E

B		A	B	C	D	E
1	A	0	1	11	12	13
0	B	1	0	10	11	12
10	C	10	9	0	1	2

C		A	B	C	D	E
10	B	1	0	10	11	12
0	C	11	10	0	1	2
1	D	11	10	1	0	1

E		A	B	C	D	E
1	D	11	10	1	0	1
0	E	12	11	2	1	0



C updates B,D; E updates D

B		A	B	C	D	E
1	A	0	1	11	12	13
0	B	1	0	10	11	12
10	C	11	10	0	1	2

D		A	B	C	D	E
1	C	11	10	0	1	2
0	D	12	11	1	0	1
1	E	12	11	2	1	0

D updates C,E

C		A	B	C	D	E
10	B	1	0	10	11	12
0	C	11	10	0	1	2
1	D	12	11	1	0	1

E		A	B	C	D	E
1	D	12	11	1	0	1
0	E	13	12	2	1	0

C updates B,D; E updates D
network stabilized

B		A	B	C	D	E
1	A	0	1	11	12	13
0	B	1	0	10	11	12
10	C	11	10	0	1	2

D		A	B	C	D	E
1	C	11	10	0	1	2
0	D	12	11	1	0	1
1	E	13	12	2	1	0

Results

Router A’s output

```
run:
new dv calculated:
127.0.0.1:10022 1 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10022 1
Update sent to all neighbors at time 20
```

```
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10021 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 2
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10024 2
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10024 3 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 4 127.0.0.1:10022
127.0.0.1:10024 3 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
Update sent to all neighbors at time 40               // network stabilized
```

```
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
Update sent to all neighbors at time 60
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
Update sent to all neighbors at time 80
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
Update sent to all neighbors at time 100
127.0.0.1:10023 2
127.0.0.1:10022 1
127.0.0.1:10025 4
127.0.0.1:10024 3
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
new dv received from 127.0.0.1:10022
127.0.0.1:10023 3
127.0.0.1:10021 1
127.0.0.1:10025 5
```

```
127.0.0.1:10024 4
new dv calculated:                                // start counting to infinity
127.0.0.1:10023 4 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 6 127.0.0.1:10022
127.0.0.1:10024 5 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 4
127.0.0.1:10022 1
127.0.0.1:10025 6
127.0.0.1:10024 5
new dv received from 127.0.0.1:10022
127.0.0.1:10023 5
127.0.0.1:10021 1
127.0.0.1:10025 7
127.0.0.1:10024 6
new dv calculated:
127.0.0.1:10023 6 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 8 127.0.0.1:10022
127.0.0.1:10024 7 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 6
127.0.0.1:10022 1
127.0.0.1:10025 8
127.0.0.1:10024 7
new dv received from 127.0.0.1:10022
127.0.0.1:10023 7
127.0.0.1:10021 1
127.0.0.1:10025 9
127.0.0.1:10024 8
new dv calculated:
127.0.0.1:10023 8 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 10 127.0.0.1:10022
127.0.0.1:10024 9 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 8
127.0.0.1:10022 1
127.0.0.1:10025 10
127.0.0.1:10024 9
new dv received from 127.0.0.1:10022
127.0.0.1:10023 9
```

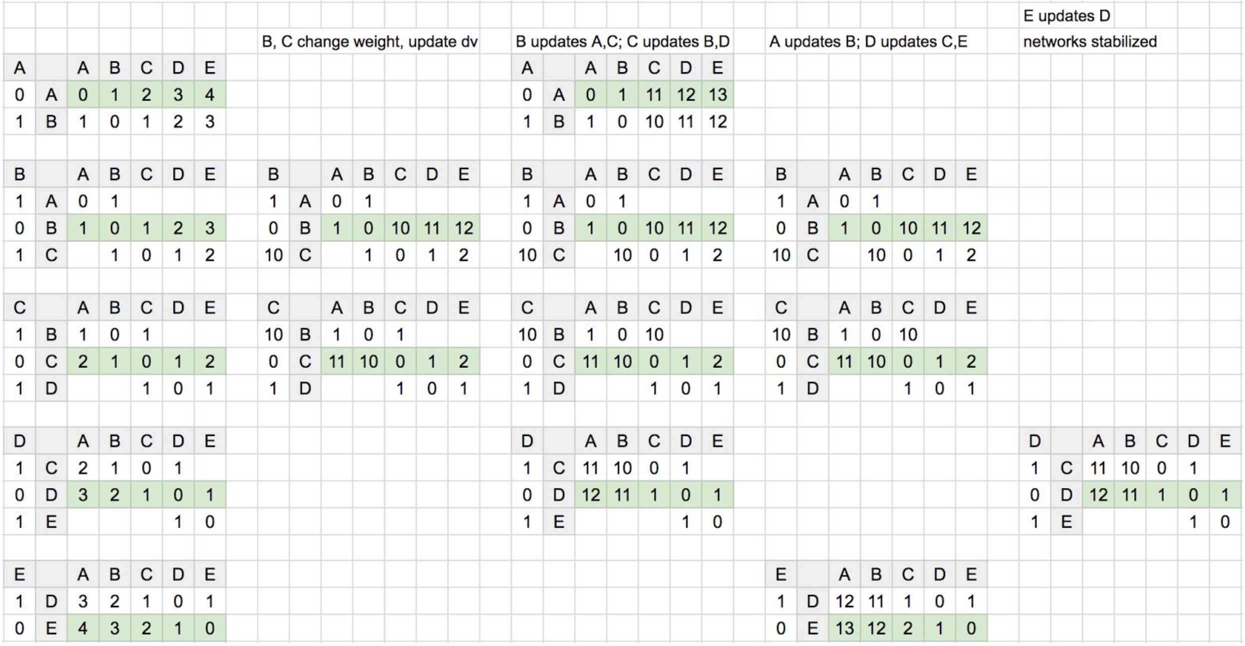
```
127.0.0.1:10021 1
127.0.0.1:10025 11
127.0.0.1:10024 10
new dv calculated:
127.0.0.1:10023 10 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 12 127.0.0.1:10022
127.0.0.1:10024 11 127.0.0.1:10022
Update sent to all neighbors
127.0.0.1:10023 10
127.0.0.1:10022 1
127.0.0.1:10025 12
127.0.0.1:10024 11
new dv received from 127.0.0.1:10022
127.0.0.1:10023 10
127.0.0.1:10021 1
127.0.0.1:10025 12
127.0.0.1:10024 11
new dv calculated:
127.0.0.1:10023 11 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 13 127.0.0.1:10022
127.0.0.1:10024 12 127.0.0.1:10022
Update sent to all neighbors                        // correct dv, stop cnt-to-inf
127.0.0.1:10023 11
127.0.0.1:10022 1
127.0.0.1:10025 13
127.0.0.1:10024 12
Update sent to all neighbors at time 120
127.0.0.1:10023 11
127.0.0.1:10022 1
127.0.0.1:10025 13
127.0.0.1:10024 12
new dv received from 127.0.0.1:10022
127.0.0.1:10023 10
127.0.0.1:10021 1
127.0.0.1:10025 12
127.0.0.1:10024 11
Update sent to all neighbors at time 140            // networks stabilized
127.0.0.1:10023 11
127.0.0.1:10022 1
127.0.0.1:10025 13
127.0.0.1:10024 12
```

c. Weight change and poison reverse

The same situation as (b) with poisoned reverse.

Expectation

** blank space = unreachable = infinite distance*



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

B, C change weight, update dv						
B		A	B	C	D	E
1	A	0	1			
0	B	1	0	10	11	12
10	C		1	0	1	2
C		A	B	C	D	E
10	B	1	0	1		
0	C	11	10	0	1	2
1	D			1	0	1

B updates A,C; C updates B,D						
A		A	B	C	D	E
0	A	0	1	11	12	13
1	B	1	0	10	11	12
B		A	B	C	D	E
1	A	0	1			
0	B	1	0	10	11	12
10	C		10	0	1	2
C		A	B	C	D	E
10	B	1	0	10		
0	C	11	10	0	1	2
1	D			1	0	1
D		A	B	C	D	E
1	C	11	10	0	1	
0	D	12	11	1	0	1
1	E				1	0

A updates B; D updates C,E						
B		A	B	C	D	E
1	A	0	1			
0	B	1	0	10	11	12
10	C		10	0	1	2
C		A	B	C	D	E
10	B	1	0	10		
0	C	11	10	0	1	2
1	D			1	0	1
E		A	B	C	D	E
1	D	12	11	1	0	1
0	E	13	12	2	1	0

E updates D						
networks stabilized						
D		A	B	C	D	E
1	C	11	10	0	1	
0	D	12	11	1	0	1
1	E				1	0

Results

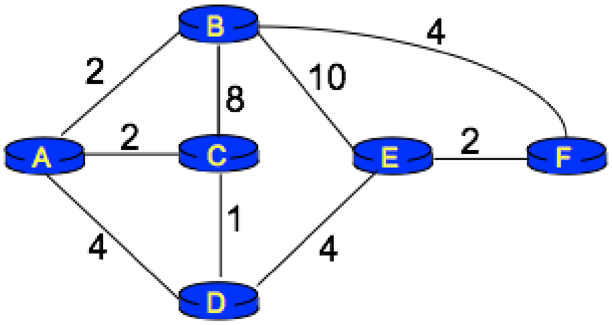
Router A's output

```
run:
new dv calculated:
127.0.0.1:10022 1 127.0.0.1:10022
Update sent to neighbor 127.0.0.1:10022
127.0.0.1:10022 1
Update sent to neighbor 127.0.0.1:10022 at time 20
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10021 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
Update sent to neighbor 127.0.0.1:10022
127.0.0.1:10022 1
```

```
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10024 2
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10024 3 127.0.0.1:10022
Update sent to neighbor 127.0.0.1:10022
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
new dv calculated:
127.0.0.1:10023 2 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 4 127.0.0.1:10022
127.0.0.1:10024 3 127.0.0.1:10022
Update sent to neighbor 127.0.0.1:10022                  // network stabilized
127.0.0.1:10022 1
Update sent to neighbor 127.0.0.1:10022 at time 40
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
Update sent to neighbor 127.0.0.1:10022 at time 60
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 1
127.0.0.1:10021 1
127.0.0.1:10025 3
127.0.0.1:10024 2
new dv received from 127.0.0.1:10022                     // weight change
127.0.0.1:10023 10
127.0.0.1:10021 1
127.0.0.1:10025 12
```

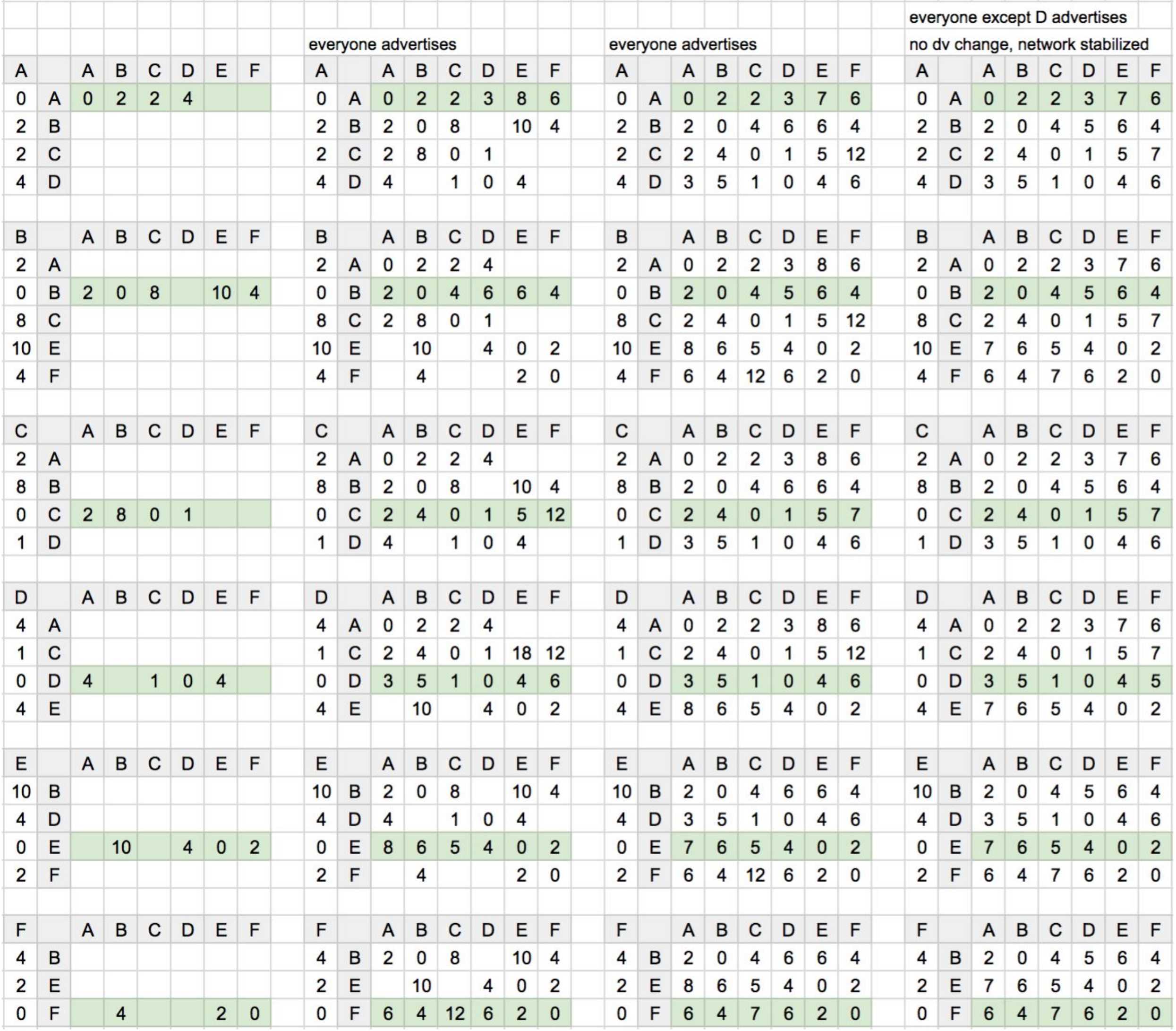
```
127.0.0.1:10024 11
new dv calculated:
127.0.0.1:10023 11 127.0.0.1:10022
127.0.0.1:10022 1 127.0.0.1:10022
127.0.0.1:10025 13 127.0.0.1:10022
127.0.0.1:10024 12 127.0.0.1:10022                    // correct dv, no cnt-to-inf
Update sent to neighbor 127.0.0.1:10022
127.0.0.1:10022 1
Update sent to neighbor 127.0.0.1:10022 at time 80
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 10
127.0.0.1:10021 1
127.0.0.1:10025 12
127.0.0.1:10024 11
Update sent to neighbor 127.0.0.1:10022 at time 100
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 10
127.0.0.1:10021 1
127.0.0.1:10025 12
127.0.0.1:10024 11
Update sent to neighbor 127.0.0.1:10022 at time 120     // network stabilized
127.0.0.1:10022 1
new dv received from 127.0.0.1:10022
127.0.0.1:10023 10
127.0.0.1:10021 1
127.0.0.1:10025 12
127.0.0.1:10024 11
Update sent to neighbor 127.0.0.1:10022 at time 140
127.0.0.1:10022 1
```

2. Complex network



a. Creation and Stabilization

Expectation



A		A	B	C	D	E	F
0	A	0	2	2	4		
2	B						
2	C						
4	D						
B		A	B	C	D	E	F
2	A						
0	B	2	0	8		10	4
8	C						
10	E						
4	F						
C		A	B	C	D	E	F
2	A						
8	B						
0	C	2	8	0	1		
1	D						
D		A	B	C	D	E	F
4	A						
1	C						
0	D	4		1	0	4	
4	E						
E		A	B	C	D	E	F
10	B						
4	D						
0	E		10		4	0	2
2	F						
F		A	B	C	D	E	F
4	B						
2	E						
0	F		4			2	0

everyone advertises							
A		A	B	C	D	E	F
0	A	0	2	2	3	8	6
2	B	2	0	8		10	4
2	C	2	8	0	1		
4	D	4		1	0	4	
B		A	B	C	D	E	F
2	A	0	2	2	4		
0	B	2	0	4	6	6	4
8	C	2	8	0	1		
10	E		10		4	0	2
4	F		4			2	0
C		A	B	C	D	E	F
2	A	0	2	2	4		
8	B	2	0	8		10	4
0	C	2	4	0	1	5	12
1	D	4		1	0	4	
D		A	B	C	D	E	F
4	A	0	2	2	4		
1	C	2	4	0	1	18	12
0	D	3	5	1	0	4	6
4	E		10		4	0	2
E		A	B	C	D	E	F
10	B	2	0	8		10	4
4	D	4		1	0	4	
0	E	8	6	5	4	0	2
2	F		4			2	0
F		A	B	C	D	E	F
4	B	2	0	8		10	4
2	E		10		4	0	2
0	F	6	4	12	6	2	0

everyone advertises							
A		A	B	C	D	E	F
0	A	0	2	2	3	7	6
2	B	2	0	4	6	6	4
2	C	2	4	0	1	5	12
4	D	3	5	1	0	4	6
B		A	B	C	D	E	F
2	A	0	2	2	3	8	6
0	B	2	0	4	5	6	4
8	C	2	4	0	1	5	12
10	E	8	6	5	4	0	2
4	F	6	4	12	6	2	0
C		A	B	C	D	E	F
2	A	0	2	2	3	8	6
8	B	2	0	4	6	6	4
0	C	2	4	0	1	5	7
1	D	3	5	1	0	4	6
D		A	B	C	D	E	F
4	A	0	2	2	3	8	6
1	C	2	4	0	1	5	12
0	D	3	5	1	0	4	6
4	E	8	6	5	4	0	2
E		A	B	C	D	E	F
10	B	2	0	4	6	6	4
4	D	3	5	1	0	4	6
0	E	7	6	5	4	0	2
2	F	6	4	12	6	2	0
F		A	B	C	D	E	F
4	B	2	0	4	6	6	4
2	E	8	6	5	4	0	2
0	F	6	4	7	6	2	0

everyone except D advertises							
no dv change, network stabilized							
A		A	B	C	D	E	F
0	A	0	2	2	3	7	6
2	B	2	0	4	5	6	4
2	C	2	4	0	1	5	7
4	D	3	5	1	0	4	6
B		A	B	C	D	E	F
2	A	0	2	2	3	7	6
0	B	2	0	4	5	6	4
8	C	2	4	0	1	5	7
10	E	7	6	5	4	0	2
4	F	6	4	7	6	2	0
C		A	B	C	D	E	F
2	A	0	2	2	3	7	6
8	B	2	0	4	5	6	4
0	C	2	4	0	1	5	7
1	D	3	5	1	0	4	6
D		A	B	C	D	E	F
4	A	0	2	2	3	7	6
1	C	2	4	0	1	5	7
0	D	3	5	1	0	4	5
4	E	7	6	5	4	0	2
E		A	B	C	D	E	F
10	B	2	0	4	5	6	4
4	D	3	5	1	0	4	6
0	E	7	6	5	4	0	2
2	F	6	4	7	6	2	0
F		A	B	C	D	E	F
4	B	2	0	4	5	6	4
2	E	7	6	5	4	0	2
0	F	6	4	7	6	2	0

Results

Router E’s output

```
run:
new dv calculated:
127.0.0.1:10032 10 127.0.0.1:10032
Update sent to all neighbors
127.0.0.1:10032 10
new dv calculated:
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10032 10 127.0.0.1:10032
```

```
Update sent to all neighbors
127.0.0.1:10034 4
127.0.0.1:10032 10
new dv calculated:
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10032 10 127.0.0.1:10032
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10034 4
127.0.0.1:10032 10
127.0.0.1:10036 2
Update sent to all neighbors at time 20
127.0.0.1:10034 4
127.0.0.1:10032 10
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10035 2
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10034 4
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv calculated:
127.0.0.1:10031 12 127.0.0.1:10032
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10033 14 127.0.0.1:10032
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 12
127.0.0.1:10034 4
127.0.0.1:10033 14
127.0.0.1:10032 6
```

```
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10035 2
127.0.0.1:10032 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 9
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 4
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv received from 127.0.0.1:10034
127.0.0.1:10031 3
127.0.0.1:10035 4
127.0.0.1:10033 1
127.0.0.1:10032 5
127.0.0.1:10036 6
new dv calculated:
127.0.0.1:10031 7 127.0.0.1:10034
127.0.0.1:10034 4 127.0.0.1:10034
```

```
127.0.0.1:10033 5 127.0.0.1:10034
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors                                // correct dv
127.0.0.1:10031 7
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 7
127.0.0.1:10032 4
Update sent to all neighbors at time 40                     // stabilized
127.0.0.1:10031 7
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
```

b. Removing node after convergence

The same as in (d) but remove node D after convergence.

Expectation

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

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

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

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

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

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

drop D

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

C		A	B	C	D	E	F
2	A	0	2	2	3	7	6
8	B	2	0	4	5	6	4
0	C	2	4	0	5	9	8

E		A	B	C	D	E	F
10	B	2	0	4	5	6	4
0	E	8	6	9	8	0	2
2	F	6	4	7	6	2	0

C updates A,B

E udpates B,F

A		A	B	C	D	E	F
0	A	0	2	2	7	8	6
2	B	2	0	4	5	6	4
2	C	2	4	0	5	9	8

B		A	B	C	D	E	F
2	A	0	2	2	3	7	6
0	B	2	0	4	5	6	4
8	C	2	4	0	5	9	8
10	E	8	6	9	8	0	2
4	F	6	4	7	6	2	0

F		A	B	C	D	E	F
4	B	2	0	4	5	6	4
2	E	8	6	9	8	0	2
0	F	6	4	8	9	2	0

A updates B,C

F updates E,B

B		A	B	C	D	E	F
2	A	0	2	2	7	8	6
0	B	2	0	4	9	6	4
8	C	2	4	0	5	9	8
10	E	8	6	9	8	0	2
4	F	6	4	8	9	2	0

C		A	B	C	D	E	F
2	A	0	2	2	7	8	6
8	B	2	0	4	5	6	4
0	C	2	4	0	9	10	8

E		A	B	C	D	E	F
10	B	2	0	4	5	6	4
0	E	8	6	10	11	0	2
2	F	6	4	8	9	2	0

B updates A,C,E,F

C updates A,B

E udpates B,F

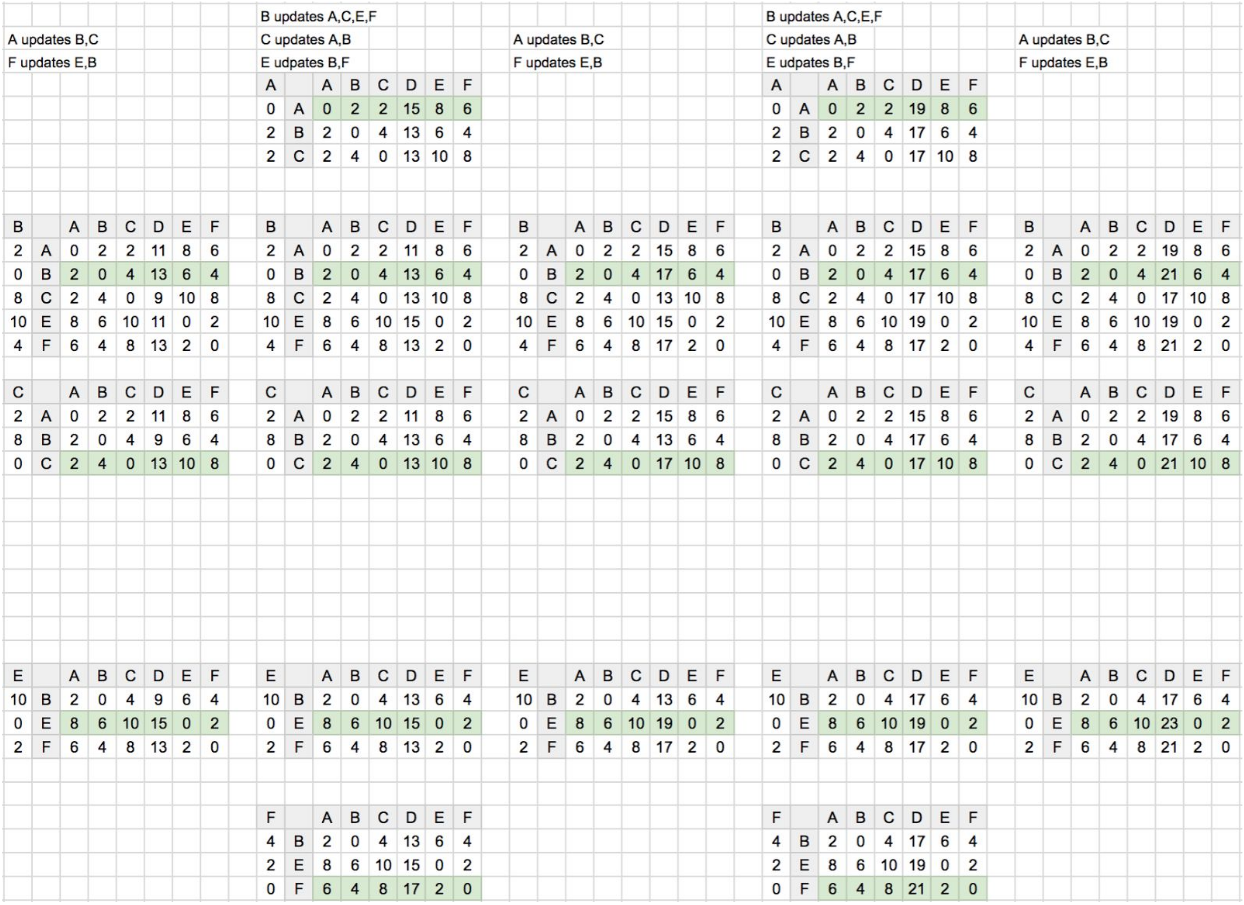
A		A	B	C	D	E	F
0	A	0	2	2	11	8	6
2	B	2	0	4	9	6	4
2	C	2	4	0	9	10	8

B		A	B	C	D	E	F
2	A	0	2	2	7	8	6
0	B	2	0	4	9	6	4
8	C	2	4	0	9	10	8
10	E	8	6	10	11	0	2
4	F	6	4	8	9	2	0

C		A	B	C	D	E	F
2	A	0	2	2	7	8	6
8	B	2	0	4	9	6	4
0	C	2	4	0	9	10	8

E		A	B	C	D	E	F
10	B	2	0	4	9	6	4
0	E	8	6	10	11	0	2
2	F	6	4	8	9	2	0

F		A	B	C	D	E	F
4	B	2	0	4	9	6	4
2	E	8	6	10	11	0	2
0	F	6	4	8	13	2	0



A updates B,C
F updates E,B

B		A	B	C	D	E	F
2	A	0	2	2	11	8	6
0	B	2	0	4	13	6	4
8	C	2	4	0	9	10	8
10	E	8	6	10	11	0	2
4	F	6	4	8	13	2	0

C		A	B	C	D	E	F
2	A	0	2	2	11	8	6
8	B	2	0	4	9	6	4
0	C	2	4	0	13	10	8

E		A	B	C	D	E	F
10	B	2	0	4	9	6	4
0	E	8	6	10	15	0	2
2	F	6	4	8	13	2	0

B updates A,C,E,F
C updates A,B
E udpates B,F

A		A	B	C	D	E	F
0	A	0	2	2	15	8	6
2	B	2	0	4	13	6	4
2	C	2	4	0	13	10	8

B		A	B	C	D	E	F
2	A	0	2	2	11	8	6
0	B	2	0	4	13	6	4
8	C	2	4	0	13	10	8
10	E	8	6	10	15	0	2
4	F	6	4	8	13	2	0

C		A	B	C	D	E	F
2	A	0	2	2	11	8	6
8	B	2	0	4	13	6	4
0	C	2	4	0	13	10	8

E		A	B	C	D	E	F
10	B	2	0	4	13	6	4
0	E	8	6	10	15	0	2
2	F	6	4	8	13	2	0

F		A	B	C	D	E	F
4	B	2	0	4	13	6	4
2	E	8	6	10	15	0	2
0	F	6	4	8	17	2	0

A updates B,C
F updates E,B

B		A	B	C	D	E	F
2	A	0	2	2	15	8	6
0	B	2	0	4	17	6	4
8	C	2	4	0	13	10	8
10	E	8	6	10	15	0	2
4	F	6	4	8	17	2	0

C		A	B	C	D	E	F
2	A	0	2	2	15	8	6
8	B	2	0	4	13	6	4
0	C	2	4	0	17	10	8

E		A	B	C	D	E	F
10	B	2	0	4	13	6	4
0	E	8	6	10	19	0	2
2	F	6	4	8	17	2	0

B updates A,C,E,F
C updates A,B
E udpates B,F

A		A	B	C	D	E	F
0	A	0	2	2	19	8	6
2	B	2	0	4	17	6	4
2	C	2	4	0	17	10	8

B		A	B	C	D	E	F
2	A	0	2	2	15	8	6
0	B	2	0	4	17	6	4
8	C	2	4	0	17	10	8
10	E	8	6	10	19	0	2
4	F	6	4	8	17	2	0

C		A	B	C	D	E	F
2	A	0	2	2	15	8	6
8	B	2	0	4	17	6	4
0	C	2	4	0	17	10	8

E		A	B	C	D	E	F
10	B	2	0	4	17	6	4
0	E	8	6	10	19	0	2
2	F	6	4	8	17	2	0

F		A	B	C	D	E	F
4	B	2	0	4	17	6	4
2	E	8	6	10	19	0	2
0	F	6	4	8	21	2	0

A updates B,C
F updates E,B

B		A	B	C	D	E	F
2	A	0	2	2	19	8	6
0	B	2	0	4	21	6	4
8	C	2	4	0	17	10	8
10	E	8	6	10	19	0	2
4	F	6	4	8	21	2	0

C		A	B	C	D	E	F
2	A	0	2	2	19	8	6
8	B	2	0	4	17	6	4
0	C	2	4	0	21	10	8

E		A	B	C	D	E	F
10	B	2	0	4	17	6	4
0	E	8	6	10	23	0	2
2	F	6	4	8	21	2	0

Results

Router E's output

```
run:
new dv calculated:
127.0.0.1:10032 10 127.0.0.1:10032
Update sent to all neighbors
127.0.0.1:10032 10
new dv calculated:
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10032 10 127.0.0.1:10032
Update sent to all neighbors
127.0.0.1:10034 4
127.0.0.1:10032 10
new dv calculated:
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10032 10 127.0.0.1:10032
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10034 4
```

```
127.0.0.1:10032 10
127.0.0.1:10036 2
Update sent to all neighbors at time 20
127.0.0.1:10034 4
127.0.0.1:10032 10
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10035 2
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10034 4
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv calculated:
127.0.0.1:10031 12 127.0.0.1:10032
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10033 14 127.0.0.1:10032
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 12
127.0.0.1:10034 4
127.0.0.1:10033 14
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10035 2
127.0.0.1:10032 4
new dv received from 127.0.0.1:10036
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10032 4
new dv received from 127.0.0.1:10036
```

```
127.0.0.1:10031 14
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 16
127.0.0.1:10032 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 4
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10034
127.0.0.1:10031 3
127.0.0.1:10035 4
127.0.0.1:10033 1
127.0.0.1:10032 5
127.0.0.1:10036 6
new dv calculated:
127.0.0.1:10031 7 127.0.0.1:10034
127.0.0.1:10034 4 127.0.0.1:10034
127.0.0.1:10033 5 127.0.0.1:10034
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors                              // correct dv
127.0.0.1:10031 7
```

```
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 7
127.0.0.1:10032 4
Update sent to all neighbors at time 40                    // stabilized
127.0.0.1:10031 7
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 7
127.0.0.1:10032 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv received from 127.0.0.1:10034
127.0.0.1:10031 3
127.0.0.1:10035 4
127.0.0.1:10033 1
127.0.0.1:10032 5
127.0.0.1:10036 6
Update sent to all neighbors at time 60
127.0.0.1:10031 7
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
```

```
127.0.0.1:10034 6
127.0.0.1:10033 7
127.0.0.1:10032 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
Update sent to all neighbors at time 80
127.0.0.1:10031 7
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 7
127.0.0.1:10032 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
127.0.0.1:10033 4
127.0.0.1:10036 4
Update sent to all neighbors at time 100
127.0.0.1:10031 7
127.0.0.1:10034 4
127.0.0.1:10033 5
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 6
127.0.0.1:10033 7
127.0.0.1:10032 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 5
```

```
127.0.0.1:10033 4
127.0.0.1:10036 4
neighbor 127.0.0.1:10034 dropped                 // drop router D
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 9
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 8 127.0.0.1:10036
127.0.0.1:10033 9 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036                // dv = (8,6,9,8,0,2)
Update sent to all neighbors                     // start counting to infinity
127.0.0.1:10031 8
127.0.0.1:10034 8
127.0.0.1:10033 9
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 9
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 11 127.0.0.1:10036
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036                // dv = (8,6,10,11,0,2)
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 11
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 10
```

```
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 12 127.0.0.1:10036
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 12
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 13
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 15 127.0.0.1:10036
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036                    // dv = (8,6,10,15,0,2)
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 15
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 10
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 9
127.0.0.1:10033 4
```

```
127.0.0.1:10036 4
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 14
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 16 127.0.0.1:10036
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 16
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
new dv received from 127.0.0.1:10032
127.0.0.1:10031 2
127.0.0.1:10035 6
127.0.0.1:10034 13
127.0.0.1:10033 4
127.0.0.1:10036 4
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 13
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 15 127.0.0.1:10036
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 15
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
```

```
new dv received from 127.0.0.1:10036
127.0.0.1:10031 6
127.0.0.1:10035 2
127.0.0.1:10034 17
127.0.0.1:10033 8
127.0.0.1:10032 4
new dv calculated:
127.0.0.1:10031 8 127.0.0.1:10036
127.0.0.1:10034 19 127.0.0.1:10036
127.0.0.1:10033 10 127.0.0.1:10036
127.0.0.1:10032 6 127.0.0.1:10036
127.0.0.1:10036 2 127.0.0.1:10036                // dv = (8,6,10,19,0,2)
Update sent to all neighbors
127.0.0.1:10031 8
127.0.0.1:10034 19
127.0.0.1:10033 10
127.0.0.1:10032 6
127.0.0.1:10036 2
```

IV. Conclusion

In conclusion, we have demonstrated the functionalities of our DV-routing simulation, accomplishing all the goals stated above. These features include updating distance vector and forwarding tables upon network changes, automatically advertising distance vector to neighbors, dropping inactive neighbors, forwarding messages to correct destination, accepting weight change and adjusting the network accordingly, and supporting poison reverse as an option.

In addition, we observed that DV networking can stabilize very swiftly, if not instantly, and function well with poison reverse. However, without poison reverse, at the event of a dropped router, the network cannot converge and thus suffers tremendously due to count-to-infinity. Through this simulation, we also see the benefits of advertising distance vector automatically. The feature is not only to keep the network update-to-date and maintain its integrity, but also, equally important, to detect if certain neighbors are inactive, or dropped, through the absence of their DV update.

V. References

[1] A. Sadovnik, “Programming Assignment 3: Simulating the distance vector algorithm,” Apr-2017. [Online]. Available: https://docs.google.com/document/u/1/d/1WzJRXv2V1GlNUbtJEUpdIr0PKIXOmBCSpVagW2gBq1o/pub. [Accessed: 06-May-2017]

[2] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach*.

Addison-Wesley Longman, 2013.