# Griffin Fujioka #11044124 CptS 455 - Project 2 (Routers)

# Comments

**The routing problem** involves around finding the least-cost path between routers in a network using an iterative, distributed and asynchronous algorithm.

**The solution implemented in router.c** implements the iterative, distributed and asynchronous algorithm touched on above. The program is implemented on multiple, communicating routers emulated by processes. Each router instance begins by reading the configuration of its local links from file at the beginning of the run; these files were provided by the professor (the professor also supplied C files readrouters.c and readrouters.h to read the router instance's configuration of its local links and constructs a data structure representation for us. Furthermore, the professor provided handy functions createSocket() and createConnection() which constructed the connected datagram sockets for talking to neighbors and constructed an array of (neighbor,socket) pairs. When an update message was received, we were able to determine who the sender was based on which socket the message came from.

Each router instance initializes its routing table with entries for all directly-connected neighbors. Non-neighbors are added to the routing table dynamically. Each instance of the router learns better paths to other routers by receiving update messages from its neighboring routers by sending datagram messages that look like:

U d cost\0

as defined in the assignment where U indicates that the message is an update message, d represents the destination and cost is the cost from the sending router to the destination router. Since the cost messages are expressed as a sequence of ASCII decimal digits, y implementation creates null-terminated U messages to handle cost messages with multiple digits. A simple loop ensured that multiple digit costs were accommodated. Upon receiving a U message, the router searches for the destination in its routing table and inserts it if it is not there with a cost of infinity.

Since the U message contains the cost of getting from the sending router to the destination router, the cost of getting from this specific router instance to the destination router includes the cost of the link from this specific router to the sending router.

The implementation also includes an unconnected datagram socket bound to the local baseport to receive L and P messages (but not U messages - those come only from neighbors). L messages are link-cost changes and look like:

L n cost

as defined in the assignment where n is a neighbor and cost is a new cost of reaching that neighbor directly. Upon receiving an L message, the router updates the link info for the specific neighbor.

Upon receiving either a U or L message the router makes appropriate changes to its routing table and if there are changes, they are sent to the router's neighbors using U messages; this is the **triggered update** portion of the assignment. A standard output message is also printed which looks like:

Received U message: U D 9

Router A making change:

Destination: D

Cost: 10

Next hop: B

Furthermore, in addition to the triggered updates, each router sends U messages for all routing table entries to each of its neighbors every 30 seconds.

Finally, the router uses the unconnected socket to receive P messages which look like:

P d

after which it prints its routing table.

**Test cases:**

* **Test case 1:** test2

This test case involves routers A through G.

Router A:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| B | 1 | B |
| C | 3 | B |
| G | 5 | B |
| D | 5 | C |
| H | 3 | B |
| E | 7 | C |

Router B:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| G | 4 | G |
| D | 3 | C |
| C | 2 | C |
| H | 2 | H |
| A | 1 | A |
| F | 10 | G |
| E | 5 | C |

Router C:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| D | 1 | D |
| E | 2 | D |
| B | 2 | B |
| A | 3 | B |
| H | 4 | B |
| F | 4 | E |

Router D:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| F | 2 | E |
| G | 1 | G |
| E | 1 | E |
| B | 3 | C |
| H | 11 | B |
| A | 5 | C |

Router E:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| F | 1 | F |
| C | 2 | D |
| D | 1 | D |
| G | 7 | F |
| B | 5 | C |
| A | 7 | C |

Router F:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| G | 4 | D |
| D | 2 | E |
| E | 1 | E |
| B | 10 | G |
| C | 4 | E |
| H | 20 | G |

Router G:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| F | 6 | F |
| D | 1 | D |
| B | 4 | B |
| H | 6 | B |
| E | 2 | D |
| A | 5 | B |
| C | 6 | B |

Router H:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| B | 2 | B |
| G | 6 | B |
| D | 11 | B |
| C | 4 | B |
| A | 3 | B |
| F | 20 | G |

When the link cost BE changes to 2, router B's routing table updates the least cost path to E to a cost of 2 using router C as the next hop. Router E's routing table updates the least cost path to G to a cost of 2 using D as next hop. Router G updates its routing table.

* **Test case 2:** test4

Router A:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| B | 1 | B |
| C | 3 | B |
| D | 5 | C |
| E | 7 | C |
| G | 5 | B |
| H | 3 | B |

Router B:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| G | 4 | G |
| D | 3 | C |
| C | 2 | C |
| H | 2 | H |
| A | 1 | A |

Router C:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| D | 1 | D |
| E | 3 | E |
| B | 2 | B |
| A | 3 | B |
| G | 6 | B |
| H | 4 | E |
| F | 4 | E |

Router D:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| F | 2 | E |
| G | 1 | G |
| E | 1 | E |
| B | 3 | C |
| H | 11 | B |
| A | 5 | C |

Router E:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| F | 1 | F |
| C | 3 | C |
| D | 1 | D |
| B | 5 | C |
| A | 7 | C |
| G | 7 | F |

Router F:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| G | 6 | G |
| D | 2 | E |
| E | 1 | E |
| C | 4 | E |

Router G:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| F | 6 | F |
| D | 1 | D |
| B | 4 | B |
| H | 6 | B |
| E | 7 | F |
| C | 6 | B |
| A | 5 | B |

Router H:

|  |  |  |
| --- | --- | --- |
| **Destination** | **Cost** | **Next hop** |
| B | 2 | B |
| G | 6 | B |
| D | 11 | B |
| C | 4 | B |
| A | 3 | B |

When the link cost BE changes to 2, router B's routing table updates the least cost path to E to a cost of 2 and a next hop of E. F is added to B's routing table with a cost of 12. Router C updates its routing table entry for Router E to a cost of 2 with next hop of E. Router C also changes the cost to G to a value of 6 with a next hop of B. More changes may occur but if you want to see them, you can run the program.

**Additional experiments**

* changelink test2 A B 5: causes similar updates as demonstrated above
* printtables test2 A: prints A's routing table
* printtables test2: prints the routing table for all routers in test2
* printtables test4: prints the routing table for all routers in test4

**Decisions that have been made that could significantly affect the behavior of this distributed routing algorithm**

* Adjust select() timeout period to a smaller number which would rapidly send updates to neighboring routers
* routers do not need to worry about new links being added but in practice they likely would be
* Implement parallel programming algorithm such that we divide resources to receive data from neighboring routers across a number of different sockets
* Optimize updating algorithm by improving data storage design

**Discrepancies between the behavior observed in the program and the behavior of the distance vector routing algorithm described in the textbook**

* This implementation does not implement poison reverse

**Conclusion**

This implementation of the distance-vector algorithm serves as a very good model and may even scale well, at least algorithmically. Some assumptions we made which simplify the implementation include:

* The fact that we don't have to worry about new links being added
* We did not implement poison reverse
* No new routers are added to the network
* We assume that we'll always know the maximum number of routers in system

Some aspects of the environment which are different and might have a significant influence on the performance of a real router are:

* Physical connection medium: Obviously very fast for two processes on a machine, but how about robust program separated by hundreds or thousands of miles?
* Number of routers in the system: We limited the size of our system to 8, but a real network could have millions of routers which means we'd be receiving and updating our table at an unfathomable rate
* Storage device for router information: In this implementation we're maintaining necessary information locally in modest data structures. A router might be connected to an external database which would scale better but would add data transfer time

I spent an estimated 35 total hours on this project, but it was a very good learning experience.