Comparison of Network Routing Algorithms

Jodi A. Hieronymus

University of Nevada, Reno

CPE 400: Computer Communication Networks

Instructor Dr. Shamik Sengupta

December 12, 2022

Abstract

The most important part of networking is being able to reliably route a packet from one device to the next in an efficient manner. Routing algorithms are network-layer operations that consider different possible paths when taking a packet to its destination. Good algorithms are able to successfully deliver a packet to its intended destination while minimizing the cost of the path taken. Additionally, algorithms must balance time complexity and reliability. A naive approach is to simply take the link that has the lowest cost at each step. Dijkstra's Algorithm and this naive approach both have their own strengths and weaknesses that should be considered in comparison to the network.

Functionality

One goal of this project was to create a simple representation of real network concepts. Three layers of the network stack were explored: the application layer and link layer briefly, with greater emphasis on the network layer. Because networks in reality are incredibly complex, many simplifications were made in order to decrease the scope of this project. Many of the simplifications made will be discussed in this section.

The Application Layer

In the Packet class, the application layer is represented by a simple string. The string is set when the packet is created and retrieved when the packet successfully reaches its destination. The string is then printed to the terminal.

The Network Layer

The network layer is the layer of the network stack that was most explored in this project. The network itself is created by connecting individual Routers in a directed graph. For simplification, the graph only moves one way, and there is no way for a packet to be sent the opposite direction of its destination. Each link between Routers has an associated cost that is used in routing calculations. The network is represented in two ways: in lists of Routers and in a 2-dimensional array. Figure 1 shows a visual representation of the network used.

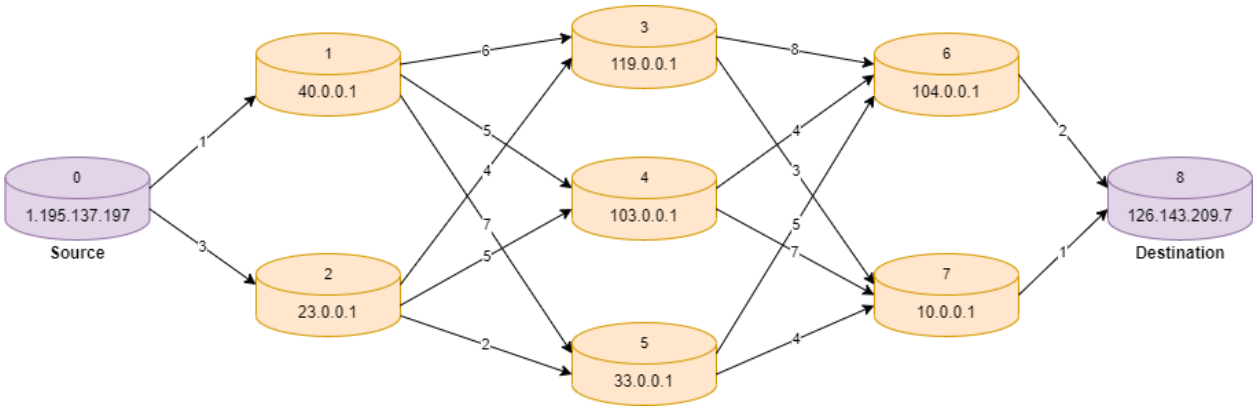

Figure 1: The Network

The IPv4Address class is used to represent the source and destination IP addresses of the Packet. The IPv4Address class only supports IPv4 addresses of class A and performs checks in the constructor to ensure proper class A addresses are used.

Another class, the Router, further implements features of the network layer. Each Router saves its network ID which should match the first byte of the class A IPv4 address. When routing a packet, each Router compares its saved network ID to the destination address in the packet to see if the destination is in-network before performing actions to pass the packet to the next router in the sequence. Each Router also holds an instance of each of the two routing algorithms implemented: A naive approach and Dijkstra's Algorithm. Both methods provide the time taken to take a router from source router to destination router.

Naive Approach

When routing using the naive approach, each router examines the costs associated with each of its immediate neighbors and chooses the lowest cost link. It

does not consider costs further down the line, instead opting for what appears to be the most immediate best choice.

The naiive approach only requires knowledge of routers that are directly connected to the router making the calculation, and therefore requires little space. This information is stored in each router using lists of routers. Each router along the packet path runs the algorithm itself.

Dijkstra's Algorithm

Dijkstra's Algorithm is a well-known and realistically used algorithm in networking. The router that runs Dijkstra's calculates the shortest path to every router in the network, saving the results so they can be reused later.

Dijkstra's Algorithm requires knowledge of every router in the network, and therefore requires more memory than the naive approach. This information is stored in each router using a 2-dimensional array of costs associated with each router. In exchange, the calculations are only run once and then reused, saving time in future packet routing.

The Link Layer

Finally, the link layer is loosely explored. Routers are titled so because of the emphasis on network layer routing algorithms in this project, however, the Router class has some functionality similar to that of link layer switches. Namely, each router is assigned a MACAddress that represents the index of the router in both the list

representation and the graph representation of the network. This is an extremely simple representation of actual MAC addresses, which are actually 48 bits long.

Results

Each algorithm was run with two tests taking a packet from Router 0 to Router 8. In one test, all routers are online and functioning as intended. In the second test, a router that was along the path in the first test's path goes offline after the first route sequence. Each test was run 15 times each.

Naive Approach

The path chosen by the naive approach when all routers are active is shown in Figure 2 below. The total cost of the path is 12.

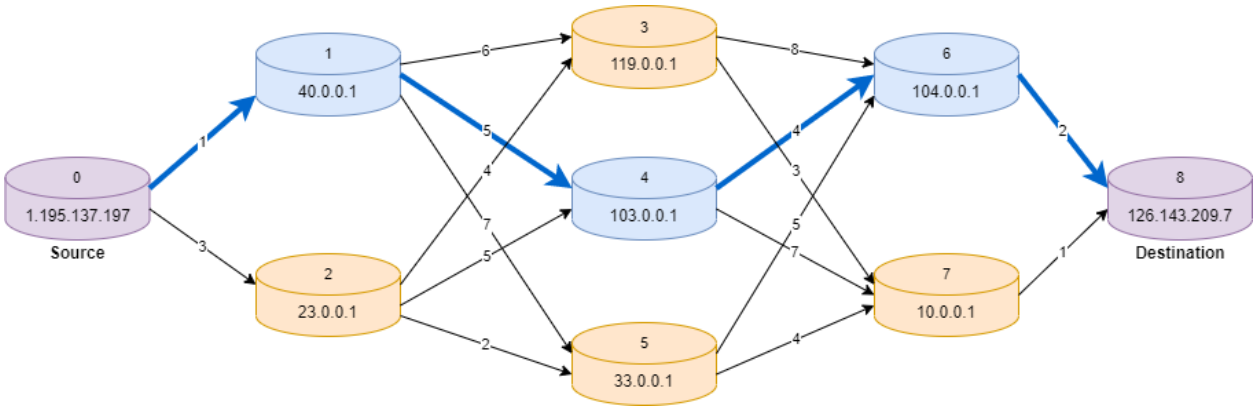

Figure 2: The path chosen by the naive approach when all routers are active.

The path chosen by the naive approach when router 4 is inactive is shown in Figure 3 below. The total cost of the path is 11.

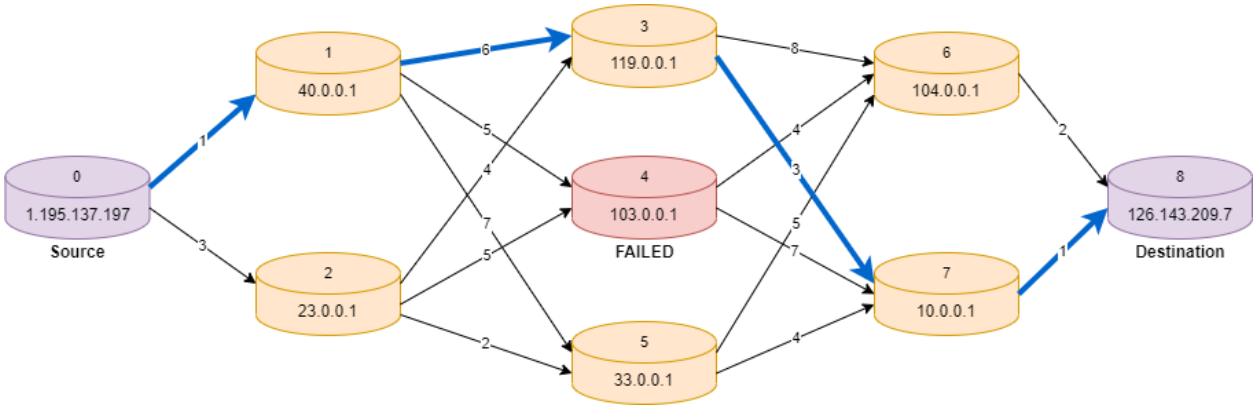

Figure 3: The path chosen by the naive approach when router 4 is inactive.

The results of the tests for the naive approach are shown in Table 1 below.

	All Routers Active		**Router 4 Failed**	
Test No.	*Path Cost*	*Time Taken*	*Path Cost*	*Time Taken*
1	12	7706700ns	12	10367600ns
2	12	7174400ns	11	5306400ns
3	12	12286200ns	11	4324300ns
4	12	10114400ns	11	3959000ns
5	12	7141700ns	11	6893500ns
6	12	7270700ns	11	10578000ns
7	12	5620100ns	11	13732400ns
8	12	4514600ns	11	7034800ns
9	12	5379200ns	11	3883500ns
10	12	4167500ns	11	5514600ns
11	12	4094600ns	11	12405600ns

	All Routers Active		Router 4 Failed	
Test No.	*Path Cost*	*Time Taken*	*Path Cost*	*Time Taken*
12	12	11522600ns	11	13554400ns
13	12	11756300ns	11	12162000ns
14	12	4904200ns	11	10011100ns
15	12	10855000ns	11	12344100ns

Table 1: The results for routing a packet using the naive approach are detailed.

When all routers were active, the naive approach routed the packet in an average of 7633880 ns, or 7.634 ms, with a path cost of 12. When router 4 failed, the packet was delivered in an average of 8693121 ns, or 8.693 ms, with an average path cost of 11.07.

Dijkstra's Algorithm

The path chosen by Dijkstra's Algorithm when all routers are active is shown in Figure 4 below. The total cost of the path is 10.

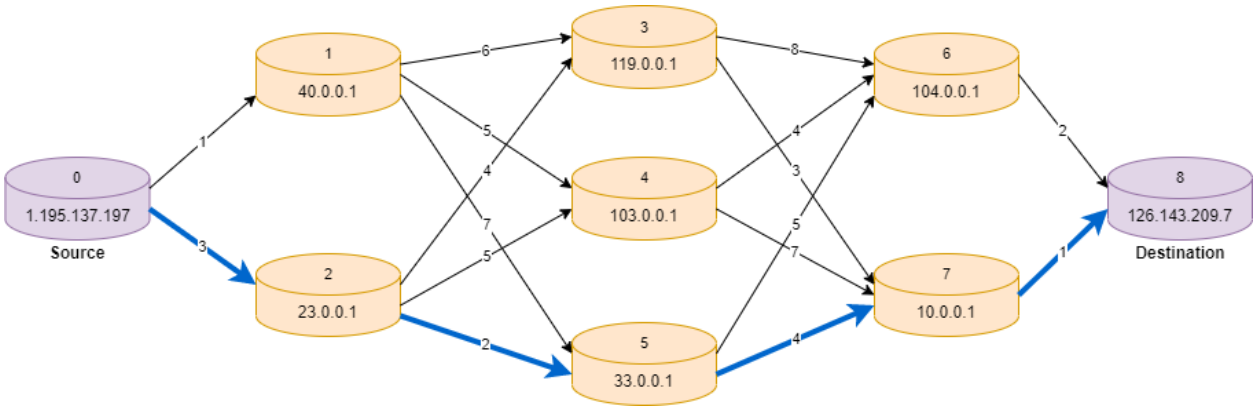

Figure 4: The path chosen by Dijkstra's Algorithm when all routers are active.

The path chosen by Dijkstra's Algorithm when router 5 is inactive is shown in Figure 5 below. The packet is lost when it is passed to router 5.

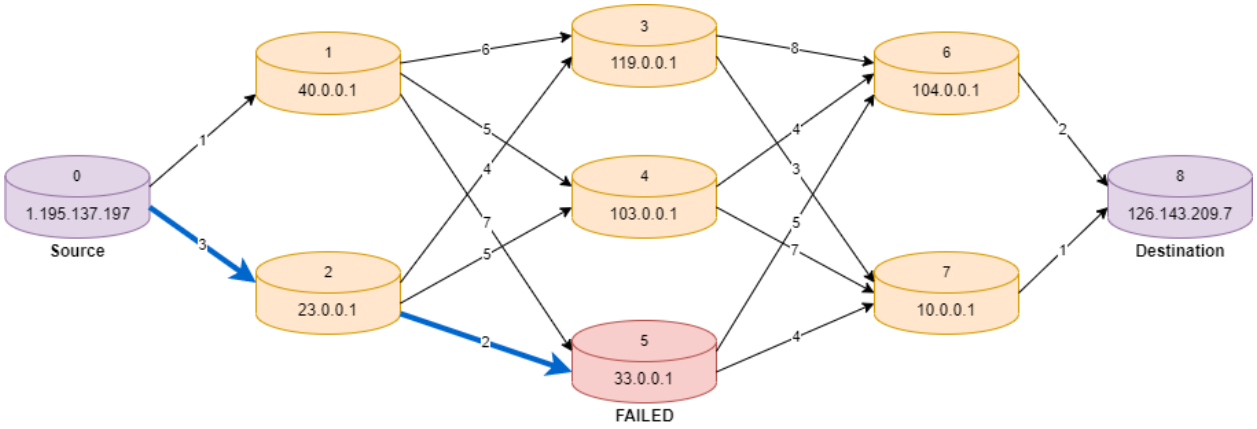

Figure 5: The path chosen by Dijkstra's Algorithm when router 5 is inactive.

The results of the tests for Dijkstra's Algorithm are shown in Table 2 below.

	All Routers Active		**Router 5 Failed**	
Test No.	*Path Cost*	*Time Taken*	*Path Cost*	*Time Taken*
1	10	73949200ns	10	3477600ns
2	10	1925200ns	Failed	Infinite
3	10	2009300ns	Failed	Infinite
4	10	2453100ns	Failed	Infinite
5	10	4167300ns	Failed	Infinite
6	10	5670700ns	Failed	Infinite
7	10	3536400ns	Failed	Infinite
8	10	5684900ns	Failed	Infinite
9	10	4540600ns	Failed	Infinite
10	10	3018200ns	Failed	Infinite
11	10	4988900ns	Failed	Infinite

	All Routers Active		Router 5 Failed	
Test No.	*Path Cost*	*Time Taken*	*Path Cost*	*Time Taken*
12	10	4614500ns	Failed	Infinite
13	10	3331700ns	Failed	Infinite
14	10	2760500ns	Failed	Infinite
15	10	3887400ns	Failed	Infinite

Table 2: The results for routing a packet using Dijkstra's Algorithm are detailed.

When all routers were active, Dijkstra started with routing the packet in 73949200ns, or 73.949 ms. There was a decrease in time routing the packets after the initial calculations were completed, with an average of 3756336 ns or 3.756 ms. This is 49.2% the time taken by the naive algorithm. However, when router number 5 failed, Dijkstra's Algorithm did not recalculate and instead kept using the same path, resulting in the packet not being delivered.

Analysis

Ultimately, the moment when the path was calculated for the packet delivery significantly affected the performance of each algorithm. Dijkstra's Algorithm only calculated the path once the first time the router was prompted to route the packet, resulting in large overhead but extremely short time taken in future packet routing. Additionally, Dijkstra's Algorithm successfully found the shortest possible path in the network, saving additional time and resources. However, Dijkstra's Algorithm was unusable in cases where a router along the path was inactive. Because the calculations were performed once at the very beginning, the router was falsely believed to be successful in future routes, and therefore the packet was lost.

In comparison, the naive approach took much longer than Dijkstra's Algorithm, taking over twice as long on average. Additionally, the path taken was not the shortest path possible, resulting in wasted time and resources. However, by calculating the next step at each router, the naive approach was still able to deliver the packet when a router failed. Both methods have clear weaknesses that could use improvement in order to improve reliability.

Improving the Naive Approach

The naive approach is extremely simple. One issue is that it does not guarantee that the chosen router will even be closer to the destination router at all. This is not a problem in this simulation since any path on the directed graph will always go towards the destination router. However, one significant improvement- or requirement in most cases- would be to follow down possible paths to ensure the destination router is at the end. A possible way to do this is explored but unused in the naive approach class. This would increase the amount of time taken to route a packet, but would also significantly improve the reliability of the approach. To lessen the cost to route a packet, the naive approach could also look farther down the path to see which path is shorter overall.

Improving Dijkstra's Algorithm

In the current implementation, Dijkstra's never recalculates the paths if the algorithm has already been run. Instead, Dijkstra's could recalculate paths each time a packet is lost. This could be accomplished by recalculating all paths. A more time-efficient way could be to only recalculate paths that include any routers along the route that failed.

Other Options

In the end, rather than improving the algorithms explored here, other approaches may be more appropriate for certain networks, like Bellman-Ford Algorithm. The naive approach in its current form is not reliable in most networks and should not be used in reality. Dijkstra's Algorithm is very beneficial given a network that has routers that are highly dependable and very rarely fail. However, networks with routers that fail often should choose a separate method than either discussed here.

References

Books

James F. Kurose and Keith W. Ross, "*Computer Networking: A Top-Down Approach*", Addison-Wesley, 8th edition.

Web

Geeks for Geeks, "*Dijkstra's Shortest Path Algorithm*". Last updated 31 Aug, 2022. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Geeks for Geeks, "*Printing Paths in Dijkstra's Shortest Path Algorithm*". Last updated 22 Aug, 2022. https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/