PROJECT		
COURSE:	EEE4121F	MOBILE AND WIRELESS NETWORKS
MODULE:	B	
AUTHOR:	NATHANAEL THOMAS	
STD NUM:	THMNAT011	
DUE DATE:	29 APRIL 2022	

Contents

Introduction

Service Requirements

The protocol will focus on telemetry. Telemetry can be defined as a collection of information from a variety of sources, this information will be transmitted to another for analysis. The collection of the data source is done through communication processes that have been automated. [1]

Telemetry within networks acts as a monitoring technology when data is collected when data is sent from devices. By monitoring the packets in the network through the process of telemetry more can be said about the network structure.

One of the main protocols presents in the telemetry as a monitoring process is traceroute. Traceroute as the name suggests can trace the routes present in the network. What it does is that it will continuously send packets throughout the network decrementing the time to live once the time to live is 1 the original router will send an ICMP message. What traceroute does is that it can tell you how many hops it takes to get from the source to the destination.

But this traceroute protocol does not tell us much about the network architecture. But rather how many hops it takes to send the packet from the source to the destination. One of the major downfalls of traceroute is that it doesn't provide the past data which makes the process of finding patterns a difficult one. [2] The traceroute protocol is also a very inefficient method of tracking as if the TTL reaches 0 before reaching the destination the packet will then be dropped. Therefore many packets need to be sent through the next work with different TTL to find the path to get from the source to the destination and this can be seen as a waste of resources in the network. One improvement that can be done in editing the header field of the packets is to provide more information about the router and the network. What this protocol can do for the end-user is provide more information about the specified network in which it is located.

Protocol Specification

To make this protocol more unique to the traceroute protocol the following features will be added to the already existing traceroute algorithm, the first being the IP address of the router it has just hopped to and the time it takes to reach the router. This will provide a little more information than before about the network. It will also help in checking for issues in the network and patterns more easily identifiable. The communication between parties will be that the packet will check if the router it is at is the

destination if it isn't the header of the packet will have an additional field holding the IP address of the router or switch it is currently at. This will then be sent back to the sender where the field can be stored. Thus being able to create a traceback pattern of the packet from the source. Then the shortest path from the source to the destination can be identified meaning at further sending of packets the shortest path to the destination can be taken thus saving network sources.

Protocol Requirements

The packets should contain the source and destination IP addresses they should also contain a field of time to live. Then it should also contain a field of the *time to reach* this will check how long it takes for the packet to send to the next hop. Lastly, a field called hop will store the switch or router the current hop is on and echoes it back to the sender. As well as a field called next-hop is where the next one with the shortest distance can be referenced and used by the packet to find the shortest path

From the sender's side initially, the packet will only contain the information of the source and the destination Ip addresses as well as a time to live. It should also be able to take the echoed header field of time to reach an IP address and store it somewhere for future use.

When it reaches a node the node should first compare if the IP address of the node is the destination IP. If these do not match then the time to reach the node should be updated and sent back to the sender and the IP address of the node should also be updated and echoed back to the sender.

When the destination node is reached the comparison will be done and confirmed. Thus in subsequent packets transferred the information gathered can be used to determine the shortest path to the destination. When the packet reaches a node it can echo the IP of that node to the sender, the sender can then access the table generated previously and check the IP of the shortest node to the destination.

Designing the protocol

Protocol Specification

For this protocol what needed to be implemented a look into the specification made from the requirements. Firstly what was done was need to determine which programming language will be used to run our program. Many languages were considered such as Java, P4, python and Mininet as implementation platforms. But after consideration of aspects such as complexity, run-time, syntax knowledge, and just general understanding of the programming language. It was concluded that Python will provide the best option for this implementation. This provided me with a strong base set to complete the protocol and easily available resources online made it a great read and assistance too if

any problems were run into. Whereas languages like Java even though well understood provided syntax issues that may prove to take quite a while to fix. Whereas P4 provided a challenge because of the lack of proficiency in the language it would provide a challenge to implement and using mininet provided an issue as limited resources and proficiency proved to make it quite difficult to implement the proposed network.

For this protocol what was needed is that a network topology needed to be created, for this a partial mesh or mesh network would be considered. Within this network, the protocol will build on the traceroute protocol, providing more detailed information about the network not showing how many hops it takes to go from the source host to the destination host. With traceroute, multiple resources are wasted in this method as multiple packets will be sent through the network to find a route from the source to the destination. The improved protocol will look into only sending one packet at first calling it a "*dummy packet*" this packet will be able to find the shortest route in the network. It will also return how long it will take to send a packet one packet from the source to the destination as well as how many switches the packet will have to go to reach the destination. As well as provide information about the source and destination host.

Solution Design

For implementing the requirements set out for the protocol the following step was taken first to find the shortest path in a network an algorithm was needed to execute this. Upon further research, the Dijkstras algorithm would provide the necessary computation to determine the shortest path for the packet to take to reach the destination from the source. As this was a readily and commonly implemented algorithm code was sourced and adapted from the python implementation by Alexey Klochay. As this provided the implementations of graphing (which would emulate how a real-world network would look like) and Dijkstra's algorithm to find the shortest path in the network. So that when packets are sent the least amount of resources are used during this process. [3]

```python
class Graph(object):
    def __init__(self, switches, init_network):
        self.switches = switches
        self.graph = self.network_creation(switches, init_network)
    #function to create the network
    def network_creation(self, switches, init_network):
        graph = {} #creating a dictionary for the network links
        #creating the switches in the dictionary
        for switch in switches:
            graph[switch] = {}
        graph.update(init_network)#updating the dictionary for the grap

        for switch, edges in graph.items():
            for adjacent_switch, value in edges.items():
                if graph[adjacent_switch].get(switch, False) == False:
                    graph[adjacent_switch][switch] = value
        return graph
    #function to get the switches
    def get_switches(self):
        return self.switches
    #getting the edges of the graph
    def get_outgoing_edges(self, switch):
        connections = []
        for out_switch in self.switches:
            if self.graph[switch].get(out_switch, False) != False:
                connections.append(out_switch)
        return connections
    #getting the values for the links
    def value(self, switch1, switch2):
        "Returns the value of an edge between two switches."
        return self.graph[switch1][switch2]
```

```python
def dijkstra_algorithm(graph, start_switch):
    unvisited_switches = list(graph.get_switches())
    shortest_path = {} #this dictionary will save the cost of visiting each switch the graph
    previous_switches = {} #this dictionary will save the shortest known path

    max_value = sys.maxsize
    for switch in unvisited_switches:
        shortest_path[switch] = max_value
    # However, we initialize the starting switch's value with 0
    shortest_path[start_switch] = 0

    # this will run until all the switches have been visited
    while unvisited_switches:
        # Find the link with the smallest cost
        current_min_switch = None
        for switch in unvisited_switches: # go over the switches
            if current_min_switch == None:
                current_min_switch = switch
            elif shortest_path[switch] < shortest_path[current_min_switch]:
                current_min_switch = switch
        # will update the current switch and the cost to the neighboring switch
        neighbors = graph.get_outgoing_edges(current_min_switch)
        for neighbor in neighbors:
            tentative_value = shortest_path[current_min_switch] + graph.value(current_min_switch, neighbor)
            if tentative_value < shortest_path[neighbor]:
                shortest_path[neighbor] = tentative_value
                # update to the lowest cost to the current switch
                previous_switches[neighbor] = current_min_switch
        # Remove the switch that has been visited from the unvisted switch array
        unvisited_switches.remove(current_min_switch)
        print("Following switch connected to", current_min_switch);
    return previous_switches, shortest_path #will return an array of all the visited switches with the lowe
```

Figure 1: Graph class and Dijkstra's algorithm

As shown above, this was the code written for the implementation of a graph class as well as the Dijkstras algorithm. The following functions are available in the *Graph class* firstly def *__init__* this function does is that it will be used to initialize the graph connections, this function work with the function def *network_creation* to create the topology where the functions *get_switches* and *get_outgoing_edges* these two functions are used to get the switches in the network and the edges of the networks and append them to an array. The function *value* what this does is that it gets the values of the links in the network. Lastly one of the key elements of the protocol is Dijkstra's algorithm, what this function will do is compare all the links in the network and compute the shortest path from the source node to the destination node. It will do this by visiting all the nodes in the graph and determining the shortest path by analysing the link weights of the connection between switches and then evaluating which path gives you the shortest path the packet can take to reach the destination host.

As a function was needed to print out the information obtained from executing the algorithm. The following function was created

```python
def print_result(previous_switches, shortest_path, start_switch, target_switch):
    path = []   #initialising an empty array
    switch = target_switch
    ttl = 0 #initialisng the time to live
    while switch != start_switch:
        path.append(switch)
        switch = previous_switches[switch]
        ttl = ttl+1
    # Add the start switch manually
    path.append(start_switch)
    #Joining the shortest path and the cost of the sending the packet from source
    value = format(shortest_path[target_switch])
    pathway = " -> ".join(reversed(path))
    hops = ttl
    return value, pathway, hops
```

Figure 2:Print_results function

What happens in this function is that the following information is obtained and printed out, it gives you the information on the shortest path, the time to live and how much it will cost the network to send the packet from the hosts.

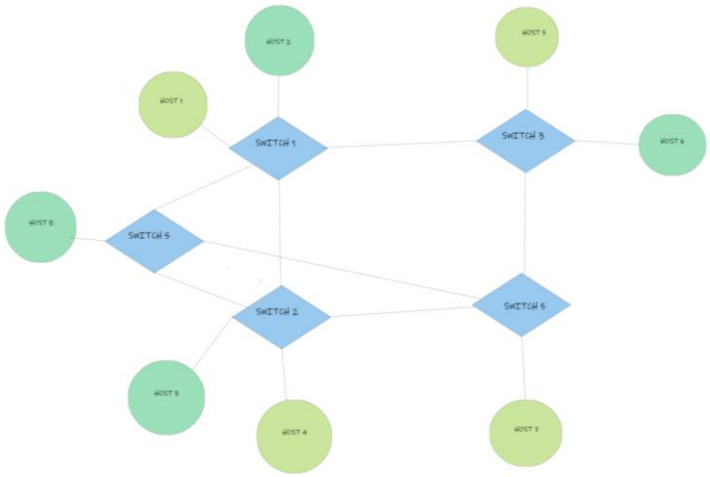

Figure 3: The partial mesh network topology that is being created for this experiment

Now that the network structure has been created, now it was time to emulate how an actual network would perform and behave. In a realistic network, we can never fully know how loaded a router will be and to implement this in Python and allow Dijkstra's algorithm to compute the shortest path we needed to give the links randomly allocated values for the links this was done using the *random.randint().* This function will give the links a random value between 1 and 10.

```
def create_links():
    print("The values for the links are")
    #switches in the network
    switches = ["s1", "s2", "s3", "s4", "s5"]
    init_network = {} #dictionary
    for switch in switches:
        init_network[switch] = {}
    #initialising the links and randomly allocating we
    a = random.randint(1,10)
    print("the value between the s1 and s2 is: ", a)
    init_network["s1"]["s2"] = a
    b = random.randint(1,10)
    print("the value between the s1 and s5 is: ", b)
    init_network["s1"]["s5"] = b
    c = random.randint(1,10)
    print("the value between the s2 and s5 is: ", c)
    init_network["s2"]["s5"] = c
    d = random.randint(1,10)
    print("the value between the s5 and s4 is: ", d)
    init_network["s5"]["s4"] = d
    e = random.randint(1,10)
    print("the value between the s1 and s3 is: ", e)
    init_network["s1"]["s3"] = e
    f = random.randint(1,10)
    print("the value between the s2 and s4 is: ", f)
    init_network["s2"]["s4"] = f
    g = random.randint(1,10)
    print("the value between the s3 and s4 is: ", g)
    init_network["s3"]["s4"] = g
    return switches, init_network
```

Figure 4: This function will randomly allocate the weights to the links as well as show the links between the switches

```
def packets(switch_src, switch_dst):
    begin = time.time() #start time
    time.sleep(random.randint(1,3))
    #Getting the switches and shortest path, value, path taken and hops from the fr
    previous_switches, shortest_path = Graph.dijkstra_algorithm(graph=graph, start_
    value, pathway, hops = Graph.print_result(previous_switches, shortest_path, sta
    end = time.time() #endtime
    duration = end-begin # finding the duration
    print(duration)
    return value, pathway, hops, duration
#function to retrieve and print out the information from the packet
def printPacket(s, s_ip,s_h, d, d_ip, d_h):
    value, pathway, hops, duration = packets(switch_src = s,switch_dst = d)
    print("-----------INFORMATION EXTRACTED FROM PACKET HEADER---------------------
    print("---------------------------PACKET_HEADER-------------------------------
    print("-----------------------------------------------------------------------
    print("SOURCE HOST INFORMATION")
    print("Source Host is : ",s_h)
    print("With IP Address of : ", s_ip)
    print("Connected to Switch : ",s)
    print("-----------------------------------------------------------------------
    print("DESTINATION HOST INFORMATION")
    print("Destination Host is : ",d_h)
    print("With IP Address of : ", d_ip)
    print("Connected to Switch : ",d)
    print("-----------------------------------------------------------------------
    print("The lowest cost to send packets from ",s_h, " to ",d_h," is ",value)
    print ("It to ", hops, "hops from source to destination")
    print("It took ",duration, " seconds to send the packet from source to destinat
    print("The shortest path the from source to destination is ", pathway)
    print("----------------END OF PACKET-------------------------------------------
```

Figure 5: Two of the functions used, def packets will execute the graph and start the timing to implement Dijkstra's algorithm the second function will just retrieve the information from the packet and print it out to the source host

Implementation Results

Through this implementation when sending packets from the source host 2 to destination host 6 the following result was obtained

```
WELCOME TO PACKET SENDING IMPROVED TRACEROUTE PROTOCOL
CREATING THE NETWORK
THE NETWORK HAS BEEN CREATED
Please chose a number between 1-8 for the sources host:
3
Please chose a number between 1-8 for the destination host:
6
NOW SENDING DUMMY PACKET THROUGH THE NETWORK
Finding the shortest route
Shortest route found
PACKET INFORMATION EXTRACTED AND SHOWN BELOW
Following switch connected to s2
Following switch connected to s1
Following switch connected to s5
Following switch connected to s3
Following switch connected to s4
-----------INFORMATION EXTRACTED FROM PACKET HEADER-----------------------
--------------------------PACKET_HEADER---------------------------------
--------------------------------------------------------------------------
SOURCE HOST INFORMATION
Source Host is :  h3
With IP Address of :  10.0.0.0.3
Connected to Switch :  s2
--------------------------------------------------------------------------
DESTINATION HOST INFORMATION
Destination Host is :  h6
With IP Address of :  10.0.0.0.6
Connected to Switch :  s3
--------------------------------------------------------------------------
The lowest cost to send packets from  h3  to  h6  is  2
It to  2 hops from source to destination
It took  2.0092198848724365  seconds to send the packet from source to destination
The shortest path the from source to destination is  s2 -> s1 -> s3
----------------END OF PACKET-----------------------------------------------
PROGRAM DONE
PACKETS RECIEVED BY HOST  h6  FROM HOST h3
GOODBYE!!!!
```

```
The values for the links are
the value between the s1 and s2 is:  1
the value between the s1 and s5 is:  9
the value between the s2 and s5 is:  1
the value between the s5 and s4 is:  8
the value between the s1 and s3 is:  1
the value between the s2 and s4 is:  4
the value between the s3 and s4 is:  5
```

Figure 6: sending packets from host 3 to host 6 and the values of the weights between the links

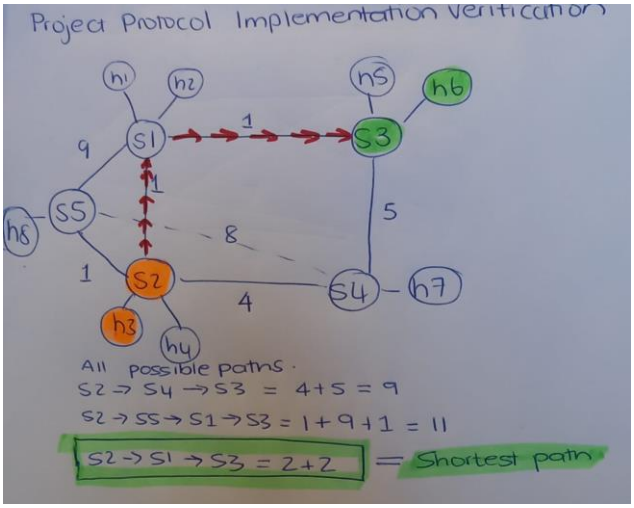

Figure 7: The handwritten solution for verification

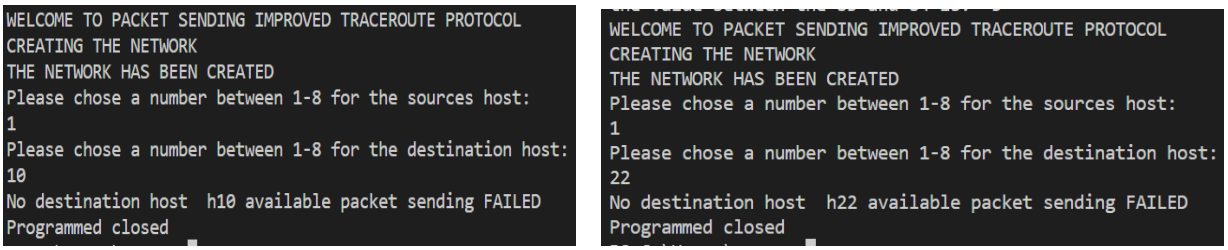

Figure 8: Output for invalid inputs when trying to send to host 10 which is not in the network.

Discussion of the results

For the experimentation of our protocol will input the source host to be h3 and will send packets to the destination host h6. From the results in fig6 the results from our program, the shortest path from host 2 to host 6 will be from s2 → s1 → s3. With the resource required to send packets to the destination will be 2 which is the lowest required when comparing the multiple paths from switch 2 to switch 3. We can confirm that the solution achieved from the report is correct the calculation was done by hand and fig 7 proves that when all the routes from s2 to s3 were taken into consideration the shortest cost path from source to the destination was from s2 → s1 → s3. Evaluating this shows that the algorithms implemented and code does produce the results that were expected. This was done a few more times and the results were the same where the python output corresponded to the hand calculated solution. Also looking at another issue that may arise if a user was trying to send packets from a host that didn't exist in the network as well as if the user tried sending to a host that wasn't present in the network these results were obtained and can be seen in fig 8.

The results obtained above indicate that the protocol was executed as expected, as it provided the user with more information about the network as it indicate the quickest route to send packets from one host to another, and also provided the information previously given by the traceroute the time it takes to find the router as well as the how many hops it took to reach the destination from the source. Doing this would improve the existing traceroute algorithm.

Performance Evaluation

When evaluating the protocol created it could prove to be quite a helpful aspect of the network. By finding the shorted path between hosts the number of network resources can be greatly reduced. This is achieved by reducing the number of packets that are required to find a path from the host to the destination. The protocol designed will only require one packet which will be called the *dummy* packet which can access the network whereas the traceroute protocol will send numerous packets till the ICMP message returns a time-to-live of 1 meaning it has not been timed out. But in a very large network, the

packet number could be high meaning a large number of resources are being wasted. Due to the time constraints set out in this implementation, not a lot more could be tested to make it more useful would be storing the path obtained in memory so that when more packets are sent through the network the host can look at this path and see if the host it indents to send packets to is present in its lookup table it can just send the packets through that path, which means more resources can be saved. But to implement this in a more realistic emulation like mininet and P4, more research needs to be done and a further understanding needs to be gained in the fields. As with the time and knowledge now is not merely enough to implement it in this way. Something else that can make this more useful will be giving a complete analysis of the switches/router in the network, and information like how congested is the router/switch. Information of how long the packet remained in the buffer. These pieces of information can prove to be vital in the analysis of the network. As it can help service providers in identifying where there is heavy congestion in the network and help ease the stress on the switch/router

Conclusion

To conclude the overall design of the protocol has worked as expected as it provided the user with valuable information about the network. The protocol has also met the requirements set out. So the results are satisfactory for the python implementation. But further time and research would be required to implement this protocol on more realistic networks using controllers etc. But with the knowledge and resources available a viable solution has been made and from the result obtained it is clear that the algorithm developed can determine the lowest cost path to send a packet from a source to a destination. As well as how long it took the packet to go from one host to the other and how many hops it took this proves to be very useful information for network providers as well as the user. By doing this initial computation the network resources can be used more efficiently. This research into this topic proves to be quite useful in the context of networks as service providers are trying to make networks more efficient. This increase in efficiency can be maintained and done through proper utilization of resources available in the network. Overall the implementation was successful and a further understanding was gained in the implementation of the traceroute. How implementing different protocols in a network can increase the efficiency of the network.

References

[1] A. Mallick, "Network Telemetry: Why Telemetry in Networking Just Makes Sense," Pluribus Networks, 28 September 2016. [Online]. Available: https://pluribusnetworks.com/blog/network-telemetry-just-makes-sense/. [Accessed 28 04 2022].

[2] "What Is Traceroute and How Does It Work? | N-able," N-able, [Online]. Available: https://www.n-able.com/blog/what-is-traceroute-how-does-it-work. [Accessed 28 4 2022].

[3] A. Klochay, "Implementing Dijkstra's Algorithm in Python," 12 October 2021. [Online]. Available: https://www.udacity.com/blog/2021/10/implementing-dijkstras-algorithm-in-python.html. [Accessed 09 05 2022].