OpenFlow-based Adaptive Routing for Wireless Networks

Alok Kulkarni
akulkar4@ncsu.edu

Angelyn Arputha Babu John
ababujo@ncsu.edu

Jignesh Darji
jndarji@ncsu.edu

Nishad Sabnis
nsabnis@ncsu.edu

`https://sites.google.com/a/ncsu.edu/openflow-based-adaptive-routing-for-wireless-networks`
`https://github.ncsu.edu/jndarji/CSC573_FALL2015_Project`
November 28th, 2015

1 Introduction

1.1 Problem Statement

Designing a system to enable adaptive routing in a wireless network in order to make a comparative analysis of the throughput efficiency of ground nodes versus aerial nodes.

1.2 Problem Description

We aim to implement OpenFlow based adaptive routing in an ad-hoc network by monitoring the link quality between wireless nodes. We anticipate that in such a network which offers multiple wireless routes between two end-points, the fluctuations in the RF link qualities between the endpoints will play an important role in determining the best end to end path. Determining the wireless link quality between each and every inter-connected node and making routing decisions based on this information constitute the two major parts of the problem.

We plan to make aerial nodes a part of the network which will be used for testing. Aerial nodes have their own set of advantages and disadvantages. They are less susceptible to electromagnetic interferences and can beam wifi over a large area if the antennae are powerful enough. However, the number of aerial nodes and naturally the number of available links through such nodes is likely to be lesser due to the low prevalence of such nodes. These trade offs need to be accounted for while making the routing decisions as well. The final aim is to ensure that the flow tables are dynamically modified to ensure effective end to end packet transmission.

2 Components

2.1 Platforms for the project

- OpenFlow v1.0
- Ubuntu
- POX OpenFlow Controller

2.2 Areas for the project

- Link Quality Monitoring
- Adaptive Routing

2.3 Major Components

2.3.1 Wireless Ground Nodes

The ground nodes will be laptops. They will have the following features:

- At least one wireless interface
- At least one interface to connect to the host
- Open vSwitch module installed
- One of the ground nodes will be the controller

2.3.2 Aerial Nodes

The aerial nodes will have the capacity to go up till 30 feet and beem signals from above. The key features of the aerial nodes are:

- At least one wireless interface
- BeagleBone black Linux boards
- Open vSwitch kernel module installed

2.3.3 Software Components

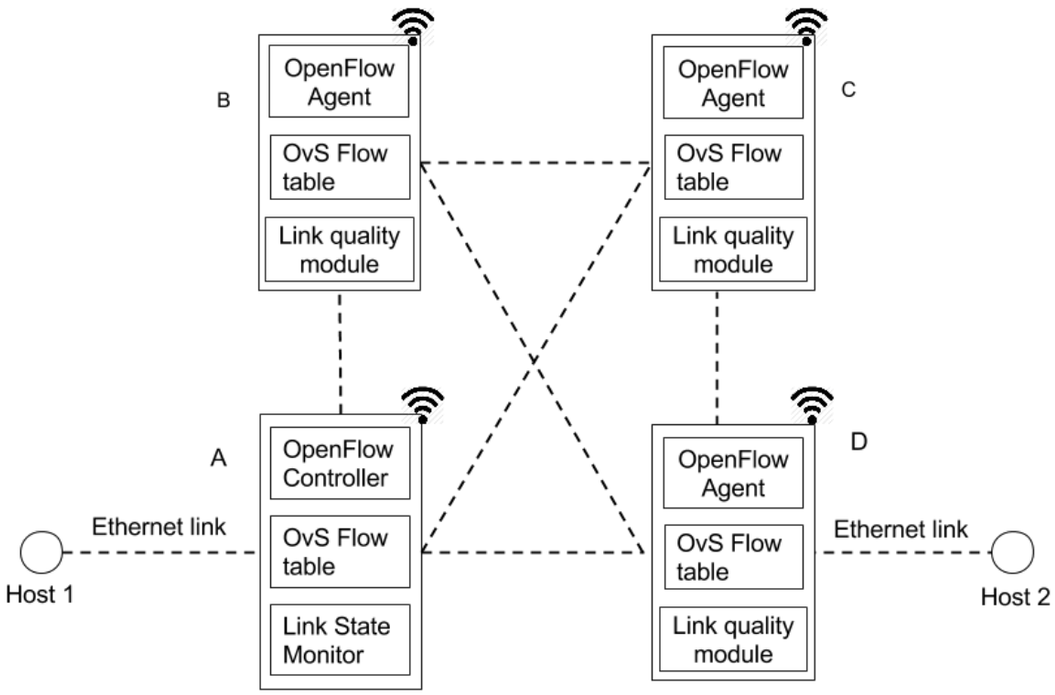
The software components will help determine the link quality and the optimum path, and they will configure the network with the optimal path.

- Link Quality Information module
- OpenFlow Control module

3 Design

3.1 Overview

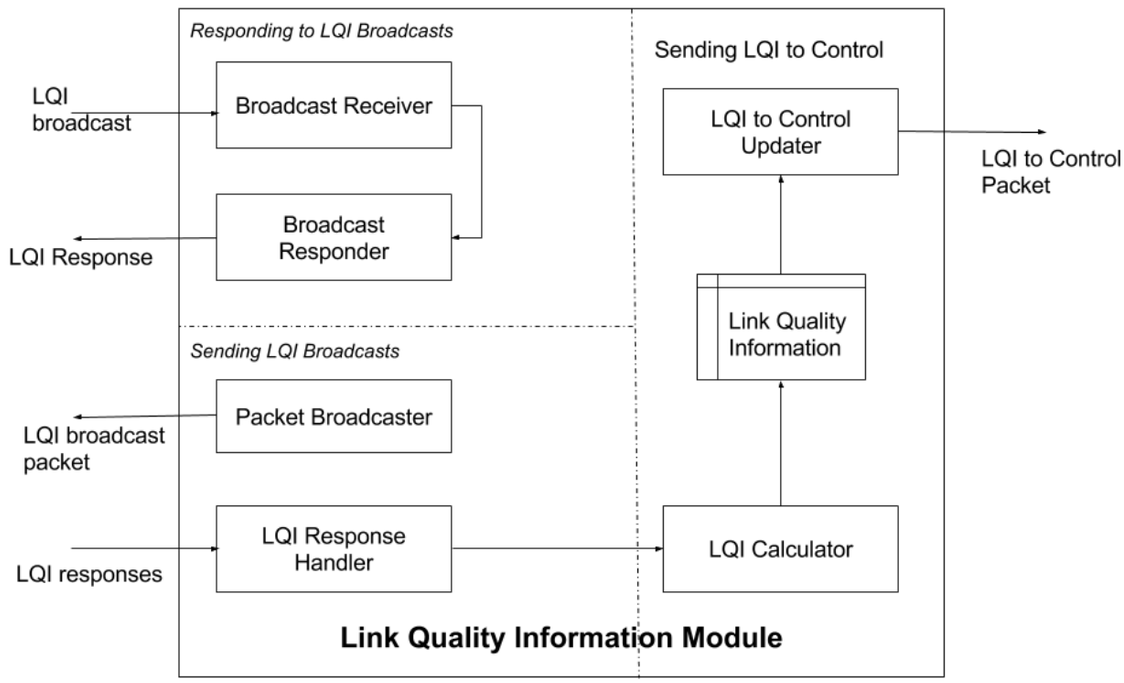
Figure 1: Design Overview



The above figure describes the overview of the components constituting this system. There exists a wireless ad-hoc network of aerial and ground nodes. One of the nodes acts as an SDN controller and the others act as agents. A link quality information (LQI) module is running on all the nodes in the network and this information is forwarded to the OpenFlow controller which makes adaptive routing decisions. The controller will use this information to compute the optimum end-to-end route between the endpoints. These routes will then be configured into the nodes using OpenFlow. The nodes will have OVS running on them which will configure the routes sent by the Controller.

3.2 Link Quality Information Module

Figure 2: Components in the Link Quality Information Module



This module is responsible for estimating the point-to-point link quality between nodes. It will run on all the nodes in the topology and give a periodically updated link quality estimate to the SDN controller. This information can then be used by the SDN controller to update the topology and find the optimal path. The module is made up of two units:

- *Request unit*: It sends out the UDP broadcast and receives the responses for these broadcasts (Packet Broadcaster, Response Handler).
- *Response unit*: It receives the UDP broadcast and responds to it (Broadcast Receiver, Broadcast Responder).

3.2.1 Packet Broadcaster

The LQI packet broadcaster will send out a UDP broadcast on the ad-hoc network. The broadcast is directed at a specific UDP port to differentiate it from normal UDP traffic. The packet broadcaster will send out a broadcast packet after a 10 second interval. The UDP frame will look as follows:

Source Port	<specific destination port>
Length	Checksum
"LQI Hello Packet" <payload>	

3.2.2 Broadcast Receiver

Will listen on a specific port for LQI Hello packets and trigger the Broadcast Responder once it has verified the frame format for each.

3.2.3 Broadcast Responder

Will prepare a unicast response for the broadcast source. The response will be sent out to a specific UDP port and will have the following format.

Source Port	<specific destination port>
Length	Checksum
"LQI Response Packet" <payload>	

3.2.4 Response Handler

The LQI Response Handler will listen at a specific port for any LQI responses, parse the LQI packet for obtaining the SSI, source and destination IP and MAC, and forward the packet to the LQI calculator. The module will have a timeout within which it expects to receive all the responses and will update the LQI Calculator after the timeout. The radiotap header (LINKTYPE_IEEE802_11_RADIOTAP) is an alternative to the normal 802.11 header (LINKTYPE_IEEE802_11) and and contains some extra information about the wireless network along with the normal 802.11 header. The radiotap header itself consists of a set of defined fields from which antenna signal (SSI) is to be extracted. The position of the Radiotap header in the frame is as shown below.

Frame(Physical Layer)	**Radiotap Header**	IP	UDP	UDP Payload

3.2.5 LQI Calculator

This module will get the SSI, source and destination IP and MAC address values from the Response Handler for all the responses it receives. These value will be converted and stored in an LQI table in a format which can be further sent to the controller.

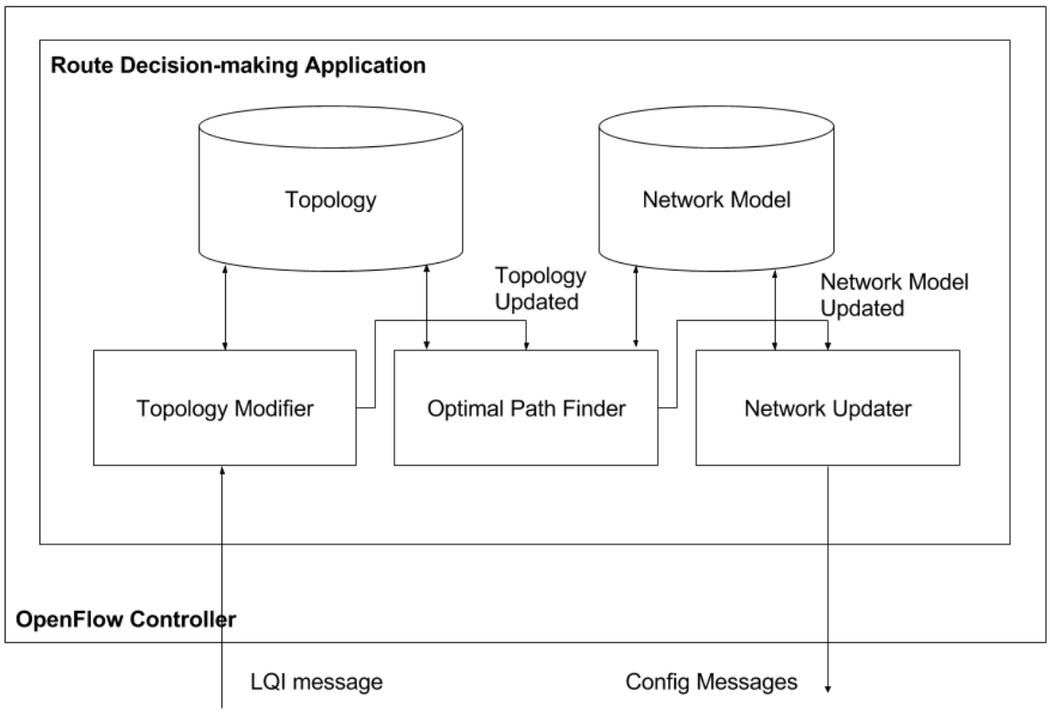
3.2.6 LQI to Control Updater

This module will refer to the values stored by the Calculator in LQI information table and prepare a payload that will be sent to the controller. This packet will be used by the controller to create a topology and find the optimal path. The format of the payload is as follows:

Source Port	Destination Port	Length	Checksum
SRC_IP			
SRC_MAC			Padding
Neighbor 1 IP			
Neighbor 1 MAC			SSI(1 byte)
Neighbor 2 IP			
Neighbor 2 MAC			SSI(1 byte)

3.3 OpenFlow Control Module

Figure 3: Components in the OpenFlow Control application



3.3.1 Topology Modifier(TM)

Function: Generate and maintain the topology of the network and intimate Optimal Path Finder about topology change.

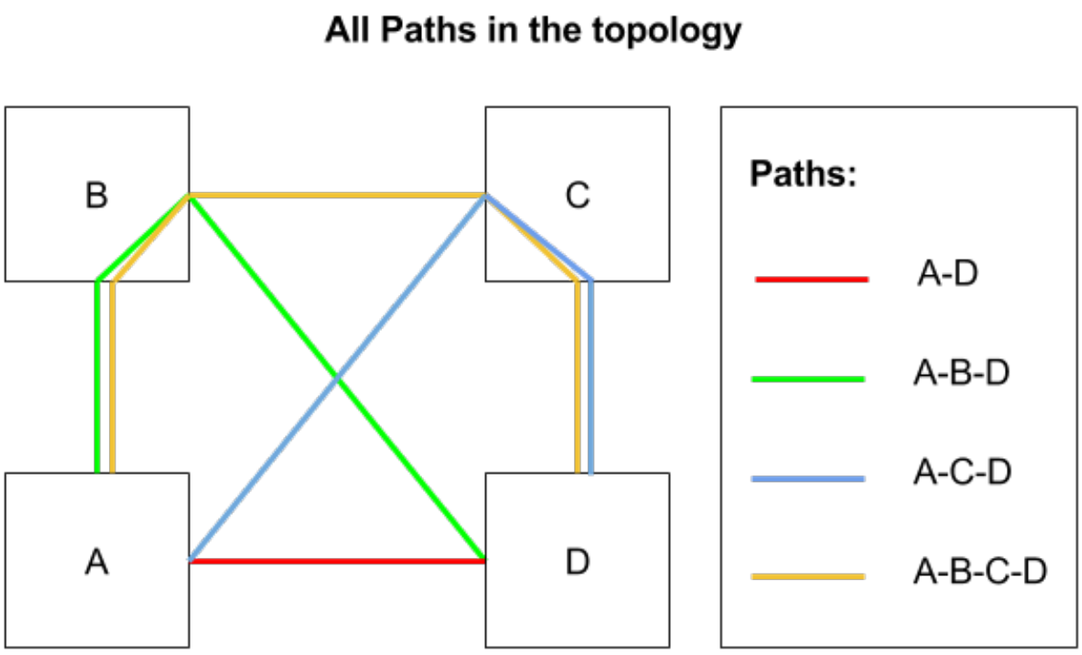
- Topology Construction: TM module accept the LQI messages and constructs the adjacency lists for all the nodes in the network. The node data structure will contain following information be as follows:
  - Node IP
  - Node MAC
  - Adjacent Nodes(IP and MAC) along with cost to reach them
- *Node Removal*: For each node, we shall have a running timer. If we do not get N consecutive LQI messages from a node, we shall remove the node from the topology. Calculation of N will be done later. We cannot have a high N because the network convergence time would be affected. We cannot keep it low because that will mean a lot of reconfigurations by the Network Updater if the node was not out of network or down but its LQI messages were dropped for a transient duration due to other factors.
- *Topology Updates*: The TM module will be invoked to update the topology when one of the following event occurs in the network:
  - *Node Addition*: When a node is added, it will be added to the adjacency list and cost to reach it from other nodes, and costs from that new node to other nodes need to be updated.
  - *Node Removal*: When a node is removed from the network(See Node Removal above), we need to remove its entry.
  - *Cost Updates*: When the LQI message from a node is received with different costs, the adjacency list of that node shall be update.
- *Callback to Optimal Path Finder*: When the topology is updated, it will send a callback to the OPF module.

3.3.2 Optimal Path Finder(OPF)

Function: Find the lowest cost path between the end hosts.

- *Limited Paths*: Since the topology is limited - 4 nodes and 2 of them connect to the end hosts - we have limited end-to-end paths between the 2 edge nodes.

Figure 4: Paths between edge nodes

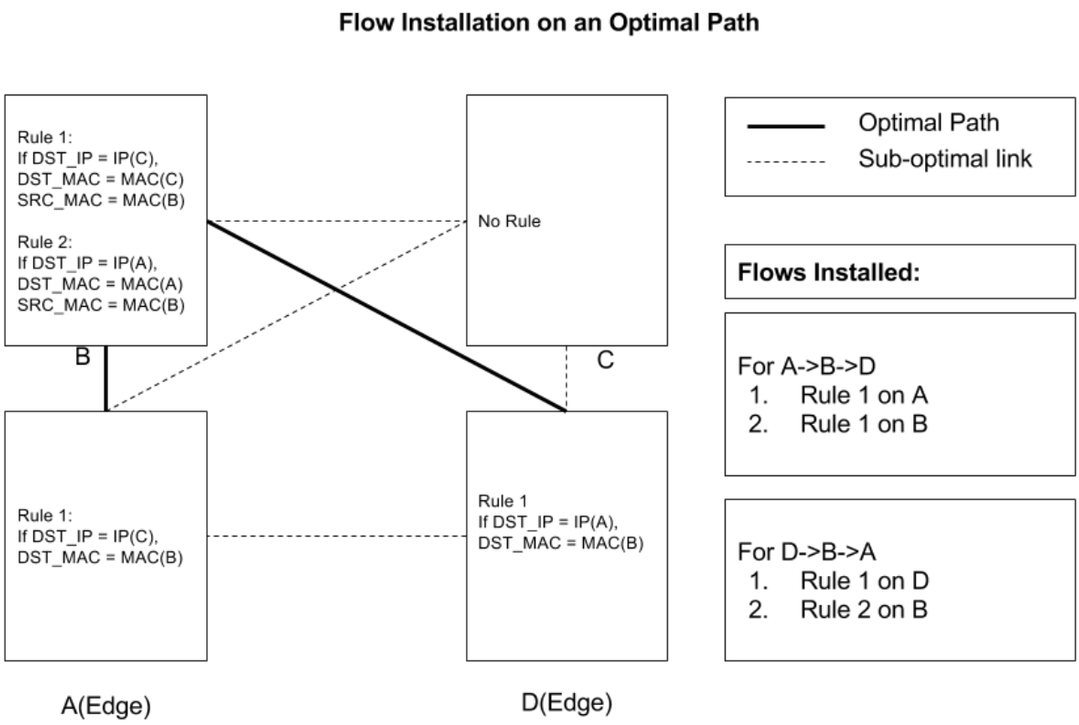


- *Path Selection*: OPF will calculate path costs for all the 4 paths and select the least cost path.
- *Callback to Network Updater*: If the new path is different from the previous path, we send a callback to the Network Updater.

3.3.3 Network Updater(NU)

- *Problem*: Since the network is wireless ad-hoc and all the nodes can generally connect to each other, the destination node will accept the packet directly. We need to enforce a method so that the packet follows the optimal path calculated by the OPF and not the direct path between edge nodes.
- *Algorithm*:
  - *Edge Nodes*: Install single rule on edge nodes to modify the DST_MAC to its next nodes MAC. Here, next node is the adjacent node in the optimal path.
  - *Internal Nodes*: Install two rules on internal nodes to modify the DST_MAC as well as SRC_MAC for both the directions. A simple illustration will clarify this step.
- *Example*: In the below illustration, the edge nodes are A and D and the optimal path is A-B-D. To enforce forwarding along A->B->D through the optimal path, A sends to B by changing the DST_MAC to Bs MAC and then B sends to D by changing the DST_MAC to Ds MAC and SRC_MAC to Bs MAC. Similar rules on D and B are required to enable for D->B->A.

Figure 5: Flow Rule Illustration



4 Per-member Responsibilities

Tasks	**Angelyn Arputha Babu John**	**Jignesh Darji**	**Nishad Sabnis**	**Alok Kulkarni**
Node Setup	Implement	Implement	Implement	Implement
Creation of ad-hoc network	Review	Review	Implement	Implement
LQI Message Handling	Review	Review	Implement	Implement
LQI Calculator	Review	Review	Review	Implement
LQI to Control Updater	Review	Review	Implement	Review
Topology Modifier	Implement	Implement	Review	Review
Optimal Path Finder	Implement	Implement	Review	Review
Network Updater	Implement	Implement	Review	Review

5 Test Results

Test ID	Test	Expected Observation	Conclusion	Result
T-1	Establishing the ad-hoc Network	All the nodes (ground and aerial) are able to ping each other within the network	Each node is able to join the ad-hoc network successfully	Pass
T-2	Add a node with low path cost	**LQI Module**: Connected nodes transmit their respective costs with new node to LQI messages **Topology Modifier**: Adds the node to the Topology; Intimate OPF **Optimal Path Finder**: Adds the path to Network Model; Intimate NU **Network Updater**: Updates the affected nodes.	The message packets are now sent through the path with least cost instead of following the costlier route	Pass
T-3	Add a node with high path cost	**LQI Module**: Connected nodes should transmit cost with the new node to the LQI messages **Topology Modifier**: Add node to Topology; intimate OPF **Optimal Path Finder**: Network Model remains same **Network Updater**: Not Affected	The message is sent through the usual path and is not changed by the addition of this new node	Pass
T-4	Remove node from Network Model	**LQI Module**: LQI messages will not contain costs to this node **Topology Modifier**: Remove node from Topology; intimate OPF **Optimal Path Finder**: Modifies Network Model; intimate NU **Network Updater**: Updates the affected nodes	The node is successfully removed from the topology with or without affecting the other nodes	Pass
T-5	Remove node not part of the Network Model	**LQI Module**: LQI messages will not contain costs to this node **Topology Modifier**: Remove node from Topology; intimate OPF **Optimal Path Finder**: Modifies Network Model; intimate NU **Network Updater**: Not Affected	The controller ignores this change as it is not part of the network	Pass
T-6	Increase associated path cost of a node in Network Model	**LQI Module**: Cost to this node should be increased in the LQI messages **Topology Modifier**: Update path costs in Topology; intimate OPF **Optimal Path Finder**: Modifies Network Model; intimate NU **Network Updater**: Update the affected nodes	Changes in the network are noticed as the controller routes the packet through the cheaper route	Pass
T-7	Increase associated path cost of a node NOT in the Network Model	**LQI Module**: Cost to this node should be increased in the LQI messages **Topology Modifier**: Update path costs in Topology; intimate OPF **Optimal Path Finder**: Network Model remains unchanged **Network Updater**: Not affected	The controller ignores this change as it is not part of the network	Pass
T-8	Decrease associated path cost of a node in Network Model	**LQI Module**: Cost to this node should be decreased in the LQI messages **Topology Modifier**: Update path costs in Topology; intimate OPF **Optimal Path Finder**: Modifies Network Model; intimate NU **Network Updater**: Update the affected nodes	The network changes as the controller routes the packet through the newest path with least cost.	Pass

T-9	Decrease associated path cost of a node NOT in the Network Model	**LQI Module**: Cost to this node should be decreasedin the LQI messages **Topology Modifier**: Update path costs in Topology; intimate OPF **Optimal Path Finder**: Modifies Network Model; intimate NU **Network Updater**: Update the affected nodes	The network changes as the controller routes the packet through the newest path with least cost.	Pass

Testing and Results obtained

For the testing, we use 4 wireless nodes such with the following configuration:

- Node A - IP:192.168.10.1 and MAC: `64:66:B3:21:B7:73`
- Node B - IP:192.168.10.2 and MAC: `C4:6E:1F:26:A1:6C`
- Node C - IP:192.168.10.3 and MAC: `60:36:DD:16:B8:EF`
- Node D - IP:192.168.10.4 and MAC: `A0:88:B4:25:1F:E8`

Below are the list of steps underwent to prove the test cases:

1. Starting the network with edge nodes A and D and setting up the ad-hoc network
   (a) Below is the ad-hoc network information and connectivity tests:

Figure 6: Ad-Hoc Network Information

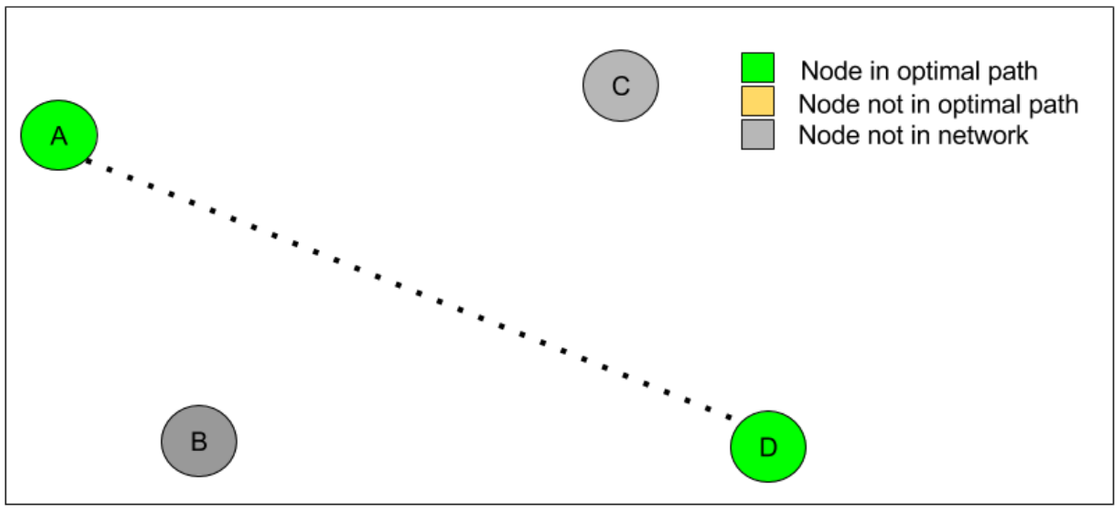
```
root@beaglebone:~# iwconfig wlan0
wlan0     IEEE 802.11bgn  ESSID:"beagle-ad-hoc"
          Mode:Ad-Hoc  Frequency:2.462 GHz  Cell: E6:61:38:86:FA:88
          Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

root@beaglebone:~#
```

Figure 7: Network connectivity test for the Ad-hoc network

```
Terminal                                                    (100%)  7:28 PM  Alok
Terminal ×   Terminal ×   Terminal ×
root@beaglebone:~# ifconfig bridge
bridge    Link encap:Ethernet  HWaddr 64:66:b3:21:b7:73
          inet addr:192.168.10.1  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::302f:2eff:fe36:4d4d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1767 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1231 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:105554 (103.0 KiB)  TX bytes:115340 (112.6 KiB)

root@beaglebone:~#
root@beaglebone:~# ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_req=1 ttl=64 time=4.05 ms
64 bytes from 192.168.10.2: icmp_req=2 ttl=64 time=20.2 ms
^C
--- 192.168.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 4.055/12.144/20.233/8.089 ms
root@beaglebone:~#
root@beaglebone:~#
root@beaglebone:~# ping 192.168.10.3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_req=1 ttl=64 time=9.04 ms
64 bytes from 192.168.10.3: icmp_req=2 ttl=64 time=38.4 ms
^C
--- 192.168.10.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 9.045/23.762/38.480/14.718 ms
root@beaglebone:~#
root@beaglebone:~# ping 192.168.10.4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
64 bytes from 192.168.10.4: icmp_req=1 ttl=64 time=9.88 ms
64 bytes from 192.168.10.4: icmp_req=2 ttl=64 time=12.9 ms
64 bytes from 192.168.10.4: icmp_req=3 ttl=64 time=2.07 ms
^C
--- 192.168.10.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 2.079/8.318/12.988/4.589 ms
```

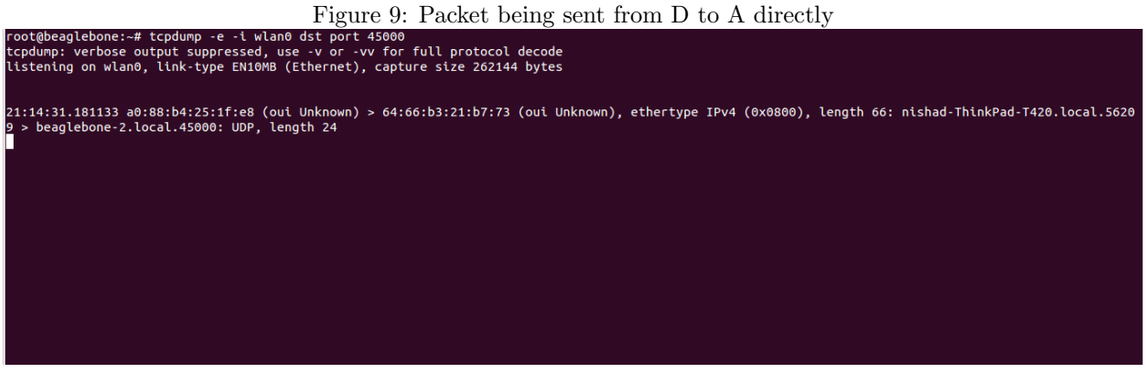
   (b) Below is an illustration of the network topology:

Figure 8: Network Topology



(c) Below is a screenshot of the packet being sent from node D(MAC = `A0:88:B4:25:1F:E8`) to node A(MAC = `64:66:B3:21:B7:73`) directly:

Figure 9: Packet being sent from D to A directly

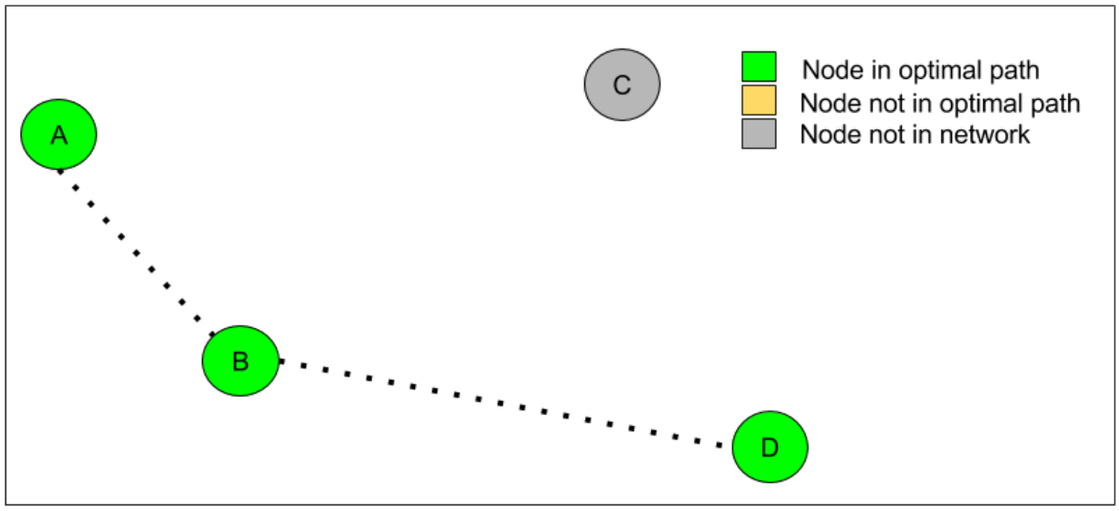


(d) This proves test case T-1

2. Now, we add node B(MAC = `C4:6E:1F:26:A1:6C`) to the network such that the optimal path is A-B-D

(a) Below is the illustration of the modified network topology:

Figure 10: Network Topology



(b) Below is a screenshot of the packet from D to A being routed through B:

Figure 11: Packet being sent from D to A through B

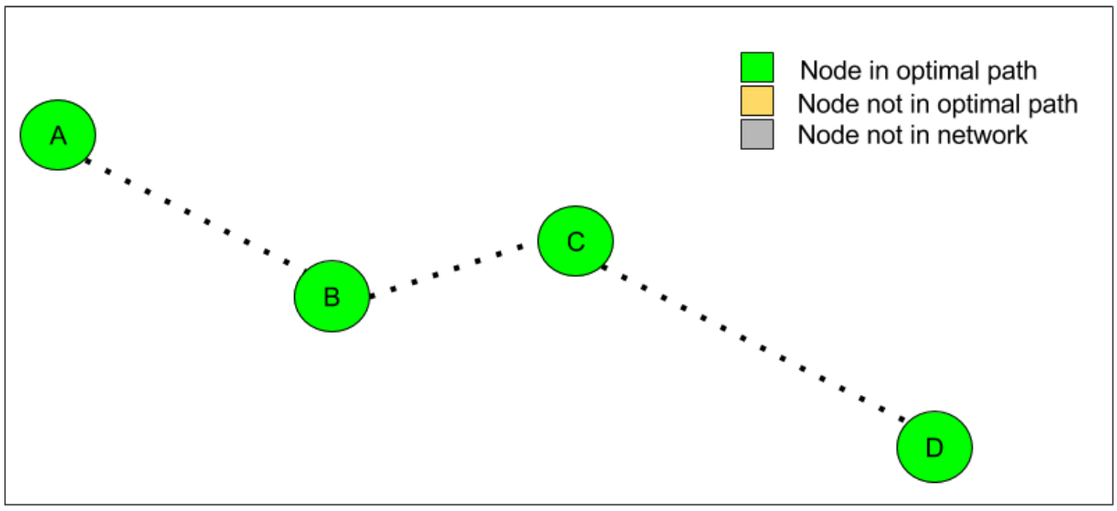
```
root@beaglebone:~# tcpdump -e -i wlan0 dst port 45000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes


21:32:17.591246 a0:88:b4:25:1f:e8 (oui Unknown) > c4:6e:1f:26:a1:6c (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.5930
3 > beaglebone-2.local.45000: UDP, length 24
21:32:17.596721 c4:6e:1f:26:a1:6c (oui Unknown) > 64:66:b3:21:b7:73 (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.5930
3 > beaglebone-2.local.45000: UDP, length 24
```

(c) This step proves test case T-2

3. Now, we add node C(MAC = `60:36:DD:16:B8:EF`) to the network now such that the optimal path is A-B-C-D

   (a) Below is the illustration of the modified network topology:

Figure 12: Network Topology



(b) Below is a screenshot of the packet from D to A being router through C and B:

Figure 13: Packet being sent through D-C-B-A path

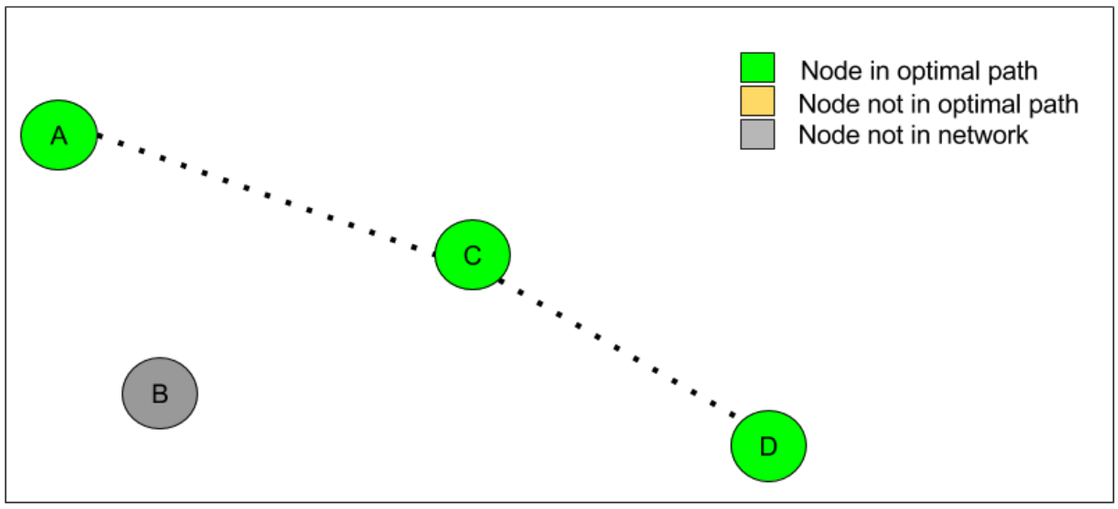
```
root@beaglebone:~# tcpdump -e -i wlan0 dst port 45000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes


21:28:18.782675 a0:88:b4:25:1f:e8 (oui Unknown) > 60:36:dd:16:b8:ef (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.4247
0 > beaglebone-2.local.45000: UDP, length 24
21:28:18.785825 60:36:dd:16:b8:ef (oui Unknown) > c4:6e:1f:26:a1:6c (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.4247
0 > beaglebone-2.local.45000: UDP, length 24
21:28:18.787492 c4:6e:1f:26:a1:6c (oui Unknown) > 64:66:b3:21:b7:73 (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.4247
0 > beaglebone-2.local.45000: UDP, length 24
```

(c) This step proves test case T-2

4. Now, we remove node B from the network itself

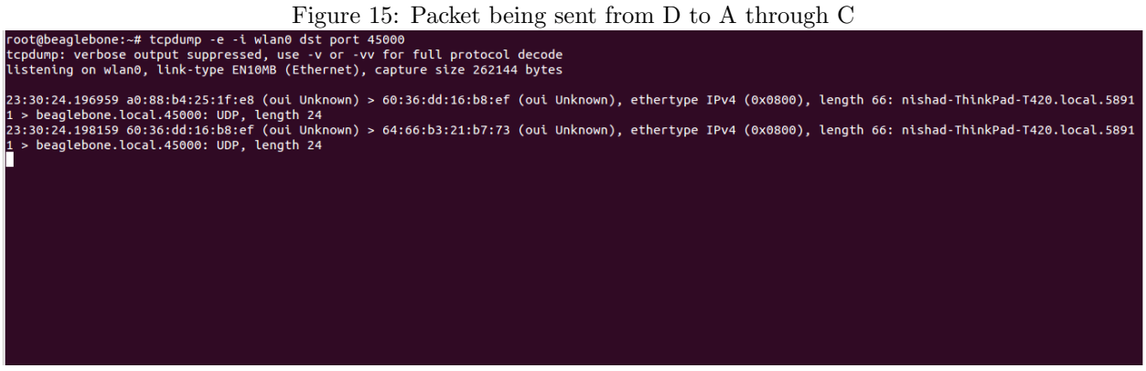
   (a) Below is the illustration of the modified network topology:

Figure 14: Network Topology



(b) Below is a screenshot of the packet from D to A being router through C since the shortest path now is A-C-D:

Figure 15: Packet being sent from D to A through C

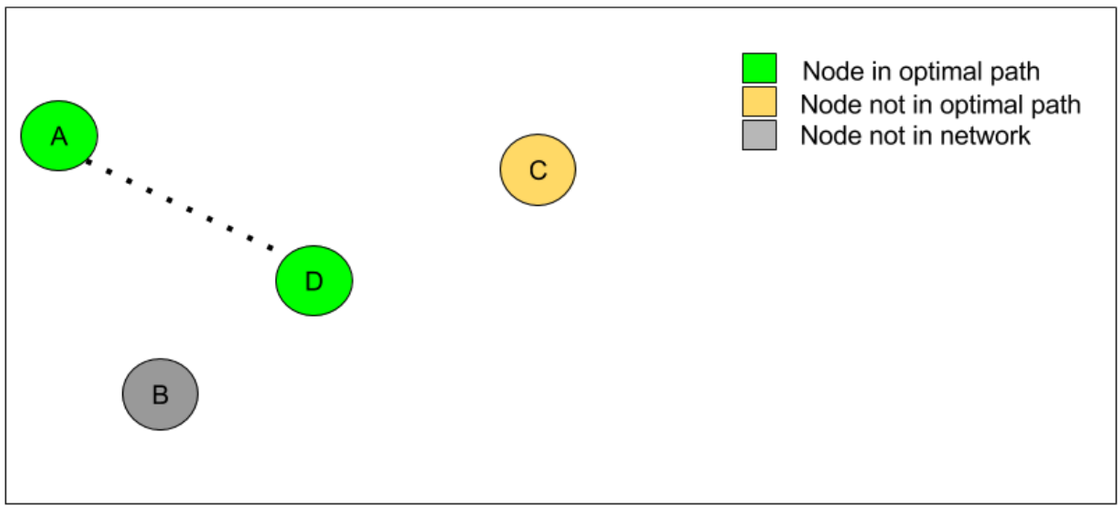


(c) This step proves test case T-4

5. Now, we move node D closer to A such that the optimal path is A-D and C is not part of the optimal path

(a) Below is the illustration of the modified network topology:

Figure 16: Network Topology



(b) Below is a screenshot of the packet from D to A being sent directly:

Figure 17: Packet being sent from D to A directly

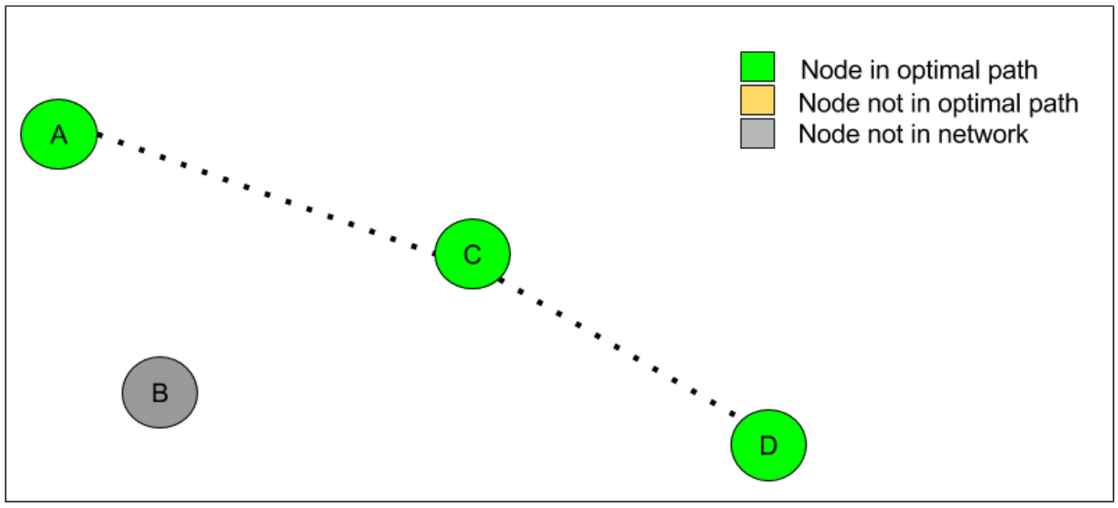
```
root@beaglebone:~# tcpdump -e -i wlan0 dst port 45000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes

21:33:58.829057 a0:88:b4:25:1f:e8 (oui Unknown) > 64:66:b3:21:b7:73 (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.3713
9 > beaglebone-2.local.45000: UDP, length 24
```

(c) This step proves test case T-8

6. Now, we move node D away from A such that the optimal path is A-C-D

   (a) Below is the illustration of the modified network topology:

Figure 18: Network Topology



(b) Below is a screenshot of the packet from D to A being sent directly:

Figure 19: Packet being sent from D to A directly

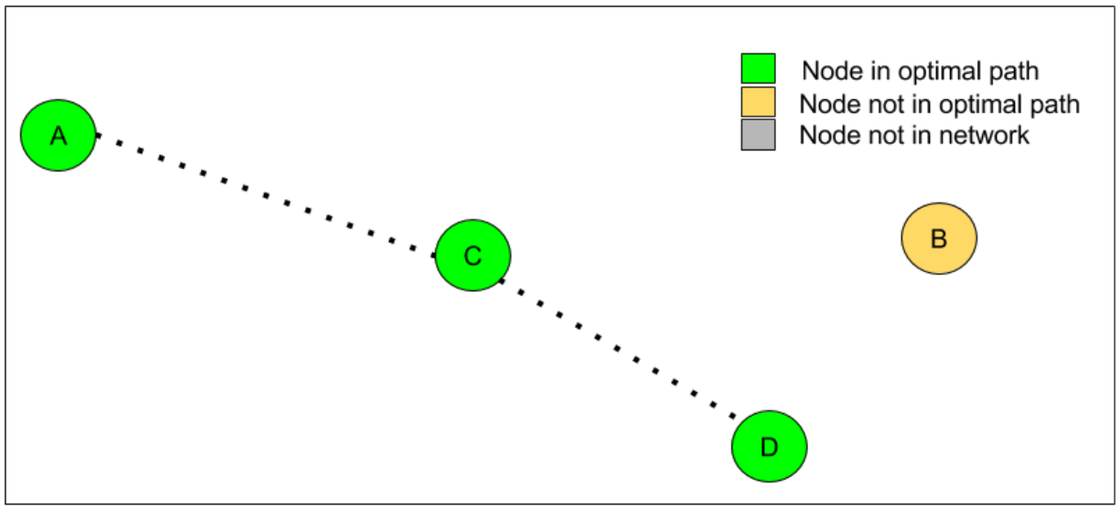
```
root@beaglebone:~# tcpdump -e -i wlan0 dst port 45000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes


23:33:12.725290 a0:88:b4:25:1f:e8 (oui Unknown) > 60:36:dd:16:b8:ef (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.4839
1 > beaglebone.local.45000: UDP, length 24
23:33:12.726594 60:36:dd:16:b8:ef (oui Unknown) > 64:66:b3:21:b7:73 (oui Unknown), ethertype IPv4 (0x0800), length 66: nishad-ThinkPad-T420.local.4839
1 > beaglebone.local.45000: UDP, length 24
```

(c) This step proves test cases T-6 and T-9

7. Now, we add node B to the network such that it's not part of the optimal path, i.e. with high path cost

   (a) Below is the illustration of the modified network topology:

Figure 20: Network Topology

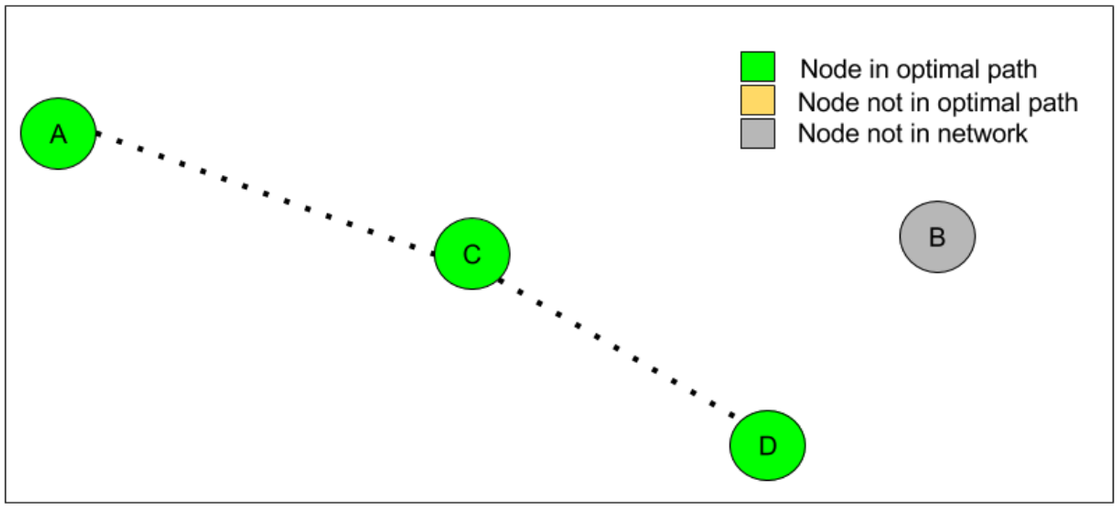


(b) This step proves test cases T-3

8. Now, we remove B from the network

(a) Below is the illustration of the modified network topology:

Figure 21: Network Topology



(b) This step proves test cases T-5 and trivially proves T-7

6 Demo Plan

Demo ID	**Scenario**	**Expected Observation**	**Conclusion**
D-1	The nodes will be arranged such that the shortest path from A to D is A->D. Then a simple data stream from Host 1 to Host 2 will be started. Tcpdump will be running on all nodes	Tcpdump for A and D nodes will show the data stream but tcpdump for B and C nodes will show no activity	The system has setup the flow in such a way that the route is direct.
D-2	The nodes will be arranged such that the shortest path from A to D is A->B->C->D. Then a simple data stream from Host 1 to Host 2 will be started. Tcpdump will be running on all nodes	Tcpdump for A,B,C,D nodes will show the data stream and indicate the packet path	The system has setup the flow in such a way that the route is not the direct route. LQI based routing has successfully taken place and routes have been configured accordingly

7 Self-study Results

7.1 Description of Base Case

The base case for our system is a 4 node wireless ad-hoc network topology. In a normal ad-hoc topology, if a transmission has to be sent from node A to node D, it will be sent directly(A->D). This takes place irrespective of the link quality between these nodes as these nodes are in the same wireless ad-hoc network. Also, it is to be noted that in case A and D are not in range of each other, the transmission is not possible.

7.2 Characteristics to Observe

The main characteristics observed are as follows:

1. *Link Quality*: Link quality was successfully observed by using the RadioTap header. A virtual network interface was created using the Linux iw wireless device configuration utility. This interface was set in monitor mode and observed and formatted all the radiotap data received on wlan0.

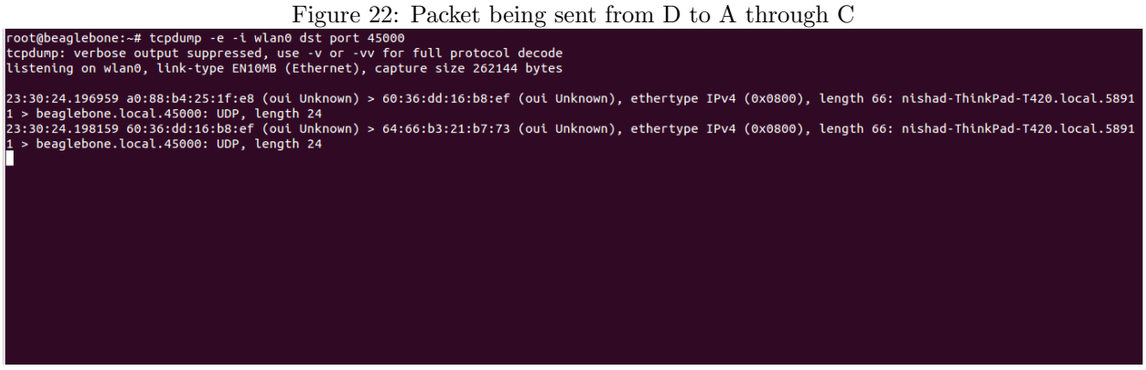
   Active link quality information was collected for every LQI packet received on every node. Every node sends out its own link quality as 1000, a random default value chosen for standardisation purposes. The actual signal strength is = -LQI dB

   A sample LQI packet as observed from the node with an IP address 192.168.10.4 is shown below:

IP Address	**LQI Value**
192.169.10.1	47
192.169.10.2	0
192.169.10.3	37
192.169.10.4	1000

2. *Packet path*: The packet path can be observed once the OpenFlow controller starts enforcing rules and setting LQI-based paths. The following figure shows the path taken by a UDP packet sent from A to D when a path A-C-D is set by the controller.

Figure 22: Packet being sent from D to A through C



3. *Latency*: As the packet may not be routed directly, some delay will be experienced for each hop it takes. This (negligible) delay can be seen in the figure above. The packet is sent via the path A-C-D. The packet reaches C at time 21:24:27.684111, and reaches D at time 21:24:27.685720, a delay of 0.001609 secs that was caused because of this hop.

7.3 Range of Scenarios to Investigate

There will be three major scenarios to investigate in this system are as follows:

1. The first scenario will be to observe edge-to-edge packet forwarding without any OpenFlow flows installed. It is expected that a packet destined to reach router D from router A, will be sent directly to router D. The main things which will be important to observe and consider is the time delay that it takes for this transmission, successful delivery of packet to D and the link quality between A and D.

2. The second scenario will be set up in such a way that the OpenFlow Flow tables are setup as mentioned in the low-level design and edge-to-edge forwarding is carried out. In this scenario it is expected that, the link quality between A- >D be lower than the link qualities between B->D and A->B. The packet will then take the path A->B->D and the timing delay will be observed. We think that the packet will take more time as it is coming via a longer route in spite of a shorter route with poorer link quality being available.

3. In the third scenario, we will set up the network so that A and D are not in range. In this case it is expected that the base case will fail to transmit but the proposed system shall deliver the packet successfully albeit with a higher delay.

8 Reflections

8.1 Per-member Learning Experiences

- **Alok Kulkarni**
  Coming into this project I had a very limited experience of working with wireless-ad-hoc networks, OpenFlow and OpenVSwitch. Working with these things throughout the duration of the course has helped me gain a deeper understanding of the nitty-gritties of fundamental networking concepts and help me get a good grip over the above mentioned things. Traffic management in ad-hoc networks, OpenVSwitch configuration and working, deeper exploration of the Linux wireless networking stack, basic scripting knowledge and working familiarity with commonly used linux command-line utilities such as tcpdump, iw were some of the important lessons I learnt while working on this project.

- **Angelyn Arputha Babu John**
  Initially I did not have a clear idea on SDN and have never worked with OpenFlow. Installing and configuring the Open VSwitch were the first few things I did in the project. We had initial problems of implement multi-hop scenarios in the ad-hoc network. We learnt how to manage the scenario modifying the L3 headers. I had initial knowledge of Python but learnt a lot from this project working in a Linux environment, which was new for me.

- **Jignesh Darji**
  Learning and working on SDN was one of the primary reasons I decided to pursue a Masters degree. This project was very challenging and the learning curve as well as the learning was steep. Starting with the most fundamental step of the project, installing OpenFlow rules over wireless network, was an important challenge and which brought immense satisfaction. In our initial approach to that, we were mangling the L3 header also, along with the L2 header, to force multi-hop approach in the ad-hoc network. Upon consultation, we learned that the mangling of L3 header was unnecessary and only modifying the dst_mac to the next hop would suffice. Next challenge was performing these steps programmatically, populating the costs in the LQI messages, and then configuring the network. The development of the project required learning Python and debugging network program using tcpdump and logging in the program.

- **Nishad Sabnis**
  While creation of the ad-hoc network was easy, the actual setup and the issues that arose helped me learn the nuances of ad-hoc networking. The ovs bridge setup and the OpenFlow rule installations have given me a lot of confidence in my understanding of the data link and network layers. The plethora of linux command line utilities and other tricks I learnt will help me in the future too. The background research done on the radiotap header increased my knowledge of network adapter chips and gave me insight into the working of tools such as tcpdump. I had never done any programming in python, so this was a good opportunity for me to learn. Working with threads, socket and timers lead to me being comfortable with python.

8.2 What went as expected?

- Creation of the ad-hoc network went smoothly and without any issues.
- The UDP broadcast mechanism for the LQI and neighbour discovery was also completed without any major issues and worked as per the design.
- The SDN Controller met all the needs and no workarounds were required to work with it.

8.3 What did not go as expected?

- The plan of using pcap to extract the signal level was not successful. Despite a lot of attempts, due to lack of documentation and examples, we were not able to properly capture packets using pcap. We tried alternatives like pcapy and scapy to no avail. Finally, due to time constraints we employed the method of creating a monitor interface using the iw command and then ran tcpdump as a subprocess in a python script. The tcpdump output was then parsed to get the radiotap header information, and the signal quality within it.

References

[1] OpenFlow Switch Specification
`http://archive.openflow.org/documents/openflow-wp-latest.pdf`

[2] POX OpenFlow Controller
`https://openflow.stanford.edu/display/ONL/POX+Wiki`

[3] RFC 2501: Wireless ad-hoc Networks
`https://tools.ietf.org/html/rfc2501`

[4] Radiotap header
`http://www.radiotap.org/`

[5] Open VSwitch
`http://openvswitch.org/`

[6] IW utility
`https://wireless.wiki.kernel.org/en/users/documentation/iw`

[7] RFC 768: User Datagram Protocol
`https://www.ietf.org/rfc/rfc768.txt`

[8] *tcpdump* utility
`http://www.tcpdump.org/`

[9] *libpcap* library
`http://www.tcpdump.org/`