Comparative analysis between SPF and BAR algorithm in SDN unicast network

SUBMISSION DATE: 24.12.17

SUBMITTED BY:
Soumen Ghosh (14101157)
Nayeem Mehedi (14101158)
Sahariar Khandoker (14101159)
Sabbir Aanwar (13201072)
Department of Computer Science and Engineering

Supervisor:

Amitabha Chakrabarty, Ph.D
Assistant Professor
Department of Computer Science and Engineering

Declaration

We, hereby declare that this thesis is based on results we have found ourselves. Materials of work from researchers conducted by others are mentioned in references.

Signature of Supervisor

Amitabha Chakrabarty, Ph.D
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Signature of Authors

Soumen Ghosh
(14101157)

Nayeem Mehedi
(14101158)

Sahariar Khandoker
(14101159)

Sabbir Anwar
(13201072)

ABSTRACT

Software Defined Networking (SDN) is a new idea of networking where Central Server takes all the decisions and finds a path for a packet to move from source to destination whereas in traditional networking a router takes decisions and finds path for the packet. The main advantage of SDN is that it reduces the time complexity of packet transfer as routers do not have to look up routing table for a path and it can reduce the packet loss to a minimum level. In traditional network, the routers make decisions according to their routing table which most of the time cannot grasp the full network topology. However, in SDN, central server possesses the entire routing table to control the data flow of network. As a result, the percentage of packet loss becomes at a minimal level. Though making packet loss zero percent cannot be achievable due to insufficient data flow path or any other physical factors, it is still less than the traditional network. In this paper, we have implemented two algorithms, Shortest Path First (SPF) and Bandwidth Aware Routing (BAR). We have simulated the algorithms in different topology using JAVA. We have noted down the packet loss, latency, path cost and bandwidth. We have developed our thesis between the comparison of Shortest Path First (SPF) and Bandwidth Aware Routing (BAR). We have simulated network graph on the both algorithms to get results of latency, bandwidth, packet loss, path cost. We hope that this comparison will help to get a clear picture about the advantages and disadvantages of these algorithms.

Keyword: Software Defined Network, SDN, Routing algorithm, Comparison, SPF, BAR.

Acknowledgement

This paper is based on Software Defined Network using SPF and BAR Algorithm in different scenario and performance analysis on the results. We would like to thank our honorable advisor Dr. Amitabha Chakrabarty who helped us in every aspect in this paper. It is a great pleasure to acknowledge our deepest thanks and gratitude to Dr. Amitabha Chakrabarty, Assistant Professor of Department of Computer Science, BRAC University for suggesting the topic and his kind supervision. It is a great honor to work under his supervision.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

In traditional network system, it is quite difficult to customize a network according to demand. Software Defined Networking (SDN) has changed the scenario of the traditional networking concept. Network control flow and routing paths of network routers have become directly programmable in Software-Defined Network. Network control flow is centralized in Software Defined Network (SDN) that maintains a global view of the network [1]. That means, central server has the all information about the routers in that particular network. With SDN system, central server selects path for packet according to bandwidth, path cost etc. If a path is down because of buffer full limitation or any physical reason, then central server will find out alternative route for that packet to move from source to destination [1]. So it is easy to maintain the network by SDN.

It is true that our network is dynamic and the costs between two routers change time to time based on their bandwidths. It is also possible that a packet has started moving from its source towards its destination. In the travel time, inner routers costs also may change and this changing can control the path selection. In our traditional network if this scenario occurs that packet will be dropped by a router as the expected route may no longer exist. However, in Software Defined Network as the central server takes all the decisions it can choose alternate paths for the data to flow. In this way, we can reduce packet loss by SDN. The Open-Flow protocol is a foundational element for Software Defined Network (SDN) [15]. Software Defined Network is a new concept in the networking field. In this paper,

we have implemented the Shortest Path First (SPF) and Bandwidth Aware Routing (BAR) algorithm according to the Software Defined Network (SDN) protocol [5][2][4].

1.1 Motivation:

Software Defined Network has brought huge change in the networking field. It has totally changed the concepts of network. Because of SDN the control plane has become centralized, programmable, flexible and more scalable. As a result, it is easier to modify the network accordingly. Traditional network can not guarantee the data transfer and the traffic flow of the network. It is one of the disadvantages of the traditional network. Day by day the network size is increasing and data flow also increasing so the problems faced in the current networks are also increasing. If this continues it will become a hindrance for the technological advancement. Thus the current network will be replaced by SDN. As a new field it needs modification and more research to improve the SDN network if not it might backfire. That is why we have tried to work with the SDN network and contribute on its development.

We have implemented SPF and BAR routing algorithms in the SDN network. After that, we have compared our results between the algorithms. We worked with the routing algorithms as SDN is a new concept, it is necessary to implement routing algorithms for the better one. So we tried to contribute a little in this aspect.

1.2 Methodology:

We have implemented the SPF routing algorithm for finding the shortest path from source to destination whose bottleneck bandwidth is maximum among the shortest path. Shortest Path First is to find out the route between source and destination in the network such that the total sum of cost is minimum. If there are multiple existing paths, we will select the maximum bottleneck. We have modified the relax operation of Dijkstra's shortest path algorithm [2][4].

Bandwidth Aware Routing focuses on a path with Maximum Bottleneck Bandwidth (MBB) of a network by modifying the SPF algorithm. BAR algorithm selects a shortest path from source to destination which has the MBB in a network. In case of same MBB, we will select the shortest path for the packet.

1.3 Objective:

Our objective is to implement SPF and BAR algorithm in Software Defined Network and simulate the algorithms to find out the hop count, average cost, average buffer size, packet per node, path size and data loss. We will compare these criteria between the two algorithms. In Software Defined Network because of its flexibility and scalability it has become possible to minimize the data loss percentage as the control plane has become centralized. So, it is possible to control the data flow and change the flow to alternate paths. If we can control the data flow the network congestion will also reduce. After comparing the SPF and BAR algorithms, we will try to come up with an improved routing solution so that it can efficiently reduce the traffic congestion and improve the latency of the

network. Our objective is to reduce the data loss and network congestion in Software Defined Network.

1.3 Thesis Overview

Chapter 1: Introduction to SDN, Methodology used and Objective of our Thesis

Chapter 2: Background concepts and Study of the ideas of the previous works

Chapter 3: Topologies used in our paper

Chapter 4: Implementation of the algorithms used in this paper

Chapter 5: Results found in each of the algorithms used in the paper

Chapter 6: Conclusion and future plan regarding the thesis

CHAPTER 2
Literature Review

Literature review is important as it gives a basic understanding of the subject and related works. Going through other works, we can grasp a comprehensive idea of the subject. It also broadens our horizon and enrich our knowledge. In our paper, we have taken some of ideas and concepts of other people. The main reason of developing Software Defined Network is that it takes shorter time for packet transfer than Traditional Network [1]. In our Traditional Network a router has a routing table where it keeps all the information of its neighboring routers. So when a packet comes to a router to move forward, that router checks its routing table to take decisions. That means if a packet need 5 routers to move from source to destination, then those 5 routers must check their routing table to select the path for that packet. Therefore, checking routing table each and every time is really a very slow process. However, in Software Defined Network routers do not take decisions for path selection for packets. Central server pre-calculates the route and distributes it to the network. In this time routers have information about the route and follow the decisions made by central server. Therefore, routers do not need to look up their routing table. Routers get the instructions from central server and act accordingly. In this way, Software Defined Network is way faster than Traditional Network. Because of these benefits SDN has become a new hot topic of networking. It is fully programmable and flexible according to the network needs [1]. Unicast routing is when there is only a single source and destination for a packet to deliver within a network. With the technological advancement in other sectors, Networking paradigms are also changing. Current

networking field is based on the routers with their decisions to take whether a packet will be delivered or not

2.1 SPF and BAR Algorithms:

Shortest Path First (SPF) algorithm locates the shortest path from source to the destination and set the bottleneck bandwidth to get maximum among every single path [2]. For different paths having a similar bottleneck bandwidth, we chose the path which has the maximum bottleneck flow entries. The authors chose a path with Maximum Bottleneck Bandwidth (MBB) of a network by modifying the SPF algorithm [5]. The relax operation of the SPF algorithm was changed. For every vertex the BAR algorithm chooses a shortest way from source to vertex which has the MBB in a network as opposed to choosing the shortest one as in Shortest Path First algorithm. When a vertex can be reached from source with a non-zero remaining transmission capacity, they lost the majority of the active edges. For the same MBB, they chose the one with the shortest path among them and sorted out all edges of smaller MBB in the original graph. Then executed the SPF algorithm on the reduced graph to find shortest path with the MBB from source to destination [5].

The authors evaluated the performance of these algorithms through simulations where they had used java-programming language to portray the SDN network [5]. They created a network with 100 switches and 400 hosts. The controller controls all the switches and the number of links between switches was 850 that were assigned to each switch randomly [5].

The links were given the same bandwidth: 1GB/s for the fairness of comparison [5].

2.2 K-SPF and K-BAR Algorithm:

k-SPF is the modified version of the SPF algorithm [2][4]. The authors proposed a k-SPF algorithm so that they can find a path with MBB among the first k shortest paths, where $k \geq 2$ is a predefined number [18]. Reversed the direction of every edge in the original network graph. At that point, utilized the SPF algorithm to discover one-to-all shortest path beginning at destination vertex and develop a shortest path tree established at destination. Utilizing the developed shortest path tree from destination to all vertices in a network diagram, creators turned around the course of each edge in the shortest path tree [18]. Along these lines, they could discover the Shortest Path from all vertices to vertex destination and get a Shortest Path Tree from all vertices to destination. The last algorithm they implemented was k-BAR where this algorithm was used to find a shortest path among the first k largest bottleneck bandwidth paths in a network, where $k \geq 2$ was a predefined number [18]. The authors obtained the k-BAR algorithm by modifying the k-SPF algorithm.

2.3 Performance analysis:

In the related works the authors evaluated the performance of different algorithms through simulations [5][7][8] [27]. They created a network with 100 switches and 400 hosts [5]. The controller controls all the switches and the number of links between switches was 850 that were assigned to each

switch randomly. The links were given the same bandwidth: 1GB/s for the fairness of comparison [5].

The criteria they followed were Average hop count or communication cost of a request, Average unsatisfied request rate, Average bandwidth satisfaction rate and Average link utilization. First, they showed the average hop count result. In the result, it is shown that the OSPF and SPF were the lowest average hop count among other algorithms. In The average unsatisfied request rate analysis the authors found that OSPF had the most unsatisfied request rate on the other hand SPF had the lowest unsatisfied request rate. The BAR algorithm had the highest average bandwidth satisfaction rates among various routing schemes. According to the authors the BAR algorithm choose a path with MBB of a network so that it had the highest bandwidth satisfaction rate among all schemes. The average link utilization increments only when the flow size of the solicitations increments. So if a demand can't be fulfilled, the consuming bandwidth of a demand will be not as much as the asked bandwidth. It suggests that the lower bandwidth satisfaction rate is the aftereffect of the lower link utilization. So the creators inferred that the average link utilization OSPF was the most minimal and BAR algorithm was the most noteworthy [5].

2.4 QoS (Quality of Service):

QoS is an important aspect of a network. Quality of Service(QoS) is the measurement of the services a network need so that the customer or clients can get the optimum benefit. QoS depends based on which type of service the clients require. As in all other networks QoS is also needed in the SDN network. QoS can be measured by the packet loss, latency, bandwidth, path

cost etc. In SDN network, the control plane is centralized so it is easier to improve the QoS of the network. Traffic Engineering can be improved a lot in a SDN network. SDN network can take care of the allotment of the resources and avoid congestion [20]. TE techniques usually works with the link weight of a network. Resource utilization is also can be considered as QoS. There is a chance that the heavily loaded networks data flow can use more resources. This can be improved by utilizing the flow migration capability [20]. Maintaining throughput efficiency can be also considered as Quality of Service. For example, in data centers there are thousands of servers. It is necessary to provide the data in a timely manner and maintain the QoS according to the clients need. Through QoS aware algorithm the throughput can be maintained in a large network [19].

2.5 Dynamic Traffic Scheduling Algorithm:

Day by day networks are growing and it is becoming difficult to manage the high traffic of the networks. Traffic engineering is based on the network traffic and distribution of flow paths according to the traffic matrix. Traffic Engineering is changing continuously as the networks are changing. The demands of the requirements of the networks are also increasing. But to keep up with the SDN network Traffic Engineering is not enough. The traditional TE measures the active flow distribution and flow measurement which cannot ensure the longevity of the network. In a SDN network load balancing paths can be measured unlike the traditional network. Traffic distribution model has been optimized through link utilization ratio [9]. It can only be used in the SDN network because of the flexibility of the control plane. The main objective of this model is to cope up with the

demand through load balancing and forwarding [9]. Load balancing can be adjusted in every switch so that it will not affect other switches and it will increase the effectiveness of the network.

2.6 Reliable Multitask Routing:

Reliable multitask routing (RMR) concept came up to transfer packet to a higher number of destinations [10]. Traditional multitask routing follows PIM-SM [16] where it connects the source and destination by a SPF (Shortest Path First) tree. A route from source and destination is calculated manually, SPF may lose many good routes by reducing the bandwidth consumption. To establish RMR, authors followed Johnson's algorithm, Mixed Integer Linear Programming. Again to complete RMR, they found two constraints Path and Tree Routing Constraints and Recovery Allocation Constraints.

The time complexity of Johnson's algorithm is $O(n) = O(|V||E| + |V2|log|V|))$ [10]. Again the overall time complexity is $O(n) = O(|V||D2| + |VT|r2|D2|)$ [10].

They found the packet loss rate of each link is between 1% to 10% and link delay from 10ms to 100ms [10]**.**

2.7 AWMR Algorithm:

The multipath routing in SDN based data center network using adaptive worst-fit multipath routing (AWMR), objective was to find a set of paths in a network for better utilization of the network resources [7][8]. The

proposed SDN-based adaptive worst-fit multipath routing (AWMR) algorithm for DCNs could choose different routing path and decide the number of paths for a new-coming flow as per the accessible bandwidth of paths and the requested bandwidth of the new-coming flow, rather than utilizing a settled number of routing paths for each flow. The proposed AWMR was made out of two phases [8]. The main phase of the proposed AWMR finds an underlying path set, including a fundamental number of paths from source host to destination host, and registers bandwidth capacity of such a path set. The second phase of the proposed AWMR, which used the Improved Widest Disjoint Path (IWDP) scheme, would then choose numerous worst-fit ways as per the requested bandwidth of the new-coming flow [8]. As the SDN provides the full scenario of the network it is possible to have the whole routing table beforehand. For that they used Link-Layer Discovery Protocol (LLDP) message and an OpenFlow OFPT_STATS_REQUEST message to get the information of network topology [8]. Using this information, the SDN-Controller can calculate the cost of from source to destination which can be made offline without waiting for the coming flow. Using these info, the authors picked paths from the routing table in an initial path set which will be used to select routing paths by the proposed AWMR. The authors calculated these from the k shortest path algorithm based on the Bellman-Ford algorithm. After that they had proposed the improved widest disjoint path in their paper. they have derived this architecture from the WDP scheme. Here is the proposed algorithm [8].

Algorithm1: Proposed AWMR algorithm

1 (a, b) = Host pair of source host a and destination host b
2 $B(F_{a,b})$ = Demanded bandwidth of a flow F from a to b
3 W (p) = Available bandwidth of path p
4 $P_{a,b}$ = Path set for host pair (a, b)
5 $W(P_{a,b})$ = Available bandwidth of a path set $P_{a,b}$

```
6 H = The necessary-connected-layer for host pair (a, b)
7 find H for (a, b);
8 for each path p whose path layer does not exceed H and
9                 W(p) =1= 0 for (a, b)
10        put path pinto P a,b;
11 end
12 compute W(Pa,b) using iwdp;
13 if W(Pa,b) >= B(Fa,b) then
14                decrease the number of paths in P a,b using Algorithm 2;
15        return Pa,b;
16 else if W(Pa,b) <  B(Fa,b) and His not the core layer then
17        put all possible paths into Pa,b;
18        compute W(P_a,b) using iwdp(improved widest disjoint path);
19        if W(P_a,b) >= B(F_a,b) then
20                decrease the number of paths in P_a,b using Algorithm 2;
21                return P_a,b;
22        else
23                return no feasible solution;
24 else if W(P_a,b) < B(F_a,b) and H is the core layer then
25        return no feasible solution;
26 end
```

Algorithm 2: Decreasing the number of routing paths

```
1 P_a,b = Initial routing path set from Algorithm 1
2 B(F_a,b) = Demanded bandwidth of a new-coming flow F from a to b
3 for each path p^current in P_a,b in ascending order of the
4 available bandwidth do
5         remove p^current from P_a,b;
6         compute W(P_a,b) using iwdp;
7         if W (P_a,b) >= B(F_a,b) then
8:        do nothing;
9         else
10                put p^current back into path set P_a,b;
11 end
```

2.8 Bandwidth-delay Constrained Routing Algorithm:

Fast and efficient bandwidth is introduced to maximize the utilization of network resources and reduce the computational complexity of central server [20]. They found that traffic engineering (TE) and bandwidth-delay routing concludes unsolvable issues in real time frame [23]. In our dynamic network, some traffic flows might allocate more resources in the

time of heavily loaded network. This causes a high chance of system failure. To solve this issue, Maximum Delay-Weighted Capacity Routing Algorithm (MDWCRA) [24]. MDWCRA measures the shortest paths between two nodes. By following this algorithm, 70% packets can be transferred from source to destination with average delay of 25ms of nodes. Authors concluded as they have made real implementation and have evaluated more realistic scenarios [20]. Here is the proposed algorithm to calculate the link weight [20].

Algorithm: calculation of link weights

```
#path.bw = bandwidth (BW) of the path, G =network graph
#DS_ind = indicator of DS traffic, src=source node, dst=destination node
function: get_weights(src,dst,B,DS_ind):
        for link in G.links() do:
          weights1[link]= 1/link.residual_bw
      weights2[link]= 0
       end for
       for (I,E) in IE_pairs do:
     if DS_ind == True and (I,E) == (src,dst) then: continue
      end if
      for path in DS_paths[(I,E)] do:
        for link in path.links() do:
          if link.residual_bw < path.bw +B then: #link is critical
              weights2[link]+=1/link.residual_bw
          end if
        end for
      end for
  end for
  normalize(weights1) # normalize wights in range [1,10]
  normalize(weights2)
  for link in G.links() do:
     weights[link]=(1-α)·weights1[link] + α·weights2[link]
   end for
end function
```

CHAPTER 3
Topologies

A SDN design comprises of four parts: Links, SDN-enabled switches, SDN controller and Hosts [27]. We assume that a SDN comprises of various switches interconnected by an arrangement of connections and have different connections between the switches [15]. A SDN network topology can be represented as a weighted, directed graph $G = (V, E)$ in which V is a set of vertices and E is an arrangement of edges interconnected vertices in V [4]. Every vertex in V represents a switch in SDN and each edge in E represents a switch link in SDN. Since various connections between a switch pair is permitted, we utilize the notation $e(u_i , v_i)$ to show the i-th edge from vertex u to v and utilize the notation $S(u, v)$ to demonstrate the arrangement of all edges from vertex u to v, where $(u, v) \varepsilon V$ [4].

For each edge $e \varepsilon E$, $w(e)$ denotes the link weight and $b(e)$ denotes the remaining bandwidth of the switch link. Let $s \varepsilon V$ be a vertex called source and $d \varepsilon V$ be a vertex called destination [11]. We note that if there is more than one path with the same cost, we will select the path whose switches have the maximum bottleneck flow entries [17] [22].

We have worked on two routing algorithms. First is Shortest Path First where we choose a path with the lowest cost [25]. Other is Bandwidth Aware Routing using the bottleneck bandwidth among the shortest path of the network [20].

3.1 Shortest Path First (SPF):

We find the shortest path from source to destination. If more than one shortest path exists, we will select the path with MBB among all shortest

paths [5]. We have modified Dijkstra's algorithm. Dijkstra's algorithm keeps up a set *S* of vertices whose final shortest path weights from the source *s* have just been determined [4]. The algorithm chooses the vertex $u \varepsilon V\text{-}S$ with the base briefest path estimate, adds *u* to *S*, and relaxes all edges leaving *u*. We utilize a low priority queue Q of vertices, which is calculated by the *d* value.

Dijkstra's algorithm run on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and s, terminates with u, $d = \delta(s, u)$ for all vertices $u \varepsilon V$ [4][2]. Finding new paths when processing a vertex *u*, the algorithm will examine all vertices $v \varepsilon Adj[u]$ [2][4]. For each vertex $v \varepsilon Adj[u]$, a new path from *s* to *v* is found (path from *s* to *u* plus new edge). For relaxation, if the new path from *s* to *u* is shorter than *d*[*v*], then update *d*[*v*] to the length of this new path [2].

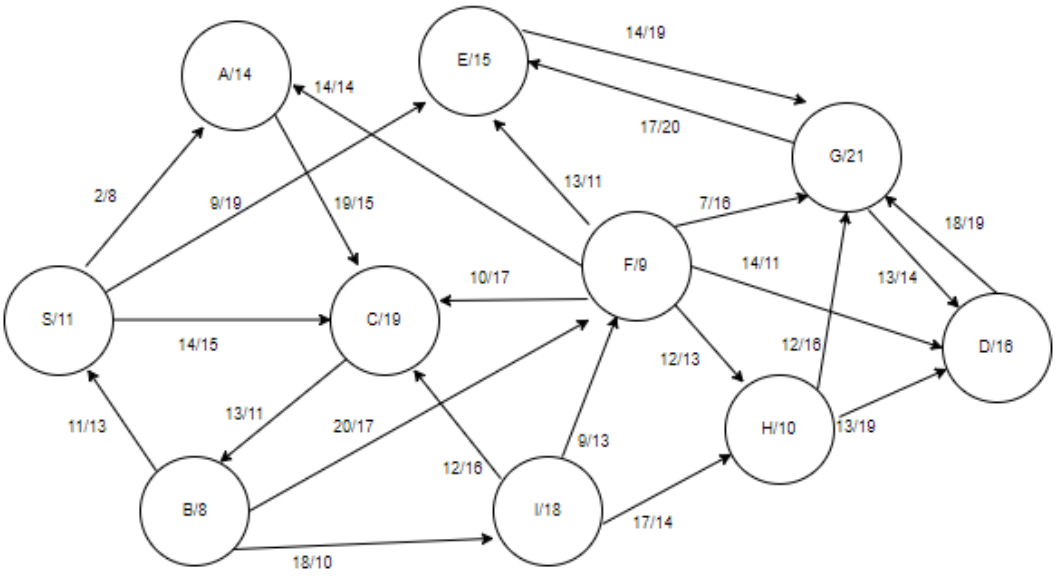

Figure 3.1: Execution of Dijkstra's algorithm

3.2 Bandwidth Aware Routing (BAR):

Bandwidth Aware Routing algorithm takes the Maximum Bottleneck Bandwidth (*MBB*) as a parameter to choose a path from *s* to *d* [24]. If there is more than one path has the same MBB than we will select the shortest path.

The following graph the alphabet in the circle is the switch *ID* and the number is available flow entries of that switch. The first value of a directed edge is the weight of that edge and the second value is the available bandwidth of the edge. We assume our source switch *S* and destination switch *D.*

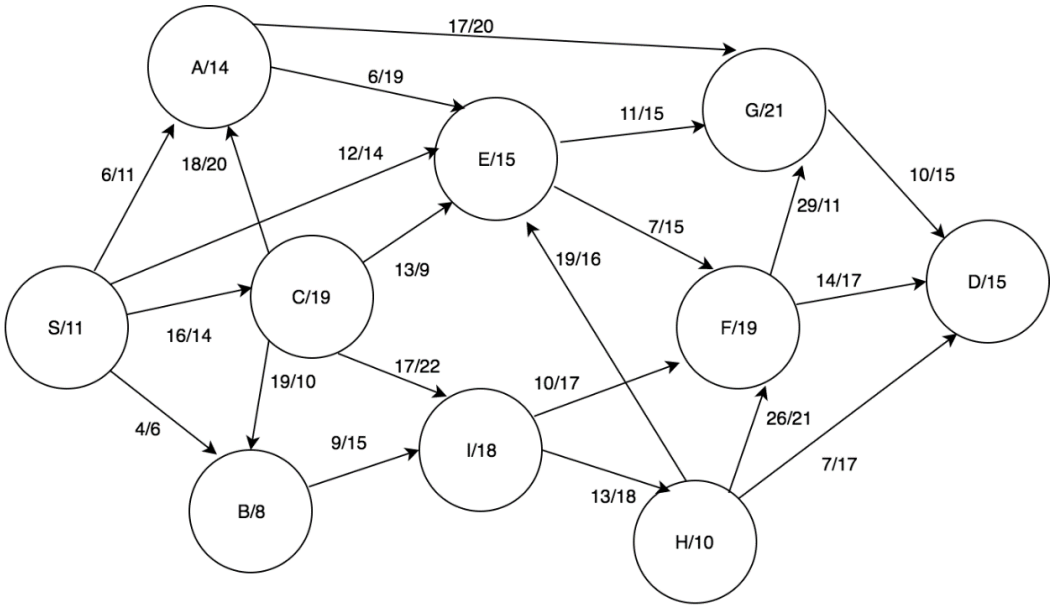

Figure 3.2: Execution of BAR algorithm

In network graph, there are multiple shortest paths from Source *S* to Destination *D*. They are (I) *S* - *E* - *F* - *D*, (II) *S* - *B* - *I* - *H* - *D* and (III) *S* - *A* - *E* - *G* - *D* and their costs are 33. All these three paths, path (I) *S* - *E* - *F* - *D* has *c* (*p*) = 14, path (II) *S* - *B* - *I* - *H* - *D* has *c* (*p*) = 6 and path (III) *S* -

A - *E* - *G* - *D* has *c* (*p*) = 11. So our final shortest path for this network graph from source S to destination D is path *S* - *B* - *I* - *H* - *D* and this path has *d* (*p*) = 33, *c* (*p*) = 6 and σ (*p*) = 8.

CHAPTER 4

Algorithm Implementation

To develop SDN we have implemented two algorithms. One is Shortest Path First (*SPF*) and another one is Bandwidth Aware Routing (*BAR*). In this chapter, we will discuss about these two algorithms in details.

4.1 Shortest Path First:

Shortest path first is our first algorithm to implement SDN to transfer packet from source to destination. To find out the shortest path for a packet we have calculated the minimum path cost. Again, if there are multiple shortest path with same bottleneck bandwidth available then we have selected the path, which has the maximum bottleneck flow entries [5] [14]. For example, in the following graph, the alphabet in the circle is the switch *ID* and the number is available *flow entries* of that switch. The first value of a directed edge is the *weight* of that edge and the second value is the available *bandwidth* of the edge [3]. In this graph, switch *S* is source and switch *D* is destination. The shortest path is *S* - *E* - *G* - *D*. The total cost of this path $d(p) = 36$. The bottleneck bandwidth is $c(p) = 14$ and $\sigma(p) = 11$. Therefore, we have selected our shortest path from this graph.

To develop shortest path algorithm, we have followed Dijkstra's shortest path algorithm [4]. We have also modified the relax operation. Here we initial some notations those we have used in SPF algorithm. Each vertex $v \in V$, $d(v)$ denotes the cost between two vertices v, $c(v)$ denotes the bandwidth between two vertices and $\sigma(v)$ denotes the available flow entries between two vertices. Lastly, $p(v)$ denotes the parent vertex of vertex v.

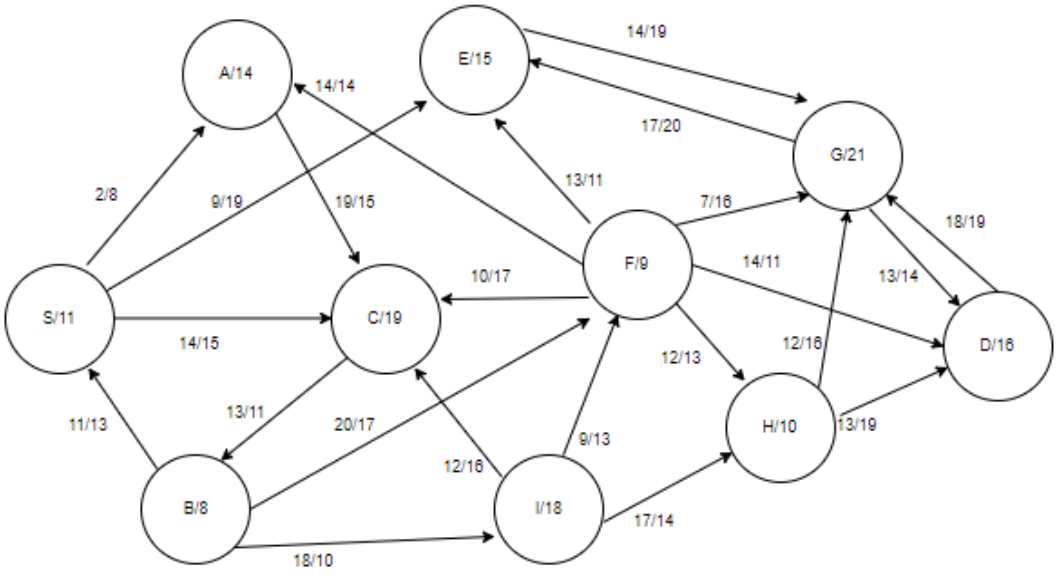

Figure 4.1: SPF topology

When a vertex $v \epsilon V$ can be reached from source with a finite cost, we can relax all of the outgoing edges $e\ (v_i, w_i) \epsilon S_u, v$ as follows.

- If $d\ (w) > d\ (v) + w\ (e\ (v_i, w_i))$, then we update the value of $d\ (w)$ with the value of $d\ (v) + w\ (e\ (v_i, w_i))$. After that, we have set $c\ (w)$ to $\min\{c\ (v), b\ (e\ (v_i, w_i))\}$. Lastly we update the value of $\sigma\ (w)$ by $\min\{\sigma\ (v), f\ (w)\}$and $p\ (v)$ is set as v [5].
- If $d\ (w) = d\ (v) + w\ (e\ (v_i, w_i))$ and $c\ (w) < \min\{c\ (v), b\ (e\ (v_i, w_i))\}$, then we have changed $c\ (w)$ to $\min\{c\ (v), b\ (e\ (v_i, w_i))\}$, $\sigma\ (w)$ is updated as $\min\{\ \sigma\ (v), f\ (w)\}$, and $p\ (v)$ is set as v [5].
- If $d\ (w) = d(v) + w(\ e\ (v_i, w_i))$, $c\ (w) = \min\{c\ (v), b\ (e\ (v_i, w_i))\}$ and $\sigma\ (w) < \min\{\sigma\ (v), f(w)\}$, we update $\sigma\ (w)$ as $\min\{\sigma\ (v), f(w)\}$ and $p\ (v)$ is set as v [5].

Initially all the vertices in this network graph are unvisited. The vertex S, initially values of $d\ (s) = 0$, $c\ (s) = \infty$, $\sigma\ (s) = \infty$, and $p\ (s) =$ null. For each

non-source vertex $v \epsilon V$, the initial values of $d(v) = \infty$, $c(v) = 0$, $\sigma(v) = 0$, and $p(v) = null$. Here in this SPF algorithm, we will visit the unvisited vertices with minimum cost and relax its neighbor vertices. After doing relax operation we will get the final result. The path we get from the following graph is S - E - G - D with $d(p) = 36$, $c(p) = 14$ and $\sigma(p) = 11$. Time complexity: $O(n) = O(|E| + |V|log|V|)$ [5][4].

Here is the pseudo code of the Dijkstra's Shortest Path First (SPF) algorithm:

Shortest Path Algorithm

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0
3      create vertex set Q
4      for each vertex v in Graph:
5          if v ≠ source
6              dist[v] ← INFINITY
7              prev[v] ← UNDEFINED
8          Q.add_with_priority(v, dist[v])
9      while Q is not empty:
10         u ← Q.extract_min()
11         for each neighbor v of u:
12             alt ← dist[u] + length(u, v)
13             if alt < dist[v]
14                 dist[v] ← alt
15                 prev[v] ← u
16                 Q.decrease_priority(v, alt)
17     return dist[], prev[]
```

4.2 Bandwidth Aware Routing:

Bandwidth Aware Routing is our last algorithm to implement SDN to transfer packet from source to destination. Here, our goal is to find a path with maximum bottleneck bandwidth (MBB) in our network graph from

source to destination. It is quite possible that there could be multiple paths with same MBB in that network graph. We will select the shortest path among the same MBB holding paths. Again, if there are multiple shortest paths then we will select the path which has the maximum bottleneck bandwidth (MBB) for our packet to move from source to destination.

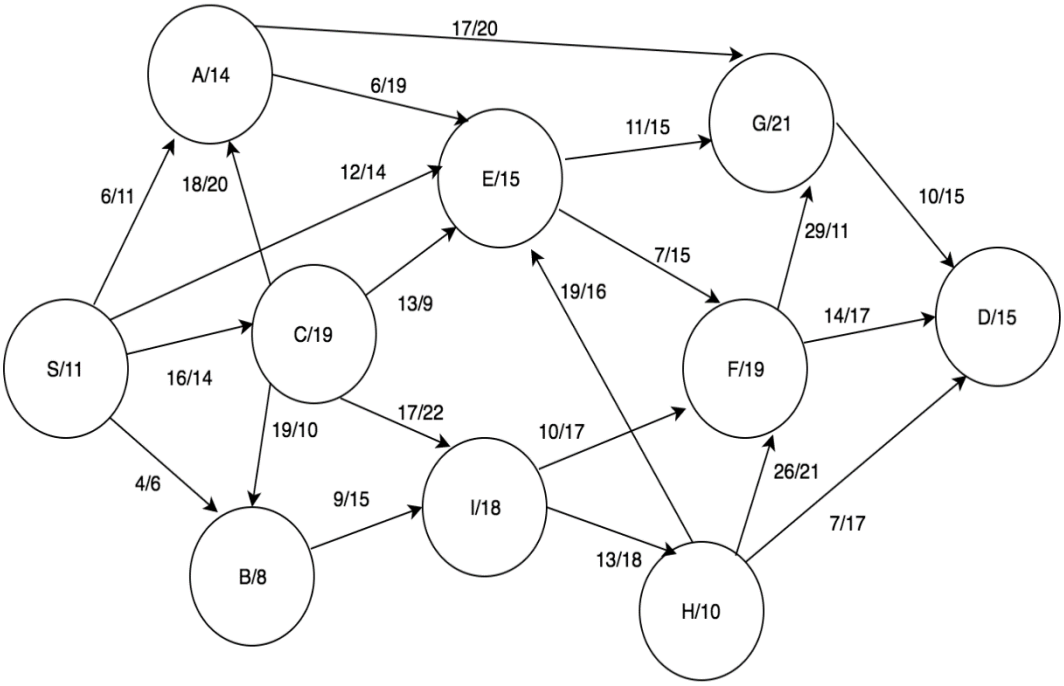

Figure 4.2: BAR topology

On the following graph, the alphabet in the circle is the switch ID and the number is available flow entries of that switch. The first value of a directed edge is the weight of that edge and the second value is the available bandwidth of the edge. We assume our source switch *S* and destination switch *D*. Here in the network graph there are multiple shortest path from Source *S* to Destination *D*. They are (I) *S* - *E* - *F* - *D*, (II) *S* - *B* - *I* - *H* - *D* and (III) *S* - *A* - *E* - *G* - *D* and their cost are 33. All three paths, (I) *S* - *E* - *F* - *D* has $c(p) = 14$, (II) *S* - *B* - *I* - *H* - *D* has $c(p) = 6$ and (III) *S* - *A* - *E* -

G - D has c (p) = 11. So our final shortest path for this network graph from source S to destination D is path S - B - I - H - D and this path has d (p) = 33, c (p) = 6 and σ (p) = 8.

To develop Bandwidth Aware Routing our main target is to find out the maximum bottleneck bandwidth (MBB) of the network graph using our shortest path first algorithm. Relax operation is modified of shortest path first algorithm [2][4]. BAR algorithm chooses the shortest path from source to v where vertex $v \epsilon V$ and that vertex has the maximum bottleneck bandwidth (MBB) in the network graph rather than following the shortest path first algorithm. At the time of selecting a vertex $v \epsilon V$ where it can be visited from S with a non-zero bandwidth, the outgoing edges e (v_i, w_i) ϵ S (u, v) will be relaxed. We will get the maximum bottleneck bandwidth value from S to D in this network graph when we complete $|V|$ times relax operations. If there are multiple paths with same MBB then we will select the shortest path from the graph and delete the edges of smaller MBB from the actual graph. After that, we will run our shortest path first (SPF) algorithm to get the shortest path with MBB from S to D.
Time complexity: $O(n) = O(|E| + |V|log|V|)$ [5][4]

Here is the pseudo code of the Bandwidth Aware Routing (BAR) Algorithm:

Bandwidth Aware Routing Algorithm

```
1 function Bar(Nodes n, Source s,Edges w):
2   Init(n,s);
3   Q=Insert(n);
4   While(Q!empty)
5     u.color=gray
```

```
6     for each vertex v adjacent to u
7     Relax(u,v,w)
8   minimum=findMiniMumPathCapacity(n)
9   foreach(w:Edges)
10     if(w.bandwidth<minimum)
11     remove w;
12  spf({n,w},s)
```

This is the definition of initialize method named as Init(),

Init method

```
1 Init(nodes[] n,source):
2  foreach(n:nodes)
3    n.pathCapacity(0)
4   if(n==source)
6    n.pathCapacity(10000)
```

This is the definition of Relax() method,

Relax method

```
1 Relax(parentNode u, childNode v, edge w):
2   min=0;
3   If(u.pathCapacity>w.availableBandwidth)
4     min= w.availableBandwidth;
5   Else
6     min= u.pathCapacity;
7  If(v.patchCapacity<min)
8    v.pathCapacity=min;
```

CHAPTER 5

Result Analysis

In our simulations, we have used our own simulator. It is written in Java. The simulator was made with the intention to evaluate different routing algorithms of Software Defined Network (SDN). The Algorithms we are using here are Shortest Path First (SPF) and Bandwidth Aware Routing (BAR) algorithm. We considered these parameters in our packet simulations: Packets Delivery, Path Bandwidth, Buffer Size of Router, Path Cost and Hop Count to destination node.

We have kept network graph source fixed, at the same time we have set the destination randomly for every time to get near accurate results for packet transfer. In the results we will be showing comparisons of SPF Algorithm first then BAR algorithm's. We transferred 100 Packets in every simulation and generated JSON file containing attributes and values for every packet transferred. Then we used some A/B testing to find out any possible correlation.

5.1 SPF ALGORITHM:

5.1.1 Packets per Node:

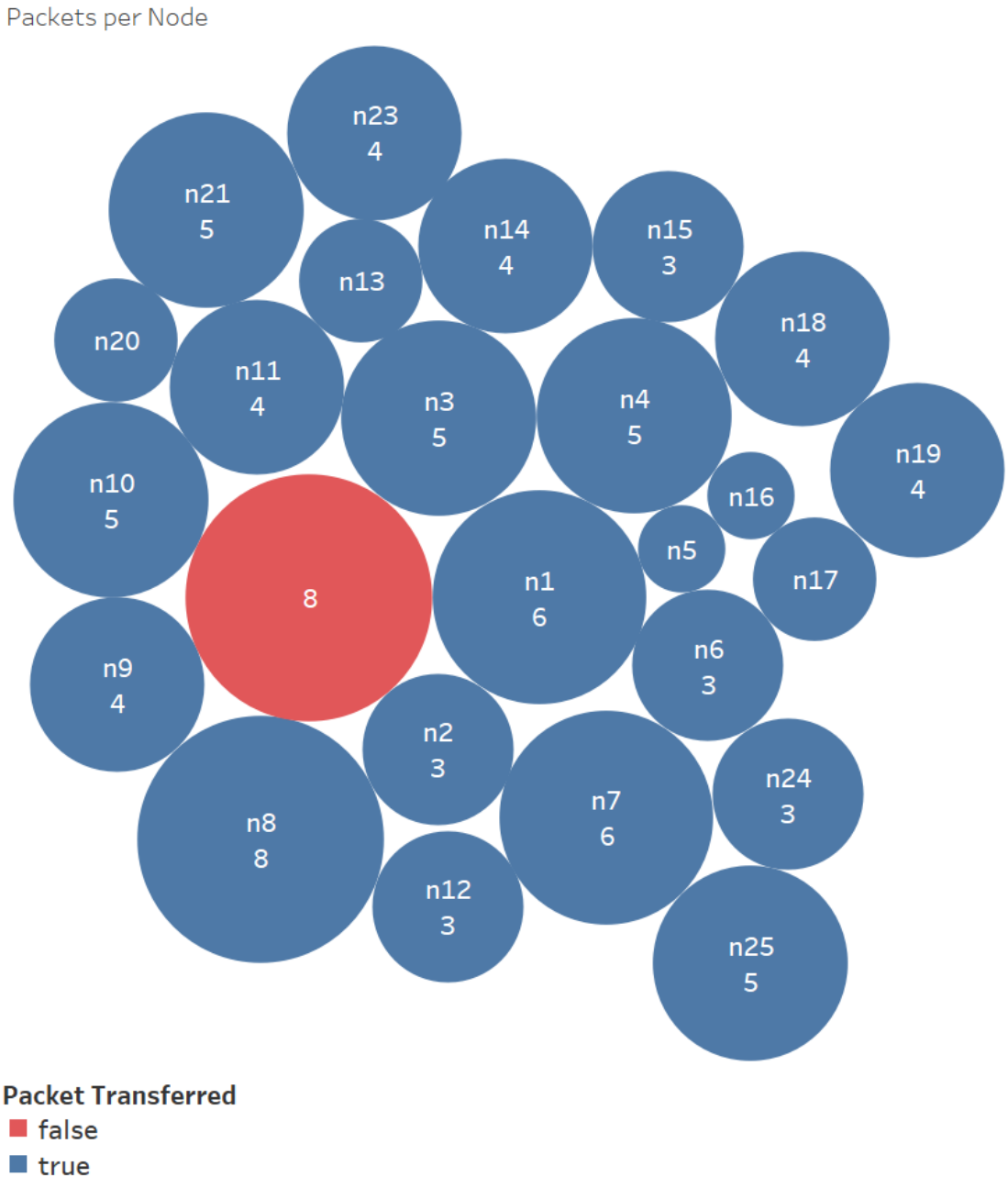

Figure 5.1.1: Packets per Node

Here, the number of each node denotes the number of packets it carries. For instances, *n3* passes 5 packets so the packet transfer rate of *n9* is 4. The

red bubble contains 8 that means, the whole process has lost 8 of the packets.

5.1.2 Packets Transferred:

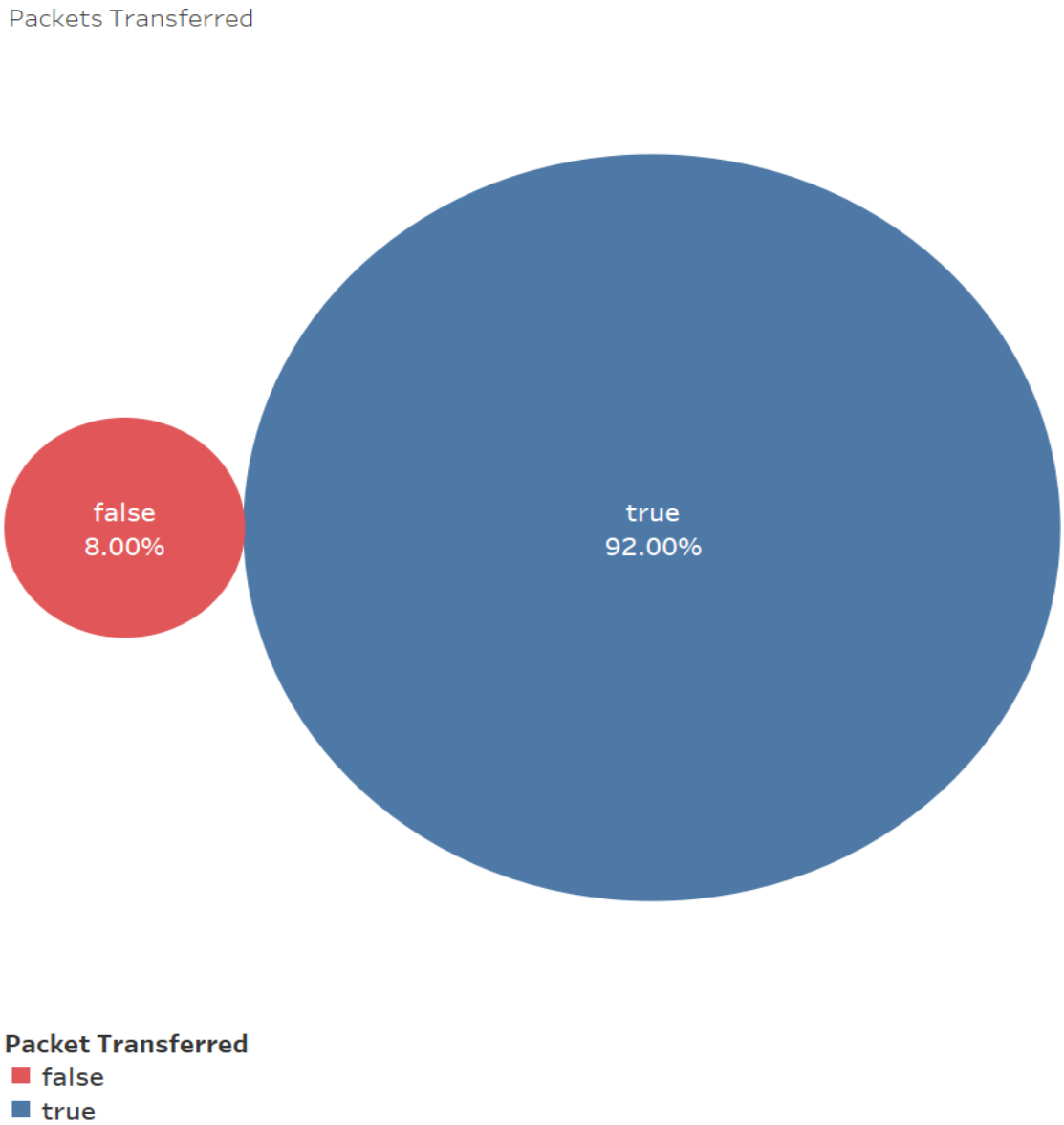

Figure 5.1.2: Packets Transferred

When we ran our network graph that has 25 nodes with 54 paths among them, we fixed our source and we change our destination randomly. For transferring 100 packets from source to various destinations and our

average packet loss is 8%. To reduce packet loss, we refresh our bandwidth buffer so that it can allocate space for next packets. Again, our network has more existing routes between two routers. So it is possible to have alternative route for every packet to move from source to destination. In our main algorithm, we have shown that, our network is changing dynamically time to time. So if a node is down because of bandwidth buffer full, the packet will find alternate route to reach its destination.

5.1.3 Hop Count to Destination:

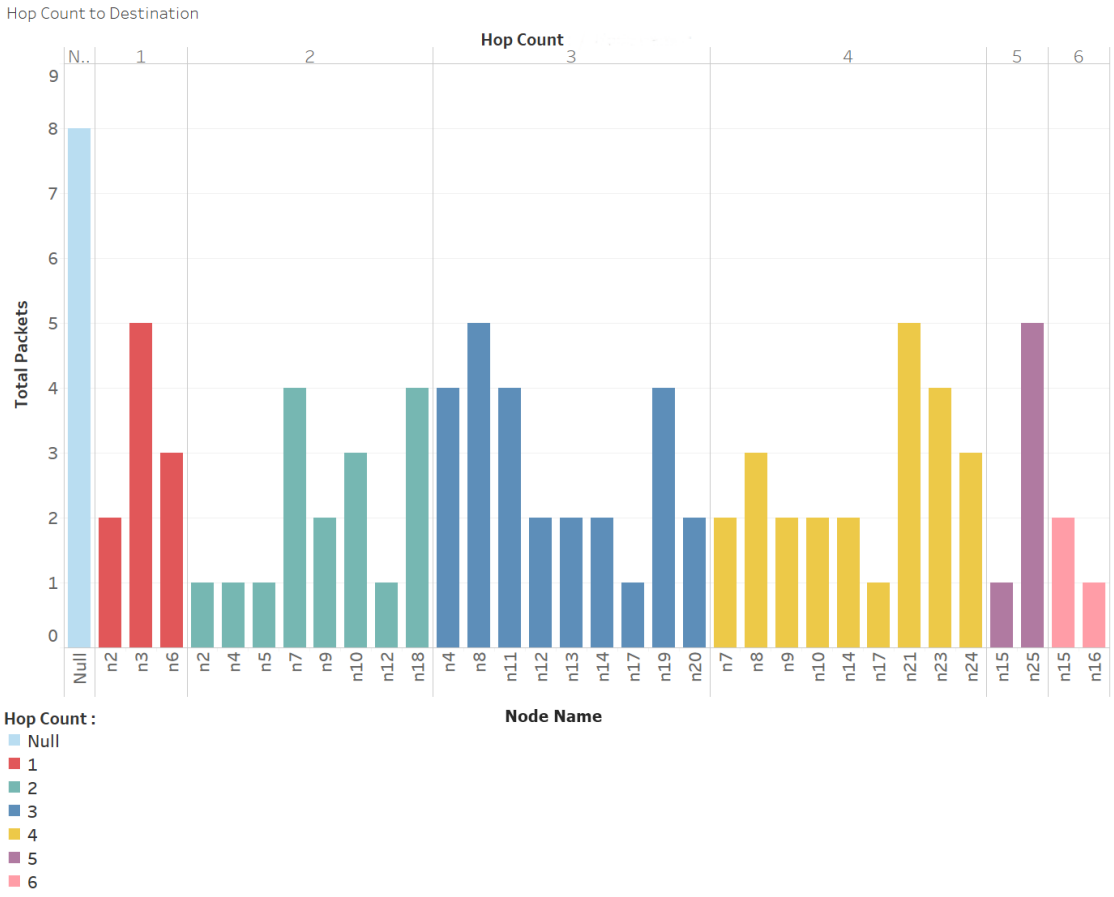

Figure 5.1.3: Hop Count to Destination

Here, we can see the number of Hops a packet needs to travel to reach the destination. We then arranged the result by the number of hops and

destination with number of packets. Like, in the column “Hop count 2” there are *n2*, *n4*, *n5*, *n7*, *n8*, *n9*, *n10*, *n12*, *n18* packets listed and at the column *n7* we can see the number of Packets are 4. So, only 4 packet reached *n7* with a hop count of 2. On the other hand, to reach *n9* node 2 packets need 2 Hops and another 2 packets needed 2 Hops.

5.1.4 Avg. Cost to Destination:

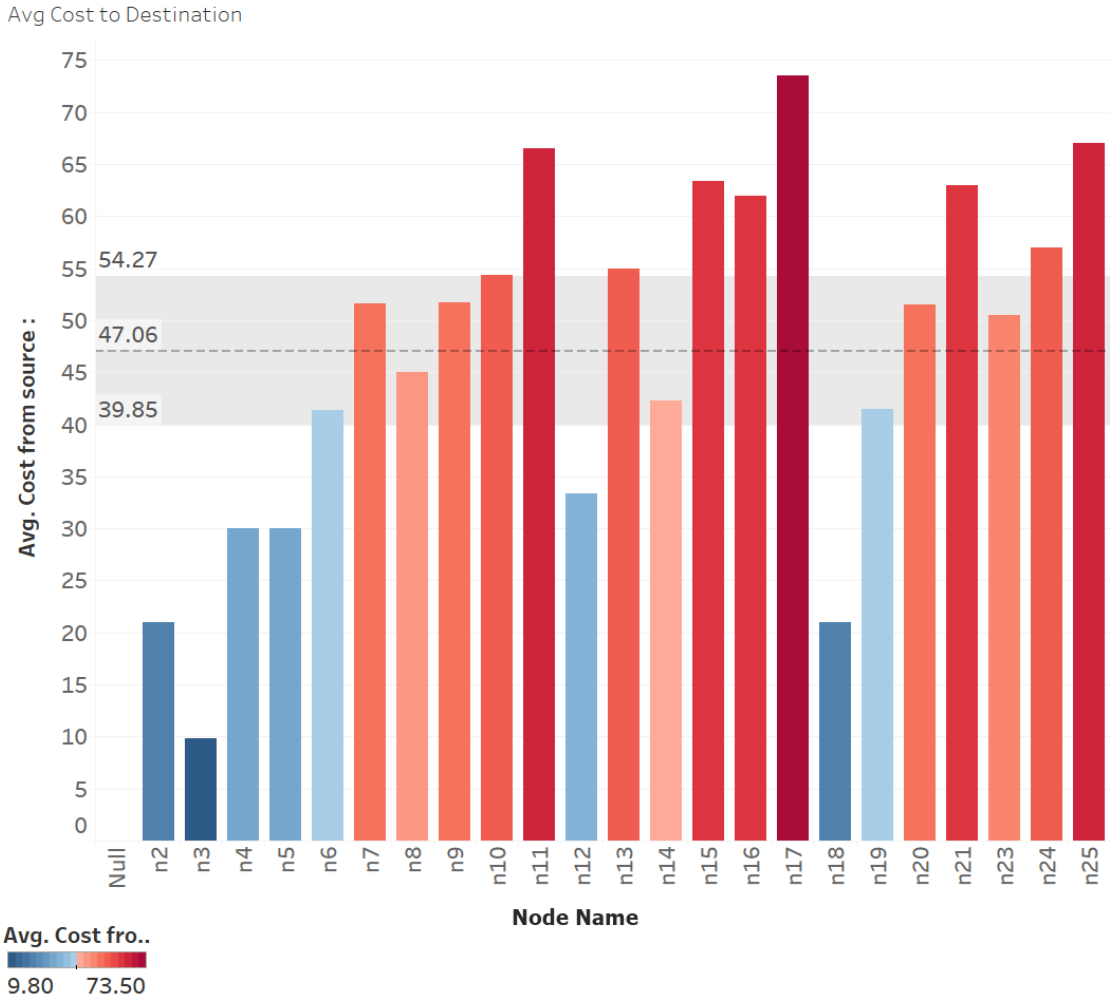

Figure 5.1.4: Avg. Cost to Destination

We calculated the Average Cost 47.06 from Source by summing up the cost for every packet to a node and then took the mean of it. We have also

put a reference line in the chart that is 47.06. The grayed area is showing the 95% confidence interval of the average. So, between 39.85 and 54.27 we will get 95% of the population.

5.1.5 Hop Count vs. Avg. Cost:

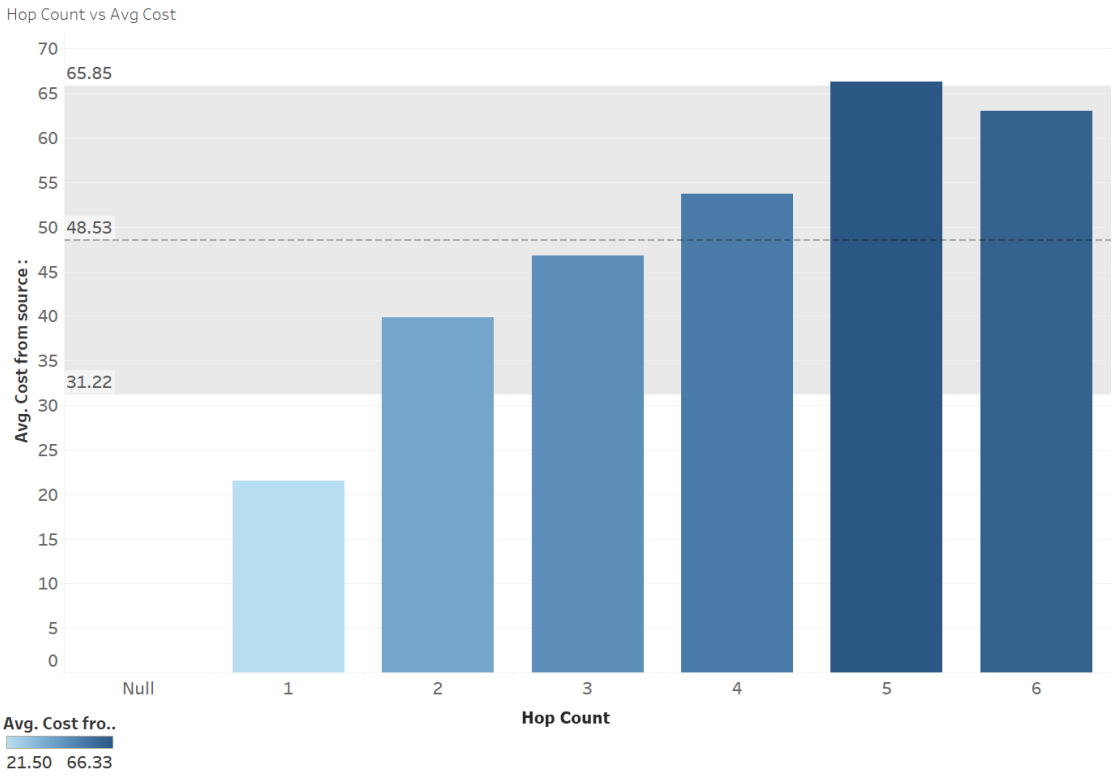

Figure 5.1.5: Hop Count vs. Avg. Cost

On the following figure, we can see the Hop Count and the Average Cost of our pre-defined network graph. We can get a relation from the result that is if the hop count is higher the cost will also be higher. So, if 1 packet travels 3 hops to reach the destination then the cost will be around 47 but if it travels 5 hops the cost will be around 65. This relation trend is true for all the packets except the last one (we can consider this as an error). In

addition, we are 95% confident that Path Cost will be between 31.22 to 65.85. As the cost is path dependent our main focus is to travel least amount of hops to reach the destination which always not possible due to the Network topology. But We tried to consider real life-like topology to get a practical result.

5.1.6 Hop Count vs. Avg. Maximum Buffer Size:

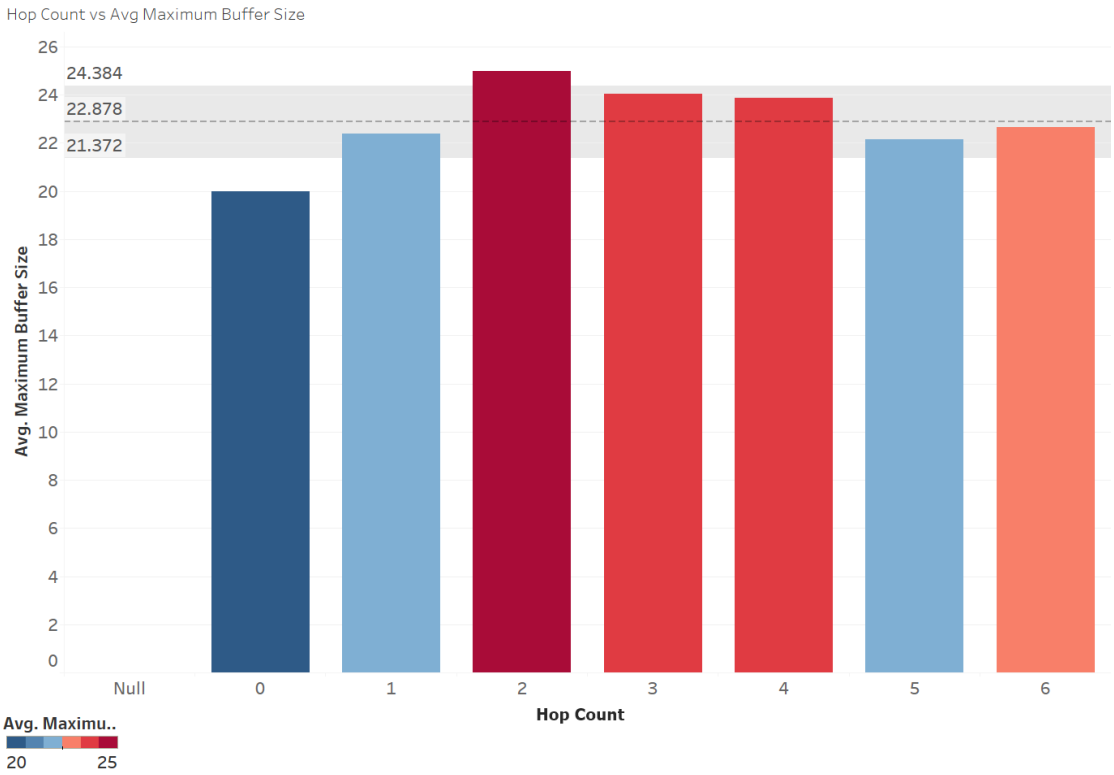

Figure 5.1.6: Hop Count vs. Maximum Buffer Size

We get an Average Maximum Buffer Size of 22.878 for all the hop count. The differences between Buffer Size is so little that we can not make any conclusion out of it. So this comparison is clearly not helping us to reach any decision.

5.1.7 Path Cost vs. Avg. Maximum Buffer Size:

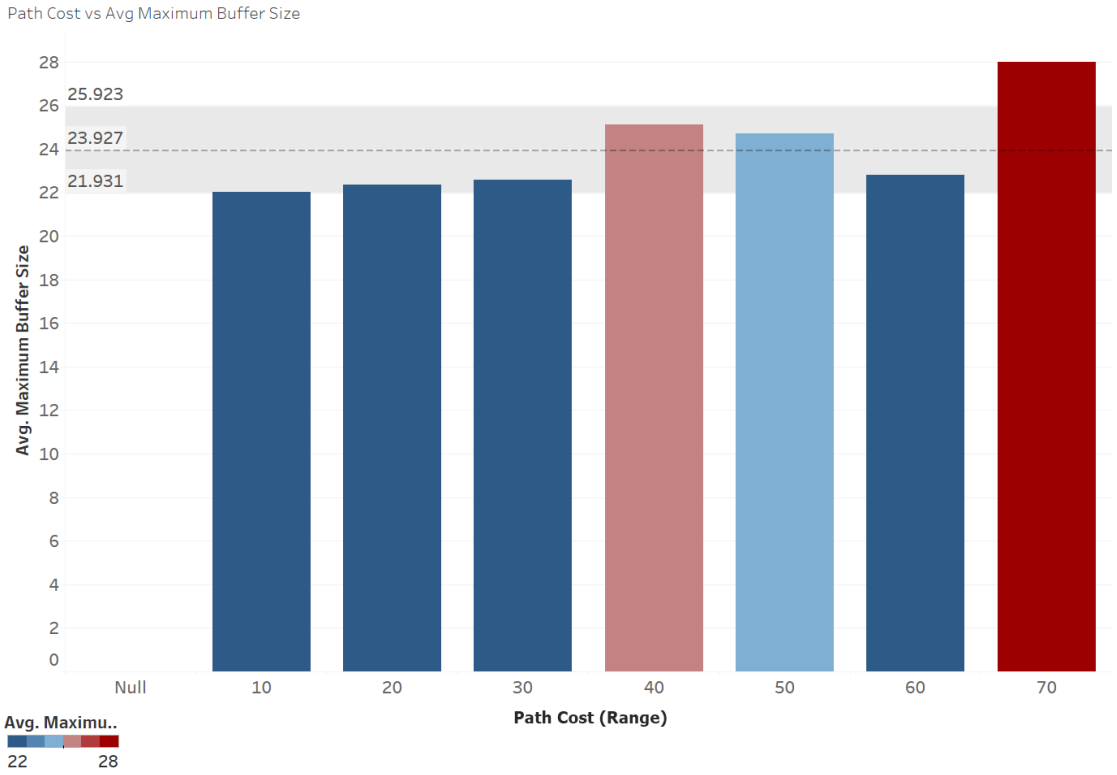

Figure 5.1.7: Path Cost vs. Avg. Maximum Buffer Size

We took a range of Cost to go to a destination from the source and made a graph with the Average Maximum Buffer Size. There is no significant relation among them but if the Maximum Buffer size is between 21.93 to 23.92 the Average cost is lower to reach a destination (assumption).

5.1.8 Path Cost vs. Avg. Remaining Buffer Size:

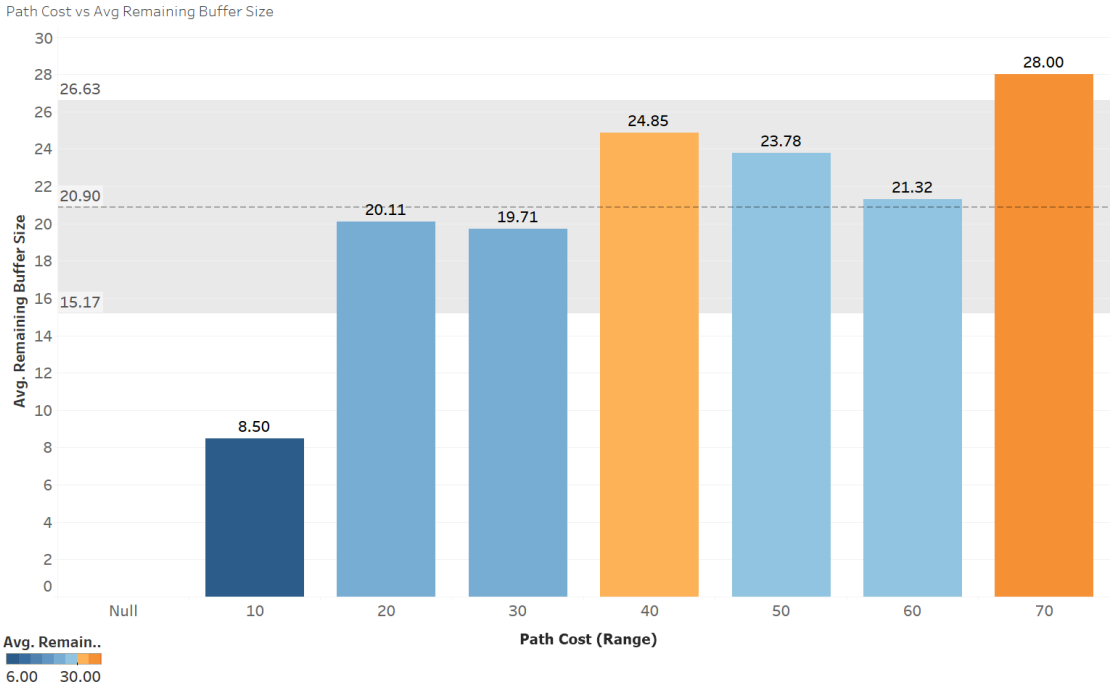

Figure 5.1.8: Path Cost vs. Avg. Remaining Buffer Size

We again tried to find out relation between Path Cost with the Remaining Buffer Size and we can see that if Remaining Buffer Size can be kept between 15.17 to 26.63, then the cost will be between 20 to 60. Packet tend to choose low cost path if less bandwidth is available.

5.1.9 Path Bandwidth vs. Hop count:

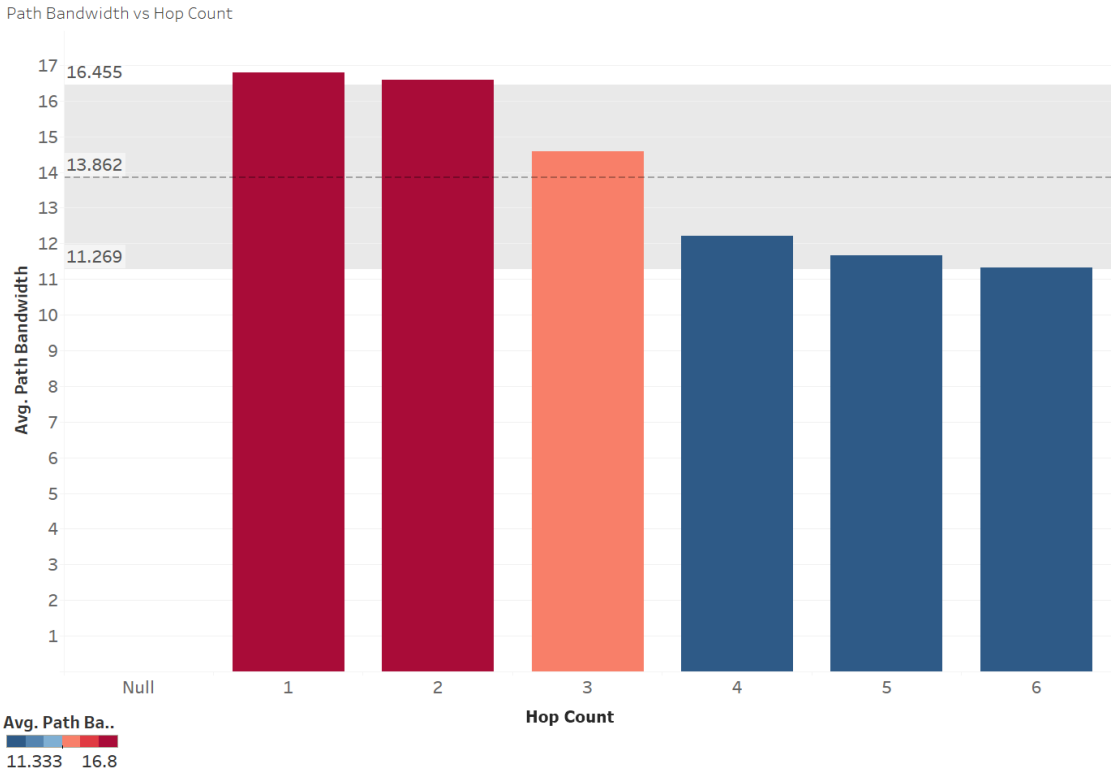

Figure 5.1.9: Path Bandwidth vs. Hop Count

Between Hop Count and Path Bandwidth if we can ensure high Bandwidth packet needs less Hop to pass. 1 hop count and 16.75 Path Bandwidth is the maximum scenario of our network graph. As the Hop Count is increasing less Bandwidth is being used. For a packet to travel from source to destination, needs 12 Bandwidth if the Hop between the source and destination is 5.

5.2 BAR ALGORITHM:

5.2.1 Packets per Node:

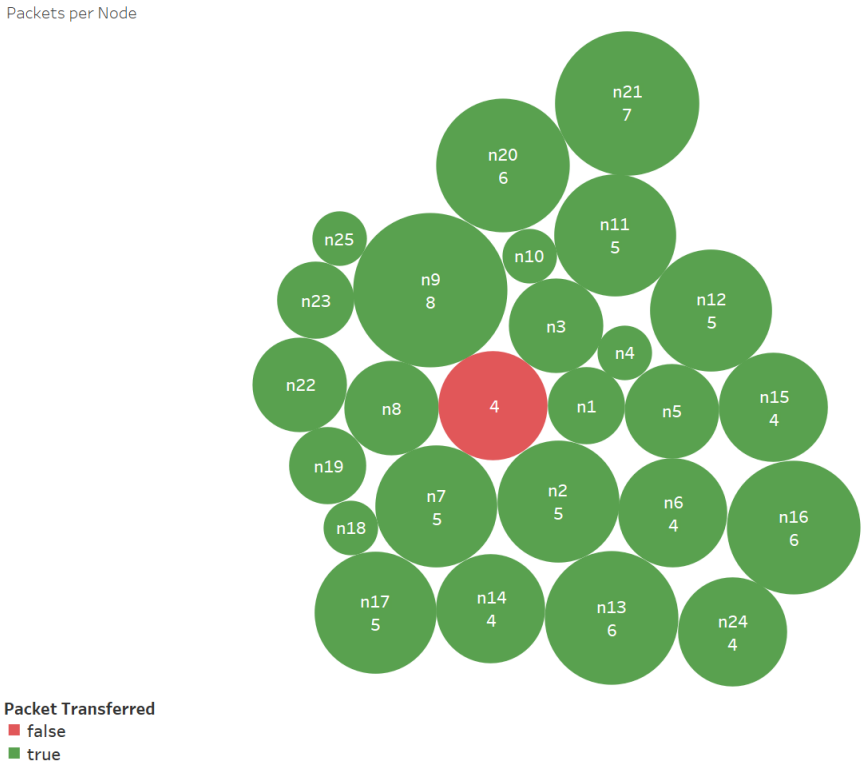

Figure 5.2.1: Packets per Node

We have kept network graph source fixed, at the same time we have set the destination randomly for every time to get near accurate results for packet transfer. Here, the number of each node denotes the number of packets it carries. For instances, *n9* passes 8 packets so the packet transfer rate of *n9* is 8. The red bubble contains 4 that means, the whole process has lost 4 of the packets.

5.2.2 Packets Transferred:

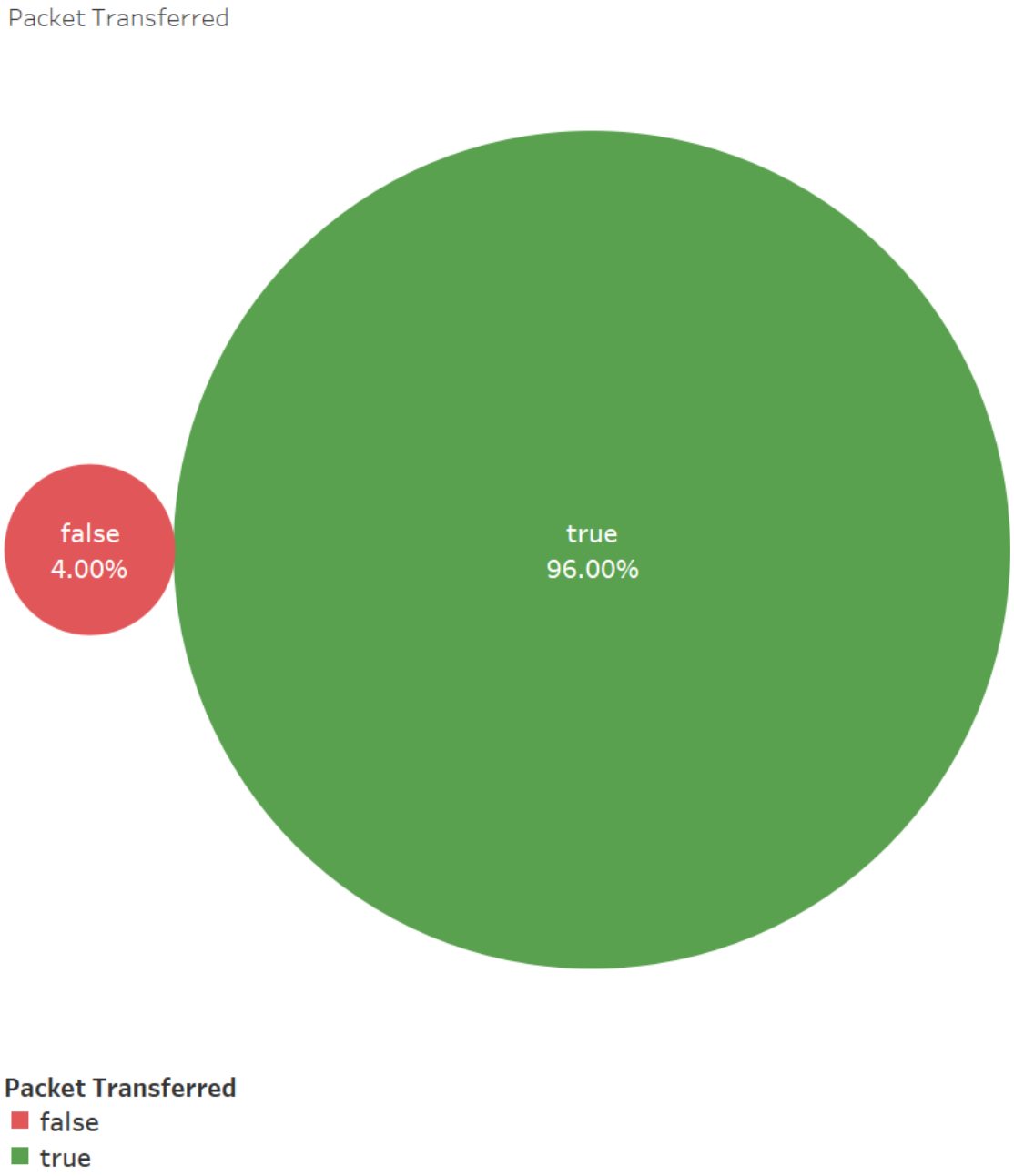

Figure 5.2.2: Packets Transferred

For transferring 100 packets from source to various destinations, our average packet loss is 4%. To reduce packet loss, we refresh our bandwidth buffer so that it can allocate space for next packets. Again, our network has more existing routes between two routers. So it is possible to have

alternative route for every packet to move from source to destination. In our main algorithm, we have shown that, our network is changing dynamically time to time. So if a node is down because of bandwidth buffer full, the packet will find alternate route to reach its destination.

5.2.3 Hop Count to Destination:

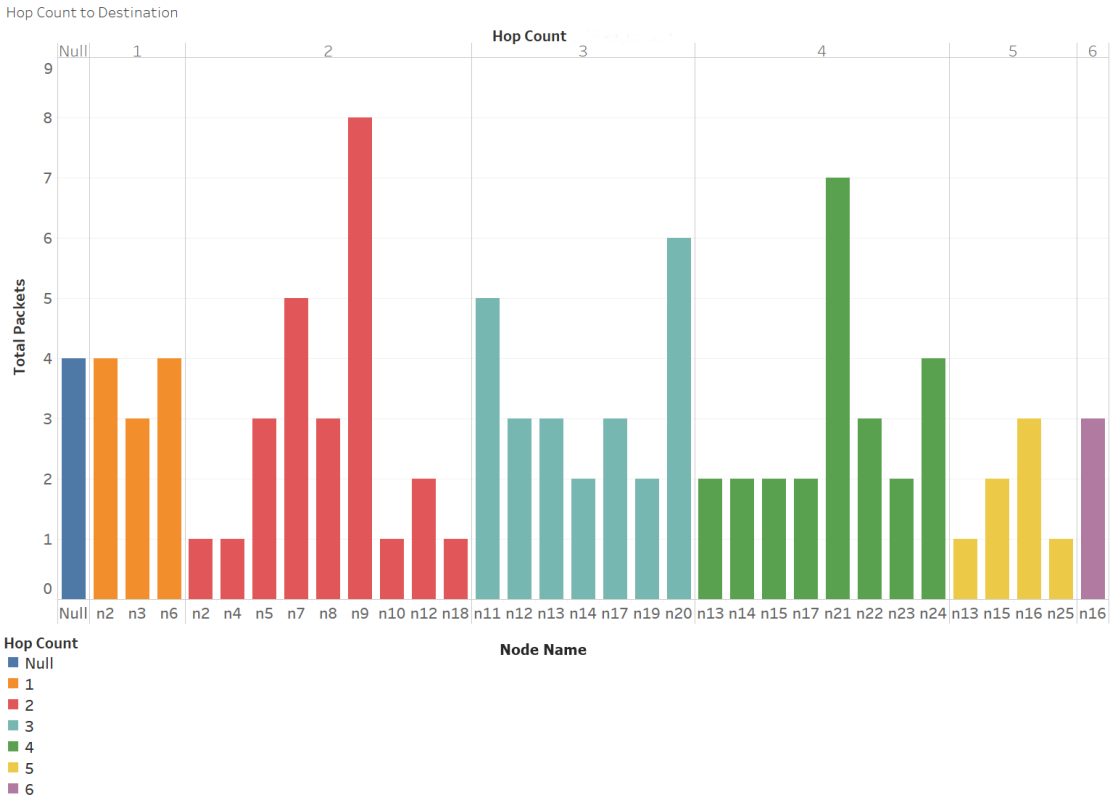

Figure 5.2.3: Hop Count to Destination

Here, we can see the number of hop (Nodes) a packet needs to reach the destination. We then arranged the result by the number of hops and destination with number of packets. Like, in the column “Hop count 2” there are *n2*, *n4*, *n5*, *n7*, *n8*, *n9* packets listed and at the column *n2* we can see the number of packets (records) are 1. So, only 1 packet reached *n2*

with a hop count of 2. On the other hand, to reach *n9* node 8 packets need 2 hop cost each.

5.2.4 Avg. Cost to Destination:

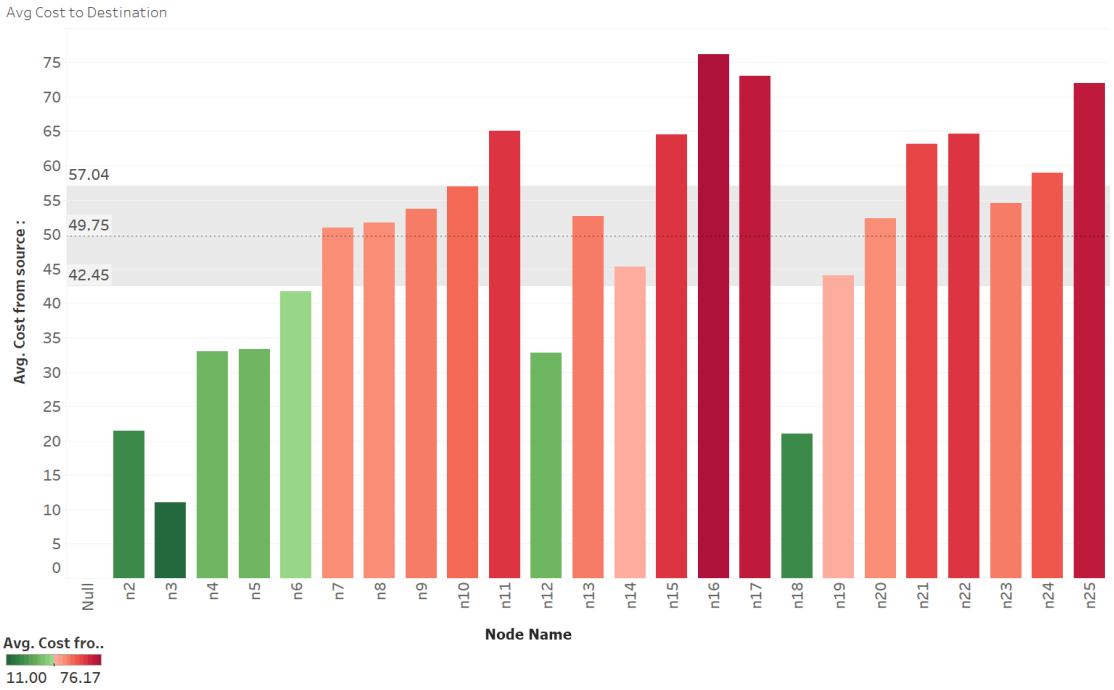

Figure 5.2.4: Avg. Cost to Destination

We can see the average cost to reach any destination. We calculated the average cost by summing up the cost for every packet to a node and then took the mean of it. We have also put a reference line at 49.75 in the chart which is the average cost to reach any node from the source *n1*.

5.2.5 Hop Count vs. Avg. Cost:

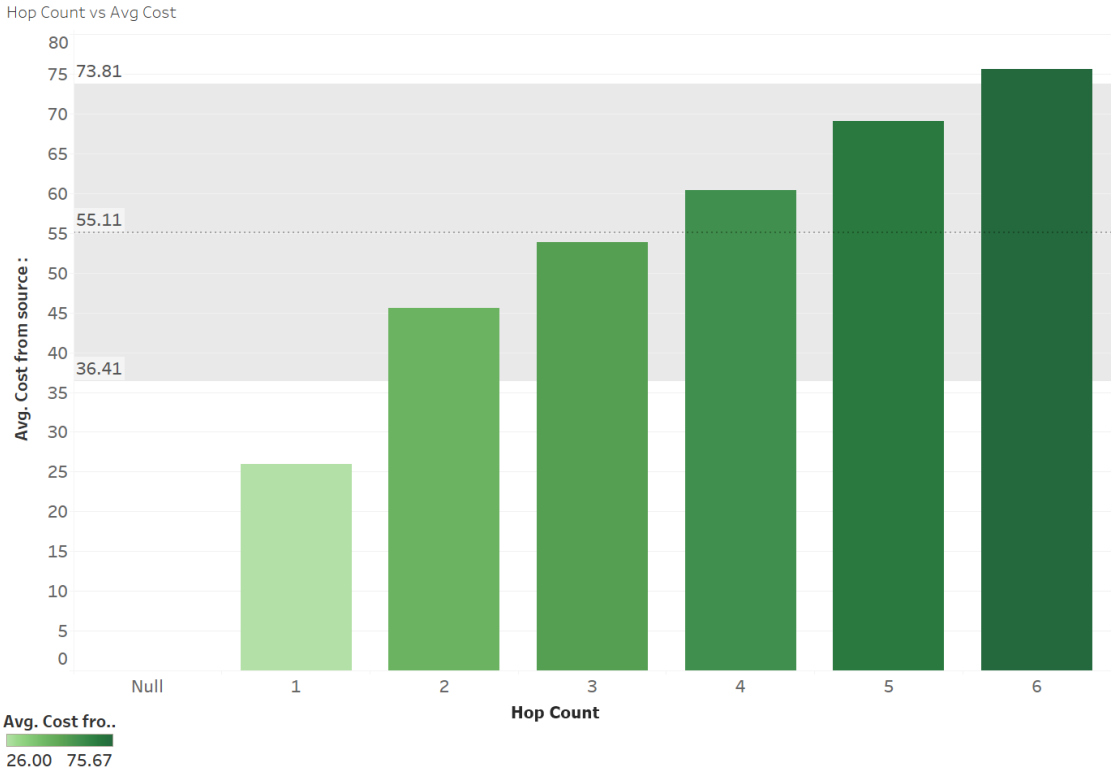

Figure 5.2.5: Hop Count vs. Avg. Cost

On the following figure, we can see the hop count and the average cost of our pre-defined network graph. We can get a relation from the result that is if the hop count is higher the cost will also be higher. So, if 1 packet travels 3 hops to reach the destination then the cost will be around 55 but if it travels 6 hops the cost will be around 75. This relation trend is true for all the packets. As the cost is path dependent, our main focus is to travel least amount of hops to reach the destination which always not possible due to the Network topology. We tried to consider real life-like topology to get a practical result.

5.2.6 Hop Count vs. Avg. Maximum Buffer Size:

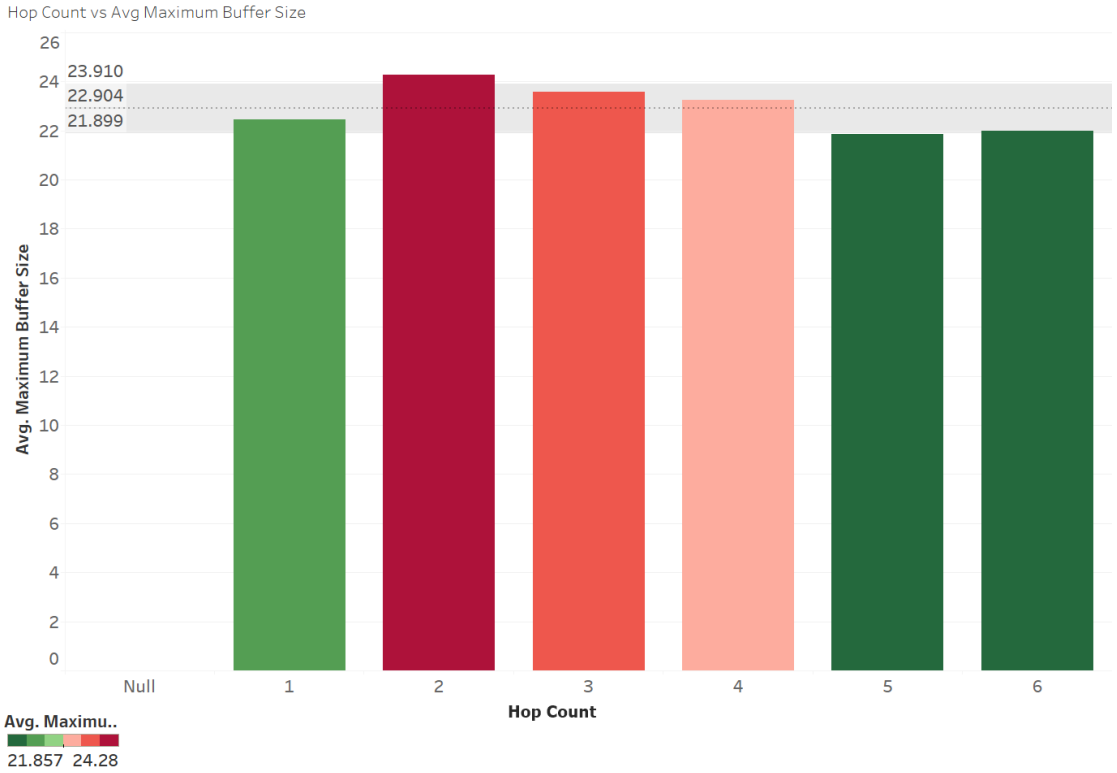

Figure 5.2.6: Hop Count vs. Maximum Buffer Size

Here we can see no special relation of Maximum Buffer Size (Flow Entries) with Hop count variation among the number of packet. For distance of hop count 3 we have Maximum Buffer Size (Flow Entries) of 19, 20, 22, 23, 25, 30. So this comparison is not helping us to reach any decision.

5.2.7 Path Cost vs. Avg. Maximum Buffer Size:

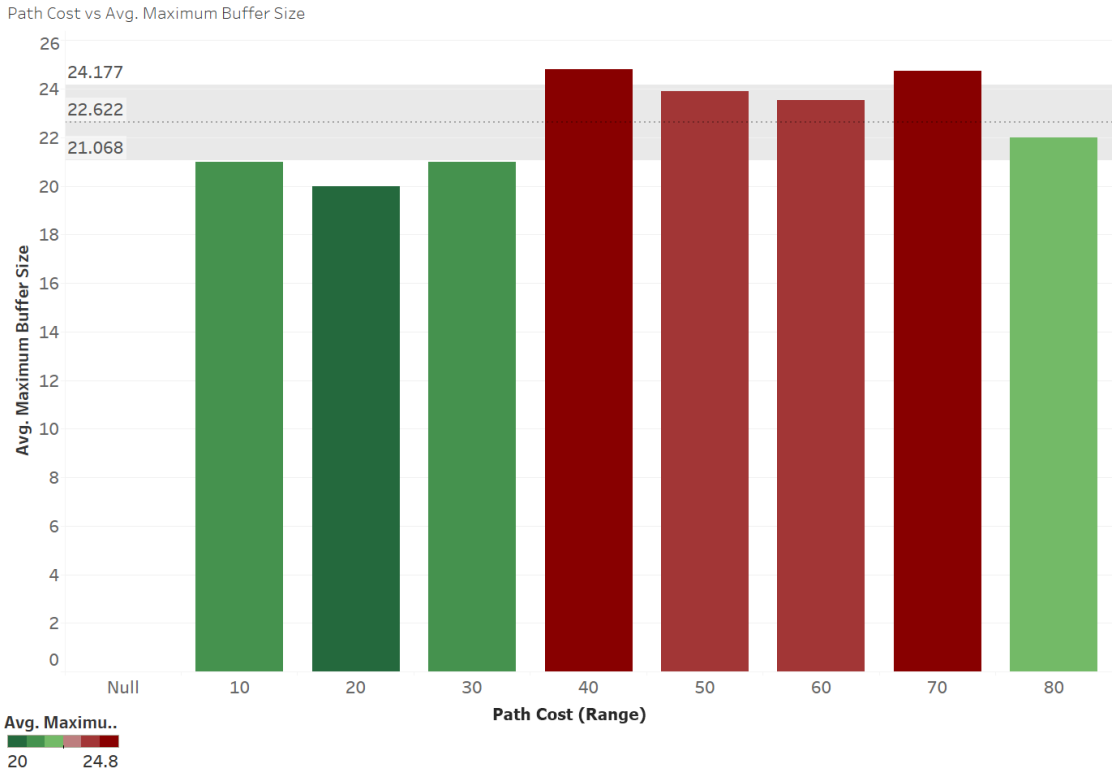

Figure 5.2.7: Path Cost vs. Avg. Maximum Buffer Size

We took a range of Cost to get a destination from the source and made a graph with the Average Maximum Buffer Size (flow entries). There is no particular relation among them. Cost does not increase or decrease with Average Maximum Buffer Size (flow entries). Therefore, we cannot come to a decision from the following figure. As our actual network changes dynamically. So we cannot predict any relation of packets' cost with Average Maximum Buffer Size (flow entries). Cost between two nodes depends on bandwidth and Buffer Size. For this reason, there is no relation between packet cost vs. Average Maximum Buffer Size (flow entries).

5.2.8 Path Cost vs. Avg. Remaining Buffer Size:

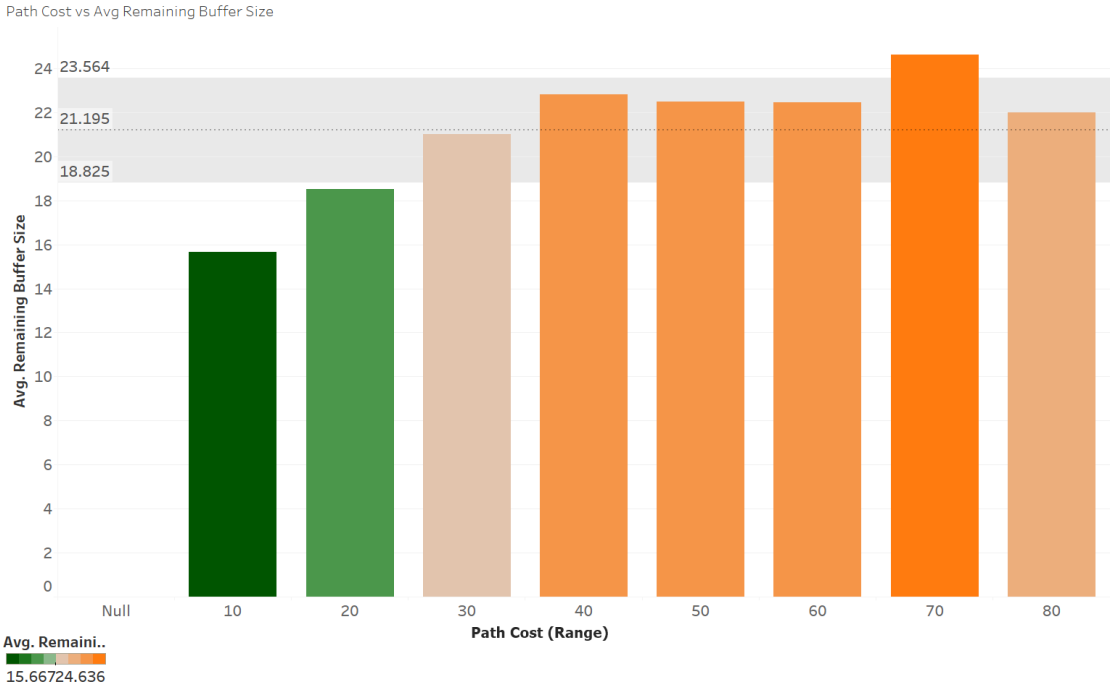

Figure 5.2.8: Path Cost vs. Avg. Remaining Buffer Size

We again tried to find out relation between the average cost range with the Remaining Buffer Size (flow entries) and we can see that if Remaining Buffer Size (flow entries) is low then the cost will get low and the cost increases with the more availability of flow entries. So packet will choose low cost path if less bandwidth is available. This trend is seen for the first few columns and later the relation is no longer exists.

5.2.9 Path Bandwidth vs. Hop count:

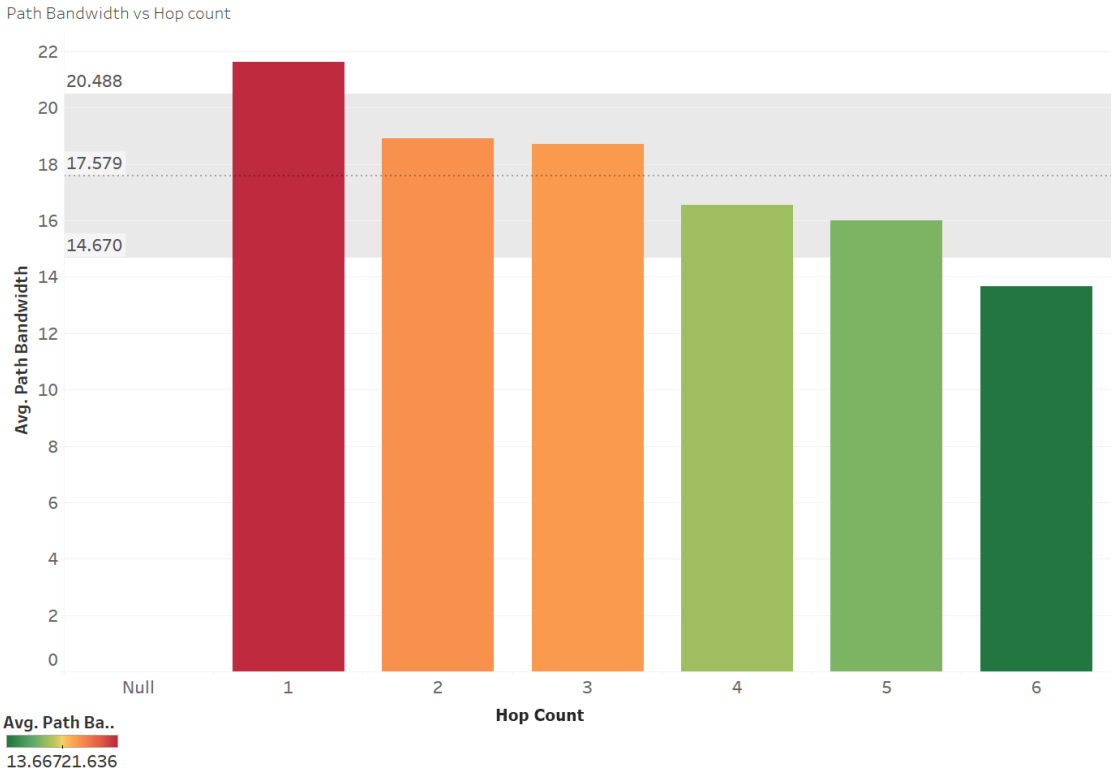

Figure 5.2.9: Path Bandwidth vs. Hop Count

Here we tried to pick up any relation between hop count with Path Bandwidth (capacity). 10 packets with 4 hop count and 17 Path Bandwidth (capacity) is the maximum scenario of our network graph. Again, there is no trend between these variables. For a packet to travel from source to destination, the existing shortest path may change time to time as we are working on a dynamic environment.

CHAPTER 6
Conclusions

Software defined network is a new concept in networking where central controller takes all the decisions for transferring packets from source to destinations. We have implemented two algorithms shortest path first (SPF) and bandwidth aware routing (BAR). From algorithm analysis, we have found that the path cost of SPF is lower than BAR. Again, the average hop count of SPF is higher than BAR. Moreover, the bottleneck bandwidth of SPF is lower than BAR. Based on network topology we can choose any of those algorithms.

6.1 Future Plan:

As in SDN, the central controller has the whole information about the network graph and it determines the routing route for any incoming packet, so routing algorithms can be changed dynamically based on packet types. So we have a plan to introduce machine learning in this scenario. Using machine learning we can determine which kind of packets are transferred most in particular time. For example, most of the users are using Skype at any particular time in a day. Machine learning will analysis the time and determine the preferred algorithms for the requested packets. It is making the networking more efficient by making the best use of resources.

6.2 Challenges:

Software Defined Networking is a new field in networking. Most of the functions are new to us. We have studied about these functionalities to have

a clear concept. To develop this project, we have faced the given challenges.

- We have created a network graph to check our simulations working according to SDN protocol or not. We have kept 25 routers and 54 routes in network.
- Network is dynamic as costs and bandwidths of routers may change time to time. We have taken values randomly and calculated SPF and BAR from those random values.
- In dynamic network, a router may not aware if its next router is down or not. If so then, the route will be down. However, we have developed our system that checks the entire graph for packet transfer. If the next router is down, then central server will reselect a new path for packet.
- We have calculated the shortest route by SPF and BAR algorithm and generate results into JSON file format. With JSON, we have developed charts and graphs for comparison between SPF and BAR algorithm.
- To simulate 25 routers and 54 routes, we have generated 100 packets to move from single source to multiple destinations. It takes 2-3 seconds to transfer these packets. 0.025 - 0.03 second for each packet transfer. We get this approximate result from our simulation. Therefore, we can speed up packet transfer.

Most importantly, our main challenge is to implement SPF and BAR algorithm in SDN environment.

REFERENCES

[1] Open Networking Foundation, "Software defined networking: the new norm for networks," Web white paper, retrieved Apr. 2015.

[2] T. H. Cormen, C. E. Leiserson, B. L. Rivest, and C. Stein, “Introduction to Algorithms,” 3rd ed., Cambridge: MIT Press, 2009, pp. 658–664.

[3] Q. Ma, and P. Steenkiste. "On path selection for traffic with bandwidth guarantees," IEEE international conference on network protocols, pp.191-202, 1997.

[4] E. W. Dijkstra, "A note on two problems in connection with graphs," Numerische Math. 1, pp. 269-271, 1959.

[5] Jang-Ping Sheu, Quan-Xiang Zeng, R. Jagadeesha and Yeh-Cheng Chang, "Efficient unicast routing algorithms in Software-Defined Networking," 2016 European Conference on Networks and Communications (EuCNC), Athens, 2016, pp. 377-381. doi: 10.1109/EuCNC.2016.7561066

[6] R. Boutaba, W. Szeto, and Y. Iraqi. "DORA: Efficient routing for MPLS traffic engineering,"

[7] Syed Asad Hussain, Shuja Akbar, Imran Raza, "A dynamic multipath scheduling protocol (DMSP) for full performance isolation of links in software defined networking (SDN)", Recent Trends in Telecommunications Research (RTTR) Workshop on, pp. 1-5, 2017.

[8] Yi-Chih Lei, Kuochen Wang and Yi-Huai Hsu, "Multipath routing in SDN-based Data Center Networks," 2015 European Conference on Networks and Communications (EuCNC), Paris, 2015, pp. 365-369. doi:10.1109/EuCNC.2015.7194100

[9] H. Ren, X. Li, J. Geng and J. Yan, "A SDN-Based Dynamic Traffic Scheduling Algorithm," 2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Chengdu, 2016, pp. 514-518. doi: 10.1109/CyberC.2016.103

[10] S. H. Shen, L. H. Huang, D. N. Yang and W. T. Chen, "Reliable multicast routing for software-defined networks," 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, 2015, pp. 181-189.doi: 10.1109/INFOCOM.2015.7218381

[11] Y. Yang, J.K. Muppala, S.T. Chanson, "Quality of service routing algorithms for bandwidth-delay constrained applications," Ninth International Conference on Network Protocols, pp. 62-70, Nov. 2001.

[12] S. Tomovic, N. Prasad, I. Radusinovic,"SDN control framework for QoS provisioning," TELFOR, pp.111-114, Belgrade, Nov. 2014.

[13] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDNbased application-aware networking on the example of youtube video streaming," EWSDN, pp. 87–92, Oct 2013.

[14] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," EWSDN, pp. 81–86, Oct 2013.

[15] "Openflow switch specification v1.3.4," 2014.

https://www.opennetworking.org/sdn-resources/onf-specifications

[16] Estrin et al., "Protocol independent multicast-sparse mode (PIM-SM): Protocol specification," IETF RFC 2362, May 1998.

[17] H. E. Egilmez, S. Civanlar and A. M. Tekalp, "An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks," IEEE Transactions on Multimedia, vol. 15, no. 3, pp. 710-715, April 2013.

[18] J. Yen, "Finding the k shortest loopless paths in a network," Management Science 17.1, pp. 712-716, 1971.

[19] S. Tariq and M. Bassiouni, "QAMO-SDN: QoS aware Multipath TCP for software defined optical networks," 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, 2015, pp. 485-491. doi: 10.1109/CCNC.2015.7158023

[20] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks," 2016 IEEE NetSoft

Conference and Workshops (NetSoft), Seoul, 2016, pp. 303-311. doi: 10.1109/NETSOFT.2016.7502426

[21] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, P. Yalagandula, "Automated and scalable QoS control for network convergence," INM/WREN"10, pp. 1-6, San Jose, CA, Apr. 2010.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” in Proceedings of ACM SIGCOMM Computer Communication Review, pp. 69–74, New York, USA, April 2008.

[23] S. Chen, "Routing Support for providing guaranteed end-to-end Quality-of-Service," PhD thesis, Computer Science, University of Illinois, Urbana-Champaign, 1999.

[24] Y. Yang, J.K. Muppala, S.T. Chanson, "Quality of service routing algorithms for bandwidth-delay constrained applications," Ninth International Conference on Network Protocols, pp. 62-70, Nov. 2001.

[25] T. He, D. Goeckel, R. Raghavendra, and D. Towsley, “Endhost-Based Shortest Path Routing in Dynamic Networks: An Online Learning Approach,” in Proceedings of IEEE INFOCOM, pp. 2202–2210, Turin,Italy, April 2013.

[26] M. Al-Fares, A. Loukissas, and A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” in Proceedings of ACM SIGCOMM, pp.63–74, Seattle, USA, August 2008.

[27] B. Shen, B. Hao, and A. Sen, “On Multipath Routing Using Widest Pair of Disjoint Paths,” in Proceedings of Workshop on High Performance Switching and Routing, pp. 134–140, Phoenix, USA, April 2004.

[28] Y. Li and D. Pan, “OpenFlow Based Load Balancing for Fat-Tree Networks with Multipath Support,” in Proceedings of IEEE ICC, pp. 1–5, Budapest, Hungary, June 2013.