Constraint-Based Rerouting mechanism to address Congestion in Software Defined Networks

Vijaya Durga Chemalamarri, Robin Braun and Mehran Abolhasan
School of Electrical and Data Engineering
University of Technology Sydney, Sydney, Australia
vijaya.d.chemalamarri@student.uts.edu.au

***Abstract*—In this paper, we propose a traffic rerouting mechanism to address congestion in Software-Defined networks. We employ back-tracking and constraint propagation techniques to find alternate paths to reroute multiple active flows simultaneously. Cost function is based on standard deviation of link-loads. We then compare traffic distribution and link utilisation with and without rerouting active flows. We measure and compare network performance using parameters such as total rate of transfer, jitter, and packet loss with that of Shortest Path First with no rerouting. Our proposed solution produces lower jitter, packet drops, and higher transfer rate. We finally conclude the paper by making observations and discussing the scope of the future work.**

***Index Terms*—SDN, Rerouting scheme, Back-tracking**

I. Introduction

Flow scheduling algorithms establish new flows based on the current state of the network. Legacy algorithms such as Equal Cost Multipath Routing (ECMP) schedule new flows on available paths in a round-robin manner. Techniques such as Weighted Cost Multi-Path Routing(WCMP) assigns weights to links and provisions flows according to the weight of the paths. While flow admission strategies provision new flows on links with less load, they cannot guarantee a congestion-free network. Rerouting few flows to delay the onset of congestion is necessary. While efficient traffic rerouting can reduce or delay the onset of congestion and help distribute traffic across various links, inefficient rerouting can lead to congestion in other parts of the network while alleviating congestion on other links. To avoid such displacement of congestion, a global view of network state is necessary.

Software-Defined Networking allows a logical separation and centralisation of the network's control-plane from the data-plane. A logically centralised control-plane configures flow-rules or instructions on the data-plane. Switches use flow-rules to handle packets. The controller configures flow-rules on the data-plane using OpenFlow(OF) or similar southbound protocol. A flow-rule is composed of matching parameters(fields) and actions. The interactions between the data and control-planes can be proactive or reactive. To proactively/reactively install flow-rules, the controller needs to maintain network information such as link-loads, number of flows and size of flows.

Rerouting of network traffic is a well-researched topic. While in traditional networks, link failures trigger traffic reroutes, SDN is a perfect platform to revisit this problem as a centralised control-plane allows for maintaining a global perspective of the network.

Rest of the paper is organised as follows: Section II briefly lists related work and motivation behind this work. Section III introduces our solution, defines keywords, presents our rerouting mechanism and various components and implementation details. Testing results and observations are presented in Section IV. We conclude in Section V.

II. Related Work and Motivation

We now review some literature on multi-path routing techniques that focus on flow admission and rerouting.

Weighted Cost Multipath Routing (WCMP) [1] distributes traffic amongst the available next-hop nodes in proportion to available link capacity.

Authors of [2] propose FlowBender that enables end-host to drive flow-level load balancing scheme by using Explicit Congestion Notification(ECN) for congestion detection and rerouting large flows.The solution uses the Equal Cost Multi-Path(ECMP) hashing mechanism to hash against a flexible field in the packet. This field is updated in case of congestion, thus allowing ECMP to handle packets differently.

In [3], authors propose OFLoad scheme that separates elephant flows from mice flows and aims to minimise network congestion by routing the elephant flows on a single shortest path. Mice flows are aggregated and routed using WCMP.

[4] proposes Mahout, which introduces a Shim layer at the end-host to detect elephant flows. Switches use the Differential Services(DS) Field of a packet to notify the controller of elephant flows. The controller places elephant flows on the best path.

In [5], authors aim at maximising aggregated network utilisation by scheduling flows dynamically. [6] proposes Niagara that uses wild card rules to split aggregate incoming traffic on the same set of next-hop nodes based on a weight vector.

In [7], weights are assigned to links and new flows are configured on the path with the least weight. In [8], Dijkstra's algorithm finds multiple equal length paths. In the event of

congestion, higher priority flows are rerouted to links with the least cost and form the shortest path.

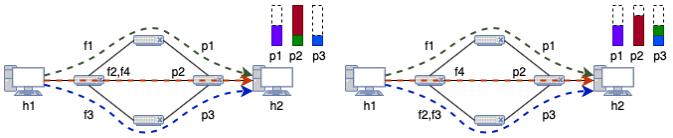
In [9] authors describe a network monitoring module that monitors the data-plane every second. A load distribution module calculates the amount of load to rerouted to backup paths.

Unlike the above stated works, in this paper we propose an algorithm to reroute multiple flows simultaneously to alleviate congestion while avoiding congestion displacement.

A. Motivation

We now discuss the motivation behind this work. Flow-scheduling algorithms distribute traffic based on current link capacity, hence require real-time network state updates. Since, the controller does not monitor the network upon installing a flow-rule,the controller provisions all the flows on the perceived best path at a given instance as shown in Fig.6(b).

Rerouting of flows can free-up resources for flows with higher demand. In Fig.1(a), active flows $f1$, $f2$ and $f3$ are provisioned on paths $p1$, $p2$, $p3$ respectively A new flow $f4$ is provisioned on perceived best path $p2$. The desired behaviour would be as shown in Fig.1(b) where $f2$ are rerouted via $p3$ to free-up resources for $f4$ on $p2$.Test1 in Section IV verifies this effect.



(a) pre-rerouting (b) post-rerouting

Fig. 1: Active flow Rerouting

Rerouting solutions usually reroute a critical flow or a flow with the highest flow-demand that might not always yield a solution due to reasons such as flow is no longer active, or insufficient resources on alternate paths. On the other hand, moving flow with moderate demand can free resources. Hence, in this work, we propose to reroute multiple active flows simultaneously on alternate paths.

Our proposed solution combines flow scheduling with periodic monitoring and rerouting of multiple flows instead of a single flow. To reroute multiple flows on alternate paths, we need to ensure that rerouting of one flow does not starve other flows of resources and does not displace congestion to other parts of network while using multiple paths to route flows. We propose a constraint-based rerouting method to reroute to minimise the standard deviation of traffic on multiple paths between a set of nodes.

III. Proposed Solution

In this section, we introduce some definition, followed by description of application components and operation.

Definitions

1) Flow: A flow is denoted by source ip address(src_ip), destination ip address(ds_ip), port(pr) tuple.
2) Path: If v is the set of all nodes in a fully connected network, and l is the set of all links, then a path p between source src and destination nodes dst is a subset of nodes v_p, sourcing at src and sinking at dst, connected via subset of links l_p
3) Residual bandwidth of a link: is defined as the available bandwidth of the link l. If bw_l is the bandwidth of the link, and current load on the link is cl_l, then residual bandwidth on link is given by $rbw_l = bw_l - cl_l$
4) Current load of a link: If tx_{t-1},tx_t are a port's transmission counters at earlier timestamp $t-1$ and current timestamp t respectively and current load of a link,cl_l is calculated as $(tx_{t-1} - tx_t)/pi$ where polling interval pi is the time interval between two captures
5) Path-capacity: If $rbw_1, rbw_2, rbw_3 ... rbw_n$ are the residual bandwidth along each link in path p, A path's available capacity is determined by bandwidth of the bottleneck link in the path given by $min(rbw_1, rbw_2, rbw_3 ... rbw_n)$

   Such a definition of path capacity, will allow the algorithm to not provision a path with a flow beyond the capacity currently offered by the most congested link in the path.
6) Path-load: If $cl_1, cl_2, cl_3 ... cl_n$ are current loads on links of a path p,then load of the path pl is $max(cl_1, cl_2, cl_3 ... cl_n)$
7) Flow-demand: flow-demand estimates the bandwidth requirement for a flow based on the flow stats collected from a switch. If f_{t-1},f_t are flow stats of a flow f at time instances $t-1$ and t respectively then flow-demand (fd) is given by $f_{t-1} - f_t/pi$ where pi is polling interval
8) Link Threshold: Link threshold is set at 80% of link capacity, upon which the link is flagged as a congested link.
9) Link-load constraint: If links with load $cl_1, cl_2, cl_3 ... cl_n$ form a path p with path-load pl,a flow f of flow-demand fd can be provisioned on this path, if and only if there exists no link whose link-load exceeds link threshold due to action $pl + fd$

   That is, when a flow is provisioned on a path, no link's load should exceed the link threshold. Such a constraint prevents the algorithm from displacing congestion from one link to another link.

A. Application components

1) Network Monitoring

Maintaining an updated network state is essential to our solution. Our application uses information from OF Flow-

Stats, OpenFlow(OF) Port-Stats messages to compute flow-demands, find paths to forward packets on, calculate the current link-loads, and identify congested links and reroute flows. The application polls network every 3 secs to gather port and flow stats.

2) New flow handler

Similar to existing solutions, our application employs the best-fit algorithm to handle new-flows by choosing a path with maximum path-capacity to provision the flow. Upon receiving a new flow, the application computes all possible paths and corresponding path-loads between the source and destination OF switches and chooses the path that offers maximum path-capacity. We employ a simple depth-first search algorithm to find all possible paths between OF switch pairs. Finding paths between a set of OF devices is a one-time activity. Path-loads and path-capacities are updated in every monitoring cycle.

3) Rerouting active flows

The network monitoring module triggers the rerouting process upon detecting link congestion. A link is identified as a congested link if the link-load exceeds 80% threshold. We have chosen this value to avoid frequent rerouting (observed for threshold set to 70%) and delayed or no rerouting (observed for threshold set to 90%). To reroute flows, we need to (i) choose flows to reroute and (ii) the paths to reroute on. To answer (i), we choose an arbitrary number of flows {F} randomly that are currently using the congested link such that $sizeof\{F\} >= 1$ allowing the application to consider flows of different flow-demands.

That brings us to (ii) finding paths to reroute the flows on. One of the risks of rerouting load is the displacement of congestion from one part of the network to another. All the paths onto which the selected flows can be rerouted make up the feasible set of paths. Not all paths can accommodate the new flow-demands, and though some paths can accommodate a flow, provisioning the flow might congest the path for other active flows. For instance, consider the Fig.3. By provisioning $f1$ on $p1$, we can no longer reroute or provision $f2$. Thus, we need to find an assignment where all the flows in $\{F\}$ have a path to be provisioned on. We employ the back-tracking and pruning mechanism with constraint checking, as explained in the next section to find such an assignment.

4) Back-Track with pruning search

Classical search techniques find a solution by systematically searching through a tree. The algorithm backtracks and navigates through new branches until a solution is found. Navigating all possible nodes is resource consuming. Backtracking search [10] with constraint propagation is a systematic search mechanism where a partial solution is extended and any assignment that fails to satisfy a constraint results in pruning of the entire branch and the algorithm backtracks to navigate and find alternate solutions. Formally, a backtracking problem can be defined as below:

If $\{V\}$ is the set of variables and set $\{D\}$ is the domain of all possible values a variable can take, then a solution is found if all variables in V have a legal value assigned from D. A legal value is a value that satisfies constraints [10]. Backtracking search assigns a variable $v1$ a legal value $d1$ from $\{D\}$, the algorithm then assigns next unassigned variable $v2$ a value $d2$. This process continues as long as constraints are satisfied, and $\{D\}$ has legal values to assign it to variable vi and search stops when a solution is found. Upon detection of a assignment that fails constraint or when a variable has no legal values for assignment, a branch is considered as not suitable and pruned entirely and allows the algorithm to backtrack early as shown in Fig.2.

Thus, if $\{F\}$ is set of all flows demands and $\{P\}$ is the set of all possible paths for flows in F, restating the problem as 'How do we assign 'n' flows to 'p' paths such that the link-load constraint is satisfied?' The algorithm backtracks when link's load exceeds the threshold.

Since a single solution might not necessarily yield the best solution, we extend the algorithm to continue searching for other possible solutions. 'Best' solution here refers to a solution that minimizes the cost function. Thus, by checking all links-load constraint, we allow the algorithm to backtrack early and prune the search space to build a feasible set of all possible solutions. A solution is found when all the chosen active flows have a path to be configured on.

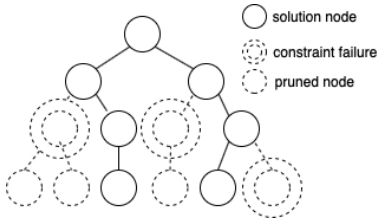


Fig. 2: Backtracking and pruning

Sample example with two flows and 2 possible paths has been provided in Fig.3

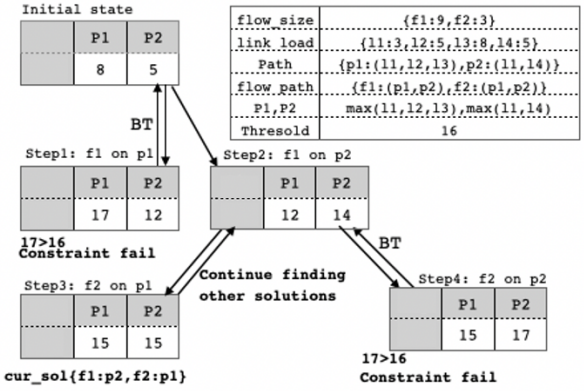


Fig. 3: Constraint propagation and backtracking

5) Cost-Function:Standard Deviation/Mean

Flows can be rerouted to meet different goals. Goals can be formulated as cost functions. For the current work, we use and minimise standard deviation of path-loads across a set of paths as cost function.

B. Operation

Upon receiving a table-miss packet from the data-plane, the controller application computes all possible paths and corresponding path-loads and available path-capacities between source and destination. As mentioned earlier, finding all paths between a set of nodes is a one-time activity as the physical topology does not change unless in the instances of link failure. Path with maximum path-capacity is chosen and flow-rules are configured proactively on the data-plane. At this instance, the path-capacities are based on the earlier network stats collected during earlier monitoring cycle. The monitoring module monitors the network periodically and updates the stored network state.

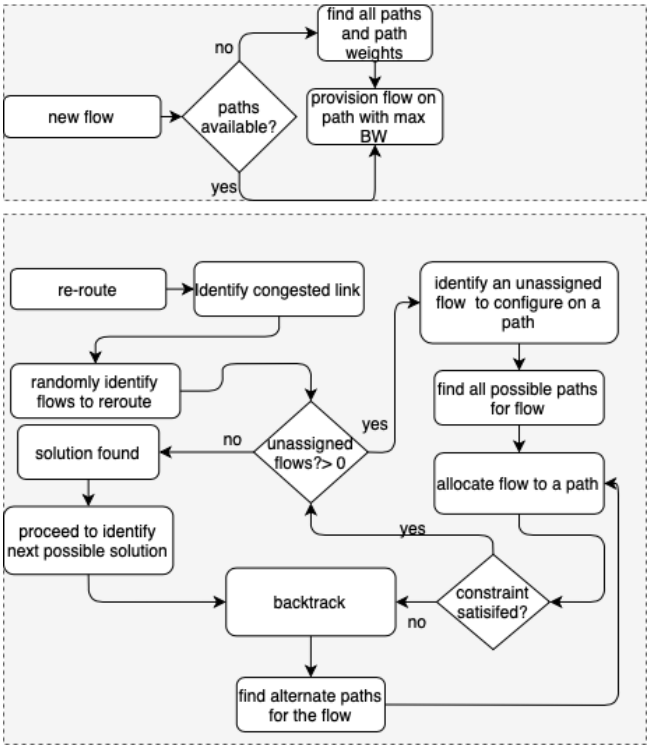


Fig. 4: Rerouting mechanism

Rerouting, on the other hand, is triggered only when one or more links are congested. A link is congested if the link-load exceeds 80% threshold as explained earlier . A set of 'n' randomly selected active flows are selected for rerouting. Given the dynamic nature of link-loads, there is no definite way of choosing a specific flow or a set of specific flows for rerouting. For instance, few flows with high flow-load can be chosen for rerouting, and no path might be able to accommodate these flows, whereas they could accommodate flows of lesser flow-demands and still reduce the load on congested links. To find a flow or set of flows that meets the criteria of available link capacities, the controller would have to iterate over multiple flows. To avoid this process, the controller randomly chooses a set of flows. This gives equal chance for all the flows to be selected.

Upon selecting n active flows to reroute, the application composes alternate paths for the flows to reroute. We apply constraint checking and back-tracking to prune any non-feasible solutions. The process of finding a solution begins with provisioning one flow at a time and adding non-provisioned flows to this set until all flows have a path to route on. A flow f of flow-demand fd_f is assigned to a path p with path load pl_p only if the link-load constraint is satisfied. Since a link can be part of multiple paths used by other flows, the algorithm checks for constraint satisfaction for all the links and proceeds to extend this current solution cur_sol by provisioning remaining flows. In case of constraint failure, while assigning flow f, the algorithm back-tracks and assigns f to alternative path and prunes all the subsequent flow assignments of the current branch. The process is repeated until all flows are assigned to a path. At this stage, the application has a complete solution cur_sol. As mentioned earlier, the process doesn't stop after finding one solution, the algorithm back-tracks and finds all possible solutions to form the $feasible\_set$.

Backtracking algorithm and corresponding constraint check mechanism are listed below

FIND-SOLUTIONS($cur\_sol, feasible\_set$)

```
1  f = select unassigned flow(F)
2  if f == None
3       feasible_set.append(cur_sol)
4       return feasible_set
5  recompute residual_bandwidths()
6  if cur_sol[f] == None
7       for p ∈ paths
8            pl_p = pl_p + fd_f
9            cur_sol[f] = p
10           if CONSTRAINT-SATISFIED()
11                FIND-SOLUTIONS(cur_sol, feasible_set)
12           cur_sol[f] = None
13           pl_p = pl_p − fd_f
14 return feasible_set
```

CONSTRAINT-SATISFIED()

```
1  for l ∈ links
2       if cl_l > link − threshold
3            return False
4  return True
```

Upon finding all possible assignments, we select the solution that yields lowest cost-function from the set of all possible solutions. At this point, a solution is available for the application. Corresponding end-to-end flow-rules are composed and configured proactively on the OF devices.

COST-FUNCTION($feasible\_set$)

1 **for** $sol \in feasible\_set$
2 **for** $f \in sol$
3 $pl_p = pl_p + fd_f$
4 sol_std = standard-deviation of link-loads
5 sol_val = sol_std / mean
6 **return** min(sol_val),sol

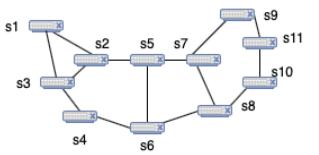
IV. Testing and Observations

As with the majority of SDN simulations, OF topology is simulated with Mininet [11]. The controller of choice is Ryu [12].

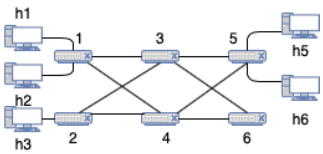
1) Test1

Abilene topology from the Topology Zoo database [13] is used to test the Proposed Solution(PS) where each link's bandwidth is set to 1 Mbps. Link-load comparison is made against Shortest Path First(SPF) algorithm with no rerouting functionality. For both PS and SPF, the network state is updated every 3 secs and the threshold for triggering rerouting process is 80% of link capacity. Hosts h1,h2,h3 connected to s1,s2,s3 send UDP traffic in an ON-OFF manner to host h11 connected to s11. The ON-OFF traffic generation pattern is listed in Table 1.

Fig.6 captures different links utilisation for the duration of the test. With the proposed rerouting algorithm, it can be noticed that the network links experience a lesser duration of high-utilisation and more links; in other words, alternate paths are utilised when active flows are rerouted, as shown in Fig.6(a). To contrast this behavior against an application running SPF and not rerouting active flows, as shown in Fig. 6(b). We can observe that a smaller set of links are highly-utilised over a longer duration of time.



(a) abilene topology for test1

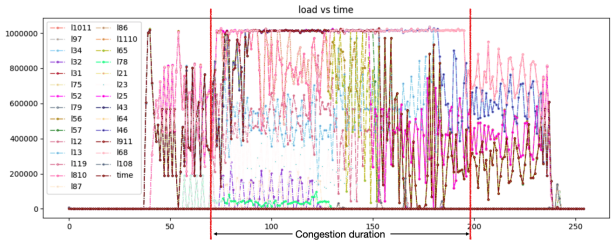


(b) Topology for test2

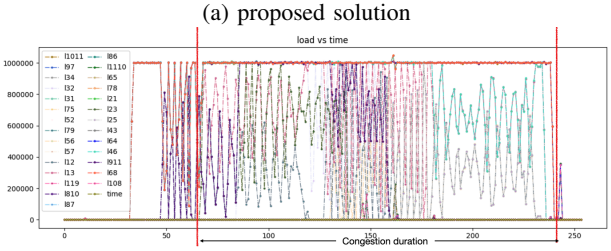
Fig. 5: topologies used in testing

TABLE I: ON-OFF traffic generation sources

Duration	Src	Dst	P
0-600secs	H1	H11	10
40-200secs	H1	H11	5
100-300secs	H2	H11	10
150-300secs	H2	H11	5
150-350secs	H3	H11	10
250-400secs	H3	H11	5
280-580secs	H1	H11	5



(a) proposed solution



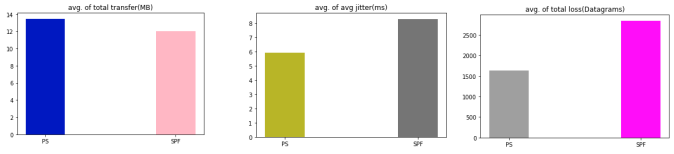
(b) shortest path first

Fig. 6: link-load comparison: Proposed Solution vs SPF

2) Test2

In Test2, we verify the effect of rerouting on transfer rates, jitter and loss. We have used iperf to generate UDP streams and register jitter and delay.Topology listed in [14] as in Fig.5(b) has been used to conduct the test. Each link's bandwidth is set of 1Mbps. h1, h2 send a UDP bit stream at 750 Kbits to h5 and h3 sends UDP stream at 750 to h6 for a duration of 60 secs. Initially,traffic from h1 to h5 is routed via path via s1,s4,s6,s3,s5 and traffic from h2 to h5 is routed via path s1,s4,s6,s3,s5 and traffic from h3 to h6 routes via s2,s4,s5. Since the topology information is updated every 3 secs, flows h1 towards h5 and h2 towards h5 are routed via same path leading to congestion. Upon rerouting, flows from h1 to h5 is rerouted on to s1,s3,s6,s4,s5.

Given that network information is updated periodically, SPF identifies same path as best path for all the flows generated between polling duration and ultimately uses the same path to forward all the flows onto thus leading to higher loss and jitter as shown in Fig.7 With our proposed solution, this is eventually addresses when the application detects a congestion in following monitoring cycles and triggers rerouting, thus moving the flows to alternate paths.



(a) avg. total transfer rates PS vs SPF (b) avg.jitter PS vs SPF (c) avg. total loss PS vs SPF

Fig. 7: Network performance comparison between SPF and PS

Observations

The cost function for this work is standard deviation, is a non-monotonic function. In sense, it doesn't display an increasing or decreasing trend and fluctuates as new flows are added to the path. Thus, greedy calculations of cost function are rendered useless. Standard deviation amongst the path-loads can only be calculated at the leaves of the tree only after all flows are assigned to paths.

Also, an upper bound on total number of possible solutions is p^f where p is number of paths for f number of flows. Such an high number of solutions can be generated only when majority of the branches are navigated and not pruned. This can occur when all the flows assignments satisfy the constraints, which is possible when high number of paths have sufficient capacity to provision the most of flows, in which case we can conclude that the paths are not highly congested-thus making the rerouting process redundant. This behaviour was observed when the threshold value was set to 70%.

In other words, the algorithm performs its worst when there is less congestion in the network. Lower congestion threshold results in a rise in the number of alternate paths for provisioning flows, thus expanding the feasible region.

In-case links are fully congested, the algorithm does not yield a solution as no alternative paths can be found that satisfies the constraints. This explains why there are some highly utilised links. As observed in Fig.6(a), where the onset of congestion is delayed but not entirely absent.

Higher polling intervals reduce the benefits of rerouting whereas smaller intervals increase polling traffic. With a certain level of link-load prediction, this can be addressed and is scoped for future work.

Since rerouting occurs only in the presence of congestion, there might be instances where traffic between a source and destination is not evenly distributed on all available paths. This behaviour is acceptable as long as it does not lead to congestion on one specific path.

A time-limit can be set for congestion duration to avoid frequent fluctuation of traffic on links. For instance, a link congested for a longer duration preempts a recently is congested link.

Finally, Since port stats from earlier time-stamps are used, multiple flows see the same link as the best available link with maximum resources at a given time. Rerouting resolves such behavior.

V. Conclusion

In this paper, we proposed a rerouting mechanism with a goal to distribute traffic across multi-path networks. This was achieved by selecting and rerouting multiple active flows on multiple paths by minimising the standard deviation of the path-loads while meeting the link constraints. We have then tested our proposed solution against SPF algorithm. We have made critical observations about the proposed rerouting mechanism.

References

[1] J. Zhou, M. Tewariy, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted cost multipathing for improved fairness in data centers," *Proceedings of the 9th European Conference on Computer Systems, EuroSys 2014*, 2014.

[2] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," *CoNEXT 2014 - Proceedings of the 2014 Conference on Emerging Networking Experiments and Technologies*, pp. 149–159, 2014.

[3] R. Trestian, K. Katrinis, and G. M. Muntean, "OFLoad: An OpenFlow-based dynamic load balancing strategy for datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 792–803, 2017.

[4] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," *Proceedings - IEEE INFOCOM*, pp. 1629–1637, 2011.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," *Proceedings of NSDI 2010: 7th USENIX Symposium on Networked Systems Design and Implementation*, pp. 281–295, 2019.

[6] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient Traffic Splitting on SDN Switches," *CoNEXT '15*, 2015.

[7] M. Shafiee and J. Ghaderi, "A Simple Congestion-Aware Algorithm for Load Balancing in Datacenter Networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3670–3682, 2017.

[8] G. N. Senthil and S. Ranjani, "Dynamic Load Balancing using Software Defined Networks," *International Journal of Computer Applications*, pp. 11 – 14, 2015.

[9] S. Attarha, K. Haji Hosseiny, G. Mirjalily, and K. Mizanian, "A load balanced congestion aware routing mechanism for Software Defined Networks," *2017 25th Iranian Conference on Electrical Engineering, ICEE 2017*, pp. 2206–2210, 2017.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[11] http://mininet.org/, [Online; accessed 2-July-2020].

[12] https://ryu.readthedocs.io/en/latest/#, [Online;accessed 2-July-2020].

[13] http://www.topology-zoo.org, [Online;accessed 2-July-2020].

[14] M. T. Kao, B. X. Huang, S. J. Kao, and H. W. Tseng, "An Effective Routing Mechanism for Link Congestion Avoidance in Software-Defined Networking," *Proceedings - 2016 International Computer Symposium, ICS 2016*, pp. 154–158, 2017.