# OpenFlow based control for Re-routing with Differentiated flows in Data Center Networks

Renuga Kanagavelu, Luke Ng Mingjie, Khin Mi Mi Aung
Data Center Technologies Division,
A*STAR, Data Storage Institute,
Singapore
{renuga_k, Luke_NG, Mi_Mi_AUNG}@dsi.a-star.edu.sg

Bu-Sung Lee,Francis , Heryandi
School of Computer Engineering,
Nanyang Technological University,
Singapore
{EBSLEE, Heryandi }@ntu.edu.sg

***Abstract*—**

**Data Center Networks need densely interconnected topologies to provide high bandwidth for various cloud computing services. It is required to fully utilize the bandwidth resource in such a network with varying traffic patterns. We propose a flow-based edge-to-edge rerouting scheme for the Data Center Networks which has the mix of large flows and short flows. We focus on re-routing the large flows to alternative paths in the event of congestion because small flows are short- lived and shifting them between paths will increase their latency and overhead. The proposed scheme has several important features: 1) It provides adaptive re-routing of flows to efficiently spread the load in accordance with the changing load conditions. 2) It ensures effective traffic redistribution upon link failures. We develop a prototype and demonstrate the effectiveness of the proposed mechanism on OpenFlow test bed.**

***Keywords-component; edge-to-edge reroute; Data Center network; load balancing***

## I. INTRODUCTION

A cloud-scale Data Center has become more and more important for enterprises by hosting business critical services. They provide support for a wide range of applications such as on-line financial transaction processing, multimedia content delivery, e-mail, file sharing, each with different requirements. To provide service for large number of applications, Data Centers require high performance network interconnect to connect tens of thousands of servers. Using single path routing in this type of networks cannot fully utilize the network capacity, leading to congestion on the oversubscribed links and underutilizing the resources on other available paths. In such networks, it is critical to employ effective load balancing schemes so that the bandwidth resources are efficiently utilized.

Large Data Centers are evolving with bigger capacity and heavy traffic among the large servers. Varying traffic patterns are generated due to the heterogeneous mix of flows that are sensitive to either latency or throughput. This traffic mix requires a Data Center to deliver high bi-section bandwidth for throughput sensitive large flows and short delay for latency-sensitive small flows. Tree based topologies [1], [2], [14]and widely adopted many multi-path routing algorithms like OSPF[3] (Open Shortest Path First), ECMP [4] (Equal-Cost Multi-Path) which uses equal multiple paths have been developed. However, a key challenge is to manage the Data Center traffic [5] which is the mix of throughput sensitive large flows and latency sensitive small flows. The widely adopted Equal-Cost Multi-Path (ECMP) routing may not ensure even traffic distribution among multiple paths because it balances the number of flows on different paths and not the bit rate which will cause the congestion across the oversubscribed links. Also ECMP works well for large number of small flows and not for large flows. Thus, an efficient flow-based routing algorithm is required to balance these mix of latency sensitive small flows and throughput sensitive large flows to avoid the hot spots.

OpenFlow [6] defines a framework to perform per-flow routing where switches and routers maintain flow tables. The flow tables can be dynamically modified by the remote NOX controller [13]. OpenFlow is a programmable protocol designed to manage and direct traffic among OpenFlow switches.

We propose flow based edge-to-edge re-routing scheme for the Data Center networks which has a mix of large flows and small flows. We define large flows as flows with a size larger than 100 KB and small flows as flows with a size less than 100 KB because according to [1] , more than 85% of flows in a Data Center are less than 100 KB. The basic idea is to perform shortest path routing to route the mix of flows across the multiple paths. In our scheme, we focus on re-routing of large flows because small flows are short- lived and shifting them between paths will increase their latency and overhead. To achieve load balancing, only large flows are changed by the controller at the time of possible congestion in the forwarding link while small flows continue their traversal. The proposed framework has several important features: 1)It provides adaptive re-routing scheme to efficiently spread the load in accordance with the changing load condition and 2)It ensures effective traffic redistribution upon link failures. We develop a prototype and demonstrate the effectiveness of the proposed mechanism on OpenFlow test bed.

The rest of the paper is organized as follows. Section II presents the background and related work on Routing-control

techniques in Data Centers. Section III presents the proposed routing control framework and its various features. Section IV details our fault tolerance mechanism. Section V studies the performance of the proposed mechanism and discusses the OpenFlow test bed results. We conclude the paper in Section VI.

## II. Background & Related Work

Recently, there has been some research work done in the literature related to the data center network topologies with dense interconnections [1], [2], including Bcube [7] and Dcell [8]. The traditional enterprise network algorithm Equal Cost Multipath (ECMP) is used for forwarding packets in Data Center networks [4]. In this routing strategy, the next hop forwarding to a single destination is based on choosing a path from among multiple paths. The decision is taken based on the hash value of certain fields of a packet. The hash based traffic distribution in ECMP attempts to balance the load in terms of the number of flows on different paths but not on the bit rate which could result in uneven traffic distribution. Therefore it works well for large number of short flows but not for large flows.

Hedera [9], is a dynamic flow scheduling system for multi-rooted tree topology which detects large flows at edge switches, estimates the natural demand for flows, computes good paths and installs them on to the OpenFlow switches dynamically and centrally. However, it considers scheduling of the long lived flows and also assumes that flows which it schedules on to lower loaded links are capable of increasing their transmit rate to fill the link. This can be violated by the flows that are end host limited and can't increase their transmission rate.

PROBE [10] (Probe and Reroute Based on ECMP) exploits the multipath routing capability provided by ECMP to realize flow-based rerouting, thus enabling fine granular traffic control. All the operations are performed at the end hosts using TCP, but not on the switches particularly using OpenFlow.

In [11], Data Center traffic are characterized and discussed at the macroscopic and microscopic level. The key findings are that the core switches are heavily utilized, the traffic pattern executes ON/OFF characteristics, and the traffic characteristics are burstier at the edge switches.

### A. *OpenFlow Routing Architecture*

OpenFlow [6] is an open standard, which enable users to easily implement experimental routing protocols via software. The key difference between the conventional router and the OpenFlow is that the forwarding (data plane) and routing (control plane) layers are decoupled. The entire network is centrally managed by a dedicated controller as shown in Fig. 1, which communicates with OpenFlow compliant switches using the OpenFlow protocol. Every OpenFlow switch maintains its own table of flow entries (flow table), in which each entry contains a set of packet fields to match, and the corresponding action to perform, (e.g. forward, drop, modify header). In the event when a switch is unable to find a match in the flow table, the packet is forwarded to the controller to make the routing decisions. After deciding how to route the new flow, the controller then installs a new flow entry at the required switches, so that the desired actions can be performed on the new flow.

With the OpenFlow, it is possible to use various routing algorithms within the controller to generate the forwarding tables that govern the flow based edge-to-edge re-routing of large flows in the data plane. We explain it in next section.

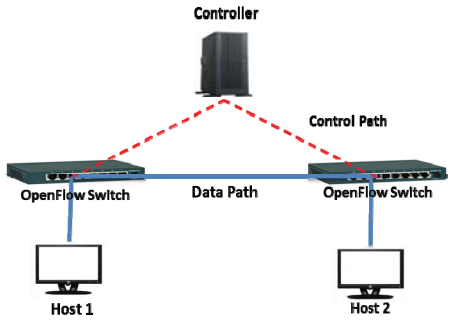


**Figure 1.OpenFlow Architecture**

## III. Routing Control Framework

We consider the OpenFlow test bed topology as shown in Fig. 2. Suppose that Host 1 is generating a mix of large flows and small flows to Host 2. In Fig. 2, if the routes A-B-D-F and A-C-D-F are selected for routing of traffic flows from Host 1 to Host 2, link D-F may be overloaded. To mitigate this problem, we propose an edge-to-edge rerouting scheme. The basic idea is to use least loaded route first. When congestion occurs on a link, we dynamically re-route one or more large flows to alternate paths and allow the small flows to continue their traversal because of their short-lived nature.

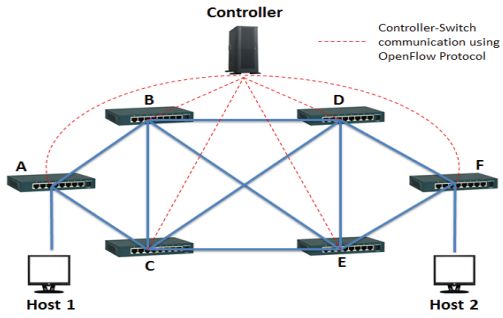


**Figure 2. OpenFlow Test Bed**

We implemented an edge-to-edge rerouting control prototype in the NOX-Classic controller [12], which is a component-based OpenFlow controller. Our rerouting framework and its functional modules are shown in Fig. 3.

### A. *Framework Modules*

In this section, we briefly describe the functions of each module in our re-routing framework.

1) Monitor

The purpose of this module is to query, consolidate and store the statistics from all OpenFlow switches. These statistics

are then used by the Routing Engine module to compute the load on various links. The key idea behind our algorithm is to route the packets along the lightly loaded path. Hence the load on the links along the possible paths needs to be determined. This module collects statistics from each OpenFlow switch by polling them at fixed intervals. The per-table, per-flow and per-port statistics are gathered from all the connected OpenFlow switches and stored in memory as a snapshot object. Each snapshot is identified by a sequence number, which increments by 1 after each interval. Only the 2 most recent snapshots are maintained in memory at instant of time. Other components can access these snapshot objects directly to obtain the required information.

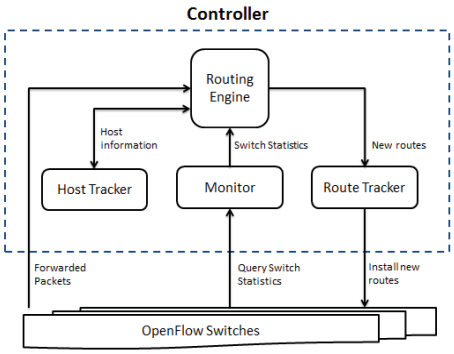


**Figure 3. OpenFlow-based Re-routing Control Framework**

2) Host Tracker

This module keeps track of all the hosts detected on the network. It stores the MAC address of the interface of the hosts connected to the network, IP address assigned to the interface of the hosts connected to the network, the OpenFlow switch id to which each of the hosts is connected and the OpenFlow switch port number to which each of the hosts is connected. Table 1 shows an example of the entries stored in this module if there are 2 hosts connected to switches A and F respectively as shown in Fig. 3, connected to port 1 of the switches.

**Table 1. Host Tracker Entries**

| MAC Address | IP | Switch ID | Port No. |
|---|---|---|---|
| 11-AA-23-FE-1C-4A | 10.200.1.1 | A | 1 |
| 22-BB-22-EA-2D-9B | 10.200.1.2 | F | 1 |

3) Routing Engine

The Routing Engine module is responsible for routing and rerouting functions. It computes the least loaded shortest candidate paths between any pair of end hosts based on the statistics collected from the switches by the Monitor module.

When a packet from a new flow from a host arrives at an OpenFlow switch the first time, the switch forwards the packet to the controller because there is no match with its installed flow entries. Upon receiving the packet, the Routing Engine module in the controller will extract the flow's source and destination host addresses. The Routing Engine will then consult the Host Discovery module to obtain the information about the switch connected to the source host and the switch connected to the destination host. Upon receiving this, it computes all the possible shortest routes. Once the set of all shortest routes R is computed, the statistics from the Monitor module will be used to compute the least loaded route.

We shall use an example to illustrate the mechanism of the Routing Engine. We consider the case of routing between Host1 and Host2 as shown in Fig. 4. R represents the set of candidate shortest paths $\{r_1, r_2, r_3, r_4\}$ where $r_1$ is A-B-D-F, $r_2$ is A-C-E-F, $r_3$ is A-B-E-F, and $r_4$ is A-C-D-F. At the initial phase, if the amount of statistics collected is insufficient, a route is randomly picked from R. The total cost of each route $r_i \in R$ is defined as $\gamma_i = \alpha_i + \beta_i$, where $\alpha_i$ is the total fixed costs of each link in the route, $link_k \in r_i$ , and $\beta_i$ is the total variable costs of $link_k \in r_i$ . Each $link_k$ has a predetermined fixed weight $w_k$, a link load of $L_k$ bps computed from the statistics, and link capacity $C_k$ bps. Currently in our algorithm, all link weights are set to 1. The value of $L_k$ is estimated from the change in byte count for all flow entries in a switch between the two most recent snapshots from the Monitor module. After computing $\gamma_i$ from all $r_i \in R$. the controller then picks the route that has the minimum $\gamma_i$ and installs the flow entries needed into the series of switches in the shortest path. Mathematically the total fixed cost and the variable cost of the route $r_i$ is given as,

$$\alpha_i = \sum_{k}^{N} w_k \quad \text{where } link_k \in r_i$$

$$\beta_i = \sum_{k}^{N} \left( \frac{L_k}{C_k} \times \frac{{w_k}^2}{\alpha_i} \right) \qquad \text{.....(1)}$$

We record the path and the cost information in the NOX controller as shown in Table 2.

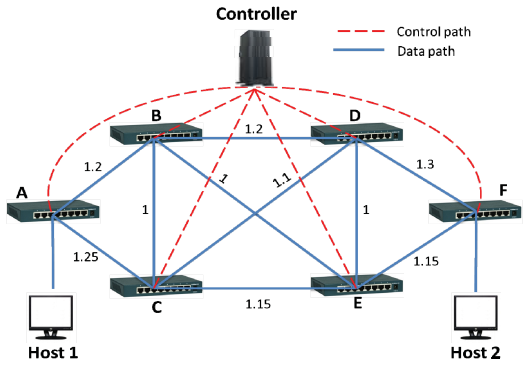


**Figure 4. Example of Path Cost Comparison**

**Table 2. Cost Information**

| i | Path | Total cost $\gamma_i$ |
|---|---|---|
| 1 | A-B-D-F | 3.7 |
| 2 | A-C-E-F | 3.55 |
| 3 | A-B-E-F | 3.35 |
| 4 | A-C-D-F | 3.65 |

Based on the total cost, the least loaded route A-B-E-F will be chosen. The Routing Engine will then pass the information of this chosen route to the Route Tracker module which will install this route as flow entries to the switches A, B, E and F.

**Rerouting Function**

The Routing Engine checks for congestion periodically across all the links. We consider 75% of link capacity as the threshold for congestion. At any instant, if any one of the link load exceeds this threshold value T, the controller will re-route one or more large flows across the link to the alternate path one by one.

We use the flow statistics mechanism to identify the large flows. Each flow is monitored at the first switch traverse by the flow. The statistics are collected from switches by the controller at a fixed interval of p seconds and used to classify the large flows. It is computed as follows:

$$\Psi_t = (b_t - b_{t-p})/p \quad \text{.....(2)}$$

where $\Psi_t$ is the estimated flow size at time t, and $b_t$ is the total bytes of the flow received by the switch at time t. In our implementation, p is set to 5.

We use an example to illustrate the re-route mechanism. Let F represent the set of mix of large flows $\{f_{1L}, f_{3L}, f_{4L}\}$ and small flows $\{ f_{2S}, f_{5S}\}$ generated by Host 1. According to Table 2, if the routes A-C-E-F and A-B-E-F are selected for routing of traffic from Host 1 to Host 2, the path load of the link E-F may exceed the congestion threshold. To mitigate this problem, the large flow $f_{3L}$ which is determined by the flow statistics mechanism, is re-routed to other least loaded alternative path A-C-D-F. Before choosing the alternative path, we will make sure that the large flow $f_{3L}$ at its current size will not overload A-C-D-F.

Upon flow completion, the controller will again be notified of the flow removal by the switches after certain period of flow entry inactivity. When the controller receives this *flow entry removal message*, the Route Tracker will check which of the installed flows is to be removed and remove that flow from the list maintained by it.

4) Route Tracker

This module installs the route chosen by the Routing Engine into the series of OpenFlow switches as flow entries. This module keeps track of all the installed routes as well, as shown in Table 2.

**Table 3. Route Tracker Entries**

| Flow entry | | Route |
|---|---|---|
| Src MAC | 11-AA-23-FE-1C-4A | A-B-D-F |
| Dst MAC | 22-BB-22-EA-2D-9B | |
| Ether Type | IP | |
| Src IP | 10.200.1.1 | |
| Dst IP | 10.200.1.2 | |
| IP Proto | TCP | |
| Src TCP/UDP Port | 51832 | |
| Dst TCP/UDP Port | 44366 | |
| Src MAC | 22-BB-22-EA-2D-9B | F-D-C-A |
| Dst MAC | 11-AA-23-FE-1C-4A | |
| Ether Type | IP | |
| Src IP | 10.200.1.2 | |
| Dst IP | 10.200.1.1 | |
| IP Proto | UDP | |
| Src TCP/UDP Port | 40023 | |
| Dst TCP/UDP Port | 54623 | |

Each entry in the table contains the information about the flow entry attributes like source MAC, Destination MAC, Type , Port and the route installed for this particular flow entry. In the event of the link failure, this module will redirect the affected installed routes.

## IV. FAULT TOLERANCE

In this section, we describe how the load will be redistributed in the event of a link failure. In the case of a link failure, one of the built-in NOX components called Discovery will notify the Route Tracker, which reacts by preparing the list of affected flows according to its record of installed route entries. For each of these affected flows, the Routing Engine will start to find alternative routes while making use of the working segment of the current (broken) route as much as possible. It will do this by considering the node closest to the destination with the end node of the failed link as the starting point for rerouting. From that node, it will attempt to find loop-free shortest route to the destination node, if it exists. If multiple loop-free shortest routes to the destination node exist, then the least-loaded route is chosen. If no loop-free route to the destination node exists, then it considers the node located just before the currently considered node in the route as the start node and attempt to find loop-free shortest route from that node to the destination node again.

We illustrate the fault-tolerant mechanism using an example. Assume that there are two flows $f_1$ and $f_2$ that are initially installed, with flow $f_1$ from Host 1 to Host 2 using the route A-B-D-F and flow $f_2$ from Host 2 to Host 1 using the route F-D-C-A. Suppose that link F-D fails.

For flow $f_1$, the first node to be considered as the start node for rerouting is node D which route to node F through D-E-F. With this alternative route, flow $f_1$ will be taking the new route A-B-D-E-F.

For flow $f_2$, the first node to be considered as the start node is node F. Note that node F has has several shortest routes to A, namely, F-E-C-A and F-E-B-A. The algorithm will take the least-loaded route among these two alternate routes. For simplicity, assume that the former route is chosen for flow $f_2$. The state of the network now is as seen in Fig. 5 below.

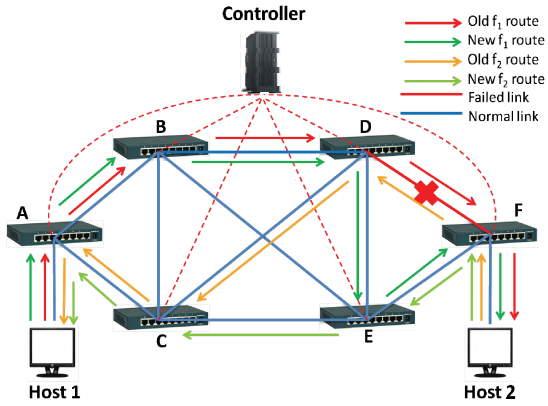


**Figure 5. Fault Tolerance - Link F-D fails**

After these rerouting, assume that another link failure happens in link C-D and link D-E. Flow $f_2$ is not affected by the link failures. Flow $f_1$, on the other hand, is affected.

The first node to be considered as the start node to reroute flow $f_1$ is D. The shortest route from D to F is D-B-C-E-F. However if that shortest route is used, the complete route when linked together with the working part of the route will be A-B-D-B-C-E-F which contains a loop. Therefore, the algorithm will now attempt to use the previous node before D, which is B, as the start node for rerouting, using a loop-free route B-E-F. With this alternate route, flow $f_1$ will be rerouted to A-B-E-F. The state of the network now is as seen in Fig. 6.

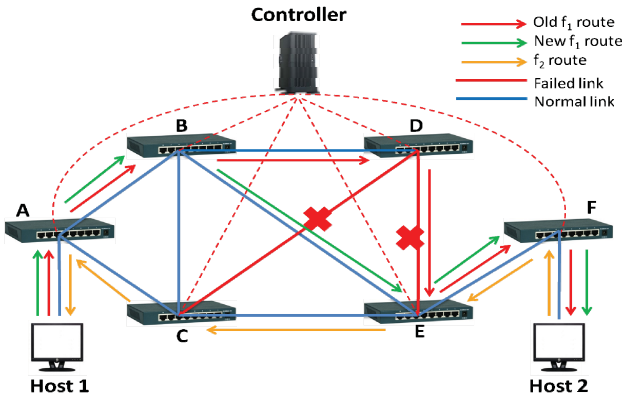


**Figure 6. Fault Tolerance - Link D-E and C-D fails**

## V. PERFORMANCE EVALUATION

We set up a test bed using 6 OpenFlow switches, 2 hosts and a controller, with the topology as shown in Fig. 2. All the OpenFlow switches are software-based OpenFlow switches, with OpenFlow reference implementation version 1.0, running on Linux server machines with 1 Gigabit Ethernet interfaces[13]. The controller is a Linux server machine running the NOX-Classic controller, Destiny branch. Our routing algorithm is running at the NOX controller. We also implemented the ECMP routing algorithm and the single path routing algorithm in the NOX controller for the purpose of performance comparison with ours.

We generate the traffic pattern which is a mix of large and small flows. We use the traffic generator device to generate various traffic patterns by adjusting the packet size as 64, 512 and 1500 bytes under 100% network load.

We use the performance metrics –load distribution, throughput, and link utilization. For Large flows, we consider the size of 100KB and more and for small flows, we can consider the size less than 100KB.

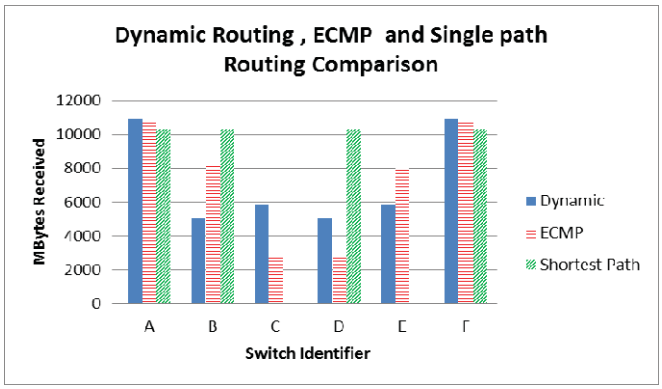


**Figure 7. Comparison of our proposed Dynamic Routing, ECMP and single path routing in terms of Load Distribution**

The plots in Fig. 7 show the load distribution at the switches in the OpenFlow test bed. It can be observed that switches A and F are overloaded in all algorithms because they are ingress and egress points. Using our Dynamic algorithm, the load gets distributed more evenly across the other switches, B, C, D and E. However ECMP overloads the switches B and E while single path routing overloads the switch B and D whereas switches C and E are underutilized for the traffic patterns considered.

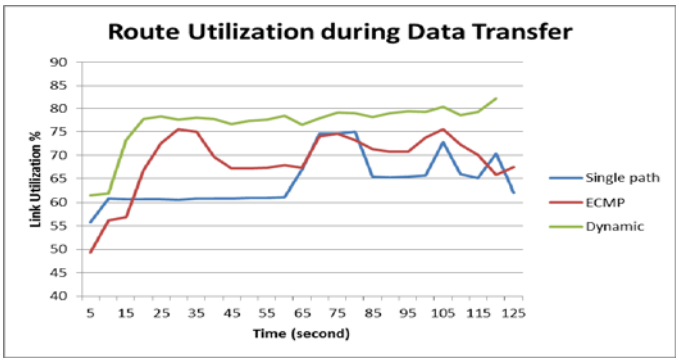


**Figure 8. Comparison of our proposed Dynamic Routing and single path Routing in terms of Link utilization**

Fig. 8 plots the link utilization using our dynamic re-routing algorithm for the traffic patterns generated from the traffic generator under 100% network load, the link utilization averages to 80% which is equal to 800 Mbps whereas for the single path routing, the link utilization averages to 70% which is equal to 700 Mbps and for ECMP, the link utilization averages to 72% which is equal to 720 Mbps.

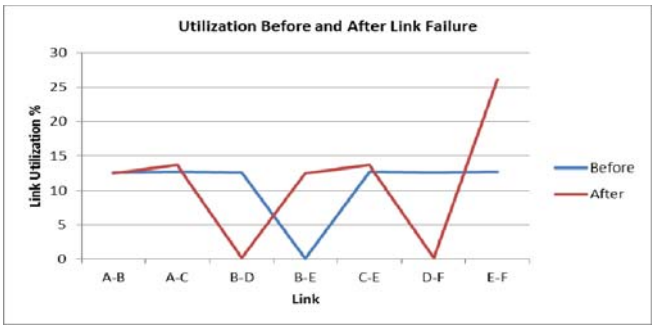


**Figure 9. Link utilization using the proposed approach without and with a link failure**

For testing the performance of the fault tolerance mechanism, we use a simple scenario where there are only 2 flows on the topology as shown in Fig. 2. and only 1 link failure. The first flow, $f_1$, is traversing through the route A-B-D-F while the second flow, $f_2$, is traversing through the route A-C-E-F.

The Fig. 9 depicts the load distribution before and after the failure of the link B-D.The failure of link B-D, causing f1 to be rerouted to A-B-E-F, thus increasing the utilization of links B-E and E-F and reducing the the utilization of the links B-D and D-F.

## VI. CONCLUSION

In this paper, we proposed a OpenFlow based re-routing control framework to address the problem of managing different flows in Datacenter networks in the event of congestion and faults. The framework enables adaptive re-routing of flows to efficiently spread the load in accordance with the changing load conditions. We also considered the case of link failures wherein traffic on the failed paths are evenly redistributed across the working paths. We implemented the framework on NOX-classic controller in the OpenFlow testbed and demonstrated the effectiveness of the proposed re-routing mechanism in comparison with other schemes.

## REFERENCES

[1]A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri,D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible datacenter network. In SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, pages 51–62, New York, NY,USA, 2009. ACM.

[2] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri,S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication,
pages 39–50, New York, NY, USA, 2009. ACM.

[3] J. Moy. OSPF Version 2. RFC 2328 (Standard), Apr. 1998.

[4] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), Nov. 2000.

[5] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson,J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev., 38(2):69–74, 2008.

[7] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a high performance, server-centric network architecture for modular data centers. In SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, pages 63–74, NewYork, NY, USA, 2009. ACM.

[8] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. DCell: A scalable and fault-tolerant network structure for data centers. In SIGCOMM'08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication, pages 75–86, New York, NY, USA, 2008. ACM.

[9] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks", in Proc. of the 7thUSENIX Symposium on Networked Systems Design and Implementation (NSDI '10), April 2010.

[10] Kang Xi, Yulei Liu and H. Jonathan Chao, "Enabling Flow-based Routing Control in DataCenter Networks using Probe and ECMP", IEEE INFOCOMM workshop on cloud computing , 2011

[11] Theophilus Benson, Ashok Anand, Aditya Akella and Ming Zhang, "Understanding Data Center Traffic Characteristics", in Proc. Of SIGCOMM'09, August 2009.

[12] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to Data center", In Proc. of 8th ACM Workshop on Hot Topics in Networks, 2009.

[13] 3Com Corporation, "Switch 5500G 10/100/1000 family data sheet." [Online]. Available:
http://www.3com.com/other/pdfs/products /en US/400908.pdf

[14] Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Towards a Next Generation Data Center Architecture: Scalability and Commoditization. In Proceedings of ACM PRESTO, 2008.