Adaptive routing for layer-2 load balancing in data center networks

Kanagavelu, Renuga; Lee, Francis Bu-Sung; Ragavendran, Vasanth; Aung, Khin Mi Mi

2012

Kanagavelu, R., Lee, Francis, B.-S., Ragavendran, V., & Aung, K. M. M. (2012). Adaptive Routing for Layer-2 Load Balancing in Data Center Networks. International Journal on Advances in Information Sciences and Service Sciences, 4(15), 1-12.

<https://hdl.handle.net/10356/90415>

<https://doi.org/10.4156/aiss.vol4.issue15.1>

© 2012 The International Association for Information, Culture, Human and Industry Technology (AICIT). This paper was published in International journal on advances in information sciences and service sciences and is made available as an electronic reprint (preprint) with permission of the International Association for Information, Culture, Human and Industry Technology (AICIT). The paper can be found at the following official DOI: <http://dx.doi.org/10.4156/aiss.vol4.issue15.1>. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper is prohibited and is subject to penalties under law.

Adaptive Routing for Layer-2 Load Balancing in Data Center Networks

¹Renuga Kanagavelu, ²Bu-Sung Lee, Francis, ³Vasanth Ragavendran, ⁴Khin Mi Mi Aung
^{1,*} Corresponding Author *Data Storage Institute, Singapore .E-mail: renuga_k@dsi.a-star.edu.sg*

²Nanyang Technological University, Singapore. E-mail: EBSLEE@ntu.edu.sg

³National university of Singapore, Singapore. E-mail: raghavendrapec@gmail.com

⁴Data Storage Institute, Singapore. E-mail: Mi_Mi_AUNG@dsi.a-star.edu.sg

Abstract

With the Internet boom over the last decade, large scale Data Centers are rapidly increasing in capacity and numbers to meet the ever increasing demand for bandwidth. There is a trend of deploying a large number of commodity layer-2 Ethernet switches in Data Centers. However, the existing Spanning Tree Protocol (STP) used in the traditional Ethernet networks becomes inefficient for Data Centers as it underutilizes the resources due to the lack of multipath capability leading to poor bandwidth utilization. In this paper we develop a layer-2 load balancing framework with a multi-path forwarding mechanism which balances the load across the network, thereby reducing the occurrence of congestion that leads to efficient utilization of the bi-section bandwidth in Data Centers. The proposed framework has several important features: 1) It provides adaptive multi path forwarding compatibility in layer-2 Ethernet switched networks to efficiently spread the load in adaption to changing load conditions. 2) It improves the bi-section bandwidth utilization in Data Center networks. 3) It is capable of achieving load balancing at the layer-2 level and 4) It ensures effective traffic redistribution upon link failures. We demonstrate the effectiveness of the proposed mechanism through simulation results.

Keywords: Data Center, Multipath Forwarding, Load-Balancing, Fat-tree

1. Introduction

With the Internet boom over the last decade, Data Centers have come to play a very crucial role. They are the crux of various giant organizations like Google, Amazon etc. The Data Center today offers very high aggregate bandwidth connecting tens of thousands of machines. With the increasing Internet users and need for more storage and processing, the capacity and number of compute and storage servers used in a Data center has been increasing. This leads to edge-core topologies that connect inexpensive edge switches with expensive monolithic core switches to provide adequate bisection bandwidth. The use of expensive core switches is impractical for large scale Data Centers. An attractive alternative is to replace the expensive core switch with multiple low cost core switches. However, to preserve high bisection bandwidth, a Data Center often employs multi-root tree topology with the existence of multiple paths between a pair of hosts.

The traditional Ethernet, which supports self-configuration of switches, forwards all the traffic through a single spanning tree [1], [2]. Conventional spanning tree architecture is not used in Data Centers as there is only one path between a given pair of communication points, leading to congestion along the single path and underutilizing the resources in other possible paths. In a Data Center, due to the use of a large number of low cost commodity switches, multiple paths exist between a given pair of hosts leading to the Fat tree architecture [3], [4]. The traditional adaptive routing OSPF protocol used in IP networks which chooses a shortest path is not effective in Data centers with the Fat tree architecture, as there are multiple paths between the end points with equal number of hops in fat tree architecture. Hence, there is a need to use other techniques for forwarding frames in a Data Center. It is desirable that such techniques distribute the load evenly across the possible multiple paths so as to minimize congestion. It is also desirable that such techniques are effective in distributing the traffic in the event of failures.

In this paper, we develop a load balancing framework to address the above challenges. We develop an adaptive multi path forwarding mechanism based on path load conditions that enhances Layer-2

Ethernet switches. The proposed mechanism balances the load across multiple paths and minimizes the occurrence of congestion. In our work, we consider traffic forwarding at Layer-2 level and this approach is more attractive when compared to a flow level approach at a higher layer [5] leading to improved link utilization and throughput. The entire framework works at Layer-2 and there is the need for a new addressing scheme because the traditional flat MAC address space is the main cause of the poor scalability in Ethernet. The modern switch's forwarding database [6, 11] can hold up to 16,000 entries. If the forwarding database is full, the frames are flooded into all the ports which cause a serious traffic storm in all the poor capacity links. In order to overcome this, we introduce a hierarchical addressing scheme which makes the routing easier and scalable.

The rest of the paper is organized as follows. Section 2 presents the background and related work on load balancing techniques in Data Centers. Section 3 describes the Data Centre network architecture used in our work. Section 4 presents the proposed load balancing framework and its various features. Section 5 studies the performance of the proposed mechanism and discusses the simulation results. We conclude the paper in Section 6.

2. Background and Related Work

Recently, there has been some research work done in the literature related to the load balancing problem in Data Center networks [3, 4, 7, and 13]. The traditional enterprise network algorithm Equal Cost Multipath (ECMP) is used for forwarding packets in Data Center networks [9]. 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 namely source address, destination address etc. and hence the name static hashing. Here, the path taken for the packets with the same source and destination address would be the same. Here all the packets pertaining to a flow will follow the same path making certain parts of the network heavily loaded while leaving the other parts lightly loaded. This technique neither considers the flow size nor the network utilization.

In [5], layer-3 routing and dynamic load balancing is considered at the flow level, i.e. all the packets pertaining to a particular flow follow the same path. Here, until reaching the threshold rate, the packets are forwarded using ECMP technique, and upon exceeding the threshold rate, Hedera algorithm is used for forwarding the flow. In short, the algorithm does the following steps. It identifies the large flows at the edge switches. Then it estimates the natural demand of the large flows and computes the best path for them. Finally it installs the path information in the switches. However, it uses a long header for distinguishing between flows. This leads to increased processing delay. In order to reduce the processing delay the header information has to be kept as short as possible. Furthermore this work, as mentioned before achieves dynamism at the flow level.

In [6], 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 is more bursty at the edge switches when compared to the core/aggregate switch.

In [4], a network address translation (NAT) layer is proposed for MAC addresses. It presents a 2-level routing table for multipath routing across the fat tree. Here, switches use destination physical MAC (PMAC) hashing to decide the next hop. The source servers can easily calculate the routing path on behalf of the switches. However this method slightly increases the lookup latency.

3. Network Architecture

The Data Center network architecture considered here is a Fat tree, a 3-tier hierarchical architecture, to interconnect commodity Ethernet switches[12]. Fat trees are a form of multi rooted trees that form the basis of Data Center topologies. The advantage of using a Fat tree is that there exists equal cost multiple paths between every pair of edge nodes, which if utilized properly can increase the

bandwidth usage. Fig. 1 represents a general three stage Fat tree built from n -port switches. A Fat tree has 3 tiers of switches - core tier in the root of the tree, aggregate tier in the middle of the tree and edge tier at the leaves of the tree as shown in Fig. 1. The links interconnecting the switches considered here are Ethernet links with Gigabit capacity. The links towards the core switches are usually of higher capacity than the links near the edge switches. This is a “scale out” method as more low cost commodity switches are employed rather than using high cost switches. It has been shown that using the Fat tree architecture with commodity switches costs lesser than a comparable network built from non-commodity switches [5]. The Fat tree is organized into n individual pods each containing two layers of $n/2$ switches. Each n -port switch in the lower layer is connected to $n/2$ hosts and the remaining $n/2$ ports are connected to $n/2$ of n ports in aggregation layer in the hierarchy. Each core switch has one port connected to each of the n -pods.

3.1. Fat Tree

The goal of our work is to develop a layer-2 framework with addressing scheme, forwarding, load balanced routing and fault tolerance. The advantage of working at Layer 2 is that reducing the congestion management at Layer 2 will minimize the congestion at upper layers. Achieving maximum bisectional bandwidth utilization in the Fat tree architecture requires distributing the outgoing traffic from any pod as evenly as possible among the core switches. In a k -ary Fat tree architecture [11], there exists $(k/2)^2$ multiple paths between any two hosts on different pods. We illustrate the existence of multiple paths between two hosts using an example. Suppose that host B wishes to initiate a flow to host G. The multiple paths which exist between these hosts are shown in Table 1. We use E_i , A_j and C_p to denote the edge switch i , aggregate switch j and core switch p respectively. Note that host B is connected to switch E_1 and host G is connected to switch E_4 .

Table 1. Multiple paths between host B and host G

Path	Route
1	$E_1-A_1-C_1-A_4-E_4$
2	$E_1-A_1-C_2-A_3-E_4$
3	$E_1-A_2-C_3-A_3-E_4$
4	$E_1-A_2-C_4-A_4-E_4$

We choose one of the paths based on the load metrics to achieve maximum bandwidth utilization and minimize congestion.

3.2. Addressing Scheme

Routing is an essential component for load balancing. To support routing we group the switches into pods or zones. The hosts have the address of the form of *podId.layerId.switchId.hostId*, where *podId* denotes the pod number containing the host (all core switches are assigned to pod0), *layerId* denotes the layer of the switch, *switchId* denotes the switch’s position in the layer and *hostId* denotes the position of the host in the Pod. We name this addressing scheme as hierarchical MAC addresses (HMAC) and it is more location specific. The Address is stored in the switch. We note that our addressing scheme is different from that used in [7] as the latter uses a scheme similar to IP addressing with longest prefix matching procedure. In our scheme upon arrival of a packet from a host, the HMAC is attached to the packet by the edge switch. This HMAC is used by various switches for routing particularly for deciding interpod or intrapod communication. This HMAC format is shown in Fig. 1.

3.3. Chief supervisor

Chief supervisor (CS) is a centralized manager which maintains the soft state about network configuration information such as topology and has the connectivity to all the switches in the network. The CS gathers the information about the load on various links as and when needed. In the event of link failure, it will notify all the edge switches about the failure so that they can quickly react to the failure. Further CS used here is stateless as it does not need to maintain any table. It identifies the incoming packet type and then takes appropriate action accordingly. It is to be noted that there has been much interest in developing a controller such as NOX [12] which uses “open flow” for communication with the switches. Therefore implementation of such CS is feasible.

Figure 1. Three Tier Fat tree architecture with Core, Aggregate and Edge Switches

4. Proposed Load balancing Framework

The key idea of our work is to measure the load on the possible paths and distribute the traffic load across multiple paths in proportion to the load information estimated. For reasons of scalability, we use this load balanced routing only when the load exceeds a certain threshold. When the load is low, we forward packets using known hash based routing technique until reaching the threshold.

Our load balancing framework has the following Modules.

- The Route computation module computes the possible candidate paths between any pair of end hosts.
- The Load Measurement Module measures the load on various links and sends this information to edge switches through the chief supervisor.
- The load-aware routing module distributes the packets to various paths based on the load information obtained for the path.
- The fault tolerant routing module redistributes the traffic on the failed links to the other working paths.

Each of the modules will be described in more detail in the following subsections.

4.1. Route computation

In a Fat tree (Fig. 1), there exists multiple paths between any two hosts. We identify two cases. In the first case, the two end hosts are located in the same pod. Here, there exist multiple paths between

these hosts traversing the edge and aggregate switches only within the pod. We don't consider the path traversing up through the core switches as they are longer and consume more bandwidth. We call this intrapod routing. In the second case, the two end hosts are located in two different pods. Here, there exist multiple paths between these hosts traversing the edge, aggregate and core switches. In fact, each of the paths from a source host reaches one core switch in the upward direction and there exist one path from a core switch to a destination host in the downward direction. We call this as interpod routing. The possible routes are pre computed and stored at the edge switch.

We use an example to illustrate the intrapod and interpod routing. Consider host A and host D which are in the same pod. The possible routes we consider are; host A-E₁-A₁-E₂-host D, host A-E₁-A₂-E₂-host D. Note that these paths do not traverse any core switch. Now we consider the case of routing between host A and host L which are in different pods, thus the requiring interpod routing. The possible routes are; host A-E₁-A₁-C₁-A₆-E₆-Host L, host A-E₁-A₂-C₃-A₅-E₆-host L, host A-E₁-A₁-C₂-A₆-E₆-host L, and Host A-E₁-A₂-C₄-A₅-E₆-Host L.

4.2. Load Measurement

The key idea of our load balancing 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. The key idea is to let the source edge switch request the chief supervisor to return the required load information on the links along various paths. We measure the link load and path load in terms of number of packets per second. The chief supervisor sends control packets to various switches along the path to get the measured link load information. It then passes the link load details to the edge switch. We now explain the procedure in detail.

The source edge switch sends the *path_load_req* control packet to the chief supervisor requesting to return the load information on the paths specified. This control packet carries information that includes source edge switch, destination edge switch, the paths between them. Upon receiving this control packet, the chief supervisor first extracts the details of various switches and links along the paths specified in the control packet. The chief supervisor prepares control packets *link_load_req* each containing the details of the switch and its associated links whose load information is requested. These control packets are then sent to the corresponding switches. Upon receiving a *link_load_req* packet, a switch returns the load information of the specified links based on its queue length information. This load information is sent back to the chief supervisor using the control packet namely *link_load_reply*. After receiving *link_load_reply* control packets from various switches, the chief supervisor calculates the path load for each of the paths as required by the source edge switch. The path load on a path is computed as the maximum load on any of the links traversed by the path. The chief supervisor then prepares *path_load_reply* controlpackets, with each containing the details of path and the associated paths load. The chief supervisor then sends each control packet to the requesting source edge switch.

We use an example to illustrate the Load measurement mechanism. We consider the case of routing between edge switch E₁ and edge switch E₄. In this case, the control packet *path_load_req* (E₁, E₄, P) is sent from the source edge switch E₁ to the chief supervisor. Here, P represents the set of paths {p₁, p₂, p₃, p₄} where p₁ is A₁-C₂-A₃, p₂ is A₁-C₁-A₄, p₃ is A₂-C₃-A₃, and p₄ is A₂-C₄-A₄. Upon receiving the *path_load_req* packet from E₁, the chief supervisor first extracts the details of various switches A₁, C₁, A₂, C₂, A₃, C₃, A₄ and C₄, and links along the paths specified in the control packet. The chief supervisor then prepares control packets *link_load_req*(s, L) for each of the switch s. Here, L is the set of the required links at s. One such packet is *link_load_req*(A₁, {A₁-C₁, A₁-C₂}). Upon receiving a *link_load_req*, switches A₁, C₁, A₂, C₂, A₃, C₃, A₄ and C₄ return the load information of the specified links based on the queue length information. The load information for each of the specified links is sent by switch s back to the chief supervisor using the control packet namely *link_load_reply*(s, {<link, load>}). For example, switch A₁ sends the packet *link_load_reply*(A₁, {<A₁-C₂, 4>, {A₁, C₁, 3>}). After receiving the *link_load_reply* control packets from various switches, the chief supervisor calculates the path load for each of the paths as as the maximum load on any of the links traversed by

the path. Mathematically this is represented as path load pl_{ij}^m where i and j are edge switches and m is the path identifier.

$$pl_{ij}^m = \max(g, l_{g,ij}^m) \forall g \in \text{links of path } m \quad \dots\dots\dots (1)$$

The chief supervisor then prepares the control packet *path_load_reply* ($\{ \langle p_1, pl_1 \rangle, \langle p_2, pl_2 \rangle, \langle p_3, pl_3 \rangle, \langle p_4, pl_4 \rangle \}$), where p_i and pl_i represent the path i and its path load, respectively. As an example, we show the path and path load information in Table 2. When edge switch E_1 receiving the *path_load_reply* packet, it stores the information in the *load_information* Table as shown in Table 2.

Table 2. Path LOAD_INFORMATION Table at Switch E_1

Path	Route	Path Load
1	$E_1-A_1-C_1-A_4-E_4$	4
2	$E_1-A_1-C_2-A_3-E_4$	2
3	$E_1-A_2-C_3-A_3-E_4$	1
4	$E_1-A_2-C_4-A_4-E_4$	4

The load measurement protocol operations are shown in Fig. 2.

Figure 2. The load measurement protocol

4.3. Load aware adaptive routing algorithm

In this section, we describe the proposed load-aware adaptive routing (LAAR) algorithm. As discussed in the previous section, an Edge switch updates the *load_information* table based on the dynamic condition. The switch then determines the proportion of the traffic to be sent along the various possible paths. Let p_i be the path i , pl_i be the path load for path i , and k be the number of paths between a given pair of edge switches. The fraction f_i , $i=1,2,\dots,k$ of the traffic to be sent through path i is calculated as follows:

$$f_i = \frac{(1/pl_i)}{\sum_{i=1}^k (1/pl_i)} \quad i=1, 2, 3, \dots k \quad \dots\dots\dots (2)$$

Now, the traffic is split among the paths according to the fractions calculated as above. The path id information is attached to the packet, so that other switches will know the path or output link through which the packet is to be forwarded. Table 3 below shows the fractions calculated for the paths between the host B and host G.

Table 3. Fraction of the traffic transmitted along the paths based on the load

Path	Route	Path Load	Fractions
1	E ₁ -A ₁ -C ₁ -A ₄ -E ₄	4	0.125
2	E ₁ -A ₁ -C ₂ -A ₃ -E ₄	2	0.25
3	E ₁ -A ₂ -C ₃ -A ₃ -E ₄	1	0.5
4	E ₁ -A ₂ -C ₄ -A ₄ -E ₄	4	0.125

The algorithm is given as follows:

Algorithm 1: Load Aware Adaptive routing algorithm

```

do  $\forall$  flows  $\mathcal{F}$ 
/* Here i and j are edge switches, k represents number of paths between i and j and m is the path identifier*/
  if (flow  $\mathcal{F}$  is new)
    if (load in source edge switch < threshold  $\mathbf{T}$  )
      Perform hash routing
    else
      if (load  $\frac{L_{ij}}{P_{ij}^m}$  determined  $\forall m$ )
        for m = 1 to k do
          
$$f_{ij}^m \leftarrow \frac{\left(\frac{1}{P_{ij}^m}\right)}{\sum_{m=1}^k \left(\frac{1}{P_{ij}^m}\right)}$$

        end-for
      for every n packets do
        Send  $n*f_1$  packets through path 1
        Send  $n*f_2$  packets through path 2
        .
        .
        .
        Send  $n*f_m$  packets through path m
      end-for
    end-if
  end-if
while no more flows

```

4.4. Fault Tolerance

In this section, we describe how the load will be redistributed in the event of link failure. When the link fails, the corresponding switches first stop transmitting through the failed link. If the switch finds that the link is down, it routes the control packet *failure_notification* packet to the Chief Supervisor about the link failure containing the identifier of the two switches connecting the failed link. The Chief

supervisor upon receiving this *failure_notification* packet forwards the packet to all the edge switches. An edge switch upon receiving the *failure_notification* packet checks if this link is traversed by any of the paths in its *load_information* table. To facilitate redistribution of the traffic on the failed paths to the working paths, the fraction of the traffic is distributed on the paths are recalculated. Without the loss of generality. Suppose that path p_k fails among the k paths between a given pair of edge switches. Each fraction f_i , $i=1, 2, \dots, k-1$ is calculated as follows:

$$f_i = \frac{\left(\frac{1}{p_i l_i}\right)}{\sum_{m=1}^{k-1} \left(\frac{1}{p_m l_m}\right)}, i=1, 2, 3 \dots k-1 \quad \dots (3)$$

As an example, suppose that link $A_2 - C_4$ fails in the fat tree shown in Fig.2. The corresponding switches A_2 and C_4 send the *notification_failure*(A_2-C_4) packet to the chief supervisor. Upon receiving this packet, the chief supervisor forwards it to all the edge switches E_1 through E_8 . Edge switch E_1 , now finds from Table III that path 4 traverses this failed link $A_2 - C_4$. It then calculates the fractions of the traffic to be distributed among the paths other than path 4. The updated values are shown in Table 4.

Table 4. Fraction of Traffic to be transmitted after the failure of link A_2-C_4

Path	Route	Path Load	Fractions
1	$E_1-A_1-C_1-A_4-E_4$	4	0.143
2	$E_1-A_1-C_2-A_3-E_4$	2	0.286
3	$E_1-A_2-C_3-A_3-E_4$	1	0.571
4	$E_1-A_2-C_4-A_4-E_4$	-	0

5. Performance Study

In this section we evaluate the performance of the proposed algorithm, called load aware adaptive routing algorithm (LAAR) by simulation. The simulation was performed using Omnet++ 4.0, an open-source, extensible, modular, component-based C++ simulation library and framework, with an Eclipse-based IDE and graphical runtime environment [8].

We use different traffic patterns, namely, Stride1, Stride2, Stride4, Stride8 and Bijective mapping. Such traffic patterns have earlier been used as benchmark in the literature [7]. The results obtained for LAAR is compared with that of the static hash-based algorithm. We use the performance metrics – bisectional bandwidth, load distribution, and link utilization. The traffic patterns considered are listed and described below.

- Stride 1 – in this pattern, each $host_i$ sends packets to $host_{(i+1) \bmod 16}$
- Stride 2 – in this pattern, each $host_i$ sends packet to $host_{(i+2) \bmod 16}$
- Stride 4 – in this pattern, each $host_i$ sends packet to $host_{(i+4) \bmod 16}$
- Stride 8 – in this pattern, each $host_i$ sends packet to $host_{(i+8) \bmod 16}$
- Bijective – here every host randomly selects a destination host and each host is a source for only one traffic flow and destination for only one traffic flow.

For each of the traffic patterns considered, each host sends 10000 packets, each of size 1500 bytes, to the destination host. Each switch in the three tier fat tree architecture considered is a four port switch and there are 16 hosts in total. Fig. 3 below shows the simulation setup using Omnetpp 4.0 simulator.

Figure 3. Simulation setup using Omnetpp simulator

For the calculation of bisection bandwidth, the network is split into two halves by a vertical divider line such that each half of the network has equal number of nodes and the two halves are symmetric. The Bisection bandwidth is the sum of the bandwidths of the links crossing the divider line. In the topology considered here, the divider line which splits the network into two equal halves is between C_2 and C_3 . Hence for the calculation of bisection bandwidth the links considered are $C_1 - A_6$, $C_1 - A_7$; $C_2 - A_6$, $C_2 - A_8$; $C_3 - A_2$, $C_3 - A_3$; $C_4 - A_2$, $C_4 - A_4$. Figure 4 depicts the divider line for the topology considered in this paper. For the sake of simplicity, the CS is not shown in the fig.4.

Figure 4. Divider line dividing the network into two equal halves

Figure 5. Comparison of hash_based and our proposed LAAR algorithm in terms of bisection bandwidth

Fig. 5 plots the bisection bandwidth obtained using various traffic patterns considered under study. From Fig. 5, it is clear that the bisection bandwidth of LAAR is higher in most of the traffic patterns compared to the hash-based routing. This is due to the fact that in LAAR, traffic is equally distributed across all the switches and roughly equal amount of data flows across the divider line which is not the case with the static hash-based routing.

Figure 6(a). Load distribution using LAAR

Figure 6(b). Load distribution using hash-based routing

The plots in Fig. 6(a) and (b) shows the load distribution across the core switches. It can be observed that using LAAR the load gets distributed more evenly across the core switches thereby not overloading a particular core switch. However hash-based routing performs worse for the traffic patterns considered.

Figure 7. Comparing the Load distribution for stride 8 traffic pattern

To get a better understanding of the load utilization, we use one of the traffic patterns and the result of the two algorithms are compared and plotted in Fig. 7. As expected LAAR has even load distribution when compared to Hash-based routing. Two of the core switches C_1 and C_4 are heavily loaded whereas the other two core switches C_2 and C_3 are lightly loaded.

Figure 8. Load utilization using LAAR algorithm

Fig. 8 plots the link utilization using LAAR for the traffic patterns, stride 4 and stride 8. For the stride 4 pattern the link utilization averages to 655 Mbps for the links considered as shown in Fig. 8. Similarly for the stride 8 pattern, the link utilization averages to 625 Mbps.

Figure 9. Load distribution using the proposed approach without and with a link failure

Fig. 9 compares the scenario of a link failure with that under normal working condition. From Fig. 9 it is clear that during the normal operation of the network the load is more evenly distributed across the core switches. The figure also depicts the load distribution for the failure of the link $A_2 - C_4$. Here the packets reaching C_4 are affected and thus packets arriving at C_4 are less compared to the other switches. Further, we can observe that the packets are more evenly re distributed among the other core switches i.e. C_1 , C_2 and C_3 .

6. Conclusions

In this paper, we addressed the problem of load balancing at layer-2 level in Data Centre networks. We developed a load balancing framework which dynamically distributes traffic across multiple paths based on the path load conditions so as to achieve load balancing. Thus the traffic is routed to adapt to the dynamic load conditions. We also considered the case of link failures wherein traffic on the failed paths are evenly redistributed across the working paths. We demonstrated the effectiveness of the proposed load balancing mechanism using various traffic patterns through simulation results.

7. References

- [1] N. Spring, R. Mahajan, and D. Wetherall, and R. Perlman, "An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN", SIGCOMM CCR, 1985.
- [2] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh.Viking, "A Multi-spanning-tree Ethernet Architecture for Metropolitan Area and Cluster Networks", In Proc. of INFOCOM, 2004.
- [3] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz Parveen, and Patel Sudipta Sengupta , "VL2: A Scalable and Flexible Data Center Network", In Proc. of SIGCOMM'09, Barcelona, Spain, August 2009.

- [4] Radhika Niranjana Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric", in Proc. of SIGCOMM'09, August 2009.
- [5] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks", in Proc. of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10), April 2010.
- [6] Theophilus Benson, Ashok Anand, Aditya Akella and Ming Zhang, "Understanding Data Center Traffic Characteristics", in Proc. Of SIGCOMM'09, August 2009.
- [7] Mohammad Al-Fares, Alexander Loukissas and Amin Vahdat, "A Scalable, Commodity Data Center Network Architecture", In Proc. of SIGCOMM'08, August 2008.
- [8] <http://www.omnetpp.org>
- [9] C. Hoops, "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, Internet Engineering Task Force, 2000.
- [10] Yong Khai Leong, Khin Mi Mi Aung, Pantelis Sophoclis Alexopoulos, "Storage System Architecture for Data Centers of the Future", IJACT: International Journal of Advancements in Computing Technology, Vol. 4, No. 9, pp. 184 ~ 192, 2012
- [11] Charles E. Leiserson, "Fat-trees: Universal Networks for Hardware-Efficient Supercomputing, IEEE Transactions on Computers, Volume 34, Issue 10 (October 1985), Pages: 892 - 901.
- [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] Roy Lao Sahagun, Shu Qin Ren, Ho Seng Beng, ,Khin Mi Mi Aung, "Development of Intelligent Network Storage System with Adaptive Decision-Making", IJACT: International Journal of Advancements in Computing Technology, Vol. 4, No. 2, pp. 122 ~ 131, 2012