Data Center Traffic Scheduling with Hot-Cold Link Detection Capabilities

Anis Yazidi
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway
anisy@oslomet.no

Hussein Abdi
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway

Boning Feng
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway

ABSTRACT

Software-Defined Networking (SDN) has been one of the most discussed areas in computer networking over the last years. The field has raised an extensive amount of research, and led to a transformation of traditional network architectures. The architecture of SDN enables the separation of the control and data planes and centralizes the network intelligence. Today's data center networks are clusters of thousands of machines. The most used routing protocol in Data centers is Equal-Cost Multi-Path Protocol (ECMP) which relies on a per-flow static hashing that is known to cause bandwidth loss because of long term collisions. In this paper, a traffic engineering approach built on the concept of SDN is presented that aims to enhance the least-loaded link routing mechanism with intelligent monitoring capabilities. In this perspective, we introduce Hot and Cold link detection (HCLD) mechanism. Our HCLD permits to dynamically re-route heavy flows from heavily utilized links (Hot links) while attracting more flows to lowly utilized links (Cold links). Comprehensive experimental results show that the devised flow scheduling solution outperforms the widely used ECMP. Results also demonstrate that dynamic monitoring of traffic statistics could be used to better utilize the total available bandwidth of the network in a reactive manner.

CCS CONCEPTS

• **Networks** → *Network resources allocation*; *Traffic engineering algorithms*;

KEYWORDS

Data Center, Traffic Engineering, SDN, Hot Link, Cold Link, ECMP

ACM Reference Format:
Anis Yazidi, Hussein Abdi, and Boning Feng. 2018. Data Center Traffic Scheduling with Hot-Cold Link Detection Capabilities. In *International Conference on Research in Adaptive and Convergent Systems (RACS '18), October 9–12, 2018, Honolulu, HI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3264746.3264797

RACS '18, October 9–12, 2018, Honolulu, HI, USA

ACM ISBN 978-1-4503-5885-9/18/10…$15.00
https://doi.org/10.1145/3264746.3264797

1 INTRODUCTION:

The number of Data centers owned by large public and private organizations has witnessed a dramatic increase during the last decade. In parallel with the rise of cloud computing, many organizations choose to move their operations, storage and computation to data centers owned by cloud computing providers [2].

A data center network is traditionally based on a *layered* [4] [9] or a three-tier approach. Such a three-tier network architecture consists of three layers of switches and routers. The layered approach is designed to enhance scalability, high performance and flexibility and to also improve the maintenance associated with data center networks. These layers are explained below:

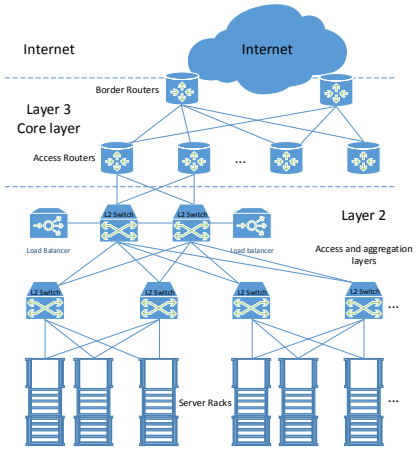

Figure 1: The architecture of a traditional layered data center.

- Access layer: This is where the servers are physically connected to the network by connections to the Layer 2 switches, also called the Access or Edge switches.
- Aggregation layer: This layer provides functions such as service module integration, Layer 2 domain definitions, spanning tree and default gateway redundancy.
- Core layer: This layer handles all the incoming and outgoing traffic that comes in and leaves the data center. This layer provides the connectivity required to various aggregation modules. It handles the Layer 3 networking with the access and border routers.

Due to the exponential growth of the cloud in data centers and the evolution of the computers, computing power is no longer the

constraining factor in data centers. The servers are becoming increasingly powerful and as the phenomenon of Cloud Computing grows, the number of communicating machines correspondingly has dramatically increases. Thus, data centers are faced with inherent problems in the traditional data center network (DCN) architecture. This leads to real problematic issues such as bandwidth bottlenecks, oversubscription in the higher layers and the underutilization of the lower layers of the data center network [3]. To resolve this, several new approaches to designing data center network topologies have been proposed in the recent years, one of which is the "tree topology" discussed below).

A tree topology: As mentioned previously, modern-day data centers usually follow traditional three-tier (or three-layer) network architectures. At the lowest level, referred to as the *access tier*, hosts connect to one or multiple access switches. Each of the access switches is connected to one or multiple aggregate switches at the aggregation layer. The aggregation switches, in turn, connect to multiple core switches at the core layer. This design creates a tree-like topology where packets are forwarded according to a Layer 2 logical topology [8]. The higher level network elements are usually enterprise-level devices and are often highly oversubscribed.

ECMP or Equal-Cost Multi-Path is considered as the state of the art forwarding protocol in data center and enterprise environments [6]. ECMP is used to statically assign flows across available paths using flow hashing. The static mapping does not take account for current network utilization or flow size resulting in hash collisions among elephant flows.

The goals of this paper is to investigate how we can remediate to existing flow-aware mechanisms such as per-flow scheduling techniques in Data Centers, namely ECMP, so that we can optimize the usage of network resources. The reliance on per-flow static hashing of current IP multi-pathing protocols tends to cause substantial bandwidth losses because of long term collisions. Therefore, the paper aims to solve these issues by creating an intelligent flow scheduling component that takes the advantage of the global view of the network provided by SDN.

2 RELATED RESEARCH

Al-Fares et al. [1] propose to leverage the high-end commodity switches to support the full aggregate bandwidth. Curtis et al. [5] introduced, Mahout, a traffic management system, aiming to solve underutilized bisection bandwidth by implementing an Openflow central controller for better detection and handling of large elephant flows. Both papers advocate dynamic traffic flows rather than the traditional static ECMP routing. In a subsequent work, Alfares et al. [2] present a dynamic flow scheduling system, called Hedera. Hedera schedules flows based on Global-first-fit. According to the Global-first-fit, a flow is assigned to the first path that fits that flow, i.e, a path that has enough capacity. Hedera implementation consists of a simulated data center with 8,192 hosts and the simulation results showed an improvement of the bisection bandwidth by 96 percent.

Trestian et al. [10] differentiates between long-lived and short-lived flows. Dedicated traffic management mechanisms are used to handle the long-lived or elephant flows while short-lived or mice flows are handled with baseline routing methods. The paper explains through an example that traditional ECMP routing mechanism can cause congestion to short-lived flows. Trestian et al. present MiceTrap, an OpenFlow-based traffic engineering method targeting short-lived flows in data centers.

Long et al. [7] introduce Laberio, a routing algorithm for balancing the traffic dynamically during data transmission in OpenFlow enabled data center networks. Laberio finds the biggest flow in busiest link, and reroutes it to a new available alternative path.

3 OUR SOLUTION: LEAST-LOADED ROUTING WITH HOT-COLD LINK DETECTION CAPABILITIES

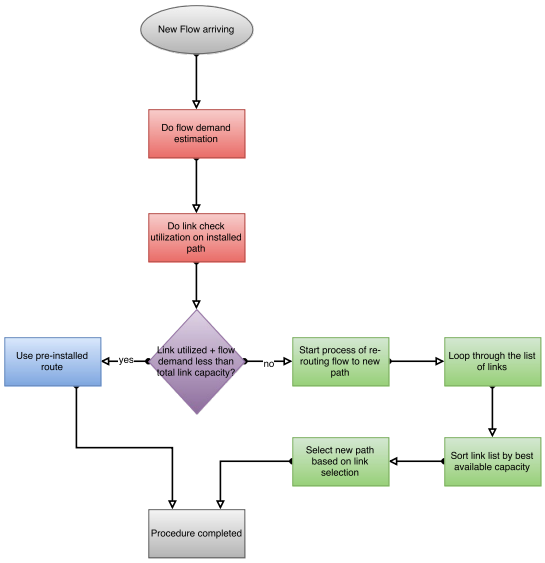

Figure 2: Flowchart of least-loaded routing function

The proposed routing component will be based on routing flows dynamically based on link utilization over all paths on the network. The overall approach is depicted in Fig. 2. The least-loaded routing component adopts the following logic: the controller has a global view of all installed paths among switches, end-hosts, and provides the foundation for creating new routing functions to forward traffic flow based on any given rule. When a new flow enters the network, the controller performs a check on all links capacities and the flow demand will be performed. If the flow demand and each link utilized values in designated paths are greater than the overall link capacities, then the routing component will be called to install the flow to a new path which has the available capacities of links.

ECMP routing component

This controller component uses a static network description to create paths. It contains a mode for routing based on a hash function, called reactive mode which will be implemented for the ECMP routing. In this mode hashing will come from the ECMP 5-tuple. The same way the proposed routing component will be

executed, this module will also be run along with Mininet and a FatTree topology.

Least-loaded routing pseudocode

The implementation of the least-loaded routing function is described as a pseudocode in detail below:

Algorithm 1: Least-loaded link function

```
Input: Set of N links to be assigned to K flows
Output: List of high capacity links to be used by flows
for i, path in paths do
    AboveCongestionThreshold = False;
    ListPaths <—- possible paths from src to dst;
    SortPaths <—- by position from left to right;
    AllPaths = Length_Paths;
    for i in range(0, AllPaths) do
        LinkID = Links(path[i], path[i + 1]);
        if Links_LinkID + FlowDmd > Capacity then
            AboveCongestionThreshold = True;
            for i in sorted(Links.values() do
                LinkID = Links_path_i,i+1;
                LinkID += FlowDmd;
            end
            return path;
        end
    end
end
```

HCLD: Flow modification

HCLD, *Hot-link and Cold-link Detection* will serve as a flow rerouting component. As more flows enter the network some of the paths and links will be more utilized than others. To achieve an overall good bisection bandwidth and an efficient utilization of the network resources, large flows that demand more than what each link can carry will be redirected to more suitable paths. Flow modification will occur each time a flow is larger than the link threshold value, which will trigger the *rerouting* method in this component to select a new route. Links that are marked as congested will be categorized as *hot-links*, which will be links to avoid routing traffic through. Consequently links that are not much utilized will be categorized as *cold-links*. The demand of each flow will calculated based total bytes divided by duration of the flow and divided by link bandwidth. All these attributes will be collected using the OpenFlow flow statistic counters, and when the demand is known the flow will either pass through using the intended paths or redirected. To install the new route, OpenFlow modification messages will be sent from the controller to the switches and then the new route for the specific flow will be installed in the switches flow tables.

Traffic Statistics and flow modification component

OpenFlow switches are equipped with multiple counters for capturing flow entries and monitoring of interfaces. To capture link usage statistics and measure each link, the received and transmitted byte counters of the interfaces will be accessed. Other information such as packet counters are also available. The collection of traffic statistics will be achieved by implementing OpenFlow's statistics request and reply messages between the controller and switches. Two types of statistics will be focused on while running the experiments, port and flow statistics.

Port and flow statistics

To strengthen the proposed routing component of Least-loaded, the used bandwidth of each link must be monitored and calculated. OpenFlow provides with port-level functionalities making it possible to get detailed statistics of ports on each switch. The controller will periodically send a *OFPortStatsRequest* message and the switches will respond with a *OFPortStatsReply* that contains information of switches retrieved from their counters. Statistics such as the number of transmitted and received bytes will be used to calculate link utilization. As the link utilization is measured a check for congestion will also be performed. A threshold at 70% of the links capacity will be set and if utilization is above or equal to this threshold then an event will be raised to the controller.

Statistics per-flow is desired to obtain from switches across the network. Every switch offers byte counters for flow entries along the flow path. Each switch has a flow-table containing the flow entries and some flow statistics. As the experiments generate traffic flows among different end-host on the network we will measure how the two routing components perform in terms of how many packets and bytes are transmitted and received, and for how long flows are alive. A summary of OpenFlow counters is given in Table 1.

Table 1: OpenFlow counters

Counters	
Per Interface	Per Flow
Received Bytes Transmitted Bytes Received Packets Transmitted Packet Received Drops Transmit Drops Receive Errors Transmit Errors Receive Frame Alignment Errors Receive Overrun Errors Collision	Received Bytes Received Packets Duration (sec) Duration (nanosec)

4 EXPERIMENTS

4.1 Network Topology

In this paper, the FatTree data center network architecture will be used to test the effect of the proposed routing component compared to the existing multipathing routing.

Characteristics

The FatTree topology will be organized by two main elements; layers and a k value. In the k FatTree there are k pods, each containing two layers (edge and aggregate) of $k/2$ switches. In the lowest layer, half of edge switch ports connect to $k/2$ hosts and the remaining will be directly connected to $k/2$ aggregate switches. Aggregate switches are connected to

$$k/2^2$$

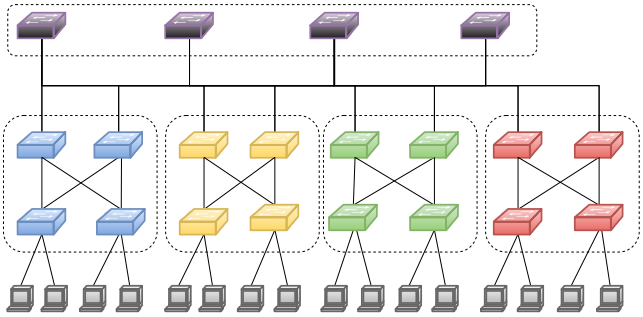

Figure 3: FatTree topology used in the paper

core switches. Core switches are responsible of connecting all the pods in the network together.

4.2 Traffic Pattern design

For all experiment sets different traffic patterns will be generated and implemented. To fully compare the performance of the proposed least-loaded component to current hash-based multipath forwarding such as ECMP, generating a variety of traffic patterns that stress the simulated network is an important aspect of this project. The different patterns are divided into three categories:

1. Stride[i]: Host x sends to host y by the rule of $y = (x + \text{i})$ *mod* (num_of_hosts).

2. Random: Host x sends to any other host in the network.

3. Hotspot: One host in the network becomes the target host all other host sends to.

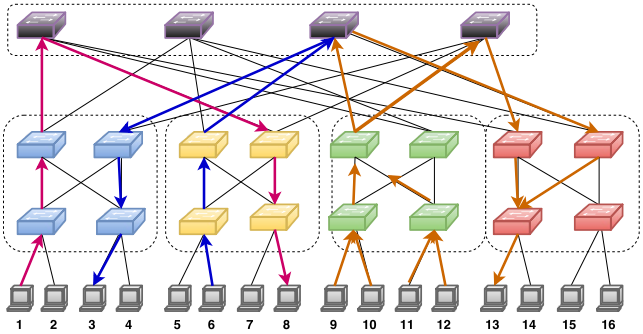

Figure 4: Traffic flow patterns

For illustration purposes, Fig. 4 shows how flows will implemented from source to destination hosts with the three traffic patterns. A stride flow will be generated as the pink flow from host 1 to host 8. The blue flow from host 6 to host 8 is an example of a random flow. Hotspot flows will in the implementation consist of one host being the target host receiving flows from all other hosts in the network. This illustrated as an example where hosts 9, 10, 11, 12 all send hostspot flows to the target host 13.

In the exception of the 'stride' traffic pattern, these patterns contain an element of randomness at the selection of destination host. The traffic patterns will be created in a python script, and besides the source and destination IP addresses it will also contain port number, flow size, start and end time of each flow and a flow sequence number. Each file will be used to generate the traffic with *iperf* and a corresponding file of results will stored and used in later analysis.

Experiment A: Compare scheduling performance of least-Loaded and ECMP on FatTree topology with 16 hosts, 20 switches and 48 links. Collect flow statistics from switch interface counters. From collected flow statistics enable HCLD component to ensure load balancing by redirecting flows from *hot-links* and attract flows to *cold-links*.

Experiment B: Compare scheduling performance of least-Loaded and ECMP on FatTree topology with 54 hosts, 45 switches and 164 links.

Experiment C: Compare scheduling performance of least-Loaded and ECMP on FatTree topology with 128 hosts, 80 switches and 348 links.

4.3 Experiment A

4.3.1 Experiment A: results. The experiment A was conducted on a $k = 4$ FatTree topology, which resulted in 16 end hosts and 20 switches.

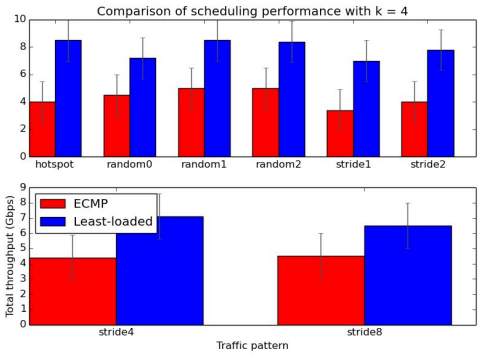

Figure 5: Experiment A: FatTree k=4

The script implemented the traffic pattern files, generated traffic flows using iperf and saved the results of total average throughput of each pattern files. The experiment was run first on the POX controller using the least-Loaded module and then on the ECMP module. Traffic flows was generated as samples and a total of 30 samples were run on all traffic patterns. Throughput was collected by adding up the the number of rx and tx bytes from each sample when generating flows on each traffic pattern. The least-Loaded component generated much higher total average throughput than ECMP, with an average of 942875000 bytes per second (7.543 Gbps and a standard deviation of 0.5345). ECMP managed to deliver an average of data transfer rate at 578125000 bytes per second (4,625 Gbps and a standard deviation of 1.0606).

4.3.2 Experiment A: analysis. In the experiment A set the size of the network made it easier to generate traffic flows from the traffic

patterns that were created. This experiment set was also the one which resulted in least variance in the results of throughput of data in both the routing components. The result shows that the least-loaded component was able to successfully transfer much more flows than ECMP for all patterns. There are a couple of reasons for these results; firstly the ECMP uses a static hash-based scheme where the forwarding paths are determined by a hash of the destination host IP. Due to this the switches in the network will route the traffic solely based on these entries installed in the flow tables, disregarding the available capacity on other paths and links in network. During the experiment time, the total average throughput would therefore be less when some flows must "wait" to pass links that don't have enough capacity to allow transmission through. Secondly, the least-Loaded component on the other hand uses a less controlled approach where flows are installed on switches based on paths with more link capacity available.

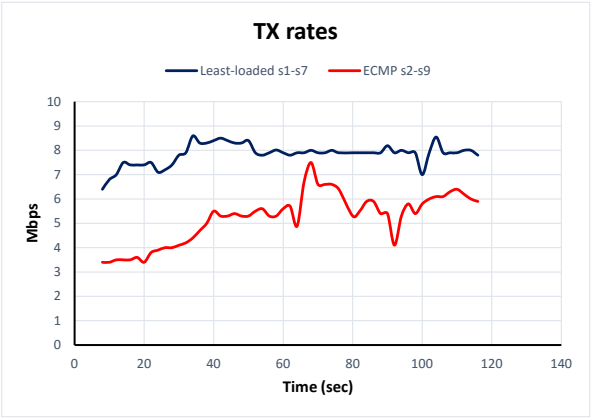

Figure 6: TX rates Experiment A

While monitoring interfaces of the switches during experiment run transmitted and received bytes were captured for both routing components. These statistics helped to view how much per interface the switches could transmit and receive when comparing the two components. Least-loaded delivered higher value of transmitted bytes per interface and held a more stable course over the experiment time. While, ECMP yielded lower performance and the values contained a notable number of variance. From these statistics it was also possible to see that ECMP would reach a limit in transmitting bytes, then as time goes and the interface waits for available capacity it then fills the link with the new flow that arrives and decreases again.

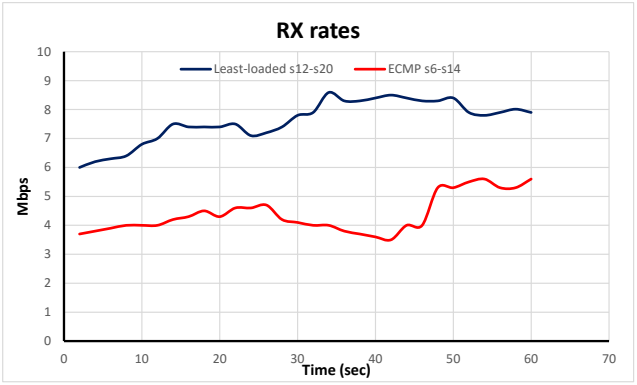

Figure 7: RX rates Experiment A

Similarly for received bytes the experiment returned higher results with least-loaded component then with ECMP. The highest number of received bytes for least-Loaded was recorded at 1125000 bytes while for ECMP the result was 625000 bytes. When measuring the throughput and comparing the scheduling performance of both modules, the mean and standard deviation of both components was collected on all traffic patterns.

Table 2: Descriptive statistics of routing modules performance in set A

Component	Mean	Stddev
Least-Loaded	7.82525	0.62498
ECMP	4.64788	0.62966

4.4 Experiment B: results

4.4.1 Experiment B: results. Experiment B was conducted on a k = 6 FatTree topology, consisting of 54 hosts and 48 switches. The same baseline experiment for experiment A was executed on this larger network. This network provided more paths and links between each source and destination and among switches. As the network was larger it took longer time to both build and wait for all switches to connect to the controller.

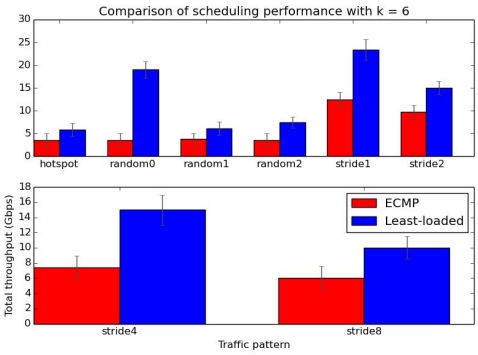

Figure 8: Experiment B: FatTree k=6

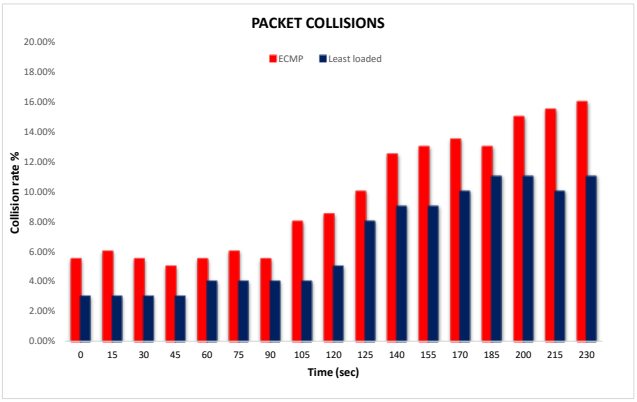

Figure 9: Packet loss rate experiment B

With larger network topology the bisection bandwidth in the network increased from experiment A. The scheduling performance of the routing modules however gave the same output results as in experiment A; least-Loaded module was able to generate better throughput than ECMP. The total average throughput for least-loaded was measured at 1604162500 bytes per second (12.833 Gbps with a standard deviation at 3.481). For ECMP the total average throughput was measured at 771125000 bytes per second (6.169 Gbps with a standard deviation at 4.297). The whole experiment took approximately 25 minutes to run.

4.4.2 Experiment B: analysis. In experiment set B the FatTree topology was increased to size $k = 6$, introducing more than double the amount of switches and hosts. The results from this experiment shows that the proposed least-loaded routing was able to outperform ECMP in scheduling performance and deliver higher throughput of data on all traffic patterns. The experiment set resulted in the largest variance of throughput in both of the modules, but most for ECMP. ECMPs low throughput on 4 out of 8 traffic patterns are the most notable. ECMP delivered an average of 4 Gbps on hotspot and all random traffic patterns before managing to deliver higher but expected throughput on the remaining traffic patterns.

Component	Mean	Stddev
Least-Loaded	7.82525	0.62498
ECMP	4.64788	0.62966

The experiment run with least-loaded component also delivered lower results than expected on the same traffic patterns as ECMP. For these communication patterns collision in path assignments, more specific for ECMP hash function, could be the reason for the variance in results as this would occur within the same switch or at a downstream switch resulting in flows being wasted not completing the transfer from source to destination.

Table 3: ECMP descriptive statistics Experiment B

Traffic Pattern	Mean (gbps)	Stddev
Hotspot	3.672	0.718
Random1	3.459	0.735
Random2	3.201	0.944
Stride1	13.735	0.931

4.5 Experiment C

4.5.1 Experiment C: result. Experiment C was conducted on a $k = 8$ FatTree topology, the largest network in the experiment with 128 hosts and 80 switches. The network and traffic flow generation was built and executed from a script. The process of building the network and establishment of switch connection was long. Traffic flow was generated manually in Mininet CLI due to inconsistency in the iperf results from the automated script.

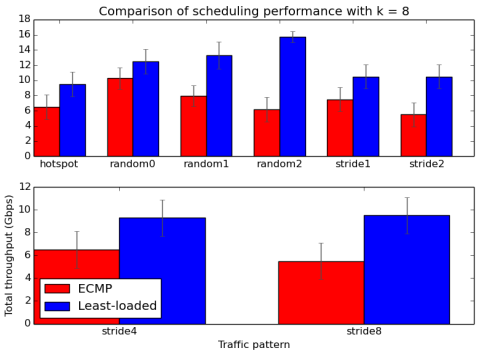

Figure 10: Experiment B: FatTree k=8

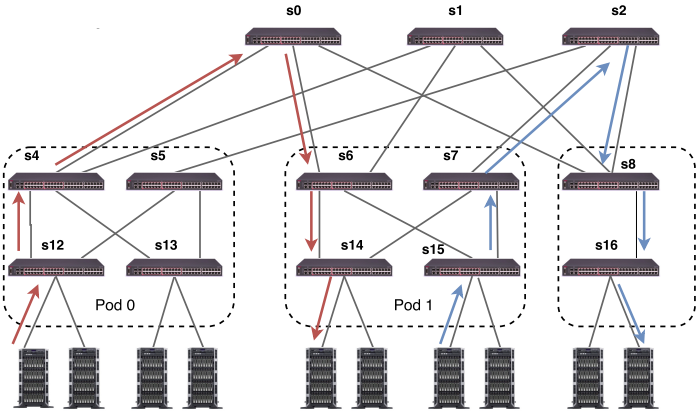

Figure 11: FatTree topology flow statistics collection

The results in this experiment set showed an increase in the total average throughput for both routing modules. The proposed least-loaded module was still able to perform better than ECMP. least-loaded measured an average of 1568375000 bytes per second (12.547 Gbps with a standard deviation of 1.607) while ECMP delivered a result of 875000000 bytes per second (7.102 Gbps with standard deviation of 1.414)

4.5.2 Experiment C: analysis. Experiment set C contained the largest number of nodes in network with 128 hosts, 80 switches and 348 links. The process of building the network was long and time-consuming. The results of the experiment shows that least-loaded component was able to deliver higher rate of throughput than ECMP and therefore provides better scheduling performance. In these results the variance of the traffic patterns were not that high as the previous experiment set, and both least-Loaded and ECMP distributed the traffic flows without any low or high spikes between the traffic patterns. ECMP managed to deliver a total average of 828125000 bytes per second (6.625 Gbps) on all traffic patterns, while least-loaded delivered a total average of 1359375000 bytes per second (10.875 GBps).

The traffic statistics component was executed along with the experiment run, collecting transmitted bytes at core switch *s0* to downstream switch *s6* in aggregate layer. Received bytes were also captured at switch *s8* from switch *s2*. As traffic flows are traversing from source to destination hosts, the statistics module was able to collect these values from the interface counters of the selected switches.

The results also shows that the least-loaded component used much less time to determine path and links selection based on the available bandwidth on the network, providing a better scheduling performance. While ECMP module runs the traffic flows based on forwarding rules determined by the hash of the destination IP, the flows being generated will run into paths containing links with less capacity as the flows are continuously sampled during the 30 iterations, resulting in lower total throughput.

4.6 Least-Loaded with HCLD

The experiment of running the FatTree topology size $k = 4$ with the least-Loaded and Hotlink and Coldlink Detection showed two kind of results; modification of large flows from heavily utilized links or *hotlinks* and assigning flows to lowly utilized links or *coldlinks*. The flow modification component was able to perform flow demand estimation and determine whether the demand was above or under the bandwidth threshold. The component captured 3 large flows that were redirected to more suitable paths and was able to assign 2 flows to links that had high capacity available. The bandwidth consumption on selected links after reassignment was monitored, for the *hotlinks* the results show that link capacity decreased with time and for *coldlinks* it increased as seen in Fig. 12.

After the reassignment of flows, whether restricting flows to pass through i.e. *hotlinks* or attracting flows i.e. *coldlinks* the results showed a distinct behavior where heavily utilized links had with time more capacity available and the underutilized links were more frequently used and became high priority links as flows were generated. This is illustrated in Fig. 13

5 CONCLUSION

The key goal of this paper was to investigate how a per-flow scheduling method could be designed and implemented in order to provide better scheduling performance against one of the most used techniques today in ECMP, and how traffic statistics could be used to help achieve this.

The paradigm of SDN was chosen as the foundation for building the data center network, designing and implementing a controller that became the intelligence of the network.

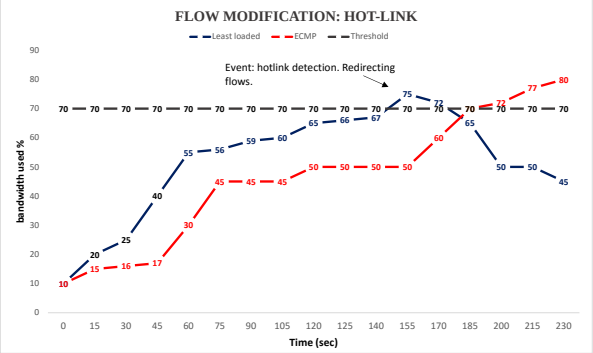

Figure 12: Redirection of flows from hotlink

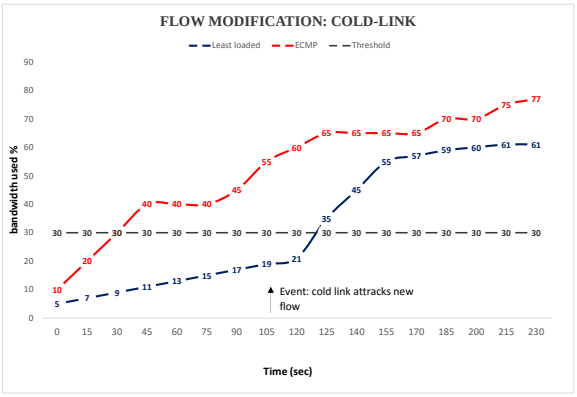

Figure 13: Assignment of flows to coldlink

The technical aspect of the problem statement is addressed by developing two components for adopting SDN in order to solve the per-flow scheduling issue of ECMP. One component was developed for routing traffic based on link utilization, and one component for statistics gathering and flow modification. For both components, an underlying infrastructure of an OpenFlow controller was implemented along with the implementation of Mininet in order to emulate a data center network. The least-loaded component was able to achieve the expected shortcomings of ECMP with the ability of routing elephant flows without long term collisions. The statistics component was developed using OpenFlow switch interface and port counters to collect flow statistics. The collected statistics was then used to determine if flows should be rerouted. The flow modification part of the component, namely, "HCLD" or Hotlink and Coldlink detection mechanism enabled the flows to be rerouted away from heavily utilized links and attracted more flows to lowly utilized links. The effect of the detection mechanism helped to improve the overall usage of the network resources.

REFERENCES

[1] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev. 38*, 4 (Aug. 2008), 63–74.

[2] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 19–19.

[3] Bilal, K., Malik, S. U. R., Khalid, O., Hameed, A., Alvarez, E., Wijaysekara, V., Irfan, R., Shrestha, S., Dwivedy, D., Ali, M., et al. A taxonomy and survey on green data center networks. *Future Generation Computer Systems 36* (2014), 189–208.

[4] Cisco. Data Center Architecture Overview . http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_1.html. [Online; accessed 22-February-2015].

[5] Curtis, A. R., Kim, W., and Yalagandula, P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings of INFOCOM* (April 2011), pp. 1629–1637.

[6] Hopps, C. Analysis of an equal-cost multi-path algorithm, 2000.

[7] Long, H., Shen, Y., Guo, M., and Tang, F. Laberio: Dynamic load-balanced routing in openflow-enabled networks. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (March 2013), pp. 290–297.

[8] Meng, X., Pappas, V., and Zhang, L. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE* (2010), IEEE, pp. 1–9.

[9] Press, C. Cisco data center infrastructure 2.5 design guide.

[10] Trestian, R., Muntean, G. M., and Katrinis, K. Micetrap: Scalable traffic engineering of datacenter mice flows using openflow. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)* (May 2013), pp. 904–907.