Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

A SIMULATION AND EMULATION STUDY OF SDN-BASED MULTIPATH ROUTING FOR FAT-TREE DATA CENTER NETWORKS

Eric Jo
Deng Pan
Jason Liu

School of Computing and Information Sciences
Florida International University
Miami, Florida, USA

Linda Butler

Department of Computer Science and Engineering
University of South Florida
Tampa, Florida, USA

ABSTRACT

The fat tree topology with multipath capability has been used in many recent data center networks (DCNs) for increased bandwidth and fault tolerance. Traditional routing protocols have only limited support for multipath routing, and cannot fully utilize the available bandwidth in such networks. In this paper, we study multipath routing for fat tree networks. We formulate the problem as a linear program and prove its NP-completeness. We propose a practical solution, which takes advantage of the emerging software-defined networking paradigm. Our algorithm relies on a central controller to collect necessary network state information in order to make optimized routing decisions. We implemented the algorithm as an OpenFlow controller module and validated it with Mininet emulation. We also developed a fluid-based DCN simulator and conducted experiments, which show that our algorithm outperforms the traditional multipath algorithm based on random assignments, both in terms of increased throughput and in reduced end-to-end delay.

1 INTRODUCTION

Data center networks (DCNs) must provide efficient inter-connection between a large number of servers in order for the data centers to achieve the necessary economy of scale (Mudigonda and Yalagandula 2010, Wang et al. 2010). In order to provide abundant bisection bandwidth, modern DCNs usually adopt a hierarchical multi-rooted network topology, such as the fat tree (Al-Fares, Loukissas, and Vahdat 2008), VL2 (Greenberg et al. 2009), DCell (Guo et al. 2008), and BCube (Guo et al. 2009). The hierarchical feature would make such topologies easy to expand, and the multi-rooted topology would allow more routing options due to multipath.

Among the existing alternatives, the fat tree topology is especially popular for its simplicity, and has been considered in a number of recent data center designs (Heller et al. 2010, Mysore et al. 2009). Figure 1 shows a 4-pod fat tree with four hierarchical layers, including, from the bottom, hosts, edge switches, aggregation switches, and core switches. The four core switches at the top are the roots of the tree. The figure shows two possible paths between host A and host B in different colors (there should be four of them). In this network, all switches have the same number of ports, and all links are of the same bandwidth. We refer the readers to Al-Fares, Loukissas, and Vahdat (2008) and Mysore et al. (2009) for a detailed description of the fat tree topology.

Traditional link-state and distance-vector routing protocols, which have been commonly used on the Internet (Kurose and Ross 2007), cannot be readily used to take advantage of the multipath capability of the fat tree topology. For example, most IP-based routing protocols calculate routes and forward packets based solely on the packet's destination address, and as a result, all packets with the same destination would follow the same path. There are indeed protocols that support multipath. For example, some protocols come with

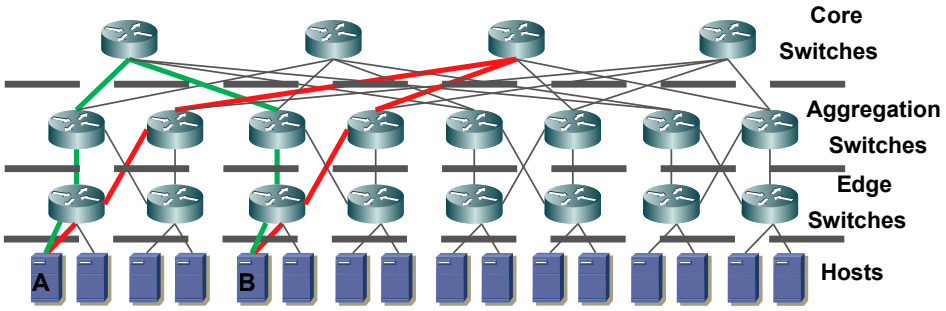

Figure 1: The fat tree topology with four pods.

the "equal cost multipath" (ECMP) feature. ECMP performs static traffic splitting based on information from the packet headers. In particular, the algorithm does not consider the bandwidth requirement of the flows. It is rather limited in terms of path optimization, since it does not consider the traffic condition of the entire network. Furthermore, the algorithm is only able to choose among the available paths with the same minimum cost. Consequently, the algorithm is limited to only a small number of alternative paths (Heller et al. 2010). One must note that DCN routing is significantly different for Internet routing. Normally, the Internet protocols would give preference to the shortest paths in order to reduce latency. By contrast, DCNs are less concerned about latencies—after all, DCNs are local area networks with close geographical distances between nodes. Routing on the DCNs should give more priority to the bandwidth utilization rather than latency.

The goal of our study is to fully explore the potential of the fat tree topology for data center networking. In particular, we design practical and efficient multipath routing algorithms with the following three objectives. First, the algorithms shall be able to improve bandwidth utilization of the entire network. In other words, the algorithms should accommodate more traffic for the same network fabric. Second, the algorithms shall be able to achieve better traffic load balance. This is to prevent the network from creating hot spots and causing congestions. Last, the algorithms shall have a lower time complexity. They should make fast routing decisions in order to handle large traffic volumes.

In this paper, we focus on load-balanced multipath routing for fat tree DCNs. First, we formulate the multipath problem as a linear program, and prove that it is NP-complete. The NP-completeness proof establishes that the problem has no known polynomial-time solutions, which directs us to seek efficient algorithms using approximation and heuristics. As a practical solution, we propose a load-balanced multipath routing algorithm that can take advantage of the emerging Software Defined Networking (SDN) paradigm (Open Networking Foundation 2012). The main idea is to use a OpenFlow controller to collect the necessary information of the entire network, based on which the algorithm can make centralized routing decisions upon each flow arrival. More specifically, our algorithm first determines all available paths that can carry the new flow, and then finds the bottleneck link of each path. A bottleneck link is defined to be the link with the minimum available bandwidth among all links along the path. The algorithm compares the available bandwidth of bottleneck links of different paths, and selects the path whose bottleneck link has the most available bandwidth. In this way, the algorithm can guarantee that the selected path minimizes the maximum link load of the entire network at the decision time. Consequently, the algorithm is able to achieve better load balance.

To validate our designs, we have implemented the proposed load-balanced multipath algorithm in two environments. First, we implemented the algorithm as a module for an open-source OpenFlow controller, which can be readily deployed in an existing data center. We used Mininet (Lantz, Heller, and McKeown 2010) to validate the implementation. Mininet is a container-based emulation environment for flexibly testing SDN applications by running the applications in networks consisted of virtual hosts and virtual switches, all running on a single physical machine. Second, we developed a fluid-based DCN simulator to

evaluate the performance of our load-balanced multipath algorithm. The simulator was implemented using the MiniSSF parallel discrete-event simulation framework (Rong, Hao, and Liu 2014). In the experiments we compared the performance of our algorithm with that of a multipath algorithm based on random path allocations. Our experiment results show that our load-balanced multipath routing algorithm consistently outperforms the traditional approach.

The main contributions in this paper are three-folds. First, we formulate the load-balanced multipath routing problem for the fat tree network as a linear program, and prove that it is NP-complete. Second, we propose a practical algorithm that can be applied in a data center using the SDN technology. Finally, through extensive emulation and simulation experiments, we demonstrate the performance superiority of the proposed algorithm over the existing approach.

The rest of the paper is organized as follows. In section 2, we review related work. In section 3, we formulate the load-balanced multipath problem as a linear program and proves its NP-completeness. In section 4, we describe the proposed SDN-based algorithm. In section 5, we describe the two implementations, one for emulation and the other for simulation, which we use to evaluate our algorithm. In section 6, we present the experiment results. Finally, we conclude the paper and outline future work in section 7.

2 RELATED WORK

Typical DCNs offer multiple routing paths for increased bandwidth and fault tolerance. Multipath routing can reduce congestion by taking advantage of the path diversity in DCNs. Multipath routing can be conducted at either layer two or layer three. Layer-two typically uses a spanning tree, where only one path is made available between a pair of nodes. A recent work by Mudigonda and Yalagandula (2010) allows multipath by exploiting the path redundancy in the network. The algorithm computes a set of available paths and merges them into a set of trees, each mapped to a separate VLAN. At layer three, equal cost multipath (ECMP) (Cisco Systems 2007) provides the multipath capability by performing static load splitting among the flows. ECMP-enabled switches are configured with several forwarding paths in a given subnet. When a packet arrives at a switch that has multiple candidate paths, the switch selects a path based on a hash function using the selected fields of the packet header, and then forwards the packet along that path. In doing so, the mechanism is able to split the traffic load among multiple paths in a subnet. It is important to note, however, ECMP does not account for flow bandwidth in making path allocation decisions, which may lead to oversubscription even for simple communication patterns. Furthermore, current ECMP implementations limit the number of choices of candidate paths to 8 or 16, which can be far fewer than what would be needed to deliver the bisection bandwidth for larger data centers (Heller et al. 2010).

There exist multipath solutions specifically designed for DCNs. Al-Fares et al. (2010) proposed two multipath algorithms: one uses global first-fit and the other uses simulated annealing. The first algorithm simply selects among all possible paths the first one that can accommodate a flow. The algorithm is simple. The drawback of the algorithm, however, is that it needs to maintain accurate information of all paths between all pairs of nodes. The second algorithm performs a probabilistic search for the optimal path. The algorithm is fast. However, it is shown that the algorithm may also converge slowly. Heller et al. (2010) proposed the ElasticTree DCN power management scheme, which includes two multipath algorithms: a greedy bin-packing algorithm and a topology-aware heuristic algorithm. The former evaluates the available paths and deterministically chooses the leftmost one with sufficient capacity. The latter uses a fast heuristic based on the topological features of the fat trees; it relies on flow splitting. Benson et al. (2010a) also proposed the MicroTE framework that supports multipath routing. Our algorithm described in this paper also considers multipath. It uses depth-first search to quickly find the possible paths among the layers of switches, and then uses worst-fit to select the candidate path. Compared with the previous approaches, our algorithm provides a fast and effective solution for achieving system-wide traffic load balance using SDN.

3 PROBLEM FORMULATION

In this section, we formulate the load-balanced multipath routing problem as a linear program, and then prove its NP-completeness for the fat tree topology by reduction from the integer partition problem.

3.1 Load-Balanced Multipath Routing Problem

A fat tree DCN can be modeled as a directed graph $G = (H \cup S, E)$, where H is the set of hosts, S is the set of switches, and E is the set of links each connecting a switch with another switch or a host. Each edge $(v_i, v_j) \in E$ has a nonnegative capacity $c(v_i, v_j) \geq 0$, which denotes the available bandwidth of the corresponding link. Let $\mathscr{F} = \{F_1, \ldots, F_n\}$ be the set of all flows in the network. A flow F_k (where $1 \leq k \leq n$) is defined as a triple (a_k, b_k, d_k), where $a_k \in H$ is the source host, $b_k \in H$ is the destination host, and d_k is the demanded bandwidth. We use $f_k(v_i, v_j) \in \{0, 1\}$ to indicate whether the flow F_k traverses the link (v_i, v_j).

The load-balanced multipath routing problem can be expressed as a linear program, whose the objective is to minimize the maximum load among all the links in the network. More formally, the load-balanced multipath routing problem can be specified as follows:

Minimize maxload

Subject to the following constraints:

$$\sum_{k=1}^{n} f_k(v_i, v_j) d_k \leq \text{maxload} \leq c(v_i, v_j),\ \forall (v_i, v_j) \in E \tag{1}$$

$$\sum_{v_i \in H \cup S} f_k(v_i, v) = \sum_{v_j \in H \cup S} f_k(v, v_j),\ \forall k \in \{1, 2, \cdots, n\}, \forall v \in H \cup S \setminus \{a_k, b_k\} \tag{2}$$

$$\sum_{v_j \in H \cup S} f_k(a_k, v_j) = \sum_{v_i \in H \cup S} f_k(v_i, b_k) = 1,\ \forall k \in \{1, 2, \cdots, n\} \tag{3}$$

Equation (1) defines *maxload*, and also states the link capacity constraint, i.e., the total demanded bandwidth on a link shall not exceed its available bandwidth. Equation (2) states the flow conservation constraint, i.e., the amount of any flow shall not change at an intermediate node. Equation (3) states the demand satisfaction constraint, i.e., for any flow, the outgoing traffic at the source and the incoming traffic at the destination shall be the same as the demand of the flow.

3.2 NP-Completeness

In this section we show the NP-completeness of the load-balanced multipath routing problem.

Theorem 1 The load-balanced multipath routing problem is NP-complete for the fat tree topology.

Proof. The proof is by reduction from the NP-complete partition problem (Garey and Johnson 1990). A partition problem decides whether a set of integers, $X = \{x_1, \ldots, x_n\}$, can be partitioned into two subsets, X_s and $X \setminus X_s$, such that the sum of elements in X_s is equal to that in $X \setminus X_s$, i.e.,

$$\exists X_s \subseteq X, \sum_{x_i \in X_s} x_i = \frac{1}{2} \sum_{x_i \in X} x_i \tag{4}$$

For reduction, we consider an instance of the partition problem with set X, and construct an instance of the load-balanced multipath routing problem for a fat-tree. The basic idea is to create a group of flows between two hosts, and the demand of each flow is one of the integers in set X. Leave only two disjoint paths between the two hosts for the flows. If there is a partition of the integer set, then we can accordingly allocate the flows to the two paths to achieve optimal load balance, and vice versa. Figure 2 shows an

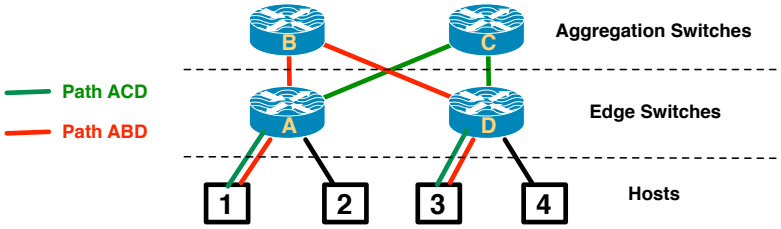

Figure 2: One pod from a four-pod fat tree topology.

example of one pod of a 4-pod fat tree, where hosts 1 and 3 have only two disjoint paths ABD and ACD between them.

The detailed reduction is as follows. First, set up a K-pod fat tree, and the capacity of each link is set as infinity to avoid violating the link capacity constraint. Second, select two hosts, denoted as h_1 and h_2, in the same pod but under different edge switches. We will designate all the flows between the two hosts. Third, create n flows each corresponding to an integer in X, where each flow F_i has $a_i = h_1$, $b_i = h_2$, and $d_i = x_i$ where $x_i \in X$. Four, create additional $K/2-2$ flows, where each flow F_k has $a_k = h_1$, $b_k = h_2$, and $d_k = \sum_{x_i \in X} x_i/2$.

Suppose that the partition problem for set X has a solution X_s. Note that for the two hosts h_1 and h_2 in a K-pod fat tree, there are $K/2$ disjoint paths between them, each through a different aggregation switch. In the fourth step of the above reduction process, we create $(K/2-2)$ flows, each with the demand of $\sum_{x_i \in X} x_i/2$. Due to load balancing, each of them will be allocated to a different path, resulting in two remaining vacant paths. Finally, each of the n flows created in the third step, if $x_i \in X_s$, we designate the corresponding flow F_i to the first of the remaining two paths. If not, we designate the corresponding flow to the second path. In this way, we find a solution for the load-balanced multipath routing problem that minimizes the maximum load to be $\sum_{x_i \in X} x_i/2$ and achieves the optimal load balance.

Suppose that the load-balanced multipath routing problem has an optimal solution with a maximum load being $\sum_{x_i \in X} x_i/2$. Each of $(K/2-2)$ flows created in the fourth step must occupy a different path. The n flows created in the third step must share the remaining two paths, based on which we can create the two subsets X_s and $X \setminus X_s$ accordingly, and thus we find a solution for the partition problem. □

4 AN SDN-BASED ALGORITHM

In this section, we propose an SDN-based load-balanced multipath routing algorithm. Software-Defined Networking (Open Networking Foundation 2012) is an emerging networking paradigm that has recently gained tremendous popularity. SDN can be realized using OpenFlow: the network has a central OpenFlow controller that communicates with the OpenFlow switches in the network using the OpenFlow protocol. The controller can be used to collect the network state and direct all traffic flows in the network. Our main idea here is to use the central controller to collect the traffic load information for all the links in the network so that our algorithm can make globally optimized routing decisions. When a new flow enters the system, the switch that first encounters the flow will inform the controller. The controller will enumerate all possible paths that can carry the flow, compare the load of the bottleneck links of those paths, and then select the one with the minimum load. After the path has been determined, the controller will inform all switches along the path and update their flow tables so that subsequent packets of the flow will be forwarded along the same path.

4.1 Design Philosophy

To optimize bandwidth utilization in DCNs, one needs a global view of the available resources in the network (Benson et al. 2010a). To do that, we implement the load-balanced multipath routing scheme at

the central controller, which communicates with the switches in the network using the OpenFlow protocol. Each OpenFlow-enabled switch contains a flow table, which the switch uses to control the flows. A flow can be flexibly defined by any combination of the ten fields in the packet header identified by the switch, including the source and destination IP addresses, the port numbers, and others. A controller can control the flows in the system by querying, inserting, or modifying the corresponding entries in the flow table of an OpenFlow switch. In particular, the central controller is able to collect bandwidth and flow information of the entire network from the switches for it to make globally optimized routing decisions, and then direct the flows by updating the flow tables at the switches along the flow paths.

OpenFlow devices are already widely available on the market today. Many recent data centers use OpenFlow as part of their network switching fabric. As such, our proposed solution can be readily applied in those data centers to improve their network performance.

4.2 The Algorithm

In the following we describe the SDN-based load-balanced multipath algorithm in more detail. The algorithm has three steps:

In the first step, the algorithm identifies the highest layer of switches a flow needs to reason based on the location of its source and destination hosts. If the source and destination hosts belong to different pods, the path will need to include the core switches. If the two hosts are attached to the same edge switch, the edge switch will simply connect them. If the two hosts are in the same pod but not under the same edge switch, one of the aggregation switches will be needed to connect them.

In the second step, the central controller determines the bottleneck link of each potential path between the source and destination hosts. Note that in the fat tree, the switch at the highest layer can uniquely identify the path between the given source and destination. Since the central controller has the information of every link in the network, it can quickly identify the bottleneck link of a path by comparing the available bandwidth on all the links along the path. The link with the smallest available bandwidth will be the bottleneck link. Arbitrary tie-breaking rules can be applied here if a path has more than one links with smallest available bandwidth.

In the third step, the controller compares the available bandwidth of bottleneck links of all potential paths and selects the one with the maximum value. If the maximum available bandwidth is greater than the demand of the flow, the corresponding path is selected for the flow. The controller will then set up the flow tables of all the switches along the chosen path accordingly. However, if the maximum available bandwidth is less than the flow demand, it means that there does not exist a viable path for this flow. The system may choose to either reject this flow, or admit this flow at the risk of jeopardizing the quality-of-service of this flow and all other existing flows in the system that may intersect with this flow.

4.3 Time Complexity

In a k-pod fat tree, which consists of $k^3/4$ hosts and $5k^2/4$ switches, there are at most k distinct paths between any two hosts. For each path, the algorithm needs to check at most 8 links (which is the maximum path length) in order to find the bottleneck. Therefore, the complexity of the routing process is $O(k)$. In other words, the proposed multipath routing algorithm has a time complexity of $O(n^{1/3})$ for a network with $O(n)$ nodes.

5 IMPLEMENTATIONS

To validate our designs, we have implemented the proposed algorithm in two environments: as a module of an open-source OpenFlow controller, and in a home grown DCN simulator.

5.1 OpenFlow Controller Module

We implemented the algorithm as a module of the Beacon controller (version 1.04) (Erickson 2013). The module has two functions: monitoring the link state and calculating paths for new flows.

The Beacon controller module periodically schedules a monitoring event, upon which the controller will send an `OFStatisticsRequest` command to every switch in the network, once every T seconds. Each switch will in turn reply with an `OFStatisticsReply` message. The controller processes the reply message by extracting the switch ID, the port number, the total number of bytes transmitted, as well as the time the reply message was received. An estimation of the flow rate for each port can be calculated based on this information. The size of the messages is quite small: 8 bytes for the `OFStatisticsRequest` message and 104 bytes for the `OFStatisticsReply` message. The overhead for the port statistics request and reply was small. In a 4-pod or 8-pod fat tree setup in Mininet (Lantz, Heller, and McKeown 2010), we noticed that by making T smaller (from 6 to 2 seconds) the CPU usage was increased only by approximately 2-5%.

The flow path scheduling function requires up-to-date port statistics for load balancing. When an OpenFlow switch receives a packet, the switch will attempt to match the packet header with the entries in its flow table. If the switch cannot find a matching flow entry, it will send the packet to the controller wrapped in an `OFPacketIn` message. Upon receiving this message, the controller will decide on an optimized path for the flow, as described in Section 4. Once the path is determined, the controller will create and send an `OFFlowMod` message, which contains the packet header information for flow matching, along with the input and output ports for each switch on the flow path.

Mininet 2.1 was selected for emulation. We studied the algorithm using a 4-pod or a 8-pod fat tree. A python script was developed to automatize the testing process, including switch initialization and connection to the Beacon Controller, host initialization, and traffic generation. Traffic are UDP flows generated by `iperf`: each host has an iperf server; and based on the traffic load, the python script creates a UDP flow from an iperf server to an iperf client chosen randomly among the hosts at certain time intervals.

5.2 DCN Simulation

We also developed a fluid-based DCN simulator to help evaluate the performance of the proposed multipath algorithm in diverse settings. The DCN simulator was implemented using the MiniSSF parallel discrete-event simulation framework (Rong, Hao, and Liu 2014), although it only used the sequential simulation functions.

The DCN simulator creates the fat tree network model with simulated hosts, switches, and links between the switches and hosts. Currently we use an exponential distribution to model the inter-arrival time of flows, which are randomly placed between the hosts according to a uniform distribution. Here we assume that the flow arrival is a stationary Poisson process. One can capture the non-stationary behavior (such as the diurnal effect) by using piece-wise constant flow arrival rates at different time intervals. Such traffic pattern can also be extended easily to include more realistic scenarios, e.g., Benson et al. (2010), Benson et al. (2010b). Upon each new flow arrival, the simulator invokes a routing oracle to determine its path. The aforementioned load-balanced multipath routing algorithm is implemented as the routing oracle.

To achieve high performance for running large-scale network models and scenarios with high traffic intensity, we developed a fluid model to represent the network traffic. Fluid models, e.g., Nicol (2001), Liu et al. (2003), can be applied to reduce the computational complexity, where one uses fluids to represent a series of packets traversing the network. It is also possible to capture the interaction between that network flows represented as fluids and those represented by individual packets, e.g., Nicol and Yan (2004), Gu, Liu, and Towsley (2004), Liu (2006), Li, Van Vorst, and Liu (2013). These models are shown to be able to achieve substantial performance gains—in some cases, more than three orders of magnitude—over the traditional packet-oriented approach, while still maintaining good accuracy.

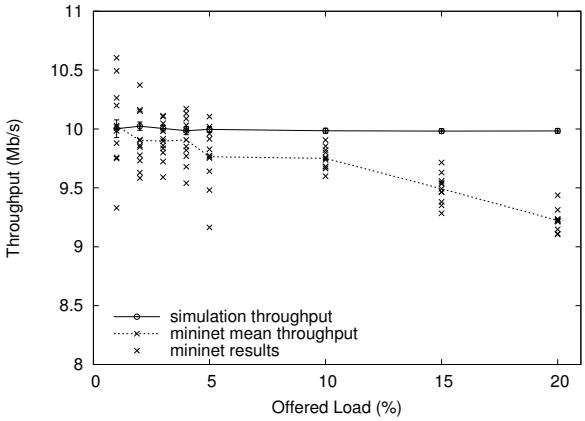

Figure 3: Mininet throughput.

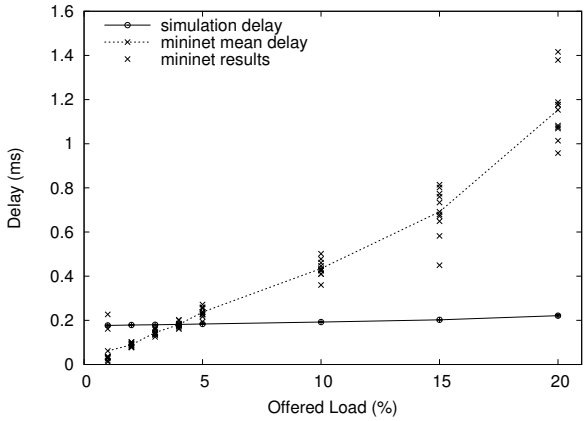

Figure 4: Mininet delay.

In our case, we developed a simple fluid model that can capture the steady-state queuing behavior. Upon the arrival of a new flow, we update the flow arrival rate of the network queues at each hop visited by the flow. We calculate the throughput and end-to-end delay of the flow traversing the queue using a steady-state approximation of the M/D/1/N queue. For simplicity, we do not capture the transient time-varying aspects of the queuing behavior. Changes in the throughput and delay at one queue are propagated immediately to the subsequent queues. The throughput and delay of other flows traversing the same queue may also need to be updated. To avoid the "ripple effect" (Nicol 2001), where a change in the arrival rate of one flow at one queue may cause changes to the rate of all other flows at the same queue as well as other queues subsequent traversed by these flows, we apply a dampening factor to prevent small changes from being propagated to subsequent queues. Our fluid model is simple and can accurately capture the fluctuations in flow rate when flows entering and leaving the system.

6 EXPERIMENTS

We conducted two sets of experiments: one with the Beacon SDN controller and the Mininet emulation platform, and the other with the MiniSSF simulator. The former validates the correctness of the proposed design in a real implemented system. The latter uses simulation to evaluate the algorithm performance for networks with various configurations and of larger scales.

6.1 Emulation

While our OpenFlow controller module implementation can be applied directly in DCN, we need to first validate its correctness. For this, we chose Mininet (version 2.1.0), which is a container-based emulation environment for testing SDN applications (Lantz, Heller, and McKeown 2010). Mininet can instantiate a set of virtual hosts and virtual switches, which can be connected as an arbitrary network, all on the same machine. Mininet runs on a single Linux kernel for all virtual host instances using a lightweight container-based virtualization solution. Using lightweight container-based virtualization provides a scalable mechanism: one can use Mininet to create hundreds and even thousands of virtual hosts. One limitation is that all instantiated virtual machines must run the same OS kernel (since each container is simply running in a separate kernel name space). The main drawback of the container-based VM, however, as we acutely observed in our experiments, is that containers do not impose resource isolation. There is a stringent limitation in the amount of traffic that can be emulated by Mininet on a single machine in real time. Moreover, the applications running on the virtual hosts are also competing for the CPU resources. For our experiment scenarios potentially with heavy traffic, Mininet cannot produce reasonable results.

To show its limitation, we used the Mininet to emulate a fat tree with $k = 4$ pods, in which case the network contains only 16 hosts and 20 switches. We set the bandwidth of all links to be 100 Mbps. We

used `iperf` to generate UDP flows; the source and destination of the flows were chosen uniformly at random. Each flow generates packets at a constant rate which is uniformly distributed between 8 to 12 Mbps; the mean is 10 Mbps. The duration of each flow is 30 seconds. We varied the offered traffic load on the network by changing the mean inter-arrival time between the flows, which is set to be exponentially distributed. For the experiment, we varied the offered load from 1% to 20% of the total bandwidth for connecting the hosts. For $k = 4$, that's 100 Mbps for each of the 16 hosts, or 1.6 Gbps in total. Each experiment ran for 300 seconds; we conducted 10 runs for each offered load.

Figure 3 shows the average throughput of all flows created in the experiment; Figure 4 shows the average delay. In the figures, we plot the results from each of the 10 runs, together with the averages. For comparison, we also show the results from simulation with a similar setting (described in more detail in the next section). From this experiment, we can make two observations. First, there exists significant variation in the Mininet results between runs, especially considering the results are averages among all flows in each experiment. Second, the throughput drops significantly when offered load increases; such decrease starts early even when the offered load is small. The same is true for the end-to-end delay, which increases with the increasing offered load. In this aspect, the emulation results diverge significantly from the simulation results, which show a rather stable behavior when the offered load is blow 20%.

There could be many possible contributing factors to this disparity. We measured the CPU utilization during the emulation experiment. The machine we ran the experiment has a four-core 3.3 GHz Intel Xeon processor with hyper-threading enable, and 12 GB of memory. At 1% traffic load, the CPU utilization was averaged at 77%. At 2% load, it was increased to 93%. For an offered load of 3% and more, the CPU utilization was consistently above 97%. As this rate, Mininet was not able to deal with the emulation events. We believe the fast increase in delay and rapid decrease in throughput can be attributed to the resource contention. Note that, with such light traffic load for a small-scale DCN, we cannot conduct meaningful experiments to compare the performance of our load-balanced multipath algorithm. In the next section, we resort to simulation for the performance evaluations.

6.2 Simulation

In this section, we present the simulation results to demonstrate the effectiveness of our algorithm. We consider fat trees of $k = 4$, 8, and 16. A fat tree contains $k^3/4$ hosts and $5k^2/4$ switches. For $k = 16$, we have 1024 hosts and 320 switches. We used the same parameters as in the Mininet experiment described in the previous section, except that we increased the range of the offered load up to 90% of the total bandwidth for connecting the hosts. For each offered load we conducted 15 trials each running 1000 seconds.

We compare the results from using our load-balanced multipath algorithm (labeled as LBMP) with those from a simple multipath algorithm, which, likes ECMP, randomly selects one of the available paths between the source and destination hosts. Figure 5 shows the average throughput achieved under different offered load, and Figure 6 shows the average end-to-end delay. We plot the 95% confidence interval for the 15 runs at each data point (which is negligible in most cases). As expected, our load-balanced multipath algorithm consistently achieved better throughput and delay compared to the random multipath algorithm. With the same offered load, the end-to-end delay of our algorithm is measured slightly higher for larger fat trees. This is due to the longer average path length. The throughput remains similar between different fat tree sizes, because the algorithm is able to balance the traffic load in the network. This is not true for the random multipath algorithm: at higher traffic intensity, congestions start to form which causes the throughput to drop more precipitously, especially for larger network size.

The size of the network queues should play an important factor in determining the packet delays. In the previous experiment, we fixed the buffer size to be 1000 packets (that's 500 KB assuming the average packet size is 500 bytes). In the next experiment, we varied the buffer size from 1000 to 16000 to examine its effect on the throughput and delay. Figures 7 and 8 shows the results for a $k = 4$ fat tree. We observe that, while the throughput does not change under different buffer sizes, a larger buffer size would increase the end-to-end delay. Both algorithms are able to spread the traffic load and the network has enough

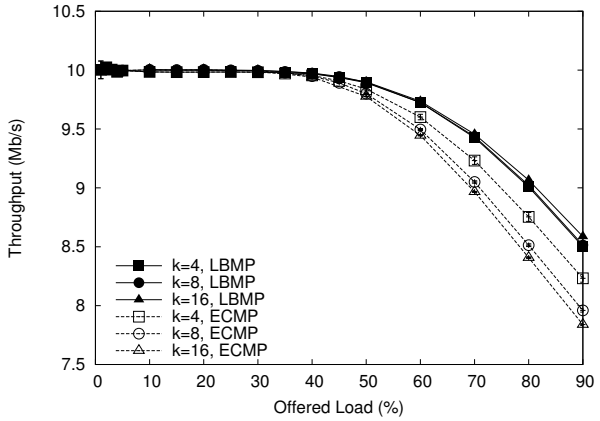

Figure 5: Simulation throughput.

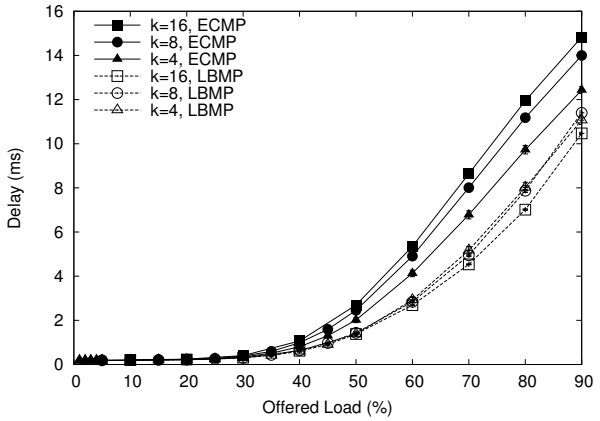

Figure 6: Simulation delay.

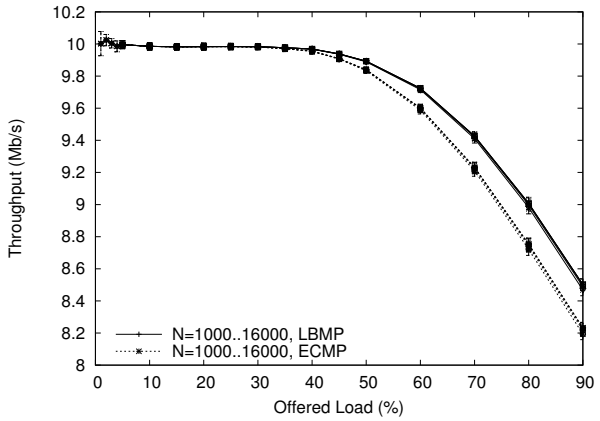

Figure 7: Throughput with varying buffer size.

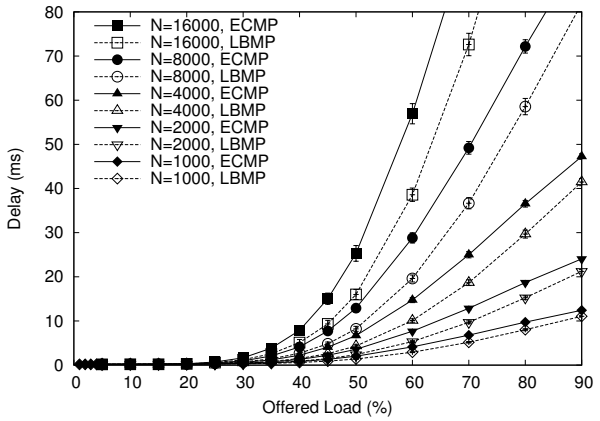

Figure 8: Delay with varying buffer size.

capacity to handle the flows originated from the hosts. The queue length remains to be relatively small and therefore the blocking probability is kept low not to significantly affect the throughput. However, with a larger buffer size, the average queue length does increase, which directly affects the queuing delay for the flows traversing the queue, as it is obviously shown in Figure 8. In all cases, our load-balanced multipath algorithm consistently achieves better performance than the random multipath algorithm, both in terms of the increased throughput and in the reduced delay.

7 CONCLUSIONS

The fat tree with multipath capability has become a popular topology for data center networks for its increased bandwidth and fault tolerance. In this paper, we propose a load-balanced multipath routing algorithm, which can better distribute the traffic load and utilize the available bandwidth in the fat tree network. We formulate the problem as a linear program and show its NP-completeness by reduction from the integer partition problem. As a practical solution, we propose a load-balanced multipath routing algorithm for SDN-based data center networks. The algorithm uses the central controller to collect information of the network state, and makes optimized load balancing decisions when admitting new flows to the network. We implemented the proposed algorithm both in Mininet, to establish the basic validation of the implementation, and in a simulator, to conduct extensive performance evaluations. Experiment results show that our algorithm consistently outperforms the traditional multipath algorithm using random assignments.

For future work, we plan to conduct more validation experiments. Especially we would like to compare the simulation results against those from a real physical SDN testbed, and against other multipath algorithms.

We will also conduct experiments with more realistic traffic load, including the use of TCP, for a better representation of the data center applications and their network behavior.

ACKNOWLEDGMENTS

This research is supported in part by NSF grant CNS-1117016, and a subcontract from the GENI Project Office at Raytheon BBN Technologies (NSF grant CNS-1346688). We thank the anonymous reviewers for their constructive comments and suggestions.

REFERENCES

Al-Fares, M., A. Loukissas, and A. Vahdat. 2008. "A scalable, commodity data center network architecture". *ACM SIGCOMM Computer Communication Review* 38 (4): 63–74.

Al-Fares, M., S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. 2010. "Hedera: dynamic flow scheduling for data center networks". In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, 19:1–19:15.

Benson, T., A. Akella, and D. A. Maltz. 2010. "Network traffic characteristics of data centers in the wild". In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 267–280.

Benson, T., A. Anand, A. Akella, and M. Zhang. 2010a. "The case for fine-grained traffic engineering in data centers". In *Proceedings of the 2010 Internet Network Management Workshop/Workshop on Research Enterprise Networking*, 2:1–2:6.

Benson, T., A. Anand, A. Akella, and M. Zhang. 2010b. "Understanding data center traffic characteristics". *ACM SIGCOMM Computer Communication Review* 40 (1): 92–99.

Cisco Systems 2007. "IP multicast load splitting - Equal Cost Multipath (ECMP) using S, G and next hop". http://www.cisco.com/en/US/docs/ios/12_2sr/12_2srb/feature/guide/srbmpath.html.

Erickson, D. 2013. "The Beacon OpenFlow controller". In *Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 13–18.

Garey, M. R., and D. S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Greenberg, A., N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. 2009. "VL2: a scalable and flexible data center network". In *ACM SIGCOMM Computer Communication Review*, Volume 39, 51–62.

Gu, Y., Y. Liu, and D. Towsley. 2004. "On integrating fluid models with packet simulation". In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Volume 4, 2856–2866.

Guo, C., G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. 2009. "BCube: a high performance, server-centric network architecture for modular data centers". *ACM SIGCOMM Computer Communication Review* 39 (4): 63–74.

Guo, C., H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. 2008. "DCell: a scalable and fault-tolerant network structure for data centers". *ACM SIGCOMM Computer Communication Review* 38 (4): 75–86.

Heller, B., S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. Mckeown. 2010. "ElasticTree: saving energy in data center networks". In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, 17:1–17:16.

Kurose, J. F., and K. W. Ross. 2007. *Computer Networking: A Top-Down Approach*. 4th ed. Addison Wesley.

Lantz, B., B. Heller, and N. McKeown. 2010. "A network in a laptop: rapid prototyping for software-defined networks". In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks*, 19:1–19:6.

Li, T., N. Van Vorst, and J. Liu. 2013. "A rate-based TCP traffic model to accelerate network simulation". *SIMULATION* 89 (4): 466–480.

Liu, J. 2006. "Packet-level integration of fluid TCP models in real-time network simulation". In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 2162–2169. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Liu, Y., F. Lo Presti, V. Misra, D. Towsley, and Y. Gu. 2003. "Fluid models and solutions for large-scale IP networks". *ACM SIGMETRICS Performance Evaluation Review* 31 (1): 91–101.

Mudigonda, J., and P. Yalagandula. 2010. "SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies". In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, 18:1–18:16.

Mysore, R. N., A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. 2009. "Portland: a scalable fault-tolerant layer2 data center network fabric". In *ACM SIGCOMM Computer Communication Review*, Volume 39, 39–50.

Nicol, D. M. 2001. "Discrete event fluid modeling of TCP". In *Proceedings of the 2001 Winter Simulation Conference*, edited by B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 1291–1299. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Nicol, D. M., and G. Yan. 2004. "Discrete event fluid modeling of background TCP traffic". *ACM Transactions on Modeling and Computer Simulation* 14 (3): 211–250.

Open Networking Foundation 2012. "Software-defined networking: The new norm for networks". https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

Rong, R., J. Hao, and J. Liu. 2014. "Performance study of a minimalistic simulator on XSEDE massively parallel systems". In *Proceedings to the 3rd Annual Conference of the Extreme Science and Engineering Discovery Environment*, Article 15.

Wang, G., D. G. Andersen, M. Kaminsky, M. Kozuch, T. S. E. Ng, K. Papagiannaki, and M. Ryan. 2010. "c-Through: part-time optics in data centers". *ACM SIGCOMM Computer Communication Review* 40 (4): 327–338.

AUTHOR BIOGRAPHIES

ERIC JO is an undergraduate student in the School of Computing and Information Sciences, Florida International University. His email address is ericcsonhung@gmail.com.

LINDA BUTLER is an undergraduate student in the Department of Computer Science and Engineering, University of South Florida. Her email address is lindabutler17@hotmail.com.

DENG PAN is an Associate Professor at the School of Computing and Information Sciences, Florida International University. His research interests include high performance switch architecture and high speed networking. He received the B.S. and M.S. degrees in Computer Science from Xian Jiaotong University, China, in 1999 and 2002, respectively, and the Ph.D. degree in Computer Science from the State University of New York at Stony Brook in 2007. His email address is pand@cis.fiu.edu; his home webpage is http://www.cis.fiu.edu/~pand/.

JASON LIU is an Associate Professor at the School of Computing and Information Sciences, Florida International University. His research focuses on parallel simulation and high-performance modeling of computer systems and communication networks. He received a B.A. degree from Beijing University of Technology in China in 1993, an M.S. degree from College of William and Mary in 2000, and a Ph.D. degree in from Dartmouth College in 2003. He served as General Chair for MASCOTS 2010, SIMUTools 2011 and PADS 2012, and also as Program Chair for PADS 2008 and SIMUTools 2010. He is an Associate Editor for the ACM Transactions on Modeling and Computer Simulation (TOMACS), and for SIMULATION, Simulation Transactions of the Society for Modeling and Simulation International. He is also a Steering Committee Member for SIGSIM-PADS. His email address is liux@cis.fiu.edu; his home webpage is http://www.cis.fiu.edu/~liux/.