# Depth-First Worst-Fit Search based Multipath Routing for Data Center Networks

Tosmate Cheocherngngarn, Hao Jin, Jean Andrian, Deng Pan, and Jason Liu
Florida International University
Miami, FL

***Abstract*—Modern data center networks (DCNs) often use multi-rooted topologies, which offer multipath capability, for increased bandwidth and fault tolerance. However, traditional routing algorithms for the Internet have no or limited support for multipath routing, and cannot fully utilize available bandwidth in such DCNs. In this paper, we study the multipath routing problem for DCNs. We first formulate the problem as an integer linear program, but it is not suitable for fast on-the-fly route calculation. For a practical solution, we propose the Depth-First Worst-Fit Search based multipath routing algorithm. The main idea is to use depth-first search to find a sequence of worst-fit links to connect the source and destination of a flow. Since DCN topologies are usually hierarchical, our algorithm uses depth-first search to quickly traverse between hierarchical layers to find a path. When there are multiple links to a neighboring layer, the worst-fit link selection criterion enables the algorithm to make the selection decision with constant time complexity by leveraging the max-heap data structure, and use a small number of selections to find all the links of a path. Further, worst-fit also achieves load balancing, and thus generates low queueing delay, which is a major component of the end-to-end delay. We have evaluated the proposed algorithm by extensive simulations, and compared its average number of link selections and average end-to-end delay with competing solutions. The simulation results fully demonstrate the superiority of our algorithm and validate the effectiveness of our designs.**

## I. Introduction

Data centers contain large numbers of servers to achieve economies of scale [17], and the number is increasing exponentially fast [11]. For example, it is estimated that Microsoft's Chicago data center has about 300,000 servers [1]. The huge number of servers have created a challenge for the data center network (DCN) to offer proportionally large bandwidth to interconnect the servers [27]. As a result, modern DCNs usually adopt multi-rooted topologies, such as the fat tree [2], VL2 [9], DCell [11], and BCube [10], which offer multipath capability for large bisection bandwidth.

However, traditional link state and distance vector based [15] routing algorithms for the Internet cannot readily utilize the multipath capability of multi-rooted topologies. In the first place, traditional routing algorithms calculate routes based on only packet destinations, and thus all packets to the same destination share the same route. Although equal cost multi-path (ECMP) [7] support multipath routing, it performs static load splitting based on packet headers without accounting for bandwidth, allows only paths of the same minimum cost, and supports an insufficiently small number of paths [12]. Further, traditional routing algorithms usually give preference to the shortest path to reduce the propagation delay. Due to small geographical distances, DCNs are less concerned about the propagation delay, but give priority to bandwidth utilization.

To fully explore bandwidth, DCNs with multi-rooted topologies need practical and efficient multipath routing algorithms, which should satisfy the following design objectives. First, the algorithm should maximize bandwidth utilization, or in other words achieve a high routing success ratio, so that the same network hardware can accommodate as much traffic as possible. Further, the algorithm must have low time complexity to make fast routing decisions, because a DCN handles a large volume of traffic. Finally, after an algorithm satisfies the above requirements, it is preferred that the packets can have low end-to-end delay, in particular queueing delay.

To optimize bandwidth utilization in DCNs, it is necessary to have a global view of the available resources and requests in the network [4], and allocate bandwidth at fine granularity [26]. For these purposes, our design sets up a central controller, which communicates with switches in the DCN by the OpenFlow protocol [24]. Each OpenFlow enabled switch has a flow table to control flows, where a flow can be flexibly defined by any combination of the ten packet headers at arbitrary granularity [16]. The controller can control the flows by querying, inserting, or modifying entries in the flow tables. In this way, the central controller can collect bandwidth and flow information of the entire network from the switches, make optimal routing decisions, and send the results back to the switches to enforce the planned routing. Multiple choices of OpenFlow devices are already available on the market [19]–[23], and OpenFlow has been adopted in many recent data center designs [3], [4], [12], [18]. The availability of OpenFlow switches makes it practical to quickly experiment with and deploy the proposed solution.

In the paper, we first formulate the multipath routing problem as an integer linear program, but it is not suitable for fast on-the-fly route calculation. Next, built on top of the OpenFlow control framework, we propose the Depth-First Worst-Fit Search based multipath routing algorithm for DCNs. The main idea is to use depth-first search to find a sequence of worst-fit links to connect the source and destination of a flow. Because DCN topologies are usually hierarchical, the depth-first search can quickly traverse between the hierarchical layers of switches to find a path. When there are multiple links to the neighboring layer, the worst-fit criterion always selects the one with the largest amount of remaining bandwidth. By using

the max-heap data structure, worst-fit can make the selection decision with constant time complexity. Further, worst-fit achieves load balancing, and therefore avoids unnecessary backtracking in the depth-first search and reduces packet queueing delay. We have evaluated the proposed algorithm by extensive simulations, and the results demonstrate that the algorithm fulfills the design objectives.

The rest of the paper is organized as follows. Section II briefly reviews the related work. Section III formulates the problem, proposes our solution, and discusses the implementation issues. Section IV presents the simulation results. Finally, Section V concludes the paper.

## II. 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. Typical layer-two forwarding uses a spanning tree, where there is only one path between source and destination nodes. A recent work, SPAIN [17], provides multipath forwarding by computing a set of paths that exploits the redundancy in a given network, and merges these paths into a set of trees, each mapped as a separate VLAN. At layer three, equal cost multipath (ECMP) [7] provides multipath forwarding by performing static load splitting among flows. ECMP-enabled switches are configured with several possible forwarding paths for a given subnet. When a packet arrives at a switch with multiple candidate paths, the switch forwards it onto the one that corresponds to a hash of selected fields of the packet header, thus splitting load to each subnet across multiple paths. However, ECMP does not account for flow bandwidth in making allocation decisions, which may lead to oversubscription even for simple communication patterns. Further, current ECMP implementations limit the multiplicity of paths to 8-16, which is fewer than what would be required to deliver high bisection bandwidth for larger data centers [12].

There exist several multipath solutions for DCNs. [3] propose two multipath algorithms: Global First-Fit and Simulated Annealing. The former simply selects among all the possible paths the first one that can accommodate a flow, but needs to maintain all paths between a pair of nodes. The latter performs a probabilistic search of the optimal path, but converges slowly. [12] proposes the ElasticTree DCN power manager with two multipath algorithms: Greedy Bin-Packing and Topology-Aware Heuristic. The former evaluates possible paths and chooses the leftmost one with sufficient capacity. The latter is a fast heuristic based on the topological feature of fat trees, but with the impractical assumption to split a flow among multiple paths. [4] presents the MicroTE framework that supports multipath routing, coordinated scheduling of traffic, and short term traffic predictability. Different from the existing solutions, our multipath routing algorithm uses depth-first search to quickly find a path between hierarchical layers, and uses worst-fit to select links with low time-complexity and avoid creating hot spots in the network.

## III. Depth-First Worst-Fit Search based Multipath Routing

In this section, we formulate the multipath routing problem, propose the Depth-First Worst-Fit Search based algorithm, and discuss its implementation.

### A. Problem Formulation

We formulate the multipath routing problem as a linear program. Model a DCN as a directed graph $G = (H \cup S, L)$, in which a node $h \in H$ is a host, a node $s \in S$ is a switch, and an edge $(n_i, n_j) \in L$ is a link connecting a switch with another switch or a host. Each edge $(n_i, n_j)$ has a nonnegative capacity $c(n_i, n_j) \geq 0$ indicating the available bandwidth of the corresponding link. There are $n$ flows $F_1, \ldots, F_n$ in the DCN. $F_k$ is defined as a triple $F_k = (a_k, b_k, d_k)$, where $a_k \in H$ is the source host, $b_k \in H$ is the destination host, and $d_k$ is the demanded bandwidth. Use $f_k(n_i, n_j)$ to indicate whether flow $F_k$ is routed via link $(n_i, n_j)$.

By translating the low latency design objective to load balancing, the objective function is thus to minimize the maximum load among all the links, i.e.

$$\textbf{minimize } maxload$$

subject to the following constraints:

$$\forall (n_i, n_j) \in L, \sum_k \frac{f_k(n_i, n_j) d_k}{c(n_i, n_j)} \leq maxload \leq 1 \quad (1)$$

$$\forall k, \forall n_i \in H \cup S \setminus \{a_k, b_k\}, \sum_{n_j \in H \cup S} f_k(n_i, n_j) = \sum_{n_j \in H \cup S} f_k(n_j, n_i) \quad (2)$$

$$\forall k, \sum_{n_i \in H \cup S} f_k(a_k, n_i) = \sum_{n_i \in H \cup S} f_k(n_i, b_k) = 1 \quad (3)$$

Equation (1) defines $maxload$, and states the link capacity constraint, i.e. the total demanded bandwidth on a link not exceeding its available bandwidth. Equation (2) states the flow conservation constraint, i.e. the amount of any flow not changing at intermediate nodes. Equation (3) states the demand satisfaction constraint, i.e. for any flow, the outgoing traffic at the source or the incoming traffic at the destination equal to the demand of the flow.

Since integer linear programing is NP-complete, the above optimal flow assignment is not suitable for fast on-the-fly route calculation.

### B. Algorithm Design Philosophy

To make multipath routing practical for DCNs, we propose the Depth-First Worst-Fit Search based multipath routing algorithm. The basic idea is to use depth-first search to find a sequence of worst-fit links to connect the source and destination hosts of a flow.

Depth-first search utilizes the hierarchical feature of DCN topologies to quickly find a path connecting the hierarchical layers. DCNs are typically organized in a hierarchical structure with multiple layers of switches and one layer of hosts [9]–[11]. For example, a fat tree [18] DCN has one layer of

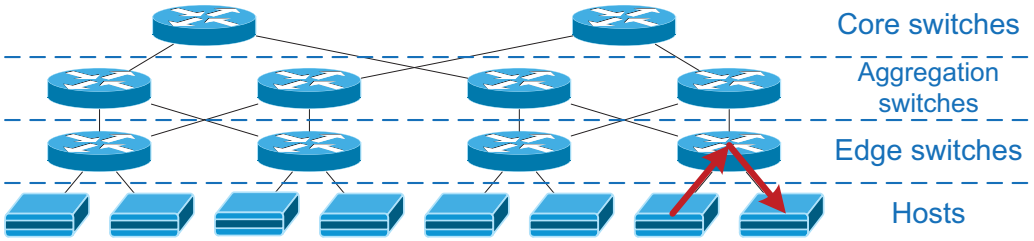



Fig. 1. Hierarchical fat tree topology.

hosts, edge switches, aggregation switches, and core switches, respectively, as shown in Figure 1. Since a path usually consists of links connecting hosts and switches at different layers, depth-first search can quickly traverse these layers. For example, a path connecting two servers of the same edge switch in a fat tree will traverse from the host layer to the edge switch layer and then back to the host layer, as shown in Figure 1. If the search has exhausted all the links in a layer and cannot proceed further, it is necessary to backtrack to the previous layer [6] and try the next candidate.

When there are multiple links to the neighboring layer, the worst-fit criterion selects the one with the largest amount of remaining bandwidth. On the other hand, the first-fit criterion in [3], [12] selects the first (or leftmost) link with sufficient remaining bandwidth, and best-fit selects the link with the least but sufficient remaining bandwidth.

Compared with first-fit or best-fit, worst-fit has the following advantages. First of all, worst-fit has time complexity of $O(1)$ by trying only the link with the largest amount of available bandwidth. Since the controller needs to search a path on the fly for each flow, constant time complexity helps accelerate the routing process. In contrast, first-fit has time complexity $O(\log N)$ to select from $N$ candidates, where $N$ grows with the DCN size, using the special winner tree data structure [13]. Similarly, best-fit has time complexity of $O(\log N)$ by conducting binary search on a pre-sorted list. Further, worst-fit achieves load balancing by evenly distribute traffic among all links, and therefore it needs fewer link selections on the average to find a path. This characteristic also helps worst-fit find a path faster than first-fit and best-fit by avoiding excessive backtracking. As a comparison, first-fit and best-fit tend to consolidate traffic to certain links and eventually block them. If all the neighboring links of a switch are blocked, the path searching has to backtrack to the previous layer, and thus needs more link selection decisions. Finally, because worst-fit achieves load balancing, it is less likely to create hot spots in the network, avoiding long packet queuing delay. On the other hand, first-fit and best-fit keep increasing the load of a link until saturate it. In this case, heavily loaded links suffer from extra latency, while some other links are still idle.

### C. Algorithm Description

In detail, the depth-first worst-fit multipath routing algorithm works as follows. The first stage is to determine the necessary layer to connect the source and destination hosts. Note that hosts in DCNs usually have IP addresses corresponding to their topological locations [5]. For example, in a fat tree based DCN, hosts in the same pod usually share the same subnet address [18]. Thus, it is easy to determine by which layer the source and destination hosts can be connected. Since the network topology and IP address assignment are known in advance, it is appropriate to do the calculation for all IP addresses in advance and store the results, so that they are handily available during path searching. Determining the connecting layer avoids wasting bandwidth of switches at higher layers, which will be available for future flows.

The second stage of the algorithm uses depth first search to select switches at each layer. Starting with the source edge switch, the search first goes upstream, and heads back downstream after reaching the necessary connecting layer. For certain topologies, such as the fat tree, the downstream path is determined after reaching the connecting layer. After arriving at each switch, the algorithm first checks if it is the destination edge switch. If yes, the search ends with a path. If no, the search continues with the worst-fit link to the next layer. In case that there is no viable link to the next layer, the search needs to backtrack to the previous layer and continue with the worst-fit link of the remaining untried links. For easy understanding, Table I gives the pseudo code description of the algorithm.

### D. Implementation

As explained in Section I, the algorithm will be running in the OpenFlow controller that has a global view of the entire network. Besides the network topology, the controller maintains two max-heaps [6] for each switch, one for upstream links and the other for downstream links. When the path search arrives at a switch, depending on the search direction, the algorithm compares the flow demand with the available bandwidth of the corresponding heap root. If the root has sufficient bandwidth, the search proceeds to the next layer; otherwise, the search backtracks to the previous layer. Since the algorithm is based on depth-first search, the time complexity is $O(|H \cup S| + |L|)$ [6]. After the algorithm successfully finds a path, it will deduct the flow demand from the available bandwidth of the links in the path and update the heaps. Fortunately, paths in DCNs usually have a small number of hops, and the heap update has time complexity of $O(\log N)$ for a size $N$ heap [6], so that the process can finish quickly.

TABLE I
PSEUDO CODE ALGORITHM DESCRIPTION

```
DFS(G, a, b, d) { // G: network, a: source, b: destination, d: demand
1   H = necessary-layer-to-connect(G, a, b);
2   path = {};
3   u = a; // temp variable indicating current location
4   next = 1; // search direction flag, 1: upstream, -1: downstream
5   return SEARCH(u, path, next);
}

SEARCH(u, path, next) {
1   path = path + u;
2   if (u = b) return true;
3   if ( layer-of(u) = H) next = −1;
    // reverse search direction after reaching connecting layer
4   if (next = −1 && layer-of(u) = 1) return false;
    // failure at bottom layer
5   neighbors = {v| layer-of(v) = layer-of(u) + next,
        and available bandwidth of link (u, v) ≥ d};
6   found = false;
7   while (neighbors ≠ ∅ && found = false) {
8       v = worst-fit(neighbors); neighbors = neighbors \ {v};
9       found = SEARCH(v, path, next);
10  };
11  return found;
}
```

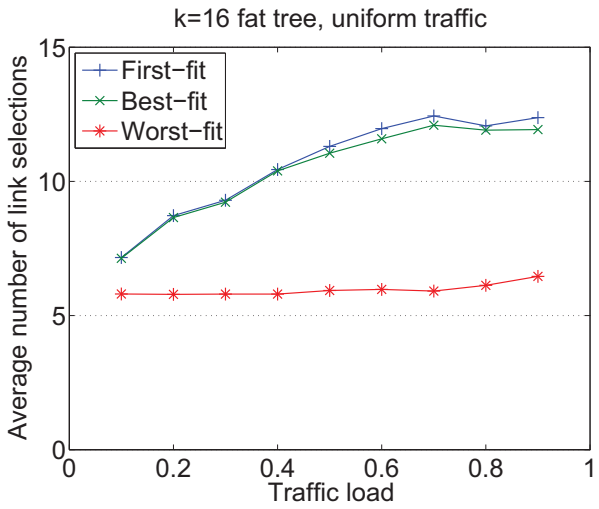
Also note that for the topologies in which a host has only one attached link, such as the fat tree, our algorithm is transparent to the hosts. Because the host has only one outgoing link, it can only send its packets to the connected switch. If the switch cannot find a matching entry in its flow table for a packet, the switch knows that the packet belongs to a new flow. The switch will then report the new flow to the controller, which will calculate a route for the flow. In this way, data center customers can choose to run their software on commodity operating systems, for enhanced security and compatibility.

We can see that the proposed algorithm fulfills the design objectives. First, it achieves high bandwidth utilization by exhaustive search with backtracking, and guarantees to find a path if there exists one. Second, the worst-fit link selection has constant time complexity; the depth-first search has linear time complexity; the average running time is much shorter with the load balancing achieved by worst-fit, as will be seen in Section IV. Third, load balancing by worst-fit reduces the queueing delay by avoiding hot spots.

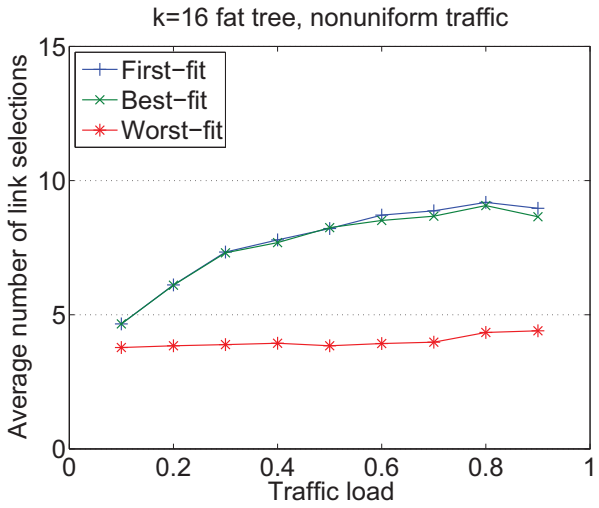
## IV. Simulation Results

In this section, we present simulation results to demonstrate the effectiveness of our designs. We compare the worst-fit criterion with first-fit and best-fit, in which first-fit was used in existing multipath routing algorithms [3], [12]. In the following, we describe our simulation settings, and look at the average number of link selections and average end-to-end delay of the three solutions.

The average number of link selections is important, because a smaller value means faster routing decisions. The routing process of each of the simulated algorithms consists of a series of link selections, and each link selection increases the routing decision time. Thus, if a routing process has a large number of failed link selections due to backtracking, it leads to long routing delay. On the other hand, average end-to-end delay is an interesting evaluation criterion, because short delay indicates good load balancing. Finally, we also observed in the simulations that all the three algorithms achieve routing success ratios of greater than 99% for admissible traffic, which demonstrate high bandwidth utilization. The high routing success ratios are not surprising due to exhaustive path searching with backtracking.




(a) Uniform traffic.




(b) Nonuniform traffic.

Fig. 2. Average number of link selections of different multipath routing algorithms.

### A. Average Number of Link Selections

Since the fat tree is the most popular DCN topology [2], [12], [18], we consider a $k = 16$ fat tree with 1024 hosts, 128 edge switches, 128 aggregation switches, and 64 core switches. Each link has bandwidth of 1 Gbps. Each flow has a demand uniformly distributed between 10 Kbps to 1 Mbps. We run the

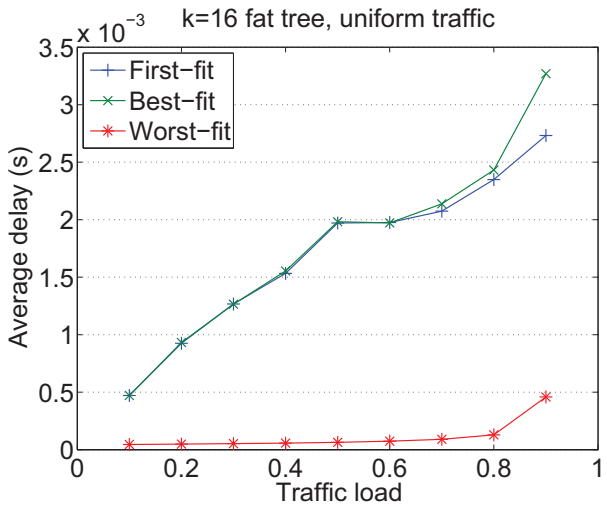
simulations under both uniform and nonuniform traffic. With uniform traffic, the flow destinations of a host are uniformly distributed among all the remaining hosts. With nonuniform traffic, 70% of the traffic generated by a host is destined for hosts in the same pod [18], which consists of all the hosts and switches connected by the same aggregation switch, and 30% traffic is destined for hosts out of the pod.

Figure 2(a) shows the average numbers of link selections of the worst-fit, first-fit, and best-fit criteria under uniform traffic. We can see that the average link selection number of worst-fit is consistently less than the other two. Specifically, it is not sensitive to the traffic load, and stabilizes at around 6. This can be explained by the fact that a path under uniform traffic needs 6 hops on average, i.e. starting from the source host, and traversing the edge switch, aggregation switch, core switch, aggregation switch, edge switch, and destination host. Although a server has only one link to its edge switch in the fat tree, and thus the hop from or to the host needs no selection decision, we still count it as one link selection, because the algorithm needs to check whether the link has sufficient bandwidth. On the other hand, for first-fit and best-fit, their average numbers of link selections grow steadily with the traffic load, up to 12 or twice as that of worst-fit. Compared with worst-fit, we can see that up to half of their link selections failed due to backtracking. The reason is that first-fit and best-fit tend to consolidate the flows to a small number of links, resulting in congestion and therefore more backtracking for the depth-first search. Note that the average number of link selections is not an indicator of route lengths but of processing time. It is also interesting to note that although first-fit and best-fit choose different links, they need a similar average number of link selections.

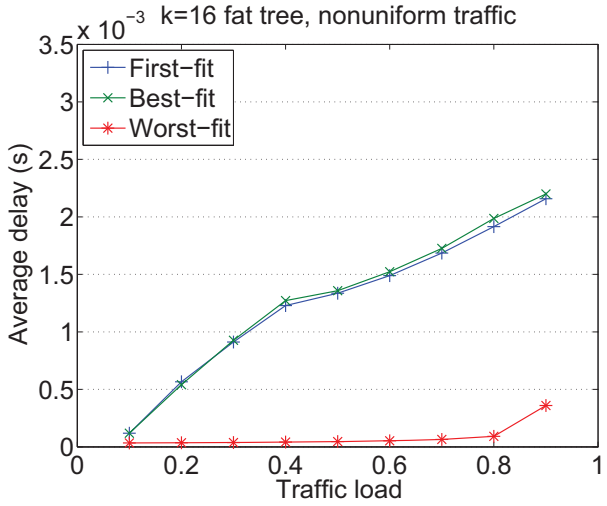
Figure 2(b) shows the simulation results under nonuniform traffic. Similarly, the data of worst-fit stabilize are not sensitive to the load change, and keep at around 4. Because under nonuniform traffic, 70% of the flows are in the same pod, the path for such a flow needs only 4 hops, i.e. staring from the source hosts, and traversing the edge switch, aggregation switch, edge switch, and destination host. Due to the big portion of in-pod traffic, first-fit and best-fit need smaller numbers of link selections as well, and the numbers increase with the traffic load, up to 9. They two still use a similar number of link selections.

### *B. Average Delay*

We now present the average end-to-end delay of the three criteria. We assume that the processing delay at each switch is zero, and the per hop delay consists of queueing delay, transmission delay, and propagation delay. Each output port of the switch has a buffer of 1 Mbytes. Every link is bidirectional with 1 Gbps for each direction. Each link has a propagation delay of 1 $\mu$s. The packet length is uniformly distributed between 40 and 1,500 bytes [8], and packet arrival of flows follows the Markov modulated Poisson process [25]. For first-fit and best-fit to avoid extreme queueing delay of saturated links, we always leave 5% of the bandwidth of a link unallocated when calculate the routes, i.e. available bandwidth = 95% of




(a) Uniform traffic.




(b) Nonuniform traffic.

Fig. 3. Average end-to-end delay of different multipath routing algorithms.

physical bandwidth. (Since worst-fit achieves load balancing, it is not likely to saturate a link when the overall traffic load is small.) Each simulation run lasts for 10 seconds.

Figure 3(a) shows the average end-to-end packet delay under uniform traffic. We can see that worst fit has much shorter average delay than the other two. Since worst-fit achieves load balancing, its average delay is not sensitive to the load, and increases only when the load becomes close to one. On the other hand, the average delay of first-fit and worst-fit grows proportionally with the load, up to six times of that of worst-fit. Because best-fit consolidates flows to a higher degree, its link is more congested and thus it has longer average delay than first-fit.

Figure 3(b) shows the data under nonuniform traffic. Consistently, worst-fit has much shorter average delay than the other two. The average delay of all three criteria decreases, because under the nonuniform traffic, most flows are in the same pod, and therefore need fewer hops.

## V. Conclusions

Data center networks (DCNs) often rely on multipath capability for increased bandwidth and fault tolerance. In this paper, we have studied multipath routing algorithms to fully utilize available bandwidth in DCNs. We first formulate the problem as a linear program, but it is not suitable for fast on-the-fly route calculation. For practical deployment, we propose the Depth-First Worst-Fit Search based multipath routing algorithm. By observing that most DCNs have hierarchical topologies, our proposed algorithm uses depth-first search to quickly traverse between hierarchical layers to find a path. On the other hand, the worst-fit link selection criterion enables the algorithm to achieve constant time complexity link selection and use fewer overall selections for a path. Further, worst-fit also reduces packet queueing delay by load-balancing the traffic. We have implemented the proposed algorithm in simulators, and conducted simulations to evaluate the average link selection number and end-to-end delay. The simulation results fully demonstrate the superiority of our algorithm over competing solutions, and validate the effectiveness of our designs.

## References


[1] "Who Has the Most Web Servers?" http://www.datacenterknowledge.com/archives/2009/10/13/facebook-now-has-30000-servers/.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[3] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *USENIX NSDI*, San Josa, CA, Apr. 2010.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "The case for fine-grained traffic engineering in data centers," in *USENIX INM/WREN*, San Jose, CA, Apr. 2010.

[5] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Generic and automatic address configuration for data center networks," in *ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[7] "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.htmlg.

[8] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot, "Packet-level traffic measurements from the sprint ip backbone," *IEEE Network*, vol. 17, no. 6, pp. 6–16, Nov. 2003.

[9] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[10] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[11] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[12] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. Mckeown, "Elastictree: saving energy in data center networks," in *USENIX NSDI*, San Josa, CA, Apr. 2010.

[13] D. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, p. 272314, Jun. 1974.

[14] R. Karp, "On the computational complexity of combinatorial problems," *Networks*, vol. 5, no. 1, pp. 45–68, Jan. 1974.

[15] J. Kurose and K. Ross, *Computer networking: a top-down approach (4th Edition)*, 4th ed. Addison Wesley, 2007.

[16] N. Mckeown, S. Shenker, T. Anderson, L. Peterson, J. Turner, H. Balakrishnan, and J. Rexford, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2006.

[17] J. Mudigonda and P. Yalagandula, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," in *USENIX NSDI*, San Josa, CA, Apr. 2010.

[18] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer2 data center network fabric," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[19] J. Naous, D. Erickson, A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *ACM/IEEE ANCS*, San Jose, CA, Nov. 2008.

[20] "Cisco OpenFlow Switches," http://blogs.cisco.com/tag/openflow/.

[21] "HP OpenFlow Switches," http://h30507.www3.hp.com/t5/HP-Networking/Take-control-of-the-network-OpenFlow-a-new-tool-for-building-and/ba-p/92201.

[22] "NEC OpenFlow Switches," http://www.necam.com/pflow/.

[23] "OpenFlow 1.0 Release," http://www.openflowswitch.org/wk/index.php/OpenFlow_v1.0.

[24] "The OpenFlow Consortium," http://www.openflowswitch.org.

[25] D. Pan and Y. Yang, "Localized independent packet scheduling for buffered crossbar switches," *IEEE Transactions on Computers*, vol. 58, no. 2, pp. 260–274, Feb. 2009.

[26] "OpenFlow Slicing," http://www.openflowswitch.org/wk/index.php/Slicing.

[27] G. Wang, D. G. Andersen, M. Kaminsky, M. Kozuch, T. S. E. Ng, K. Papagiannaki, and M. Ryan, "c-through: part-time optics in data centers," in *ACM SIGCOMM*, New Delhi, India, Aug. 2010.