Performance Analysis of a Graph-Theoretic Load Balancing Method for Data Centers

Walaa M. AlShammari[1], Mohammed J.F. Alenazi[2]
College of Computer and Information Sciences,
Department of Computer Engineering,
King Saud University, Riyadh, Saudi Arabia

***Abstract*—Modern data centers can process a massive amount of data in a short time with minimal errors. Data center networks (DCNs) use equal-cost, multi-path topologies to deliver split flows across alternative paths between the core layer and hosted servers, which could lead to significant overload if path scheduling is inefficient. Thus, distributing incoming requests among these paths is crucial for providing higher throughput and protection against link or switch failures. Several approaches have been proposed for path selection, mainly relying on round-robin and least-congested methods. In this paper, we propose a load-balancing method based on betweenness centrality to improve the overall performance of a data center in terms of throughput, delay, and energy consumption. For evaluation, we compare our method with baseline methods of different DCN topologies: fat-tree, DCell, and BCube. On average, the evaluation results show that our method outperforms the others. It increases throughput by 202% and 33% while reducing delay by 20% and 22%, and energy consumption by 40% and 41% compared to the round-robin and least-congested methods, respectively.**

***Keywords*—Data center; load balancing; path diversity; network management; load management; throughput; topology; performance metrics; betweenness centrality; flow scheduling; modeling; DCNs**

I. Introduction

Nowadays, studies in computer networking focus on data center networks (DCNs) and challenges involved in scheduling the paths of these networks. A DCN is a construction that links a large number of servers, switches, and routers to connected devices. Data center switches are designed to forward data between endpoints, while servers process the data [1], [2], [3]. The importance of these data centers is increasing, as a considerable number of networks are now linked. Moreover, the main functions of data centers are the analysis, processing, and storage of large data.

DCNs have two types of architecture: two- and three-tier architectures. The most commonly used architecture in current DCNs is the three-tier architecture, comprising a core layer, an aggregate layer, and an access layer, from top to bottom, as shown in Fig. 1 [3], [2], [4]. When DCNs process applications' requests, the requests first arrive at the core layer. Then, the core layer forwards the requests to the destination server across multiple paths through the aggregate and access layers. Computations by servers complete applications' requests; thus, massive requests require high-performance computing servers [5], [6].

Data center traffic can be classified into two main types: mice and elephant. Mice traffic makes up a data flow of small

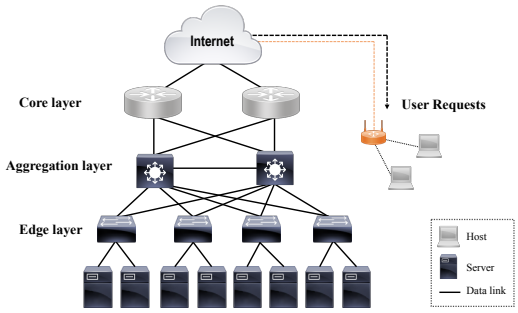


Fig. 1. Three-Tier Data Center Architecture

sizes, which is sensitive to delays (time-sensitive). An example of a mice flow is web-searching. Elephant traffic is defined as a flow that consumes more channel bandwidth. Thus, it is data-sensitive; for example, when downloading or uploading files [3], [1], [7]. Dealing with data center traffic causes a major issue known as congestion. Congestion occurs when a traffic imbalance ensues between network paths, leading to significant overload if path scheduling is inefficient—for instance, using a static approach to scheduling requests, e.g., equal-cost multipath (ECMP) or round-robin, in which requests are sequentially assigned to paths, regardless of the paths' status. Thus, some paths are overloaded, while multiple other paths are available, affecting overall DCN performance [8], [9]. Therefore, an efficient load balance of requests across multiple links (paths) must be considered to achieve the best possible data center performance. A link-based load balancing method, in general, aims to distribute requests across multiple links to avoid congestion in a single path and guide traffic to the best available destination (path). However, this task is currently one of the major challenges facing data centers [10], [11], [12], [2].

In this paper, we propose a load balancing method that utilizes a graph-theoretic betweenness centrality (C_B) metric, which attempts to forward new traffic to the least congested paths within DCN topologies. For evaluation, we compare our proposed method with baseline methods: round-robin and least-congested. We then present a comparative performance analysis between the above load-balancing algorithms and the proposed method in commonly used DCN topologies, fat-tree, BCube, and DCell, using NetworkX, a Python library for graphics and networking.

The remainder of this paper is organized as follows. In

Section II, the necessary background for the proposed method is covered, including DCN topologies, graph theory, and load balancing algorithms. Section II-B introduces the proposed load-balancing method. Section IV discusses the dataset and performance metrics. Section V presents the implementation and modeling results. Finally, Section VI presents conclusions and suggestions for future work.

II. Background and Related Work

In this section, we present a graph-theoretic background based on centrality. Furthermore, we show the most popular DCN topologies. All topologies presented were implemented and evaluated in our study. Moreover, related studies are presented in this context.

A. Graph-Theoretic Centrality Metrics

Several metrics have been introduced to measure the centrality of a node or link in a graph: to measure the importance of a vertex or an edge [13], [14]. *Degree centrality* (C_D), the degree of a node in a graph refers to the number of links attached to that node [15], [14]. A node with a high degree of centrality has a critical position in a network, since most of the links pass through it. *Closeness centrality* (C_C), is defined as the sum of the shortest paths from one node to all other nodes in a graph. *Betweenness centrality* (C_B) is a metric that is used for nodes and edges. The C_B of a node is defined as the number of shortest paths passing through that node in a graph, while the C_B of a link refers to the number of shortest paths passing through that link in a graph [16], [14]. C_B is a vital metric since the C_B value of a link or node is changed based on graph structures. The function used to calculate the C_B of a link l in networks is defined as follows:

$$C_B(l) = \sum_{u,z \in V, l \in L} \frac{|Fu, l, z|}{|Fu, z|} \quad (1)$$

where the V refers to the number of vertices, such as source or destination vertices in a graph, and L is a set of links in a graph.

set of shortest paths from a vertex u to a vertex z, passing through an edge l.

$|Fu, l, z| \Rightarrow$ set of shortest paths from a vertex u to a vertex z, passing through an edge l.

$|Fu, z| \Rightarrow$ set of all shortest paths between vertices u and z.

B. Data Center Network Architectures

DCN topologies can be classified into three categories: switch-centric, server-centric, and hybrid structure. In this section, we present examples of switch-centric and hybrid architectures.

1) Fat-tree topology: Fat-tree topology is an example of switch-centric architecture that has only switches for communication and computing tasks. It is the most widely used topology in DCNs due to a full mesh connection. Fat-tree, as illustrated in Fig. 2b, comprises three layers: a core layer, an aggregate layer, and an edge layer [17]. Servers at the bottom of a graph are directly attached to switches in the edge layer. Each edge switch comprises an n-port, which connects an n/2 server. The rest of the ports are linked to the aggregate switches. Fat-tree topology offers multipath routing between any peer hosts, which reduces the likelihood of link or switch failures. The total number of links in the fat-tree topology is $\frac{3 \times n^3}{n}$ link, where n is the number of switches [11].

2) BCube and DCell topologies: These are hybrid structure topologies that use both switches and servers for data forwarding and computing functions. They have different architectures, but both have recursively defined structures [11], [17].

$BCube_0$ comprises n count servers, where each server has a direct link to a switch at each level ($BCube_0$ and $BCube_1$ levels) through a port. Thus, servers cannot directly communicate with each other because no direct link exists between them. Moreover, as shown in Fig. 2a, switches at the $BCube_1$ level can communicate with switches at the $BCube_0$ level through an attached server, where BCube topology can construct up to three levels (k = 3). Furthermore, the number of complete links in the BCube topology is $2 * n^2$ link, where n is the number of switches [18], [11].

Moreover, the DCell architecture, as illustrated in Fig. 2c, comprises servers, mini-switches, and links. Each mini-switch comprises n servers, which are connected through a link. In addition, servers from different DCells are directly connected through connection links, and mini-switches from different DCells can communicate through an attached server. The total number of links in the DCell topology is $\frac{3n \times (n+1)}{2}$ links, where n is the number of switches [11].

C. Path Selection Methods

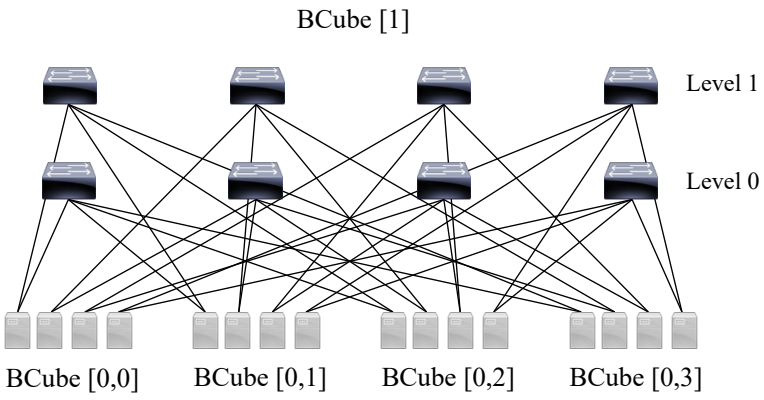
In this section, we discuss the difference between the two kinds of path scheduling algorithms: static and dynamic.

In static scheduling, traffic scheduling decisions are applied at the configuration stage, and decisions taken at this stage are independent of subsequent network statuses. A round-robin algorithm is an example of a static scheduling algorithm, which distributes requests to specific servers through sequential paths. Thus, the capacity of the server or link is not considered [11], [19]. In contrast, dynamic scheduling algorithms dynamically distribute flows through optimum links by considering the current state of a network. The 'least congested' algorithm is an example of a dynamic scheduling algorithm, which forwards a new request to the least loaded or congested link [11], [20], [21], [22].

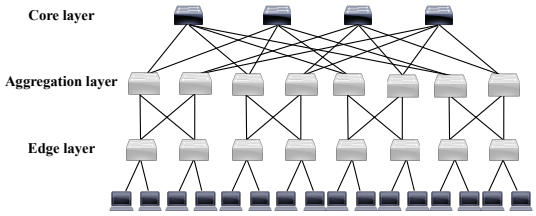
D. Related Work

Several studies have proposed approaches to improve DCN performance. In this section, we investigate related approaches.

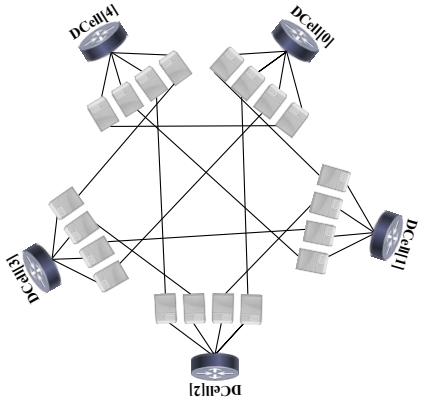
Author in [23] used Luopan, a congestion-aware load-balancing method, to distribute flowcells (sub-flows after



(a) BCube



(b) Fat-tree



(c) DCell

Fig. 2. Popular Topologies of Data Center Architectures

breaking an inserted flow) into a few available paths based on sampling. The idea is to take a few paths and distribute flowcells, relying on the least congested paths. Results show that this method outperforms the Presto algorithm, which improves the flow completion time for mice and elephant flows by 35% and 30%, respectively. However, their proposed method does not use a C_B as a metric for flow scheduling.

Zakia et al. [24] proposed a dynamic SDN-based path selection method to improve DCN load balancing. They used the Dijkstra algorithm for calculating all the shortest paths between two nodes. Then, a new flow will be assigned to the path with the least congestion and minimum cost. However, if the path has been overloaded, the path's flow will be forwarded to an alternative least-congested and lowest-cost path. However, C_B is not used in their path selection method.

Alenezi et al. [25] proposed a comprehensive comparison technique to study DCN robustness in the event of a single point of failure. They addressed this issue by increasing the number of links to balance the edge-based BC yields. Thus, they found that utilizing the BC metric and increasing the number of links improves the robustness of a network upon attacks. However, their approach lacks path scheduling, since they used C_B to increase network robustness.

Author in [26] proposed a load-balancing method that balances the workload in SDN-based DCNs. Their method monitors bandwidth utilization and the rate of packet loss before congestion occurs. When the bandwidth utilization and packet loss rate exceed a specified threshold, flows are rerouted from congested paths to alternative, less congested paths. However, if some metrics, such as latency and throughput, reach a specified threshold, a controller will set new flow rules to distribute flows among multiple paths. However, their method uses a least-congested metric to dynamically schedule the flow and they do not use the C_B metric to schedule the flow.

Shafiee et al. [27] applied a congestion-aware load-balancing method to distribute flows to paths with minimum cost, which are paths that accumulate low cost; the estimated cost is the sum of convex functions of link utilization. Moreover, assigning a flow to paths does not rely on splitting the flow into small flowlets. Thus, they reduced the difficulties associated with TCP packet reordering. Simulation results show that their method reduced the overall cost in fat-tree and JellyFish DCNs. Still, C_B is not used in their dynamic

approach for flow scheduling.

Challa et al. [28] introduced a routing method based on software-defined networking (SDN), called CentFlow. Their method utilizes the degree centrality metric to detect the node or link with the highest utilization value and avoids using that node (link) to minimize cost, since the degree of utilization affects a node's cost in a network. Thus, CentFlow selects the node or link with the lowest cost based on the Dijkstra algorithm. However, C_B does not exist in its routing method.

To briefly summarize the above-mentioned related works, we noticed that the procedure described in [24], selects the least congested path based on the shortest-path algorithm, while that in [26], considers the least-congested path and packet loss rate metric to select an optimum path. Moreover, [27], selected a path based on the least-congested metric, while [23] followed the same approach but with partial flow into multiple flowcells, which were then distributed to the least congested paths. Furthermore, [25], uses the C_B metric as an indicator to increase the number of links and network robustness against attacks, while [28], utilizes the C_B and the degree of centrality to select the optimum paths from the available shortest paths.

The approaches discussed above use only static or dynamic approaches in scheduling flows among paths. In our method, we utilize both C_B and least-congested metrics to select the optimum path for a flow from diverse paths. We evaluate and compare the performance of our method with other load-balancing methods in widely adopted DCNs.

III. A Load-Balancing Method

In this section, we present our proposed load-balancing method, which utilizes C_B and least-congested metrics to select paths between the core and edge layers.

We propose a C_B-based dynamic load balancing method that performs load balancing in a DCN topology by measuring the minimum C_B of all links as well as the least congested path at a particular time. The method obtains all simple paths between a single source and a single destination node through which flows can be sent. Traffic flows are of the same priority and size. Among the selected paths, the path with the least C_B and congestion is selected, and traffic flow is forwarded along that path. The performance of the proposed method is evaluated in a state-of-the-art DCN topology by collecting data from links and switches. Fig. 3 illustrates a flow diagram of our load-balancing method.

To demonstrate our proposed method, we apply it to a simple topology, as shown in Fig. 4. The graph has two, three, and four nodes from top to bottom and has a link of 1 Gbps capacity connecting the nodes. For instance, host1 sends data to host2 through multiple paths. We evaluate, compute, and assign a C_B value to each edge using NetworkX, as shown in Fig. 4.

Suppose we have three paths from node 0 to node 6, where each path has a number of files; for example, path 1 ([0, 1, 3, 2, 6]) has one file, path2 ([0, 1, 4, 2, 6]) has three files, and path3 ([0, 2, 6]) has one file. The question arises as to which paths the new files are to be assigned. Our proposed method is based on a four-step process. First, the C_B values of all edges in the paths are summed. Second, the C_B values of each path are normalized by dividing each value by the maximum C_B value. Then, the congestion at each path is normalized: the number of files in each path is divided by the maximum number of files. Table I illustrates the C_B and normalized values. Finally, each path is assigned a value equal to the sum of the normalized C_B and congestion values, and the path with the lowest value is selected. In the example shown in Fig. 4, we see that path 3 ([0, 2, 6]) has the lowest value. Therefore, the new file will be forwarded to this path.

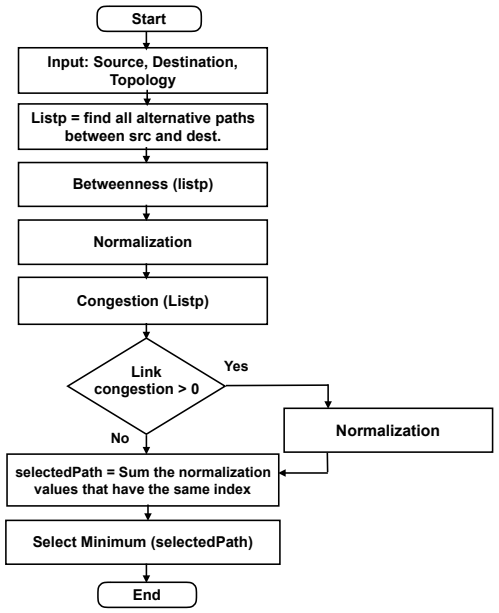


Fig. 3. The Proposed Load-Balancing Method.

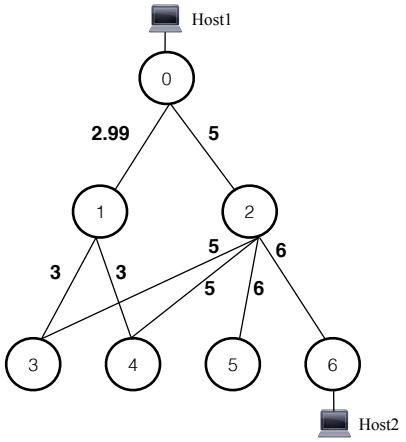


Fig. 4. A DCN Topology to Illustrate the Proposed Load-Balancing Method. Values that Appear at Each Edge in the Figure are the Betweenness Centrality.

IV. Evaluation

This section discusses our evaluation and presents the dataset of the DCN topologies and performance metrics used.

TABLE I. BETWEENNESS AND NORMALIZATION VALUES BASED ON THE EXAMPLE IN FIG.4

Paths	Betweenness Values	Normalization
[0,1,3,2,6]	16.99	1
[0,1,4,2,6]	16.99	1
[0,2,6]	11	0.65

TABLE II. DATASET USED IN THE EVALUATION OF THE PROPOSED WORK.

Fat-tree DCN	Number of Pods	4
	Number of nodes	20
BCube DCN k-level	1	
DCell DCN level	1	
Link bandwidth	1 Gbps	
Requests number	200	
Packet size	1500 KB	

Thereafter, the evaluation results are presented and discussed.

A. Dataset

The DCN topologies mentioned in Section II are implemented and evaluated. The dataset used to evaluate the proposed method against the other load balancing methods is shown in Table II. To evaluate the fat-tree topology, we use a network of four core switches, eight aggregation switches, and edge switches. Each core switch is connected to an edge switch through aggregation switches via 1 Gbps links.

The BCube topology is implemented for level one (k = 1) with four $BCube_1$ and $BCube_0$ switches. Each $BCube_0$ switch connects to four servers, making up a total of 16 servers. Each link used to connect the layers has a capacity of 1 Gbps.

The DCell topology is implemented for level 1 and comprises five switches, each connecting to four servers through a direct link of 1 Gbps. Each server is connected to other servers in different $DCell_0$ through a 1 Gbps link.

B. Performance Metrics

In this section, we present three metrics for measuring the impact of the three load-balancing methods on the performance of DCN topologies.

1) End-to-end throughput: To calculate the throughput of a DCN topology from a single-core source to a server destination, we calculate the total time for all files to be delivered, while considering bottleneck links. Completion time is the time taken to complete a specific task [29]. Then, the total size of all the files is divided by the total completion time. The approximate completion time for each path is calculated as follows:

$$(F-1) \times (T_d + P_d) \tag{2}$$

where F is the total number of files, Td is the transmission delay, and Pd is the propagation delay. To simplify the calculations for when congestion occurs, we assume that the files of the relevant paths are sent last. This is because we are only interested in the maximum completion time.

2) End-to-end delay: The following equation is used to calculate the end-to-end delay:

$$TotalDelay = T_d + P_d + U_d + Q_d \tag{3}$$

where T_d is transmission delay, which is the required time to transmit a packet in a channel; P_d is propagation delay, which is the time taken to forward packets across the channel; U_d is a processing delay, which refers to the time taken to process a task; and Q_d is a queuing delay, which is the packet's waiting time in a buffer [30]. In our measurements, U_d is equal to zero, and T_d can be calculated using $\frac{Request\ Size}{Link\ Capacity}$, where 1.5 GB is the request size and 1 Gbps is the link capacity. P_d can be calculated using $\frac{Channel\ Length}{Propagation\ Speed}$, where the propagation speed is the speed of light ($2 \times 10^8\ m/s$), and the link length is 10km. To calculate the queuing delay, we focus on calculating the average queuing delay for all links. Thus, we first find the total number of sent files (F) and subtract them one by one, starting with the first file, which has no waiting time. Then, we multiply the total number of files minus one by (T_d). Finally, we divide the calculated value by two, which affords us the average Q_d.

3) Energy consumption: Energy consumption is one of the biggest challenges faced by DCNs. Significant effort has been made to reduce the power consumed by data center components, while ensuring high network performance [11]. Thus, the use of the proposed load-balancing method for selecting an optimum path for sending packets is an appropriate way to reduce energy consumption [6]. Thus, a large workload in links increases energy consumption in a DCN [31], [32]. In our evaluation, we aimed to estimate energy consumption by considering the number of switches in paths, regardless of any other components in the DCN. This is because switches consume more energy than servers [11].

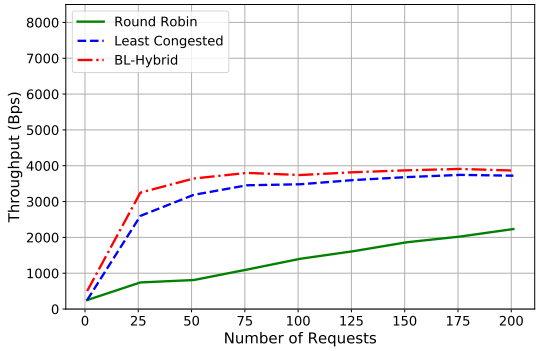
V. RESULTS AND DISCUSSIONS

In this section, we present and discuss the results of the three load-balancing methods applied to the three data center topologies, based on the experimental setup presented in Section IV. Our evaluation is based on three performance metrics: throughput, end-to-end delay, and energy consumption.

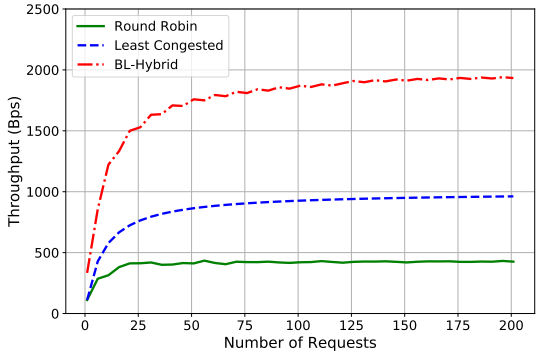
A. Throughput Results

The throughput results of the application of the three load-balancing methods to the three DCN topologies are shown in Fig. 5.

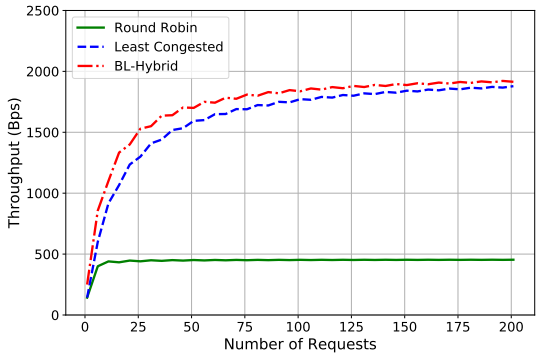
The throughput results in the fat-tree topology, as displayed in Fig. 5a show that our method has the best results compared to the other load-balancing methods. We notice that, for request number 25, the round-robin decreased the throughput by approximately 69% and 76% compared to the least-congested and our method, respectively. In contrast, we see that our scheme increased the total throughput by 3% and 65%, compared to the least-congested and round-robin algorithms, respectively. Furthermore, we observed that the round-robin has the worst results, achieving the lowest throughput compared to the other methods because it assigns flows to paths regardless of the paths' load status, reducing the throughput of the DCN.



(a) Fat-tree



(b) BCube



(c) DCell

Fig. 5. Throughput Analysis for Data Center Topologies

Moreover, the throughput results in the BCube topology, as demonstrated in Fig. 5b, validated that our method increases throughput by approximately 92% and 296% compared to the least congested and round-robin algorithms, respectively. It is followed by the least-congested algorithm, with a throughput 106% higher than the round-robin, which achieves the worst throughput results due to its static approach.

The throughput results in the DCell topology are shown in Fig. 5c. Fig. 5c shows that our method has the highest throughput compared to the other methods. We notice that, at request number 25, the round-robin achieves a throughput of approximately 60% and 68% lower compared to the least-congested method and our own, respectively. Conversely, our method increases the total throughput by 3% and 247% compared to the least-congested and round-robin, respectively. On the other hand, we observe that the round-robin algorithm has the worst throughput results because it does not consider the load status of paths in each case, affecting the overall throughput results.

By studying all the throughput results in different DCN topologies, we can see that our proposed method outperforms the other load-balancing algorithms. This is mainly because our method selects a high-bandwidth path, while avoiding the highly congested paths and bottleneck links. On the other hand, the round-robin algorithm achieves the poorest throughput results due to its static scheduling approach which assigns flows regardless of paths' load status. The round-robin algorithm reduces the throughput results by 40%, 75%, and 71% in the fat-tree, BCube, and DCell topologies, respectively, compared to our method. Moreover, it reduces the throughput results by 38%, 52%, and 70%, respectively, compared to the least-congested algorithm.

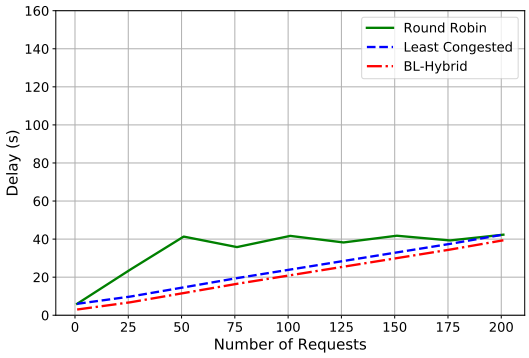
B. End-To-End Delay Results

End-to-end delay results of the three load balancing methods in the three DCN topologies are shown in Fig. 6.

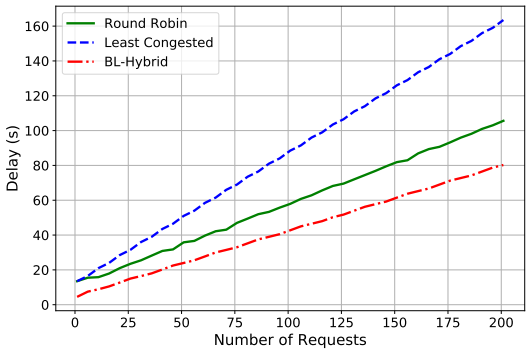
The delay results in the fat-tree topology are shown in Fig. 6a. We observe that our scheme reduces the delay by 11% compared to the least-congested and round-robin algorithms. Thus, it concisely achieves the best results, because our load-balancing method avoids congested bottleneck links. We observe that the least-congested and round-robin methods produce the same results at the end of requests. However, the round-robin algorithm incurs the highest delay at the start of requests until request number 175, which is a negative sign of load balance.

Next, Fig. 6b shows the delay results in the BCube topology. We can distinguish that our method outperforms the other load-balancing methods in terms of delay time in the BCube topology. It reduces the delay by approximately 51% and 30% compared to the least-congested and round-robin methods, respectively. Moreover, we observe that the least-congested method has the highest delay because it uses bottleneck links despite the presence of congestion.

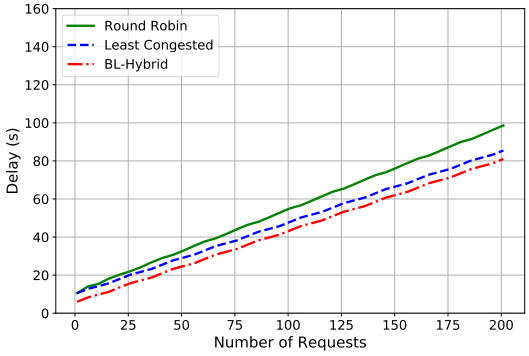
The delay results in the DCell topology are demonstrated in Fig. 6c, which shows that our load-balancing method results in has the lowest delay compared to the other methods. Our method reduces overall delay by approximately 5% and 18%



(a) Fat-tree

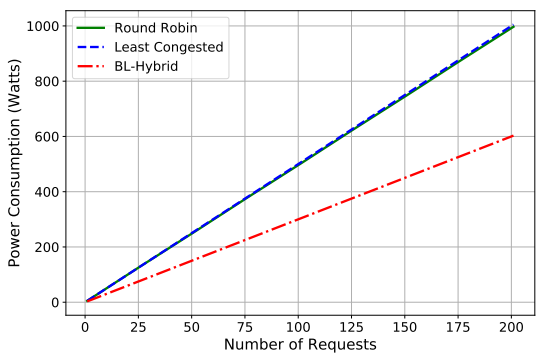


(b) BCube

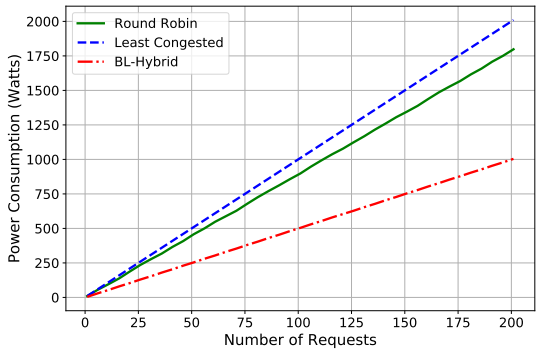


(c) DCell

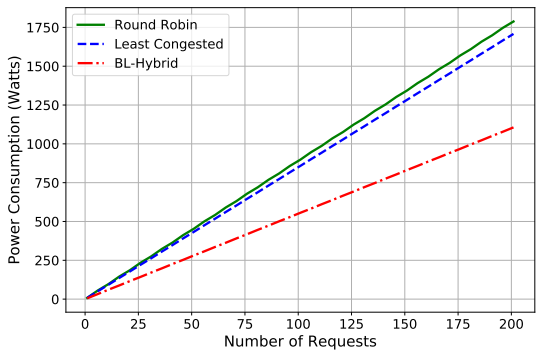
Fig. 6. End-To-End Delay Analysis for Data Center Topologies



(a) Fat-tree



(b) BCube



(c) DCell

Fig. 7. Energy Consumption Analysis for Data Center Topologies

compared to the least-congested and round-robin methods, respectively, from smaller to larger requests. Furthermore, the round-robin algorithm incurs the worst delay results, while increasing the number of requests because it distributes flows regardless of paths' load status.

We can conclude that our method has better delay results than the other load-balancing methods studied in several DCN topologies. This is primarily because our method bypasses highly congested links with the help of the BC value of each path, which, as mentioned earlier, directly affects network latency. Moreover, we note that the round-robin method provides the worst results in both fat-tree and DCell topologies because it forwards requests regardless of paths' availability. As a result, the round-robin algorithm increased the network delay by 13%, 44%, and 21% in the fat-tree, BCube, and DCell DCNs, respectively, compared to our method. It increased the delay by approximately 16% in the DCell topology, while it has the same delay result in the fat-tree topology, and in the BCube topology, about 41% higher delay than the least-congested method.

C. Energy Consumption Results

Energy consumption results from the three load balancing methods in the evaluated DCN topologies are displayed in Fig. 7.

Fig. 7a shows the energy consumption results in the fat-tree topology. We observe that, in the fat-tree topology, our method affords the optimum results, while the round-robin and least-congested methods achieve the same results. Moreover, both the round-robin and least-congested methods consume up to 1000 watts, while our method consumes only 600 watts. As a result, our method reduces energy consumption by 40% in the fat-tree topology compared to the other algorithms.

Furthermore, the energy consumption results in the BCube topology are shown in Fig. 7b. We notice that the round-robin algorithm energy consumption results are approximately 12% lower than those of the least-congested algorithm and 76% higher than those of our method. Thus, we notice that the least-congested method consumes more energy than the other methods because it uses specific links with the maximum number of hops. On the other hand, the round-robin algorithm assigns flows in sequential order, regardless of the number of switches in links, while our method uses links with fewer hops because these links have the lowest C_B values. Thus, our method achieves the lowest energy consumption compared to other methods.

The energy consumption results in the DCell DCN, as illustrated in Fig. 7c, show that our load-balancing method has the best results, consuming approximately 1100 watts. It is followed by the least-congested method and then the round-robin method, which consume approximately 1650 watts and 1755 watts, respectively. As a result, we see that our method reduces energy consumption by approximately 33% and 38% compared to the least-congested and round-robin algorithms, respectively.

Our proposed method consistently improves energy consumption in DCNs, because it uses paths with the lowest C_B values, which are the paths with a low number of hops since using paths that have a large number of hops will increase energy consumption. On the other hand, the least-congested method achieves the worst energy consumption results in the BCube DCN, because it forwards flows to the least congested paths, regardless of the number of hops in those paths. In addition, the round-robin consumes more energy in the DCell topology, increasing the energy consumption by 46% and 6% compared to our method and the least-congested method, respectively, in the DCell topology. Moreover, we notice that the least-congested method and round-robin have the same results in the fat-tree topology, both consuming more energy than our method.

To summarize these results, we studied and analyzed all the performance results in the different DCN topologies investigated. We found that our load-balancing method outperforms the other methods. On average, the throughput results show that our method achieves a total throughput that is 202% and 33% higher compared to the round-robin and least-congested algorithms, respectively. Moreover, it reduces network delay by 20% and 22%, and energy consumption by 40% and 41% compared to the round-robin and least-congested methods, respectively. This is because our method bypasses bottleneck links, while considering the lowest edge C_B values. On the other hand, we notice that the round-robin algorithm has the worst throughput results, because it assigns flows sequentially, regardless of the paths' load status. We also observed that the least-congested algorithm has the worst results in delay and energy consumption, since it assigns a flow to a path despite the bottleneck links and the number of hops in that path.

VI. Conclusion

Improving the performance of data center systems would allow DCNs to be scalable and more customizable, helping to connect more devices around the world. This paper presented a comparative performance analysis of some load-balancing methods in three DCN topologies: fat-tree, BCube, and DCell. We proposed a load-balancing method based on BC. Our method was compared to other load-balancing algorithms in different DCN topologies using a one-to-one traffic scenario.

We conclude that our method outperforms the existing load-balancing methods in all the three DCN topologies investigated. It improves throughput by 202% and 33%, while decreasing delay by 20% and 22%, and energy consumption by 40% and 41% compared to the round-robin and least-congested methods, respectively. The evaluation and DCN topology graphs were implemented using NetworkX. In our future studies, we will evaluate the performance metrics in different traffic scenarios. Moreover, this study will be evaluated in a simulation program, i.e., ns3.

Acknowledgment

The authors acknowledge the support of the Researchers Support & Services Unit, King Saud University, Riyadh, Saudi Arabia.

References

[1] F. Estrada-Solano, O. M. Caicedo, and N. L. S. Da Fonseca, "Nelly: Flow detection using incremental learning at the server side of sdn-based data centers," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1362–1372, 2020.

[2] H. Geng, H. Zhang, X. Shi, Z. Wang, and X. Yin, "Efficient computation of loop-free alternates," *Journal of Network and Computer Applications*, vol. 151, p. 102501, 2020.

[3] M. A. S. Saber, M. Ghorbani, A. Bayati, K.-K. Nguyen, and M. Cheriet, "Online data center traffic classification based on inter-flow correlations," *IEEE Access*, vol. 8, pp. 60401–60416, 2020.

[4] L. Wang, X. Wang, M. Tornatore, K. J. Kim, S. M. Kim, D. Kim, K. Han, and B. Mukherjee, "Scheduling with machine-learning-based flow detection for packet-switched optical data center networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, pp. 365–375, April 2018.

[5] J. Ye, L. Feng, Z. Xie, J. Huang, and X. Li, "Fine-grained congestion control for multipath tcp in data center networks," *IEEE Access*, vol. 7, pp. 31782–31790, 2019.

[6] Y. Wang and S. You, "An efficient route management framework for load balance and overhead reduction in sdn-based data center networks," *IEEE Transactions on Network and Service Management*, vol. 15, pp. 1422–1434, Dec 2018.

[7] W. Wang, Y. Sun, K. Zheng, M. A. Kaafar, D. Li, and Z. Li, "Freeway: Adaptively isolating the elephant and mice flows on different transmission paths," in *2014 IEEE 22nd International Conference on Network Protocols*, pp. 362–367, IEEE, 2014.

[8] R. Dewanto, "Improved load balancing on software defined network-based equal cost multipath routing in data center network," *JURNAL INFOTEL*, vol. 10, no. 3, pp. 157–162, 2020.

[9] B. He, D. Zhang, and C. Zhao, "Hidden markov model-based load balancing in data center networks," *The Computer Journal*, 2019.

[10] H. Sufiev, Y. Haddad, L. Barenboim, and J. Soler, "Dynamic sdn controller load balancing," *Future Internet*, vol. 11, no. 3, p. 75, 2019.

[11] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 2324–2352, thirdquarter 2018.

[12] S. M. Shetty and S. Shetty, "Analysis of load balancing in cloud data centers," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–9, 2019.

[13] E. Bergamini, P. Crescenzi, G. D'angelo, H. Meyerhenke, L. Severini, and Y. Velaj, "Improving the betweenness centrality of a node by adding links," *Journal of Experimental Algorithmics (JEA)*, vol. 23, pp. 1–32, 2018.

[14] M. J. F. Alenazi and J. P. G. Sterbenz, "Comprehensive comparison and accuracy of graph metrics in predicting network resilience," in *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 157–164, March 2015.

[15] A. Elibol and N.-Y. Chong, "Topology graph pruning for optical mapping methods using edge betweenness centrality," in *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 954–959, IEEE, 2019.

[16] F. Jamour, S. Skiadopoulos, and P. Kalnis, "Parallel algorithm for incremental betweenness centrality on large graphs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, pp. 659–672, March 2018.

[17] H. M. Helal and R. E. Ahmed, "Performance evaluation of datacenter network topologies with link failures," in *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, pp. 1–5, April 2017.

[18] Z. Han and L. Yu, "A survey of the bcube data center network topology," in *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, pp. 229–231, May 2018.

[19] J. Li, X. Chang, Y. Ren, Z. Zhang, and G. Wang, "An effective path load balancing mechanism based on sdn," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 527–533, Sep. 2014.

[20] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 149–160, 2014.

[21] Y.-L. Lan, K. Wang, and Y.-H. Hsu, "Dynamic load-balanced path optimization in sdn-based data center networks," in *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pp. 1–6, IEEE, 2016.

[22] N. S. B. Saeed and M. J. Alenazi, "Utilizing sdn to deliver maximum tcp flow for data centers," in *Proceedings of the 2020 The 3rd International Conference on Information Science and System*, ICISS 2020, (New York, NY, USA), p. 181–187, Association for Computing Machinery, 2020.

[23] P. Wang, G. Trimponias, H. Xu, and Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, pp. 133–145, Jan 2019.

[24] U. Zakia and H. Ben Yedder, "Dynamic load balancing in sdn-based data center networks," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 242–247, Oct 2017.

[25] M. J. F. Alenazi and J. P. G. Sterbenz, "Evaluation and comparison of several graph robustness metrics to improve network resilience," in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, pp. 7–13, Oct 2015.

[26] F. S. Fizi and S. Askar, "A novel load balancing algorithm for software defined network based datacenters," in *2016 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*, pp. 1–6, Sep. 2016.

[27] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," *IEEE/ACM Transactions on Networking*, vol. 25, pp. 3670–3682, Dec 2017.

[28] R. Challa, S. Jeon, D. S. Kim, and H. Choo, "Centflow: Centrality-based flow balancing and traffic distribution for higher network utilization," *IEEE Access*, vol. 5, pp. 17045–17058, 2017.

[29] A. Rabih, A. Ghandour, Y. N. Shnaiwer, and S. Al-Ghadhban, "Rate-aware instantly decodable network codes for heterogeneous cellular networks," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pp. 1–5, IEEE, 2019.

[30] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Dx: Latency-based congestion control for datacenters," *IEEE/ACM Transactions on Networking*, vol. 25, pp. 335–348, Feb 2017.

[31] O. Popoola and B. Pranggono, "On energy consumption of switch-centric data center networks," *The Journal of Supercomputing*, vol. 74, no. 1, pp. 334–369, 2018.

[32] L. Gyarmati and T. A. Trinh, "How can architecture help to reduce energy consumption in data center networking?," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pp. 183–186, ACM, 2010.