Performance Impact Inference with Failures in Data Center Networks

Che Zhang*, Hong Xu*, Chengchen Hu†
*NetX Lab, City University of Hong Kong, Hong Kong
†Department of Computer Science and Technology, Xi'an Jiaotong University, China

***Abstract*—Maintaining a data center network (DCN) is crucial to many services running on top of it, especially given its large scale with tens of thousands of network components. In this paper, we propose a method to infer performance change before failures really happen in data center networks, called Sibyl. Different from previous work, Sibyl relies on network topology information to infer network performance under failure scenarios without the overhead of active measurements. Specifically, we demonstrate that most important performance metrics can be obtained from two fundamental topological metrics. We develop efficient algorithms to obtain these two fundamental metrics, leveraging graph automorphism of various DCN topologies.**

I. Introduction

As the underlying infrastructure of the cloud, data center networks (DCN) typically have tens of thousands of network devices and cost millions of dollars to build and maintain [6, 10]. In such a large scale network, failures caused by hardware/software malfunctions and human errors are the norm rather than exception [5, 12]. Failures may cause severe performance degradation to the Internet services running on top of the data center network. Thus, it is crucial to design an automatic system to compute the network performance change after failures for operators to locate and pinpoint the failed devices.

While many efforts have been devoted to failure detection [9], diagnosis [3, 15], and mitigation/recovery [19], relatively little work has been done on inference of performance degradation *before* failures actually happen. We argue that performance inference with failures is instrumental in many scenarios. For example it can help operators and developers better test their software under failures, by providing the network performance impact for possibly all combinations of failures. Currently this is done either through communications with network engineers which takes time and is inaccurate; or by using a chaos monkey approach [14] that proactively introduces failures to production networks to see how the software reacts, which may disrupt production services. A better understanding of the performance loss caused by failures can also help the operator design more resilient traffic engineering schemes, by computing and installing backup tunnels for failures that have significant impact on the network.

In this paper, we propose a novel performance impact inference system for network failures, called Sibyl. Our main idea is that we can use topology information to infer network performance after failures. Specifically, we demonstrate that various performance metrics, including throughput, load balance, network delay, *etc.*, can be deduced accurately from just two basic topological properties — the shortest path and the maximum number of edge-disjoint paths between any two nodes. By enumerating through the possible topologies with different failure scenarios, one can thus calculate the impact of them to different performance metrics, without implementing any new protocols or modifying the existing network.

The technical challenge in Sibyl becomes: how to efficiently compute various performance metrics and assess the performance degradation given the topology information. Efficiency is important as the operator may need to evaluate a large number of possible failure cases in practice for better resilience. On the other hand, the sheer size of a data center network with tens of thousands of forwarding devices poses significant difficulty here.

Our main contribution in this paper is that we rely on graph automorphism for symmetric DCN topologies to develop efficient algorithms to compute network performance metrics. Although there are many algorithms solving the shortest path and the maximum number of edge-disjoint paths problems individually, most of them are for general topologies and do not consider the unique characteristics of DCN topologies. Using automorphism sets that can be readily obtained from DCN topologies, including fat-tree [1], BCube [7], and DCell [8], our algorithms can calculate the two basic topological properties to reduce the time complexity. Finally, for any scale of fat-tree and BCube, we can reduce $N(N-1)$ times calculation of paths to constant times and the time complexity of computing is $O(1)$. Although DCell is a little different from them due to the complex connections, we can still reduce the times of calculation.

The rest of the paper is organized as follows. Section II introduces the DCN topologies and performance metrics. The main focus of this paper, performance inference component, is described in details in Section III. We validate our design via extensive experiments and simulations in Section IV. Related works are shown in Section V. Finally, we conclude the paper in Section VI.

II. Topology and Performance Metrics

We first introduce the performance metrics we consider in this paper, then demonstrate that they can all be calculated from two key properties of the network topology, namely the shortest

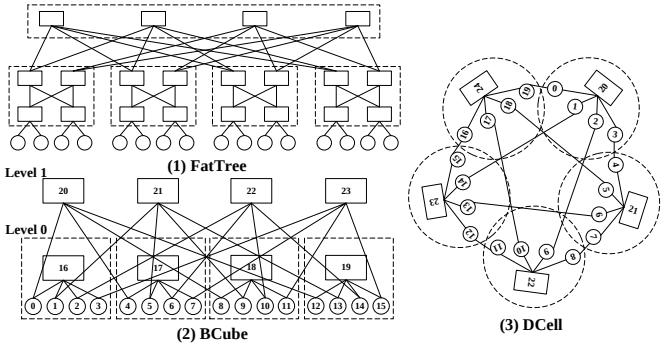


Fig. 1. Three topologies

path and the maximum number of edge-disjoint paths between two nodes.

A. Background and Assumptions

Data center networks usually follow richly connected symmetric topologies between a pair of hosts. We mainly consider three DCN topologies: fat-tree [1], BCube [7], and DCell [8] as shown in Fig. 1.

A k-pod fat-tree built from k-port switches can support non-blocking communication among $k^3/4$ end hosts using $5k^2/4$ individual k-port switches. The switches of fat-tree are split into three layers: edge, aggregation and core. A fat-tree consists of k pods, each having $k^2/4$ hosts. Fig 1(1) is a 4-pod fat-tree.

BCube is a recursively defined structure as shown in Fig. 1(2). A BCube_0 is simply n servers connecting to an n-port switch. A BCube_k ($k \geq 1$)) is constructed from n BCube_{k-1}s and n^k n-port switches.

DCell is also a recursively defined structure. We use DCell_k to denote a level-k DCell. A DCell_0 is simply n servers connecting to an n-port switch. A level-1 DCell_1 is constructed using $n+1$ DCell_0s, each DCell_0 is connected to all the other DCell_0s with one link. We treat each DCell_{k-1} as a virtual node and fully connect these virtual nodes to form a complete graph—DCell_k. Fig. 1(3) is a DCell_1 network with $n = 4$.

Network performance critically depends on the traffic pattern, which varies across time and is hard to model [2, 13]. For simplicity, in Sibyl we restrict ourselves to considering only the all-to-all traffic pattern where each server communicates with all the other servers in the network. This is consistent with much existing work on DCN [7, 8] as it reflects the worst-case performance for networks.

Network performance also hinges on how traffic is routed among all the available paths in the network. In practice DCNs use equal-cost multi-path routing (ECMP) to perform multipath load balancing uniformly at random across all of the equal-cost paths [16]. We thus assume each host's flows are routed in a round-robin fashion starting from the leftmost path in Fig. 1. This results in identical performance compared to ECMP without having to take into account the randomness.

B. Performance Metrics

We consider six common performance metrics widely used in DCN. Sibyl can be configured to compute all or a subset of these metrics depending on the need. Before explaining their definitions we introduce some notations first.

We use l to denote a link, n_l to denote the number of flows traversing l, c to denote capacity of the link, and f to denote a flow in the network. Flow f traverses through a path p_f in the network. Let n denote the total number of all-to-all flows.

The following metrics are defined with respect to a pair of hosts or nodes in the network.

Throughput: The fair share rate achievable by TCP on link l is $b_l = c/n_l$. Thus the throughput of flow f is equal to its bottleneck fair share rate along its path p_f: $b_f = \min_{l \in p_l}\{b_l\} = \min_{l \in p_l}\{c/n_l\}$.

Network connectivity: This is defined as the minimum number of links whose removal disconnects the given two nodes in the topology, i.e. the min-cut between them.

Path redundancy: To demonstrate the availability of equal-cost paths, we define path redundancy as the maximum number of edge-disjoint paths between two given nodes.

Network delay: We define network delay simply as the length of the shortest path. Modeling queueing delay in DCN is challenging and beyond the scope of this paper.

The following metrics are defined for the network.

Aggregate bottleneck throughput (ABT): We use the same definition for ABT from [7] here: $\text{ABT} = n * \min_f\{b_f\}$.

Network throughput: Let T denote the throughput of the network across all flows. $T = \sum_f b_f$.

C. Metrics and Topological Properties

We observe that all performance metrics are closely related to the graph theoretical properties of the topology. With the all-to-all traffic pattern and round-robin based uniform multipath routing, throughput can be readily determined from shortest-path calculation, and hence ABT and network throughput. Network delay is defined based on shortest path. Network connectivity is related to the min-cut, and path redundancy depends on the maximum number of edge-disjoint paths. Moreover, we have the following:

Lemma 1. *In a undirected graph, the maximum number of edge-disjoint paths is equal to the minimum number of edges whose removal separates the two nodes, i.e. the min-cut [11].*

Therefore the following holds:

Theorem 1. *All the performance metrics can be calculated from the shortest path and the maximum number of edge-disjoint path in a data center network.*

Table I summarizes the relationship between metrics and topological properties of the DCN. We refer to the two properties, the shortest path and the maximum number of edge-disjoint paths, as the *path tuple* between any two nodes in the network.

TABLE I
METHODS OF COMPUTING PERFORMANCE METRICS

Property	Network performance metrics
shortest path	throughput, network delay, ABT, network throughput
maximum number of edge-disjoint paths	path redundancy, network connectivity

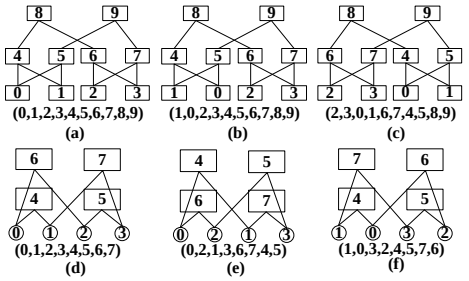


Fig. 2. Examples of graph automorphism.

III. Performance Inference Algorithms

We have shown that inferring performance metrics in a DCN boils down to calculating the path tuple of all node pairs. When there are N servers in a data center, a naive approach requires to calculate $N(N-1)$ path tuples for $N(N-1)$ pairs of source-destination nodes on a general graph. This is computationally expensive for large-scale DCNs. Since the DCN topology is largely symmetric even with failures, we propose a new algorithm to accelerate the path tuple computation. We find that all three topologies shown in Fig. 1 have automorphism.

A. Graph automorphism

We start by introducing graph automorphism. From [18] we have the following:

Definition 1. *An* automorphism *of a graph $G = (V, E)$ is a permutation σ of the vertex set V, such that the pair of vertices (u, v) form an edge if and only if the pair $(\sigma(u), \sigma(v))$ also form an edge. That is, it is a graph isomorphism from G to itself.*

Theorem 2. *If exchanging two nodes in the graph by keeping the edges of the graph unchanged does not change the positions of all edges in the graph, these two nodes are automorphic.*

An example is as shown in Fig. 2. The three graphs on the top are the topologies of half of a 4-pod fat-tree with different labeling of switches. The first graph is the original graph Fig. 2(a) with the labels from left to right and then bottom up. To see that all edge switches in the graph are automorphic, we choose the pairs $(0, 1)$ and $(0, 2)$. When we exchange switches 0 and 1 by keeping their edges unchanged, we obtain the second graph Fig. 2(b) whose connections are identical with the first graph. Thus the two graphs are an automorphism of each other, and nodes 0 and 1 are automorphic.

Moreover we can extend to the following:

Theorem 3. *If exchanging two units in the graph by keeping their edges unchanged does not change the positions of all edges in the graph, the corresponding nodes in these two units of the graph are automorphic.*

We call these *automorphic units*. In Fig. 2, we choose the pods $(0, 1, 4, 5)$ and $(2, 3, 6, 7)$ in Fig. 2(a) to be the units. After exchanging them in the first graph, we obtain the graph Fig. 2(c). Clearly $(0, 1, 4, 5)$ corresponds to $(2, 3, 6, 7)$ and they are automorphic units. As another example, a simple BCube is shown in Fig. 2(d). We choose $(4, 5)$ and $(6, 7)$ to be units and exchange them to obtain Fig. 2(e). In order to keep the positions of edges unchanged, we change the positions of switches $2, 1, 3$ and find the graph is the same as the original. Thus the two units are automorphic correspondingly. Similarly, When we exchange 0 and 1, we obtain Fig. 2(f).

We now introduce the definition of automorphic set as follows:

Definition 2. *An automorphic set is a subset of nodes in graph G such that two nodes u and v are in the same automorphic set if there exists an automorphism of G that maps u to v.*

We can then prove that all three DCN topologies are composed of extensive automorphic sets.

Theorem 4. *A fat-tree topology is composed of four mutually exclusive automorphic sets: the core/aggregation/edge switch set, and the server set. BCube is composed of two mutually exclusive automorphic sets: the server set and the switch set. $DCell_1$ has two automorphic sets including the server set and the switch set as well. $DCell_k, (k > 1)$ has $s/2$ automorphic sets for servers (s stands for the number of servers in $DCell_{k-1}$).*

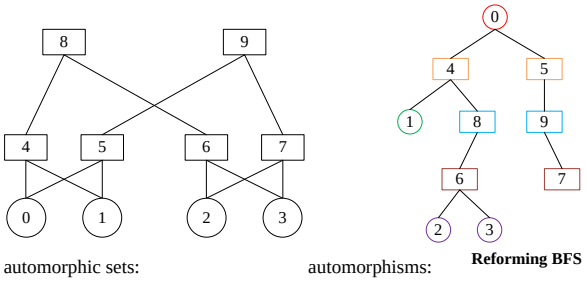
The proof is omitted here for space and all the proof of theorems in this paper can be got from our technical report [20]. We sketch the basic idea here. First we find the automorphic units with the largest cardinality in each topology. Then we identify that some node pairs in the automorphic unit are also automorphic. Now we can use the reflexive (node u is automorphic with itself), symmetric (if node u and node v are automorphic, v and u are automorphic), and transitive (if node u and node v are automorphic, v and w are automorphic, then u and w are automorphic) properties of automorphism relation to identify and prove the automorphic set in which each node is automorphic to each other.

B. Algorithm

Using the automorphic property of DCN topologies, we propose a new algorithm to reduce the number of path tuples needed to infer performance from $N(N-1)$ to a constant for fat-tree, BCube and $DCell_1$. For $DCell_k, (k > 1)$, we can also reduce it using the following lemma although not all the servers of $DCell_k$ are automorphic.

Lemma 2. *Given the paths $\mathcal{P}$ from u to all other nodes in an automorphic set, the paths from v to all the other nodes in the same automorphic set can be readily obtained from $\mathcal{P}$ by applying the permutation σ where $\sigma(v) = u$ to all the nodes on a path.*

This lemma implies that, since all the servers are automorphic in fat-tree, BCube and $DCell_1$, we can reduce the number of tuples to be calculated from $N(N-1)$ to $N-1$. That is we only need to calculate all $N-1$ path tuples from a server s to other servers. Path tuples from s' to other servers can be obtained by applying automorphism. As an example in Fig. 3, we know that all the shortest paths from server 0 to all other servers. Suppose now we want to find the shortest paths from server 2 to server 0. We know that one permutation for server 2 as shown in the bottom right of Fig. 3 maps 2 to 0, and 0 to



automorphic sets:
(0,1,2,3),(4,5,6,7),(8,9)
cells:
0 1 2 2 3 4
(0),(4,5),(1),(8,9),(6,7)(2,3)
server cells:
(0),(1),(2,3)
server pairs needing to
calculating paths:(0,1),(0,2)

automorphisms:
path (0,3) 0,1,**2,3**,4,5,6,7,8,9
0,1,**3,2**,4,5,6,7,8,9
server 1 **0,1**,2,3,4,5,6,7,8,9
1,0,2,3,4,5,6,7,8,9
server 2 **0,1,2,3,4,5,6,7**,8,9
2,3,0,1,6,7,4,5,8,9
server 3 **0,1,2,3,4,5,6,7**,8,9
3,2,0,1,6,7,4,5,8,9

Fig. 3. An example of the algorithm

2. Thus according to automorphism it is equivalent to finding shortest paths from server 0 to 2. One such path is via switches 4, 8, and 6. The corresponding shortest path from server 2 to 0 is then 6, 8, 4, again according to the same permutation for server 2.

To further reduce the number of tuples from N to constant for fat-tree, BCube and DCell$_1$ whose servers are all automorphic, we need the following:

Definition 3. *Starting from a server s, an equal-cost automorphic group (ECAG) is the group of nodes that are automorphic to each other, and have the same length of shortest path to s. The corresponding nodes in their shortest path to s are also belonging to corresponding ECAGs.*

We can efficiently find equal-cost automorphic groups using the following procedure based on breadth-first search (BFS), called reforming BFS. First we identify automorphic sets. Then choose any arbitrary server s, and run BFS. For all nodes at the same depth of the search, we see if a node is automorphic or not to others according to automorphic sets, and form the ECAGs accordingly. For example in Fig. 3, we start from server 0 and run BFS. With depth 1, BFS finds switches 4 and 5 which belong to the same ECAG. At depth 2, we have server 1 and switches 8 and 9. Server 1 is not automorphic to switches 8 and 9. Thus only 8 and 9 form an ECAG, and server 1 forms another.

Theorem 5. *For fat-tree, BCube and DCell$_1$, for a given server s, the path tuples from s to u and s to v are automorphic if u and v belong to the same equal-cost automorphic group.*

This implies that we only need to calculate path tuples for once from s to any node in an ECAG, and use automorphism to obtain path tuples from s to all other nodes in the same ECAG. In other words, we can further reduce the $N-1$ path tuples to just a constant for fat-tree, BCube and DCell$_1$. Table II shows the exact number of path tuples that need to be computed for each of the three topologies we consider. As DCell$_3$ already has enough servers (e.g. when p=6, DCell$_{3,6}$ can have 3.26-million servers), we only list to DCell$_3$. For the same example, we can reduce 3.26-million*(3.26-million-1) times computation of paths tuples to 903*(3.26-million-1) times.

TABLE II
THE TIMES OF COMPUTATION OF PATH TUPLE.

Topologies	**The times of computation of path tuple**
fat-tree	3
BCube$_k$	$k+1$
DCell$_1$	6 ($p>2$, p stands for the port number of a switch)
DCell$_2$	$(N-1)p(p+1)/2$
DCell$_3$	$(N-1)p(p+1)(p(p+1)+1)/2$

With failures in the network, since a DCN is large-scale and failures are not extensive, a majority of the topology is still automorphic and we can still use our algorithm to compute the affected path tuples. One simple way is that we can record all the paths during the path tuples computation of constant node pair shown in Table II for fat-tree, BCube and DCell$_1$. According to our algorithm, we can get all the paths for the left node pairs using automorphism. So the difference to compute the affected path tuples is that we need to judge whether the path passes through the failed node or link. Note that many automorphisms can be recorded in a smarter way to save space, like server 1 in Fig. 3, we only need to record the different parts. As we can construct the automorphism directly, we can even save more space by constructing the automorphism when using it instead of constructing once and saving the automorphism.

IV. EVALUATION

In this section, we perform the evaluations of Sibyl under different settings using simulations to verify its effectiveness.

A. Setup

We use fat-tree and BCube with different scales as summarized in Table III as the topologies in the simulation. The number of link failures varies from 5 to 50. The failure patterns include random failures involving both servers and switches, server failures only, and switch failures at every level of the topology.

TABLE III
TOPOLOGIES USED IN EVALUATION.

fat-tree(k)	F(40)=18000	F(60)=58500	F(80)=136000
BCube(n,k)	B(3,4)=648	B(4,4)=2304	B(5,4)=6250

In the experiments, we first compute the baseline performance according to the complete topology of the network. For each failure pattern, we then generate 10 topologies with failures by randomly choosing the components to fail. For each topology, we select the influenced paths to route around the failures randomly and repeat the performance metric calculation for 10 times. We compare the performance under failures against the baseline to obtain the *drop ratio* of each metric, and report the average value across the total 100 runs for each topology with failures. Next we will analyse the experiment results to answer four questions. 1. Whether the change of metric is different under different failure patterns? 2. Whether different performance metrics have different drop ratio under the same failure pattern? 3. Whether the result can reflect the automorphic property of DCN topologies? 4. Whether the failures which have severe impact are few so that the manager of the data center could repair them first?

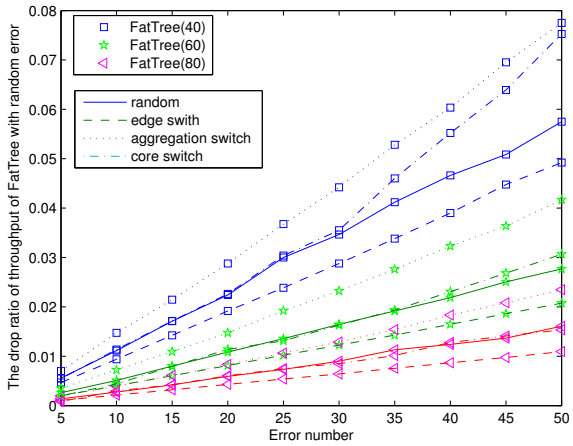


Fig. 4. The drop ratio of network throughput of FatTree

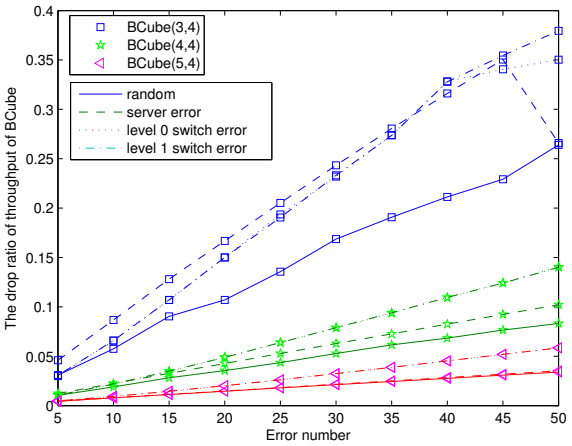


Fig. 6. The drop ratio of network throughput of BCube

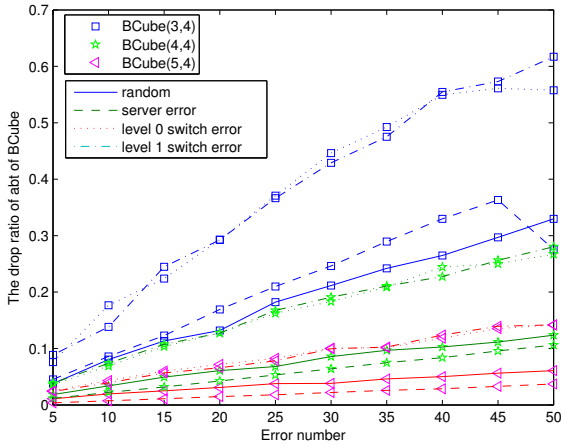


Fig. 5. The drop ratio of ABT of BCube

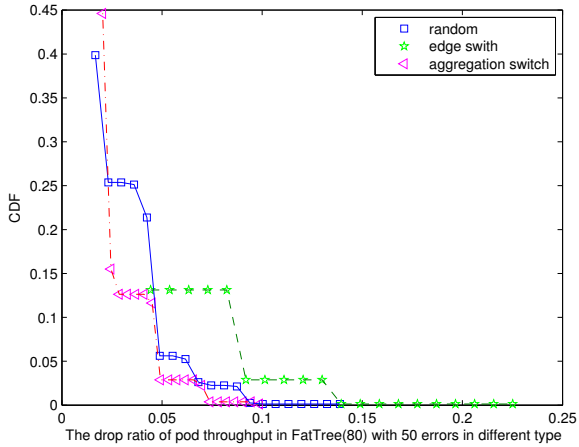


Fig. 7. CDF of the drop ratio of pod throughput in FatTree(80) with 50 errors in different types

B. Impact of failure patterns

Fig.4, 5, and 6 show the drop ratio of throughput in fat-tree and the drop ratios of throughput and ABT in BCube with different failure patterns in different scales. We can observe that under different failure patterns, the metrics all have different drop ratios. As the network scale increases, the slope of the curves which represents the relative impact of failures are quite different. The drop ratio of throughput has the fastest growth under the failure pattern of aggregation switch. It implies that when an aggregation switch fails, it has more serious impact on the network. For BCube, compared with failures in servers, failures in switches have more serious effect.

C. Various metrics

With the increase of the number of failure, the drop ratio of ABT (Fig. 5) and throughput (Fig. 6) increase almost linearly, but their growth rate is different. For example, the drop ratio of ABT and throughput of BCube(3,4) under 50 failures for level 1 switch error are almost 0.6 and 0.38 respectively. Analysing all kinds of the metrics for failure diagnosis is necessary, because we can't judge the degree of impact of failures on network performance only by the number of failures. Also, different company may pay attention to different performance metrics.

D. Failure patterns and automorphic property

For both level 0 switch failure and level 1 switch failure patterns of BCube, the curves of drop ratio of ABT and throughput are almost the same as the number of failures increases. It is because all the switches for BCube are automorphic, which means that we can also use the automorphic property of DCN topologies to reduce the number of failure cases. Conversely, the curves of drop ratio of metrics for other failure patterns like edge aggregation and core switch failures are very different, as they are not automorphic.

E. Failure patterns with severe performance impact

Fig. 7 is the CDF of the drop ratio of pod throughput in FatTree(80) with 50 errors in different failure patterns. We can find that the proportion of pods whose drop ratio of throughput is larger than 0.15 is much small. So the operator can focus on dealing with those few severe performance impact failures when designing the routing or load balancing methods.

TABLE IV
THE AVERAGE CALCULATION TIME OF BCUBE

	path tuple computing	metrics computing
BCube(3,4)	35 ms	28 ms
BCube(4,4)	466 ms	571 ms
BCube(5,4)	3.5 s	7.2 s

According to the above analysis of our results, we can find that Sibyl can be used in different topologies and deal with different failure patterns. Table IV further shows the average computation time of automorphism and the path tuples using our algorithm, and the average computation time of the metrics. We can find that the computation time is only several seconds or smaller for BCube. The computation time for other topologies is quantitatively the same and omitted here. Thus the overhead of our algorithm is mild.

V. RELATED WORK

We discuss related work in this section.

Performance monitoring and failure diagnosis: There has been much work on monitoring performance in DCNs. Most systems rely on active measurements [15, 17] that introduces overhead to the network. Sibyl only uses topology information and does not have any measurement overhead. Failure diagnosis is also an active research area. FlowDiff [3] adopts a modeling approach to identify applications and diagnose operational problems. Everflow [21] debugs faults in the network by tracing specific packets with a powerful packet filter on top of match and mirror functionality of commodity switches. Pingmesh [9] is designed to measure and analyze latency performance in a large-scale DCN. While these systems focus on diagnosis after some failures happen, Sibyl aims to better understand the performance degradation caused by failures before they happen.

Graph automorphism in DCN: Some work also applies graph automorphism in DCN research. DAC [4] and ETAC [12] for example uses graph isomorphism and induced graph isomorphism to solve the automatic address configuration problem in DCNs. To our knowledge, Sibyl is the first work that uses graph automorphism for performance impact inference with failures in DCNs.

VI. CONCLUSION

In this paper, we proposed a novel topology based performance impact inference system Sibyl for data center networks. We demonstrated that a number of network performance metrics can be inferred from just the shortest path and the maximum number of edge-disjoint paths. Utilizing the automorphism property of data center topologies (fat-tree, BCube, and DCell), we developed an efficient algorithm to compute these topological metrics that reduces the number of path tuples from $N(N-1)$ to at most constant. Simulation studies were conducted to evaluate our system under different failure scenarios in different topologies.

VII. ACKNOWLEDGEMENTS

This work is partly supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China: ECS 9048007 (CityU 21201714), GRF 9042179 (CityU 11202315), CRF C7036-15G, the NSFC (No.61272459), Program for New Century Excellent Talents in University (NCET-13-0450), and the Fundamental Research Funds for the Central Universities of China.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, 2008.
[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, 2010.
[3] A. Arefin, V. K. Singh, G. Jiang, Y. Zhang, and C. Lumezanu. Diagnosing data center behavior flow by flow. In *Proc. IEEE ICDCS*, 2013.
[4] 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 *Proc. ACM SIGCOMM*, 2010.
[5] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proc. ACM SIGCOMM*, 2011.
[6] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.
[7] 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 *Proc. ACM SIGCOMM*, 2009.
[8] 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 *Proc. ACM SIGCOMM*, 2008.
[9] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proc. ACM SIGCOMM*, 2015.
[10] C. Hu, M. Yang, K. Zheng, K. Chen, X. Zhang, B. Liu, and X. Guan. Automatically configuring the network layer of data centers for cloud computing. *IBM Journal of Research and Development*, 2011.
[11] J. Kleinberg and E. Tardos. *Algorithm Design*. 2005.
[12] X. Ma, C. Hu, K. Chen, C. Zhang, H. Zhang, K. Zheng, Y. Chen, and X. Sun. Error tolerant address configuration for data center networks with malfunctioning devices. In *Proc. IEEE ICDCS*, 2012.
[13] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proc. ACM SIGCOMM*, 2015.
[14] N. Shelly, B. Tschaen, K.-T. Förster, M. Chang, T. Benson, and L. Vanbever. Destroying networks for fun (and profit). In *Proc. ACM HotNets*, 2015.
[15] M. Shibuya, A. Tachibana, and T. Hasegawa. Efficient performance diagnosis in openflow networks based on active measurements. *ICN 2014*, page 279, 2014.
[16] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proc. ACM SIGCOMM*, 2015.
[17] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–8. IEEE, 2014.
[18] D. B. West. *Introduction to Graph Theory*. 2001.
[19] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. Netpilot: Automating datacenter network failure mitigation. *SIGCOMM Comput. Commun. Rev.*, 42(4):419–430, Aug. 2012.
[20] C. Zhang, H. Xu, and C. Hu. Performance impact inference with failures in data center networks. https://www.dropbox.com/s/fkkdkguhs2bq1me/report.pdf?dl=0, Technical Report, 2016.
[21] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng. Packet-level telemetry in large datacenter networks. In *Proc. ACM SIGCOMM*, 2015.