Agile Traffic Merging for DCNs*

Qing Yi and Suresh Singh

Portland State University, Department of Computer Science,
Portland, Oregon 97207
{yiq,singh}@cs.pdx.edu

Abstract. Data center networks (DCNs) have been growing in size and their power consumption is becoming a matter of concern. Many recent papers, including ElasticTree and CARPO, propose new near-energy-proportional DCNs, aiming at reducing the power consumption by dynamically powering off idle network switches and links. In this paper, we examine the power optimization model for DCNs, and present a scalable heuristic algorithm that finds a near-optimal subset of network switches and links that satisfies a given traffic load and consumes minimal power. Furthermore, we apply merge networks to each switch in order to power off the idle interfaces of the active switches, thus further reducing the energy consumption of active switches and achieving greater energy savings than ElasticTree. We finish by simulating large-scale fat-tree DCNs and comparing the energy cost of our techniques versus the ElasticTree method. The results demonstrate that our solution is more energy-efficient.

Keywords: Data center, routing, merging, fat-tree

1 Introduction

Data center networks (DCNs) are designed to support high communication bandwidth between servers. However, since many data centers have light loading for significant lengths of time or localized loading, large parts of these networks remain under-utilized. Networking equipment continues to consume energy even when sitting idle, and therefore contributes significantly to the overall operating costs of the data center over time. Two notable approaches have been studied to address this problem. In ElasticTree [12], the network forces traffic to the leftmost switches in a fat-tree topology to allow powering off unused switches. An orthogonal approach [16] replaces larger switches with many smaller ones in a fat-tree DCN to enable better packing of traffic into fewer switches compared with ElasticTree, hence achieving greater energy savings. Additionally, there are other approaches that primarily focus on changing link rate in response to load.

All these approaches achieve the goal of saving energy, but they have not proved to be able to adapt to changes in loading patterns efficiently. For instance, the approach proposed in [16] is static since the switch sizes and topology is fixed

* This work was funded by the NSF under award No. 1217996.

at design time. As a result, the design is only energy efficient for the specific loads that the network was designed for. Indeed, as shown in the paper, only when loads are smaller than 30% is the topology using many small switches more energy efficient than the one using large switches. ElasticTree is a more adaptive mechanism for saving energy since it computes routes every second. However, there is still considerable amount of energy wasted by switches that are powered on with light traffic on all its interfaces. Indeed, the number of such lightly loaded switches is significant, and, as a result, the overall energy savings are sub-optimal.

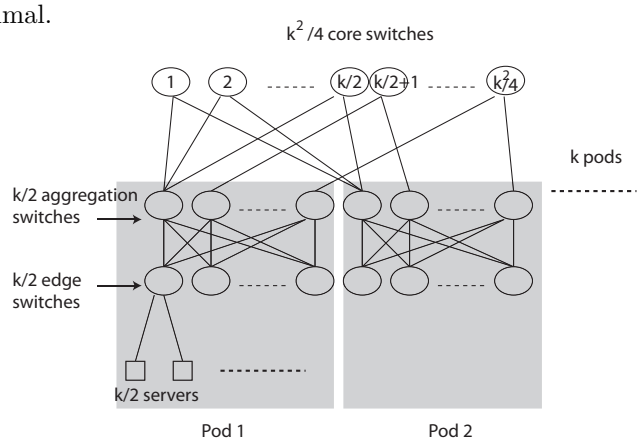


Fig. 1: Fat-tree model

To explain the deficiency of the prior approaches as well as to motivate our contribution, we consider the 3-layer fat-tree DCN shown in Fig. 1. The fat-tree network is divided into k pods, each of which has two layers of switches. The bottom layer of $k/2$ switches are called edge switches while the upper layer of $k/2$ switches are called aggregation switches. $k/2$ servers are attached to each edge switch and each edge switch is connected to each of the aggregation switches in the same pod. The leftmost aggregation switch in each pod is connected to the first $k/2$ core switches, and so on. Thus there are $k^2/4$ core switches.

In previous approaches such as ElasticTree, the $k^2/2$ edge switches are always fully powered on as they are connected to servers. Although link rate adaptation at low loads will reduce energy consumption, the reduction is only a very small fraction of the interface energy cost. At the aggregation layer, switches that are powered on in ElasticTree do not fully load their interfaces (facing the edge switches) because each interface is connected to an edge switch. Even if the edge switch has very little traffic going to the aggregation switch, the link is fully powered on but very lightly loaded. *Our contribution in this paper is to enable powering off a subset of interfaces in active switches. This is accomplished by merging traffic carefully.*

1.1 Our Approach: Merging

Consider the case of an edge switch connected to $k/2$ servers, each of which

offers a load of λ (expressed as a fraction of link rate). Then the total traffic to this switch from the servers is $k\lambda/2$. If $k = 8$, then for $\lambda \leq 0.25$, one switch interface will suffice to handle the traffic from all four servers. In other words, if there was a way to *merge* the traffic from the four servers, we could potentially power off three of the four switch interfaces connected to the servers. In a previous paper [14], we provided a design of a hardware device called *merge network*. Rather than repeating that discussion here, we provide a *functional* model of what such a network does, and then use it in the remainder of this paper. Fig. 2 shows a $\frac{k}{2} \times \frac{k}{2}$ merge network connected to $k/2$ servers on one side and to the $k/2$ ports of an edge switch on the other side.

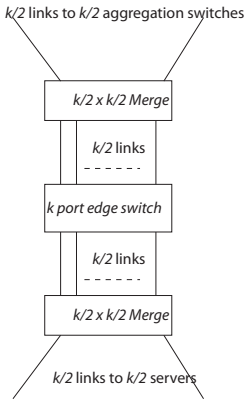


Fig. 2: Merge networks applied to a switch

1. The merge network is a fully analog device with no transceivers and, as a result, its power consumption is below one watt. The merge network can be visualized as a train switching station where trains are re-routed by switching the tracks (rather than store-and-forward).
2. Consider the uplink from the servers to the merge network. All traffic coming into the merge network is output on the *leftmost* $m \leq k/2$ links connected to the m leftmost interfaces of the switch, where $m = \lceil k\lambda/2 \rceil$ (assuming a normalized unit capacity for links). This is accomplished internally by sensing packets on links and automatically redirecting them to the leftmost output from the merge network that is free.
3. On the downlink to the servers, traffic from the switch to the $k/2$ servers is sent out along the leftmost $m \leq k/2$ switch interfaces to the merge network. The packets are then sent out along the $k/2$ links attached to the servers from the output of the merge network. The manner in which this is accomplished is described in [14] (note that the challenge is to correctly route the packets flowing through the merge network to the appropriate destinations).

To apply merge networks to a fat-tree network, we add two $\frac{k}{2} \times \frac{k}{2}$ merge networks to each edge switch as shown in Fig. 2. The connections are similar for each aggregation switch. For the core switches, we connect a $k \times k$ merge network.

1.2 Contributions and Paper Organization

In this paper, we revisit the problem of reducing energy consumption in fat-tree DCNs by attaching merge networks to each switch. In addition to the savings we obtain by forcing traffic to the left as in ElasticTree, we achieve significant

additional savings by powering off unused interfaces in active switches which is made possible by merge networks.

The remainder of the paper is organized as follows. In the next section, we present an optimization model for computing routes with the goal of minimizing energy consumption. This model is different from those developed in previous papers because we also consider merge networks and our minimization function includes the number of active interfaces as a parameter. In section 3, we present an algorithm that computes routes every second based on traffic load. The results of the optimization are compared against the simulated algorithm for a variety of loading scenarios, which show good agreement between the two. Finally, in section 4, we present the results of simulating more realistic larger fat-tree networks and analyze the energy savings obtained when using merge networks. Section 5 presents related work and section 6 summarizes the main contributions and future work.

2 Minimizing Energy Consumption

To compute the minimal power required by a DCN, we formulate a power model for all network elements including switches and links. A network $G(V, E)$ is given, where V is the set of nodes in the network and E is the set of links. We consider both the end hosts and the switches as network nodes and thus we have $V = V_1 + V_2$, where V_1 is the set of end hosts and V_2 is the set of switches. Link $(u, v) \in E$ connects node u and node v $(u, v \in V)$. Assuming each switch consumes power P_s and each link consumes power P_l, the total power consumed by the entire network can be expressed as

$$P_{total} = \frac{1}{2} \sum_{u \in V_2} k_u \times P_l + n \times P_s + \frac{\epsilon}{2} \times \sum_{u \in V, w \in V_u} f_{u,w} \tag{1}$$

where n is the number of active switches and k_u is the number of active interfaces of switch u. V_u is the set of nodes connecting to node u. ϵ is the dynamic energy consumption factor representing the power consumption per unit data transmitted through a link. $f_{u,v}$ is amount of traffic flow assigned to link (u, v). We use binary variables y_u and $x_{u,v}$ to represent the power state of node u and link (u, v), respectively. For instance, if $x_{u,v} = 1$, link (u, v) is active; if it is 0, link (u, v) is idle and can be powered off. Therefore, k_u and n can be written as

$$n = \sum_{u \in V_2} y_u \tag{2}$$

$$\forall u \in V_2, \;\; k_u = \sum_{w \in V_u} x_{u,w} \tag{3}$$

2.1 Optimization Model

Based on the power model defined above, we define an optimization problem in order to find the optimal flow assignment that involves a minimum subset

of active network elements, (n, k_u), with the minimal total power consumption P_{total} for a given network topology and a traffic load. This optimization problem is a Mixed Integer linear Programming problem (MIP), and is an extension to the capacitated Minimum-Cost MultiCommodity Flow problem (MCMCF). A classical MCMCF problem is subject to three constraints - *capacity constraint*, *flow conservation* and *demand satisfaction*, which are written as

$$\forall (u,v) \in E, \quad f_{u,v} \leq c x_{u,v} \tag{4}$$

$$\forall u,\ u \notin S \ and \ \ u \notin D, \quad \sum_{w \in V_u} f_{u,w} - \sum_{w \in V_u} f_{w,u} = 0 \tag{5}$$

$$\begin{cases} \forall s \in S, \sum\limits_{w \in V_s} g^i_{s,w} - \sum\limits_{w \in V_s} g^i_{w,s} = t^i_{s,d} \\ \forall d \in D, \sum\limits_{w \in V_d} g^i_{w,d} - \sum\limits_{w \in V_d} g^i_{d,w} = t^i_{s,d} \end{cases} \tag{6}$$

where c is the capacity for each link. S is the set of source nodes and D is the set of destination nodes. V_s and V_d is the set of switches that connect to source node s and sink node d, respectively. $f_{u,w}$ is the total flow assigned on link (u, w) and $f_{u,w} = \sum\limits_i g^i_{u,w}$, where $g^i_{u,v}$ represents the flow of the ith traffic demand $t^i_{s,d}$ routed through link (u, v).

Capacity constraint (4) takes account of maximum link utilization and ensures that the total traffic flow assigned to a link does not surpass the link capacity. The *capacity constraint* also forces flows to go through active links only. For example, inactive link (u, v) has $x_{u,v} = 0$, which causes $f_{u,v} = 0$ meaning no traffic flow is assigned to this link. *Flow conservation* (5) ensures that traffic entering an intermediate node equals to traffic exiting from it. *Demand satisfaction* (6) describes that the overall traffic departing a source node or entering a destination node equals to the traffic demand.

Besides these three constraints, the *bidirectional link* rule ensures that both directions of a link are powered on if there is a flow assigned to either direction of the link. The *bidirectional link* constraint is expressed as

$$\forall (u,v) \in E, \quad x_{u,v} = x_{v,u} \tag{7}$$

Additionally, we include constraints that correlate the power states of switches and links. For each node u and the connected links (u, w) and (w, u), we have

$$\forall u \in V,\ \forall w \in V_u,\ x_{u,w} \leq y_u \ \ and \ \ x_{w,u} \leq y_u \tag{8}$$

$$\forall u \in V, \quad y_u \leq \sum_{w \in V_u} (x_{u,w} + x_{w,u}) \tag{9}$$

Constraint (8) makes sure that a switch is powered off only when all its connected links are powered off, and constraint (9) ensures that a switch be powered off when all its connected links are powered off. Optionally, we can include a *non-splitting* constraint as follows to prevent flow splitting:

$$\forall i, \forall (u,v) \in E, \quad g^i_{u,v} = t^i \times r^i_{u,v} \tag{10}$$

where $r^i_{u,v}$ is a binary decision variable that indicates whether the traffic demand t_i is assigned to link (u, v). Constraint (10) ensures that $g^i_{u,v}$, the flow assignment to link (u, v), is either equal to the ith traffic demand t_i or equal to zero.

Furthermore, we define heuristic constraints to reduce the problem size. For example, since a k-ary fat-tree network has $5k^2/4$ switches and each switch has at most k active links, we explicitly apply an upper bound and a lower bound to k_u and n as $0 \leq k_u \leq k$ and $0 \leq n \leq \frac{5}{4}k^2$, which can greatly improve convergence time for the problem.

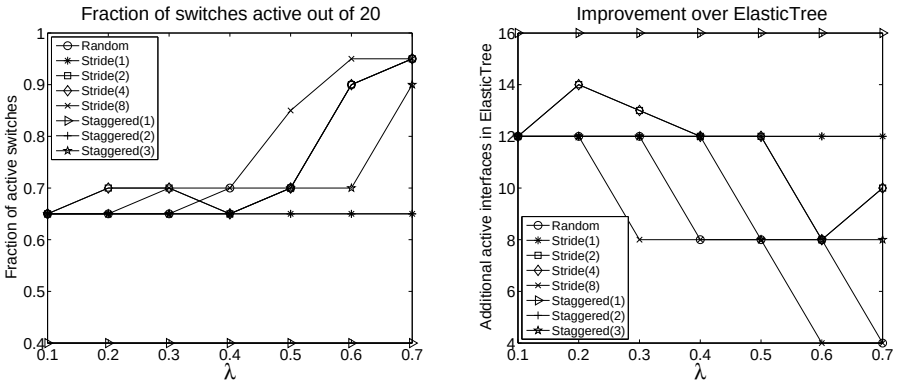
We implement the power optimization model using CPLEX, which is an optimization solver for integer programming problems. For a given traffic matrix, the optimization model outputs the numbers of active switches and links, and the flow assignment to each link corresponding to every traffic flow demand. Our model is implemented with both flow-splitting and non-flow-splitting options.

2.2 Energy Savings Due to Traffic Merging

A primary contribution of this paper is to illustrate the additional energy savings achieved by merge networks when compared with approaches such as ElasticTree. To quantify this benefit, we run the optimization problem on several different types of network loadings for a small fat-tree topology of size $k = 4$. In this topology, there are 8 edge switches, 8 aggregation switches and 4 core switches. For each edge switch, there are 2 servers connected for a total of 16 servers in 4 pods. We assume that there is a 2×2 merge network connected to either side of each edge and aggregation switch and there is a 4×4 merge network connected to each core switch.

Traffic patterns in data centers can vary greatly, and to ensure our results are widely applicable, we run the optimization algorithm on the following types of traffic: *Random*, *Stride*(n), *Staggered*(n) [4]. In *Random*, the source and destination are randomly selected from among the servers. For *Stride*(n), the destination of a flow from server i is server $[(i + n) \mod 16]$, where servers are numbered left to right as $0, 1, \cdots, 15$. For example, in a $k = 4$ fat-tree network, *Stride*(1) has almost half of the traffic goes between servers connected to the same edge switch and the other half traffic goes to aggregation and core switches. On the other hand, *Stride*(4) sends all traffic between pods, resulting in a larger number of switches to participate in forwarding traffic. The *Staggered* traffic model assigns a probability p_1 for traffic going to a server in the same subnet (i.e., connected to the same edge switch), a probability p_2 for traffic going to a server in the same pod but different subnet, and a probability $1 - p_1 - p_2$ where the flow is destined to a server in a different pod. By varying these probabilities, we can generate a large number of different loading patterns.

Fig. 3a plots the percentage of active switches for our approach as well as for ElasticTree for different loading patterns and different loads. As we have expected, the number of active switches for *Stride*(1) does not vary with λ. This is because almost all the traffic goes to the server in the same subnet or in the same pod and therefore, the active switches required are always the eight edge switches, one aggregation switch per pod and one core switch. *Stride*(8) shows



(a) Number of active switches. (b) Total number of active interfaces

Fig. 3: Difference in number of active switches and active interfaces network-wide

the highest number of active switches because all the traffic is inter-pod traffic and hence more core switches are used.

In order to illustrate the potential benefits of traffic merging, we take a difference between the total number of active interfaces when using ElasticTree and using traffic merging with the above optimization. The results, shown in Fig. 3b, clearly illustrate the benefits of merging. In the case of $Stride(1)$, ElasticTree uses 12 more interfaces than merging. The reason is that one aggregation switch is active per pod. In ElasticTree, all the four interfaces to this switch are active (albeit with very low traffic). In our approach, in contrast, we merge the traffic using a merge network and use only a single interface of the switch.

The overall energy cost of a switch can be roughly partitioned into the cost of the chassis and the cost of the interfaces. As described in [8, 16], a reasonable approximation to the cost of a switch is

$$\text{Switch Cost } = C + m \log m + m$$

where m is the number of active switch ports. The constant C accounts for static costs of a switch such as fan, etc. The second term corresponds to the cost of the interconnection fabric within the switch, which is a significant contributor to energy consumption (typically $30\% \sim 40\%$). This cost scales as $m \log m$ for a switch with m active ports. The last term is the cost contribution from the active interfaces. This term folds into itself the cost of the line cards that the interfaces are on. For the purpose of comparing the *overall cost reduction* of traffic merging relative to ElasticTree, we set C to 50% of the maximum switch cost and express it as

$$C = m_{\max} \log m_{\max} + m_{\max}$$

where $m_{\max}$ is the number of switch ports. If the traffic load fraction going to a switch is λ, the merge network will switch the traffic to the leftmost $k = \lceil \lambda m \rceil$ ports. Thus, the cost of a switch with merge networks is written as

$$\text{Traffic Merging Switch Cost } = C + k \log k + k$$

Therefore, the fraction of cost savings of traffic merging over ElasticTree is calculated as

$$\text{Cost Savings} = \frac{m \log m - k \log k + m - k}{C + m \log m + m}$$

Fig. 4 plots the fraction of reduction of network cost using traffic merging over ElasticTree. It is noteworthy that, for all traffic patterns and across all loads, the traffic merging reduces the overall energy cost even for a small-sized network consisting of 20 switches. These savings are more substantial when we consider realistic DCNs as we do later in this paper.

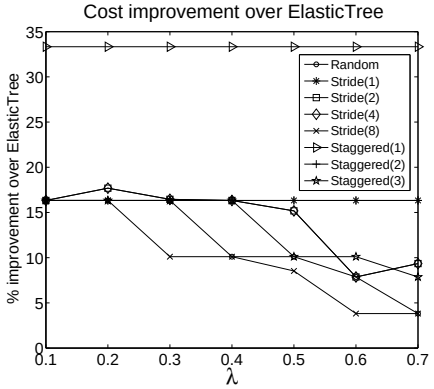


Fig. 4: Reduction in total cost when using traffic merging

3 Greedy Flow Assignment

The optimization model can find the optimal flow assignment for a given network topology and traffic loading. However, since a MCMCF problem is NP-hard, the optimization problem for a large-sized DCN cannot be solved within a reasonable time frame. To address this problem, we propose a heuristic greedy algorithm to find a near-optimal flow assignment.

3.1 Algorithm

Our greedy flow assignment algorithm is based on Dijkstra's algorithm that solves the shortest path problem. For a given network topology and a given traffic flow, our algorithm finds a route between the source node and the destination node with sufficient bandwidth and the lowest cost. We define the cost of a route as the sum of the cost of the nodes and links along the route. By carefully defining the value of the cost of each node and each link, our greedy algorithm finds the lowest-cost route for each traffic flow incrementally and ultimately obtains the optimal routing for all the traffic flows that uses the minimum number of switches and links. The greedy algorithm is described as in Algorithm 1.

Each link in the network has a *fixed capacity*. We only assign a flow to a link when there is available capacity in that link. Once a flow is assigned to a link,

Algorithm 1 Flow assignment algorithm

```
1: function FLOWASSIGN(s, d, t)
2:     for each vertex v in Graph do
3:         dist[v] ← Infinity
4:         previous[v] ← nil
5:     dist[s] ← 0
6:     insert (s, dist[s]) to Q
7:     while Q is not empty do
8:         u ← first pair in Q
9:         remove u from Q
10:        if u == d then
11:            break
12:        for each neighbor v of u do
13:            if capacity(u, v) > t then
14:                alt ← dist[u] + cost(v) +
                       cost(u, v)
15:            else
16:                alt ← Infinity
17:            if alt < dist[v] then
18:                erase (v, dist[v]) from Q
19:                dist[v] ← alt
20:                previous[v] ← u
21:                insert (v, dist[v]) to Q
22:    v ← s
23:    while v! = nil do
24:        insert v to route
25:        v ← previous[v]
26:    return route
```

the corresponding amount of capacity is subtracted from the available capacity of the link. The $cost(u, v)$ is a constant value of 2 for all links, which counts each link along the route, no matter whether it was used previously or not. Therefore, we will always find the shortest route that involves a minimum number of links. $Cost(v)$ is the cost of node v and the value is initialized as 1 for all nodes. *Once a node was used for a route once, its cost value will be updated to* 0. This will make sure that a switch that has been used in a previous route has a higher priority to be reused. As a result, we can minimize the overall number of active switches. We set higher cost for links than for switches to avoid routing loops.

3.2 Validation of Greedy Algorithm

The greedy algorithm is not optimal but, as we show below, the routes produced by the algorithm are very close to those produced by solving the optimization formulation in section 2. We use the same fat-tree topology as in section 2.2 with $k = 4$. For a given traffic load, we generate a number of packet traces following certain DCN traffic patterns [5]. The packet traces in each one-second interval are organized as a traffic matrix and is fed into the CPLEX optimization model and the simulated greedy algorithm. We obtain the number of active switches and active interfaces for the eight traffic patterns and seven traffic loads shown in Table 1.

The results we get from the simulated greedy algorithm are very close to those get from the CPLEX optimization model, especially for the lighter loads. Since the optimization model can only scale to a fat-tree DCN with $k = 6$, we use the greedy algorithm to simulate the optimization of a large-scale fat-tree network in the next part of this paper.

Table 1: Number of active switches and interfaces from optimization vs. from simulated greedy algorithm

load	Staggered(1)				Staggered(2)				Staggered(3)				Random			
	act SW		act I/F		act SW		act I/F		act SW		act I/F		act SW		act I/F	
	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim
10%	8	8	16	16	13	13	40	40	13	13	40	40	13	13	40	40
20%	8	8	16	16	13	13	40	40	13	13	40	40	13	13	40	40
30%	8	8	16	16	13	13	40	40	13	13	40	40	14	14	48	48
40%	8	8	16	16	13	13	40	40	13	13	40	40	14	14	48	48
50%	8	8	16	16	13	13	40	40	14	14	48	47.2	14	14	48	48
60%	8	8	16	16	13	13	40	40	14	14	48	53.4	18	19	64	72
70%	8	8	16	16	13	13	40	40	18	19	64	72	19	19	72	72

load	Stride(1)				Stride(2)				Stride(4)				Stride(8)			
	act SW		act I/F		act SW		act I/F		act SW		act I/F		act SW		act I/F	
	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim	opt	sim
10%	13	13	40	40	13	13	40	40	13	13	40	40	13	13	40	40
20%	13	13	40	40	13	13	40	40	13	13	40	40	13	13	40	40
30%	13	13	40	40	13	13	40	40	14	14	48	48	13	14	40	44
40%	13	13	40	40	13	13	40	40	14	14	48	48	14	14	48	48
50%	13	13	40	40	17	17	58	56.2	17	17	60	60.8	17	17	60	63.6
60%	13	13	40	40	18	18	64	64	19	19	72	72	19	20	72	75.2
70%	13	13	40	40	19	18	66	64	19	19	72	72	19	20	72	75.6

4 Simulation Results

We simulate a $k = 12$ fat-tree network which supports 432 servers and 180 12-port switches. In this network, there are 12 pods and each of which has six edge switches and six aggregation switches. We assume that each of the core switches has extra ports to be connected to external Internet through border routers. We assign 1Gbps capacity to each link. We experiment with synthetic traffic data from a traffic generator and real packet traces from a university data center. *Since flow splitting will incur packet reordering cost, which is not a desirable practice in real data centers, we implement our simulation using non-splitting flow assignment.*

4.1 Synthetic Traffic Data

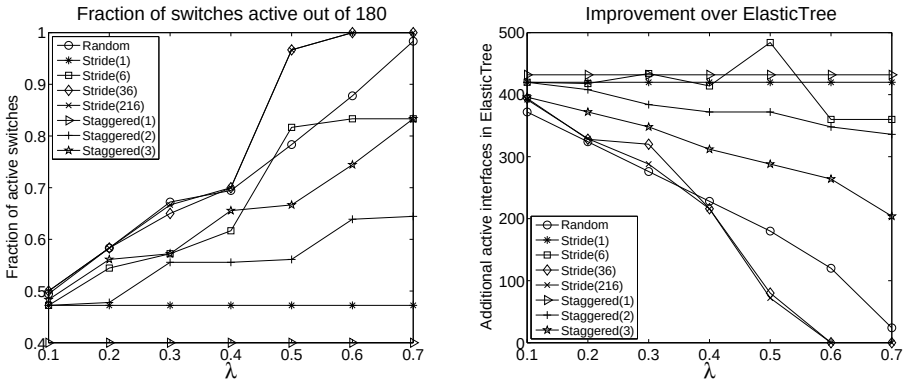
We generate network traffic following ON/OFF patterns derived from many production data centers [5,6]. The duration of the ON and OFF periods and the packet interarrival time follow the lognormal distribution. Like in Section 2.2, we study traffic patterns *Random*, *Stride*(n) and *Staggered*(n). In a $k = 12$ fat-tree network, every edge switch is connected to six servers. For *Stride*(1), flows sourcing from the first five servers of the edge switch go to servers in the same subnet, and flows from the sixth server travel to the server in the next subnet or in the next pod. In contrast, all the flows in *Stride*(6) go to the neighboring subnet, and all the traffic in *Stride*(36) and *Stride*(216) is inter-pod traffic. For

Staggered(n), it has fixed values for p_1 and p_2 as the probabilities of flow going to the same subnet and other subnets of the same pod, respectively. Table 2 shows these values for all traffic suites studied.

Table 2: Probabilities of flows going to the same subnet (p_1), to other subnets in the same pod (p_2), and to different pods ($1 - p_1 - p_2$) for all traffic suites studied.

Traffic Suite	p_1	p_2	$1-p_1-p_2$	Traffic Suite	p_1	p_2	$1-p_1-p_2$
Staggered(1)	100%	0%	0%	*Stride(1)*	83.3%	13.9%	3%
Staggered(2)	50%	30%	20%	*Stride(6)*	0%	83.3%	16.7%
Staggered(3)	20%	30%	50%	*Stride(36)*	0%	0%	100%
Random	1.2%	7%	91.8%	*Stride(216)*	0%	0%	100%

The load fraction λ offered by each server varied from 0.1 to 0.7. Our simulation outputs the number of active switches (Fig. 5a) and the number of active interfaces of each switch with varies traffic loads and patterns. In general, the number of active switches increases with the traffic load. However, both *Stride*(1) and *Staggered*(1) have constant number of active switches and active interfaces. This is because, for *Stride*(1), all loads can be satisfied by using a minimum spanning tree. For *Staggered*(1), only edge switches are used since all the traffic flows are local traffic within the same subnet.



(a) Number of active switches. (b) Total number of active interfaces.

Fig. 5: Number of active switches and active interfaces network-wide for a $k = 12$ fat-tree network

Fig. 5b illustrates the difference of total numbers of active interfaces of a DCN using merge networks versus ElasticTree. It shows that more interfaces of the active switches become idle when the traffic is light, which demonstrates that traffic merging can save more energy with lighter traffic (Fig. 6). *Stride*(1) achieves the most energy savings over ElasticTree (around 42%) because, for each active edge switch, the energy consumed by the five idle interfaces is wasted.

Staggered(1) saves 30% energy consumption since for the entire network, only half of the interfaces (facing the severs) of the edge switches are used.

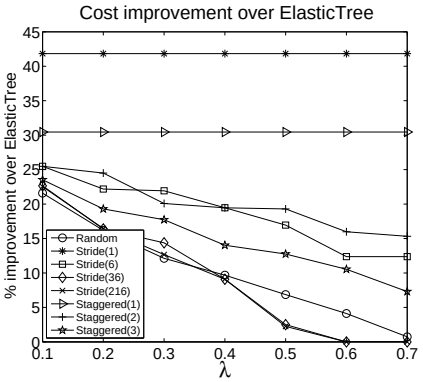


Fig. 6: Reduction in total cost when using traffic merging

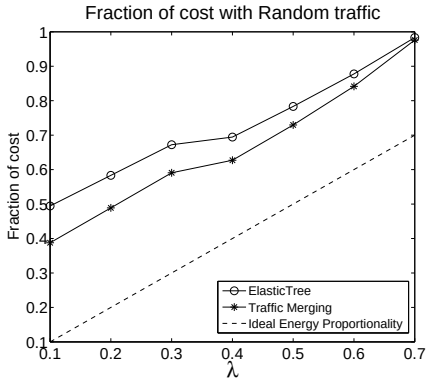


Fig. 7: Compare the total cost of ElasticTree and traffic merging

ElasticTree provides an energy-efficient solution for DCNs. However, the drawback of ElasticTree is that, a DCN still consumes a large amount of power with light load [12]. In contrast, as shown in Fig. 7, our approach reduces energy consumption when the network is lightly loaded, which demonstrates that traffic merging achieves better energy proportionality than ElasticTree.

4.2 Empirical Traffic Data

We use packet traces from a university data center published by Benson *et al.* [5]. This university data center has about 500 servers providing services for campus users. 60% of the traffic is for Web services and the rest is for other applications such as file sharing services. Traffic traces are captured by a sniffer installed at a randomly selected switch in the data center. Fig. 8 illustrates the total load of the packet traces within 50 minutes. The overall load is very small for a high-bandwidth fat-tree topology. We observe that power cost decreases from 30% to 17% when applying merge networks compared with ElasticTree (Fig. 9).

5 Related Work

The development of Internet communication and service applications requires increased bandwidth support and more powerful routing protocols for a DCN. For the past few years, many new DCN topologies have been proposed, including fat-tree [4], Clos [7] and flattened butterfly [13]. These hierarchical interconnection topologies are designed to maximize cross-section bandwidth and optimize the cost-effect ratio. Alternatively, some server-centric DCN architecture, such as DCell [10], BCube [9] and CamCube [2], use simple switches and push network routing to the servers, thus obtaining better scalability and fault-tolerance.

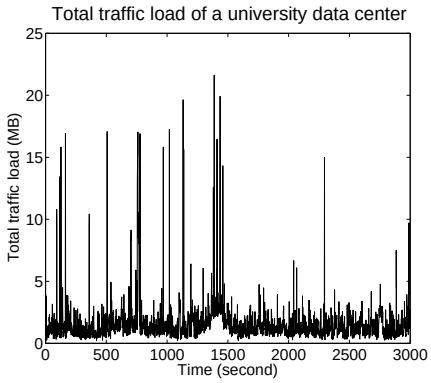


Fig. 8: Traffic load of a university data center

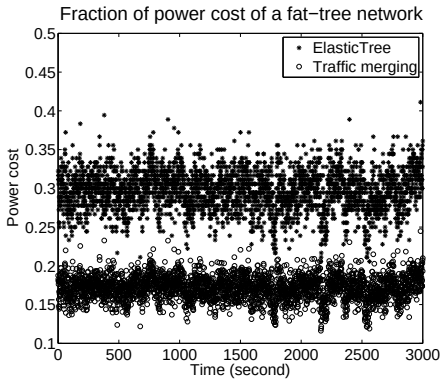


Fig. 9: Energy savings when using traffic merging

In general, these proposed network topologies are intended to support increasing number of servers and provide high capacity for bandwidth-hungry services. However, the rising energy cost of DCNs has attracted the attention of many researchers. New designs of energy-efficient network devices have been examined. For example, Abts *et al.* [1] explore dynamically tuning the link rate according to traffic intensity to save energy. Inspired by the earlier work of Gupta *et al.* [11], other researchers propose energy-proportional DCN topologies through powering off idle interfaces or devices. For example, Heller *et al.* proposed ElasticTree [12] that adapts the network topology to varying traffic loads. *CARPO* [15] examines the dynamic topology by consolidating timely-negative-correlated flows into a smaller set of links and shutting off unused ones. More recently, Adnan and Gupta propose an online path-consolidation algorithm to right-size network dynamically [3]. Widiaja *et al.* [16] compare the energy savings of optimizing fat-tree networks deployed with different sizes of switches and conclude that, with the same number of servers, it is more energy efficient to use more smaller-sized switches than using less large-sized switches when the traffic is highly localized.

Our work complements prior work by utilizing a universal greedy flow assignment algorithm to find the optimal network subset. The greedy bin-packing algorithm used in ElasticTree leverages the regularity of hierarchical DCNs and uses left-most heuristics to find the shortest route. Our greedy algorithm can find flow assignments close to the MIP model, for not just hierarchical network topologies, but also random or irregular DCN topologies. Furthermore, we apply merge networks to each switch and scale switch energy cost to the number of busy interfaces of each switch.

6 Conclusions

This paper addresses the power optimization problem of DCNs. We present a greedy algorithm that is applicable to all types of DCN topologies. We demonstrate that this algorithm can find near-optimal flow assignments comparable to

solutions achieved from optimization model. In addition, by applying merge networks to each switch, we further reduce power consumption of active switches. With very light load, our approach saves 20% $\sim$ 40% energy cost compared with ElasticTree, depending on the traffic types. Traffic with small number of inter-pod and inter-subnet flows can benefit even more from traffic merging. In the future, we will apply merge networks to switches in different ways to explore methods that further reduce energy consumption of DCNs.

References

1. Abts, D., Marty, M. R., Wells, P. M., Klausler, P., and Liu, H. Energy Proportional Datacenter Networks. In *ISCA* (2010).
2. Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., and Donnelly, A. Symbiotic Routing in Future Data Centers. In *SIGCOMM* (2010), pp. 51–62.
3. Adnan, M. A., and Gupta, R. Path Consolidation for Dynamic Right-Sizing of Data Center Networks. In *Proceedings IEEE Sixth International Conference on Cloud Computing* (2013).
4. Al-Fares, M., Loukissas, A., and Vahdat, A. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM* (2008), pp. 63–74.
5. Benson, T., Akella, A., and Maltz, D. A. Network Traffic Characteristics of Data Centers in the Wild. In *IMC* (2010).
6. Benson, T., Anand, A., Akella, A., and Zhang, M. Understanding Data Center Traffic Characteristics. In *WREN* (2009).
7. Clos, C. A Study of Non-Blocking Switching Networks. *The Bell System Technical Journal 32*, 2 (March 1953), 406–424.
8. Eramo, V., Germoni, A., Cianfrani, A., Miucci, E., and Listanti, M. Comparison in Power Consumption of MVMC and BENES Optical Packet Switches. In *Proceedings IEEE NOC (Network on Chip)* (2011), pp. 125–128.
9. Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *SIGCOMM* (2009), pp. 63–74.
10. Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *SIGCOMM* (2008), pp. 75–86.
11. Gupta, M., and Singh, S. Greening of the Internet. In *Proceedings of ACM SIGCOMM* (2003).
12. Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., and McKeown, N. ElasticTree: Saving Energy in Data Center Networks. In *NSDI* (2010).
13. Kim, J., Dally, W. J., and Abts, D. Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks. In *ISCA* (2007), pp. 126–137.
14. Singh, S., and Yiu, C. Putting the Cart Before the Horse: Merging Traffic for Energy Conservation. In *IEEE Communications Magazine*. June 2011.
15. Wang, X., Yao, Y., Wang, X., Lu, K., and Cao, Q. CARPO: Correlation-Aware Power Optimization in Data Center Networks. In *INFOCOM* (2012), pp. 1125–1133.
16. Widjaja, I., Walid, A., Luo, Y., Xu, Y., and Chao, H. J. Switch Sizing for Energy-Efficient Datacenter Networks. In *Proceedings GreenMetrics 2013 Workshop (in conjunction with ACM Sigmetrics 2013)* (Pittsburgh, PA, June 2013).