Willow: Saving Data Center Network Energy for Network-Limited Flows

Dan Li, Yirong Yu, Wu He, Kai Zheng, *Senior Member, IEEE*, and Bingsheng He

Abstract—Today's giant data centers are power hungry. Data center energy saving not only helps control the operational cost, but also benefits the sustainable growth of cloud services. Due to the adoption of much more switches in modern data centers as well as the mature server-side power management techniques, energy saving for the data center network is becoming increasingly important. Most previous works on saving data center network energy focus on aggregating flows to as few switches as possible. However, in this paper we argue that this method may not work for network-limited flows, the throughputs of which are elastic based on the competing flows. To save the network energy consumed by this kind of elastic flows, we propose a flow scheduling approach called *Willow*, which takes both the number of switches involved and their active working durations into consideration. We formulate this problem by programming and design a greedy approximate algorithm to schedule flows in an online manner. Simulations based on MapReduce traces show that Willow can save up to 60 percent network energy compared with ECMP scheduling in typical settings, and outperforms other classical heuristic algorithms such as simulated annealing and particle swarm optimization. Testbed Experiments demonstrate that this kind of dynamic energy-efficient flow scheduling causes negligible impact on upper-layer applications.

Index Terms—Data center network, energy efficiency, flow scheduling

1 Introduction

Data centers are built around the world to run large-scale distributed computations, such as scientific calculation, batch processing and cloud based service. Modern data centers are quite power hungry [1]. As an important citizen for the operating cost of data centers, energy consumption is even demonstrated to surpass the equipment cost [2], [3]. Hence, energy cost now becomes a headache for many data center owners. Energy saving in data centers is important not only for economical and environmental reasons, but also for the sustainable growth of cloud computing, due to the challenges imposed by power delivery to and heat removal from giant data centers.

Energy consumption in data centers goes to cooling, power distribution, IT equipments, etc. Usually, power usage effectiveness, or PUE, is used to measure the efficiency of the data center power distribution. PUE is defined as the total facility power over the IT equipment power [15]. The lower the PUE is, the more power is delivered to IT equipments. In the past years, researchers and engineers are struggling to reduce the PUE of data centers. In April 2011, the Open Compute Project of Facebook announced a PUE of 1.07 [4]. It indicates that 93.5 percent of the data center facility power can be delivered to the IT equipments. Hence, for energy saving in modern and future data centers, it is more important to reduce the energy consumption of IT equipments, than that of cooling or power distribution parts.

As for power management on IT equipments, past efforts have been primarily spent on the server side. However, the networking side also occupies 10$\sim$23 percent of the total data center power consumption [6], [34], [35], which cannot be neglected. If the mature power saving techniques on the servers are employed [5], the proportion of network power can even be up to 50 percent [33]. Moreover, the technical trend of data center design is to use much more switches to provide high bisection bandwidth [9], [16], [28], which will further increase the network energy. In order to reduce the energy consumption of data center network, previous works studied how to use aggregate flows into a minimal number of switches, while letting the other switches "sleep on idle" [6], [29], [33].

However, in this paper, we challenge that although the widely-used flow aggregation method works for saving energy of application-limited flows, it does not work for network-limited flows. For a network-limited flow [8], the application generates traffic at a very fast speed and the throughput of the flow depends on the network capacity of the routing path as well as the number of competing flows in the bottleneck link of the path. As a result, although flow aggregation reduces the number of switches involved, it on the other side leads to lower throughputs for the flows and thus increases the active running duration of the switches. Therefore, saving the network energy of network-limited flows cannot be simply translated as minimizing the number of involved switches or maximizing the flows' throughputs. Instead, there should be a balancing point in choosing the routing paths of the flows, so as to minimize the overall network energy footprint.

We design *Willow*, which schedules network-limited flows for energy saving by leveraging the controlled

- *D. Li, Y. Yu, and W. He are with the Computer Science Department, Tsinghua University, Beijing 100084, China. E-mail: tolidan@tsinghua.edu.cn, {yyr2046, hewu34}@126.com.*
- *K. Zheng is with the IBM China Research Lab, Beijing 100094, China. E-mail: zhengkai@cnibm.com.*
- *B. He is with the School of Computer Science and Engineering, Nanyang Technological University, Nanyang Ave, Singapore. E-mail: bshe@ntu.edu.sg.*

Manuscript received 9 Apr. 2014; revised 18 July 2014; accepted 14 Aug. 2014. Date of publication 21 Aug. 2014; date of current version 7 Aug. 2015. Recommended for acceptance by K. Wu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2350990

environment of data centers and the emerging software defined network (SDN) technique. The SDN controller collects the flows' size and deadline information to determine the best routing paths for the flows in terms of energy saving, and periodically updates the routes to accommodate flow dynamics, topology dynamics as well as inaccurate estimation. We modelize the scheduling problem in a multi-path data center network (e.g., Fat-Tree, BCube) and formulate it by programming, with consideration of the flow deadline constraints. In order to schedule flows in an online manner, we design a greedy approximation algorithm and optimize it in Fat-Tree network. The computation complexity of the algorithm in Fat-Tree network is $O(x * m)$, where x is the number of elephant flows to schedule, and m is the number of core switches in Fat-Tree.

Based on the realworld MapReduce workloads, we conduct simulations to evaluate Willow. Simulation results show that the online greedy approximation algorithm in Willow can save up to 60 percent network energy compared with traditional ECMP scheduling, in a container-sized Fat-Tree data center network running tens of thousands of elephant flows. The time required by the greedy approximate scheduling is less than 1 second on a commodity server, and it generally outperforms the other two heuristic algorithms in terms of network energy saving. In order to study the impact of dynamic flow scheduling on the application's performance, we implement Willow on SDN controller and run experiments in a small test bed. The results show that this kind of flow scheduling algorithm acts so quickly that the dynamic forwarding entry updates causes negligible impact on upper-layer applications.

The rest of this paper is organized as follows. Section 2 states the problem. Section 3 describes the algorithm design. Sections 4 and 5 evaluate our flow scheduling algorithm by simulations and experiments, respectively. Section 6 discusses the related work. Finally, Section 7 concludes the paper.

2 Network Energy of Network-Limited Flows in Data Centers

In this section we first introduce the background of power management of data center switches and data center network topology, and then state the problem of saving network energy consumed by network-limited flows in data centers.

2.1 Power Management of Data Center Switches

Generally speaking, the power consumption of a switch in data center can be expressed as $P_{idle} + f(b)$, where P_{idle} is the power consumed at the idle state, and $f(b)$ is the power consumed when traffic rate is at b. Measurements have shown that P_{idle} is the dominant contributor [17]. Though rate adaptation, i.e., changing the operating rate of the switch ports according to the traffic load, can save energy, its saving is moderate compared to putting the entire switch into sleep. Besides, existing hardware lacks support for smooth rate adaptation on-the-fly and port-level sleeping [18]. Hence in this paper we assume that the switch power consumption is almost constant during its active working period, and focus on saving energy by putting the whole switches into sleeping mode when none of its ports carry any traffic. Note that the energy consumed in sleeping state is almost negligible.

We assume that a switch can go to sleep when idle by centralized configuration or sleep-on-idle (SoI) technique, and a sleeping switch can be waken up when used for forwarding packets, either by centralized management or wake-on-arrival (WoA). Both the sleeping process and waking process take some time to transit. Usually a switch can go to sleep very quickly, but waking up may take a relatively longer time, depending on the type of the forwarding card and the amount of forwarding information needed to load. Modern hardware is capable of supporting SoI and WoA [27], but not implemented in switches, partly due to the lack of an effective power-aware management system in the network. Throughout this paper, we just assume that switches can go to sleep or wake up based on the local traffic states, and do not take the transition time into account in evaluation.

2.2 Data Center Network Topology

In recent years many advanced data center network architectures are proposed to increase the network capacity. These architectures can be divided into two categories, namely, switch-centric and server-centric. The former puts the interconnection and routing intelligence on switches, while the latter brings data center servers into the networking part and gets them involved in packet forwarding.

The switch-centric architecture proposals either use a totally new switch infrastructure to replace the tree structure, or make enhancements upon the tree to improve the bisection bandwidth. Fat-Tree [9] and VL2 [10] both use low-end, commodity switches to form a three-layer Clos network. Each server still uses one NIC port to connect an edge-level switch. Both the two architectures can provide an oversubscription ratio of 1:1 to all the servers in the network. The difference of VL2 from Fat-Tree is that higher-speed switches, e.g., those with 10GE ports, are used in higher levels of the Clos network to reduce the wiring complexity.

In server-centric data center architectures, each server uses multiple NIC ports to join the network infrastructure and participate in packet forwarding. In FiConn [16], a recursive, level-based structure is designed to connect servers via mini-switches and dual server NIC ports. FiConn not only eliminates the necessity to add server NIC ports during data center expansion, but also reduces the wiring cost. BCube [11] targets at building a data center container, typically with 1k$\sim$4k servers. BCube is also a recursive structure. Each server uses multiple ports to connect different levels of switches. The link resource in BCube is so rich that 1:1 oversubscription ratio is guaranteed.

2.3 Network Energy Consumed by Network-Limited Flows

In this work we focus on the network energy consumed by network-limited flows in data centers. For this kind of flows, the application generates traffic at a very fast speed. The throughput a flow gets depends on the routing path it chooses and the number of competing flows in the bottleneck link. Therefore, path selection for network-limited flows will affect both the number of active switches involved and the working duration of the switches. Assume the set of switches in the network is S, the power of an active switch s is P_s ($s \in S$), and the active working duration for a switch s is t_s ($s \in S$), the overall energy consumption of the network is

$$E = \sum_{s \in S} P_s * t_s. \tag{1}$$

If the data center network is built from homogeneous switches, i.e., the power consumption of each switch is the same, say, P, Eq. (1) can further be simplified as

$$E = P * \sum_{s \in S} t_s. \tag{2}$$

We call $\sum_{s \in S} t_s$ from Eq. (2) the *aggregate working duration* of the network.

We further assume the flow deadline, D_f, and the flow size, Z_f, can be obtained before choosing the routing path for a flow f. Flow deadline is assigned by application, e.g., the time required to return the result for an online search query. When selecting the routing path, we need to guarantee that the flow can meet its deadline, otherwise the transmission of the flow is useless [37]. The flow sizes can also be initiated in advance by many applications, e.g., the reading or writing of data chunks in distributed file systems [38], or data shuffling in MapReduce computations [13], [14]. Even if the flow size cannot be precisely determined, applications can usually provide a good estimate of it before the running time [37]. Based on the flow size and the expected throughput a flow can get, we can roughly estimate the task finish time of a flow before running it. Specifically, assume the set of flows is F, and the throughput of f is h_f, we should guarantee

$$\frac{Z_f}{h_f} \leq D_f, \quad \forall f \in F. \tag{3}$$

Although many existing energy-efficient flow scheduling schemes in data center networks focus on minimizing the number of switches involved or maximizing the network throughput, we will show that they do not work for network-limited flows by the following examples.

Fig. 1 shows the scenario in part of a multi-rooted tree network. There are two flows, i.e., flow $f1$ from server 1 to server 3, and flow $f2$ from server 2 to server 4. Assume the sizes of the two flows are both Z, the bandwidth capacity of each link is B, the power consumption of each switch is P, and the deadlines for the two flows are both $\frac{2*Z}{B}$. For the example in Fig. 1a, the scheduling algorithm, which maximizes the network throughput, generates the routing paths in the left figure i.e., $f1$ taking the path of $h \rightarrow e \rightarrow i$ while $f2$ taking the path of $h \rightarrow g \rightarrow i$. Four switches are used and their active working durations are $\frac{Z}{B}$. Hence, the total network energy consumption is $\frac{4*Z*P}{B}$. Meanwhile, the right figure shows the routing paths generated by the scheduling algorithm minimizing the number of used switches, both $f1$ and $f2$ taking the path of $h \rightarrow g \rightarrow i$. Three switches are involved; but the two flows share the same path, and the active working duration becomes $\frac{2*Z}{B}$. As a result, the overall network energy consumption is $\frac{6*Z*P}{B}$. Note that both the scheduling methods meet the flow deadlines, but the scheduling algorithm which maximizes the network throughput is more energy efficient in this example.

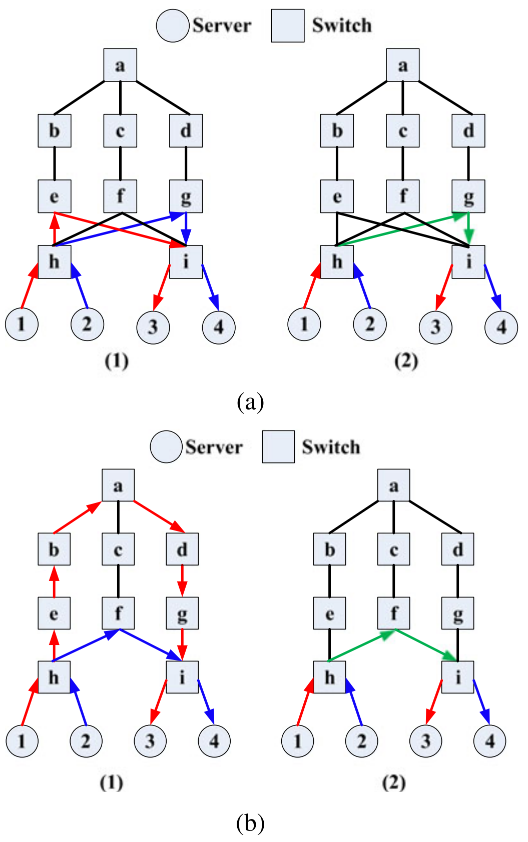

Fig. 1. Flow $f1$ is from server 1 to server 3, and flow $f2$ is from server 2 to server 4. (1) Two flows take different paths to maximize the network throughput. (2) Two flows take the same path to minimize the number of switches used. In example (a), (1) is more network energy efficient; while in example (b), (2) is more network energy efficient.

Then we check another example in Fig. 1b. The network topology is similar as example (a), except there is link failure in $h - g$ and $e - i$. The left figure shows the flow scheduling maximizing the network throughput, i.e., $f1$ taking the path of $h \rightarrow e \rightarrow b \rightarrow a \rightarrow d \rightarrow g \rightarrow i$ while $f2$ taking the path of $h \rightarrow f \rightarrow i$. It gets the active working duration of $\frac{Z}{B}$, but eight switches are used. So the network energy consumption is $\frac{8*Z*P}{B}$. The flow scheduling algorithm minimizing the number of switches shown in the right figure, in which both the flows take the path of $h \rightarrow f \rightarrow i$, results in three involved switches and an active working duration of $\frac{2*Z}{B}$. Hence, the overall network energy consumption is $\frac{6*Z*P}{B}$. In this case, we find that the flow scheduling which minimizes the number of switches consumes less network energy than the one maximizing the throughput.

From the two examples, we conclude that existing flow scheduling algorithms may not work well for minimizing the network energy of network-limited flows. Instead, to save network energy of this kind of elastic flows, we need to take both the number of switches involved and the active working durations of switches into account.

3 Willow Design

In this section we design Willow, an energy-efficient flow scheduling approach for network-limited flows in data center networks.

3.1 Basic Idea

SDN based flow scheduling. Willow leverages the SDN framework and works in a centralized fashion. The SDN controller collects the flow information including the flow deadlines and sizes, maintains the data center network topology (including switch/link failures), computes the routing path for each flow, and configures the forwarding entries on switches. In order to accommodate flow dynamics, topology dynamics as well as inaccurate estimations, the controller re-schedules flows periodically.

Routing path selection. The design of Willow does not depend on the topology of the data center network. Usually, a richly-connected data center network has multiple equal-cost paths between any pair of servers, which leaves high flexibility to the routing path selection algorithm for optimizing a certain metric. For instance, in Fat-Tree there are $\frac{k^2}{4}$ equal-cost paths between two servers in different pods, where k is the number of switch ports in the network. In BCube, there are k disjoint paths between any two servers, where k is the number of ports in a server. If considering paths with overlapping nodes, there are even more candidates to choose.

Differentiation between elephant flows and mice flows. For network-limited flows, both elephant flows and mice flows exist. Except for the data-intensive flows which exchange large volumes of data, there are also background flows with light traffic to transmit, e.g., the MapReduce protocol messages between masters and workers. It has been shown that the small number of elephant flows dominate the traffic in data center networks [8], [10]. Therefore, to control the scheduling complexity, we focus on scheduling the elephant flows with relatively large amount of data to transmit. For mice flows, we just randomly reuse the paths assigned to elephant flows. In what follows we discuss how Willow schedules elephant flows on the SDN controller for network energy saving.

3.2 Programming Solution

We formulate the problem of energy-efficient scheduling of network-limited flows by programming. The constants, programming variables, intermediate variables, constraints and targets are shown as follows.

Constants:

S: set of switches.
P: set of all the possible paths.
L: set of links in the topology.
F: set of flows to assign.
B_l: bandwidth of link $l \in L$
Z_f: traffic size of flow $f \in F$.
D_f: deadline of flow $f \in F$.
α_{sl}: is 1 when a link l is connected to switch s, 0 otherwise
β_{lp}: is 1 when a link l is in path p, 0 otherwise

Programming Variables:

y_{fp}: is 1 if $f \in F$ assigned to path p, 0 otherwise

Intermediate Variables:

t_s: time of a switch $s \in S$ to run
γ_l: time of link $l \in L$ to run
n_l: number of flows in a link $l \in L$

Constraints:

$$\forall f \in F: \quad \sum_p y_{fp} = 1 \tag{1}$$
$$\forall s \in S, l \in L: \quad t_s \geq \alpha_{sl} * \gamma_l \tag{2}$$
$$\forall l \in L: \quad \gamma_l * B_l \geq \sum_p \sum_f Z_f * y_{fp} * \beta_{lp} \tag{3}$$
$$\forall l \in L: \quad n_l = \sum_p \sum_f y_{fp} * \beta_{lp} \tag{4}$$
$$\forall f \in F, l \in L: \quad D_f * \frac{b_l}{n_l} \geq \sum_p \beta_{lp} * y_{fp} * Z_f \tag{5}$$

Target:

Min$(\sum_s t_s)$

Constraint (1) ensures that each flow $f \in F$ is assigned to exactly one path, so as to keep ordered delivery for the same flow. Constraint (2) ensures that each switch is actively running if any of its links is running. Constraint (3) ensures that each link is running before all the flows assigned to it finish transmission. Constraint (4) and (5) satisfy the flow deadline requirements.

Since the formulation covers all the possible flow scheduling solutions, the programming output is considered to be optimal, and plays as a benchmark for any flow scheduling algorithm in terms of network energy saving. However, it is a mixed integer programming, and we prove it is NP-hard as follows.

Proof. Finding the flow routes in an irregular network while not exceeding the capacity of each link is called the multi-commodity flow (MCF) problem, which is NP-complete for integer flows [24]. To prove the scheduling problem for network-limited flows in our model to be NP-hard, we construct a sequence of sub-problems, and show that they can be reduced to the famous MCF problem.

Let $P_0(F, D, Z, G)$ denotes the original problem, where F is the set of network-limited flows, D is the set of flow deadlines, Z is the set of flow size and G is the input topology. First, we reduce P_0 to a sub-problem $P_1(\overline{F}, D, Z, G)$ by adding a constrain that each host holds no more than one flow, i.e., a host can be the source of at most one flow. Second, we assume the deadlines and flow sizes of all flows to be exactly the same. Thus, we have a sub-problem $P_2(\overline{F}, \{\overline{d}\}, \{\overline{z}\}, G)$. Let $\overline{r} = \frac{\overline{z}}{\overline{d}}$. In this sub-problem, the average rate for each flow should not be lower than $\overline{r}$ to meet the deadline. Third, we constrain all the edge links connected to the hosts to be the same capacity of $\overline{r}$. Here, we have $P_3(\overline{F}, \{\overline{d}\}, \{\overline{z}\}, \overline{G})$, which limits the maximum rate of each flow. In P_3, the flow rate of each flow is forced to be exactly $\overline{r}$ and remains unchanged until it finishes. A feasible solution for P_3 should also be feasible for the $MCF(\overline{F}, \{\overline{r}\}, \overline{G})$ problem and vice versa. That is to say, both problems are equivalent. As a result, the original scheduling problem of network-limited flows P_0 is proved to be NP-complete. Since P_0 is not a NP problem, then we find that it is NP-Hard. □

In our experiment, it takes a few minutes for the programming to get the optimal solution for hundreds of flows. Obviously, the cost of programming is too high for an ideal online flow scheduling approach, which handles tens of thousands of flows within seconds. Hence, we turn to some heuristic methods for a more practical and efficient scheduling algorithms.

3.3 Classical Heuristic Schedulings

We first consider using classical heuristic algorithms to online schedule flows in Willow, such as genetic algorithm [23], simulated annealing [21] and particle swarm optimization [22]. The key idea of these algorithms is to heuristically search from the global solution space and step-by-step improve the result. The performance of the search depends on the time consumed. We apply all the three algorithms into our problem, but find that the genetic algorithm takes extremely long time to get a reasonable result. Hence we focus on the simulated annealing and particle swarm optimization algorithms.

The simulated annealing [21] is inspired by annealing in metallurgy. In our problem, the flow scheduling solution can be regarded as a molecule in metal, whose kinetic energy is related to the temperature of the metal. There is an initial solution and its energy consumption. Then simulated annealing repeatedly changes a flow's routing path to generate the neighboring solution, with flow-deadline guarantee. If the neighboring solution is better in energy saving, it is accepted; otherwise, it is accepted with a low probability.

The particle swarm optimization [22] is inspired by predation of birds. In our case, each particle represents a flow scheduling solution and the algorithm will maintain the amount of particles (generally 30 particles) to find the optimal solution. At the beginning, each particle randomly searches the solution space. During the searching process, every particle records its best previous location with flow-deadline guarantee, and all particles record a global best location. Each particle uses a specified formula to determine its searching direction, velocity and location. Note that in both simulated annealing and particle swarm optimization, a solution is skipped if it violates the deadline requirements of the flows.

However, our study shows that both simulated annealing and particle swarm optimization algorithms are subjected to the scale of search space, and the quality of the solution is non-deterministic. It is also time consuming for the algorithms to get satisfactory energy saving ratio (refer to Section 4), which cannot meet the online requirement for flow scheduling. We thus turn to a more efficient approximation algorithm.

3.4 Willow Scheduling Algorithm

A major challenge for the heuristic solutions above is the selection of the initial solution. To make it worse, all the flows in our model are TCP flows, i.e., the traffic rate of a flow is subjected to the bottleneck link and thus depends heavily on the other flows. Because of the vast solution space and the complicated correlation among different flows, there's no way to find a good initial state within reasonable time. In fact, a randomly chosen initial solution is likely to be far from the optimal point. Also, heuristic methods may be captured in local optimal solutions.

We then turn to a greedy approximation approach in Willow scheduling. The key insight is to make full utilization of the abundance of ports in each switch, and thus, aggregating all the flows onto a subset of the given topology with as few switches as possible, without increasing the network running duration. To achieve high multiplexing of the switches, we take an incremental approach to generate a solution, i.e., scheduling all the flows one after another. For each flow, we examine all the candidate paths and choose one that minimizes the increment in the total working time of all switches. This practice is far more efficient than the heuristic approaches, because it starts with an ideal initial state, i.e., no switches in use, and chooses the best paths at each step, which keeps it close to the optimal solution. What's more, our approach doesn't have to calculate the cost of the overall network at each step. We just need to compute the increment after one flow is scheduled, which makes this greedy approach outperforms the other heuristics in terms of computation overhead.

Input: F (set of flows to schedule), P_f (set of candidate paths for flow $f \in F$)
Output: W_f (candidate path assigned to flow $f \in F$)

```
01: foreach flow $f \in F$ with ascending order of deadline
02:     $\Delta T_{min} \leftarrow$ INFINITY;
03:     $p_{assigned} \leftarrow \phi$;
04:     foreach $p \in P_f$
05:         $W_f \leftarrow p$;
06:         if deadlines of affected flows are violated
07:             continue;
08:         $\Delta T \leftarrow$increment in running time of all switches;
09:         if $\Delta T = 0$
10:             break;
11:         else if $\Delta T < \Delta T_{min}$
12:             $\Delta T_{min} \leftarrow \Delta T$;
13:             $p_{assigned} \leftarrow p$;
14:         end if
15:         $W_f \leftarrow \phi$;
16:     end for
17:     if $W_f = \phi$
18:         $W_f \leftarrow p_{assigned}$;
19:     end if
20: end for
```

Fig. 2. The framework of Willow scheduling algorithm.

Fig. 2 shows the pseudocode of the Willow framework. First, we sort all the flows in an ascending order in terms of the flow deadlines (line 1), since the flows with short deadlines have lower flexibility in selecting the routing paths. If an assignment violates the deadlines of affected flows (the flows sharing links with the concerned flow), the candidate path is skipped (line 6-7). From all the candidate paths that can satisfy the flow deadlines, we choose the one which minimizes the increment in aggregate working duration of the current switches, ΔT (line 11-13, line 18). For acceleration, we terminate the search for this flow if finding a candidate path which does not increase any duration compared with the previous round (line 9-10). If no candidate path can meet the deadline requirement of a flow, the algorithm will not schedule the flow ($p_{assigned} = \phi$).

We use Fig. 3 as an example to demonstrate the working process and effectiveness of Willow scheduling. In the example network, there are 16 servers and 4 flows to schedule in a Fat-Tree network. Flow $f1$ is from server $v0$ to $v8$, flow $f2$ is from server $v2$ to $v10$, flow $f3$ is from server $v4$ to $v12$, and flow $f4$ is from server $v6$ to $v14$. The network capacity of every link is 1 Gbps. Each flow has the equal size of 1 GB and the deadline of 8 seconds. Note that in a Fat-Tree topology,

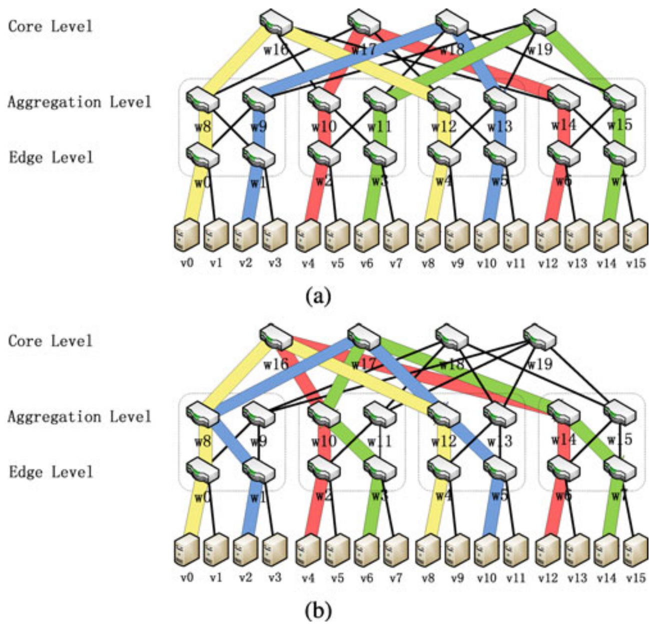

Fig. 3. ECMP flow scheduling (a) and energy efficient flow scheduling (b). The link capacity is 1 Gbps and the flow size is 1 GB. The aggregate working durations of switches are 160 and 112 seconds for the two algorithms, respectively.

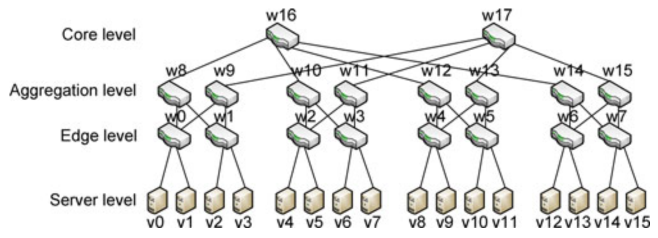

Fig. 4. A blocking Fat-Tree architecture with oversubscription ratio of 2:1.

the path between hosts of different pods is decided by the core-level switch selection. If we use traditional ECMP scheduling for Fig. 3, it is highly probable that every flow is assigned with a different core-level switch, which is shown in Fig. 3a. Hence, all the 20 switches will be used to carry the flows, and every switch will run 8 seconds. The aggregate working duration of the switches is 160 seconds.

However, if we run Willow scheduling in Fig. 2, the process is as follows. For each flow, four candidate paths with different core-level switches are available. First, for flow $f1$, the path with the first core-level switch $w16$ is assigned. Then we consider flow $f2$. If the core-level switch $w16$ is assigned to $f2$, the two flows will have to share the links $(w8, w16)$, $(w16, w12)$, and the deadline requirements are violated. As a result, the core-level switch $w17$ is chosen for $f2$. As for flow $f3$, all the four core-level switches can meet the flow deadlines, but $w16$ results in the least aggregate working duration, because it multiplexes the links in $w16$. Similarly, flow $f4$ is assigned with core-level switch $w17$. By the energy efficient flow scheduling, we finally use only two core-level switches and four aggregation-level switches, with each switch running 8 seconds as well. The total number of switches involved is thus 14 and the aggregate working duration of the switches is 112 seconds. Therefore, our approach can save 30 percent network energy compared with ECMP scheduling(160 seconds) in this example.

3.5 Optimizing Willow in Specific Data Center Network

As is shown in Fig. 2, Willow scheduling can be applied to any generic topology, only if the candidate paths for each flow are given. Considering the diversity of data center topologies and the various routing mechanism, in this work, we focus on one of the most important kinds of data center structure, namely, Fat-Tree. We will study the performance of our approximation on the Fat-tree topology, including the blocking Fat-Tree. To take advantage of the regular characteristic of Fat-Tree, we make some improvement based on the pseudocode in Fig. 2, which significantly reduces the computation time.

First, to reduce the time of path testing, we classify the candidate paths into different equivalent classes, and test only one path of each class. For example, in the case of Fig. 3, when scheduling flow $f1$ from $v0$ to $v8$, the path $v0 - w0 - w8 - w16 - w12 - w4 - v8$ (the yellow path) is equivalent to $v0 - w0 - w8 - w17 - w12 - w4 - v8$, since they only differ in the core-level switches and both of these two switches are the same in working time. The cost of selecting both paths are the equal. Thus, testing one path of the equivalent class is enough. Second, to reduce the computation time for the incremental network energy, we record the states of each switch and each flow, including the current working duration, the number of flows going through each port, the number of competitors on bottleneck links for each flow, etc. In this way, we only have to check the impacted switches, saving the useless checkout on other irrelevant switches. Third, we use early termination strategy, i.e., if the intermediate sum of incremental cost of impacted switches already exceeds the ever recorded minimum increment, we terminate this test instantly and turn to the next candidate path. All these optimization efforts lead to a speed-up by over 100X and our approach can thus be able to handle an input of 20k flows within 1 second in Fat-Tree network (refer to Section 4).

4 SIMULATION

In this section we carry out simulations to evaluate the effectiveness of Willow in saving network energy for network-limited flows.

4.1 Simulation Setup

Network topology. We use both Fat-Tree and blocking Fat-Tree as the representative network topology. The capacity of each link is 1 Gbps. For *blocking Fat-Tree*, we proportionally remove some core-level switches from the correspondent Fat-Tree structure. For instance, a blocking Fat-Tree with oversubscription ratio of 2:1 uses only half of the core-level switches than a correspondent Fat-Tree, as shown in Fig. 4.

Workload traces. We collect the MapReduce computation trace from a testbed data center, which has 50 mappers and 20 reducers. In order to measure the performance under more general cases, we use the flow size distribution of the real trace to synthesize other MapReduce patterns, given that the MapReduce computation model is highly structured.

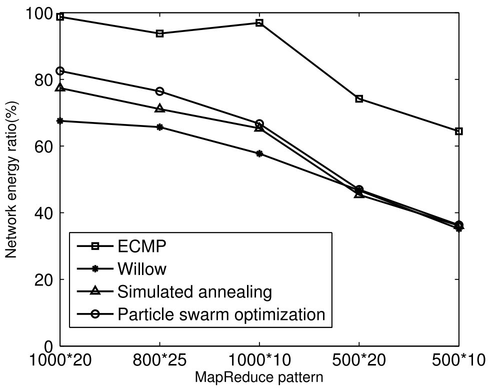

Fig. 5. Network energy ratios of the four flow scheduling algorithms against the MapReduce pattern.

Evaluation metrics. We run the greedy approximated flow scheduling used in Willow as well as the two heuristic scheduling algorithms, i.e., simulated annealing and particle swarm optimization. To accelerate these two heuristics, we bundle those flows, which share the same source and destination edge switches, for path selection. For comparison, we also run ECMP scheduling which enables sleep-on-idle. We define *network energy ratio* of an energy-aware flow scheduling algorithm as the total network energy consumed by the scheduling algorithm over that of default routing without SoI. We further compare the network energy saving by Willow with that of offline programming solution.

Running time. We run all the flow scheduling algorithms on a server installed with AMD Opteron (tm) Processor 4176 HE 2.4G CPU*12 and 32 GB DRAM. Since it takes 1 second to complete the flow scheduling for the largest set of flows we test (20,000 flows) by Willow, we run simulated annealing and particle swarm optimization for 1 second, too. We know that these two heuristic algorithms perform better when searching more from the solution space. But by running the two algorithms for longer time, we find that the improvement is marginal and longer search time does not fit the online scheduling requirement.

4.2 MapReduce Pattern

We use a Fat-Tree network composed of 16-port switches, which has 1,024 servers and fits the size of a containerized data center. We then vary the MapReduce patterns to evaluate the impact of both the number of flows and the skewness between mappers and reducers. More specifically, we set the number of mappers and reducers in each application as 1,000*20, 800*25, 1,000*10, 500*20, 500*10, respectively. The total number of (elephant) flows thus varies from 20,000, 10,000 to 5,000. We evenly place the mappers and reducers onto all the physical servers. Then we run the four flow scheduling algorithms and calculate their network energy saving ratios as shown in Fig. 5.

From the figure we find that Willow scheduling, simulated annealing and particle swarm optimization can all save network energy compared with ECMP, in particular the Willow scheduling algorithm. Generally when there are less flows in the network, the network energy ratio is lower, because more switches have opportunities to sleep. Simulated annealing and particle swarm optimization work particularly well with less flows, because they also have more

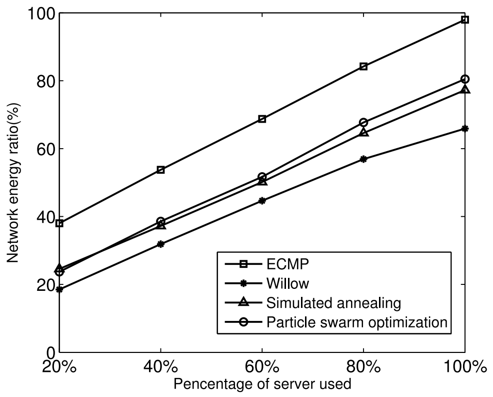

Fig. 6. Network energy ratios of the four flow scheduling algorithms against the percentage of physical servers used for MapReduce computation.

chances to search for a better solution from the smaller solution space. In the 1000*20 and 1000*10 patterns, ECMP scheduling uses almost all the switches and barely saves any network energy. In the 500*10 pattern, the greedy approximation in Willow can even save more than 60 percent network energy compared with ECMP.

4.3 Percentage of Servers to Place Flows

Then we use the 1,000*20 MapReduce pattern, and choose different numbers of physical servers to place the mappers and reducers. The Fat-Tree network is also constructed by 16-port switches. We vary the percentage of physical servers used over the total number of physical servers from 20 to 100 percent. Mappers and reducers are evenly placed onto physical servers available. Fig. 6 shows the network energy ratios of the four flow scheduling algorithms.

Generally speaking, the network energy ratios of all the four flow scheduling algorithms increase with more servers used to transmit flows. Greedy approximation in Willow enjoys the least network energy ratio while ECMP scheduling has the highest. When the percentage of servers used is 20 and 100 percent, Willow greedy approximation can save about 50 and 40 percent network energy compared with ECMP scheduling, respectively.

4.4 Data Center Size

We then test the impact of data center size on the network energy ratio. We use the Fat-Tree topology composed by 8-port, 16-port, 24-port, 32-port, and 40-port switches, respectively. Hence, the number of servers in the data center is 128, 1,024, 3,456, 8,192 and 16,000, respectively. We run the MapReduce task with 2,000*10 pattern, and the mappers and reducers are evenly placed onto all the physical servers. The network energy ratios of the four scheduling algorithms are shown in Fig. 7.

We can observe that the network energy ratio in each algorithm decreases in larger networks, because more switches will have opportunities to sleep given the same workload. Consistent with the previous simulations, the greedy approximation in Willow exposes the lowest network energy ratio among the four, while ECMP scheduling has the highest. In small-sized network such as eight-port and 16-port networks, ECMP scheduling almost uses all the switches all the time, and no network energy can be saved.

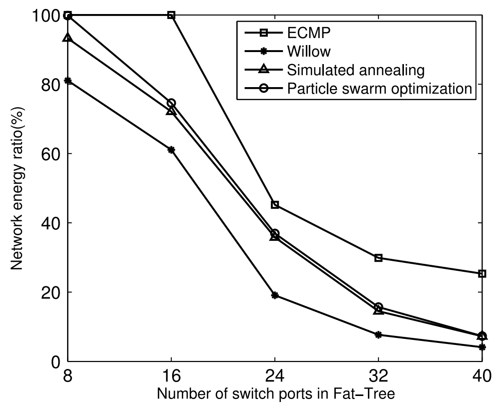

Fig. 7. Network energy ratios of the four flow scheduling algorithms against the data center size.

4.5 Blocking Fat-Tree

Though data center network architectures with non-blocking network capacity such as Fat-Tree are more and more widely applied in data centers, there are still many data center networks with oversubscriptions. We proportionally remove the core-level switches in a Fat-Tree network with 16-port switches to build blocking Fat-Trees, as shown in Fig. 4. We then run MapReduce task with 1,000*20 pattern in blocking Fat-Tree networks with oversubscription ratios from 2:1 to 8:1, respectively. Fig. 8 shows the result.

We get the following observations from the simulation. First, compared with ECMP scheduling, the other three algorithms can save much more network energy in networks with higher oversubscription ratios, among which Willow performs the best. Second, in some cases (oversubscription ratio of 2:1), the network energy ratios of simulated annealing and particle swarm optimization are even higher than that of ECMP scheduling, which is an indication that these two heuristic algorithms may be unstable sometimes, because of undetermined searches from the global solution space within the limited time. Third, in Willow, the network energy ratio first decreases from 2:1 to 4:1, then increases from 4:1 to 5:1, and decreases again with even higher oversubscription ratios. The reason is as follows. With the increase of the oversubscription ratio, the absolute network energy consumed by the default routing increases due to more traffic congestion and thus more active working duration. When the oversubscription ratio changes from

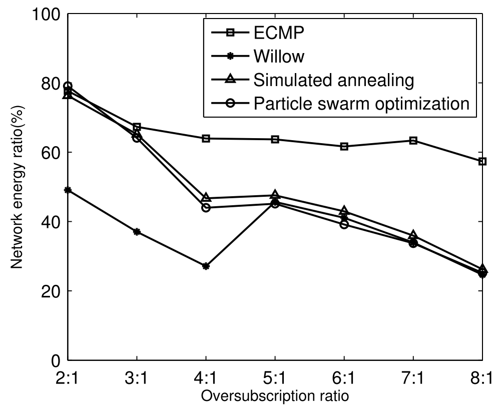

Fig. 8. Network energy ratios of the four flow scheduling algorithms against the oversubscription ratio of the Fat-Tree network.

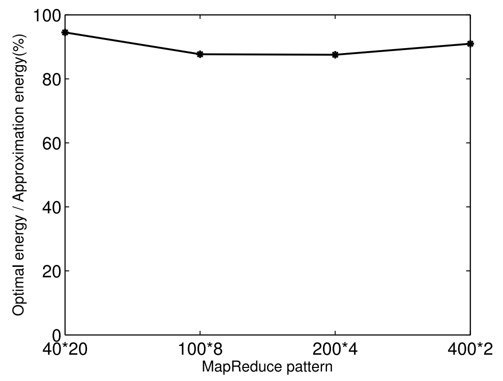

Fig. 9. Comparison of the network energy ratios of Willow greedy approximation and optimal solution.

2:1 to 4:1, there is no traffic congestion in Willow, and thus the network energy ratio decreases. But when the oversubscription ratio becomes 5:1, Willow also experiences traffic congestion, which results in much higher absolute network energy and also higher network energy ratio relative to the default routing. But if the oversubscription ratio gets even higher, Willow scheduling can save more network energy.

4.6 Comparison with Offline Optimization

Our offline optimal solution by programming in Section 3.2 plays as a benchmark to evaluate the heuristic algorithms. Since it takes too long time to get the solution by CPlex, we can only test small-scale networks with a small number of flows. We use the Fat-Tree network composed of 8-port switches (128 servers in total), and run MapReduce tasks with patterns of 40*20, 100*8, 200*4, and 400*2, respectively. Fig. 9 shows the proportion of the energy consumption by offline optimization over that of Willow scheduling. In all the cases we test, the proportion is higher than 90 percent. For the 40*20 pattern, it is about 95 percent. It implies that the online greedy approximated flow scheduling used in Willow can achieve high approximation to the offline optimization in practice.

5 Testbed Experiments

Energy-efficient flow scheduling requires dynamically updating the forwarding entries in switches. In order to measure the impact of online forwarding table update on the upper applications, we implement the Willow scheduling on the SDN controller and conduct experiments in a Fat-Tree test bed composed of 16 servers. The flow size and flow deadline are obtained from application controllers, e.g., MapReduce Master. The network topology is the same as Fig. 3. Due to the lack of SDN/OpenFlow switches, we use regular layer-2 switches conducting destination MAC based forwarding. But the forwarding tables of the switches are configured by the controller as the Willow scheduling algorithm requires. All the servers are Desktops with AMD Athlon(tm) II X2 245 2.91G CPU and 2 GB DRAM. The link speeds in all switches are 1 Gbps. To decouple the network performance from disk, we use iperf to generate 4 flows as in Fig. 3. The SDN controller first uses ECMP algorithm to schedule the 4 flows for 10 seconds, and then switches to Willow scheduling. The routing paths selected by the two

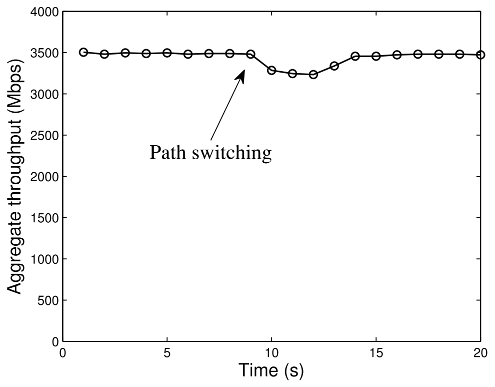

Fig. 10. Aggregate throughput of the network in the whole process. Before $t = 10$ s, the SDN controller uses ECMP scheduling. From $t = 10$ s on, the SDN controller switches to Willow scheduling. There is no obvious throughput degradation during the switching.

scheduling approaches are exactly the same as Figs. 3a and 3b, respectively. We evaluate the aggregate throughput of the network during the whole process, and the result is shown in Fig. 10.

The figure tells that although Willow uses less core-level switches to carry the flows, the aggregate throughput is not less than ECMP scheduling. Therefore, Willow saves network energy without performance compromise. When Willow switches the routing paths at $t = 10$ s, there is slight throughput degradation. This is because in our experiment setting, the switch does not support direct update of the forwarding table. We have to first delete the old forwarding entry and then add the new ones. During the interval, packets are forwarded by flooding and the aggregate throughput is thus affected. If employing OpenFlow switches, the path switching process is expected to be more smooth.

6 Related Work

In this section we discuss the related work, including the flow scheduling algorithms for data center networks, greening of the Internet, as well as greening of data centers.

6.1 Flow Scheduling Algorithms for Data Center Networks

Many flow scheduling algorithms are proposed in data center networks, with different optimization goals.

Maximizing network throughput. In Fat-Tree network [9], a centralized flow scheduling approach is mentioned to minimize the overlap of large flows. In Hedera [8], a scalable, dynamic flow scheduling system is developed to adaptively schedule the flows in Fat-Tree network in order to efficiently utilize the aggregate network resources.

Minimizing the number of switches. In Elastic Tree [6], a limited number of core-level switches in Fat-Tree network are dynamically used to accommodate the network traffic rates, so as to save network energy. Both greedy bin-packing algorithm and topology-aware algorithm are developed for the optimization. Shang et al. propose an energy-aware flow scheduling algorithm in data center networks [29], to minimize the number of switches used to carry the flows, with performance guarantee.

The flow scheduling in Hedera [8] is not for energy saving. Elastic tree [6] targets at application-constrained flows, the traffic rates of which are fixed. But our flow scheduling algorithm deals with saving network energy of network-limited flows, which considers both the number of used switches and the active working durations of the switches.

6.2 Greening of the Internet

Gupta and Singh first argue that it is necessary to suitably put network interfaces, routers and switches to sleep for energy saving along with the trends of technology and energy development [25]. After that, many works emerge for saving energy in the Internet. Nedevschi et al. suggest reducing network energy consumption via putting network components into sleep or adjusting the link rates to accommodate the traffic status [26]. Cianfrani et al. modify the OSPF protocol for saving network energy in intra-domain networks [30]. The basic idea is to coordinate the shortest-path tree built on the routers, in order to use less links to carry traffic. Zhang et al. put forward power-aware traffic engineering mechanism. The problem is abstracted as a mixed integer programming problem, and heuristic algorithms are developed [31]. Vasic et al. propose identifying energy-critical paths in the Internet based on the historical traffic, and use other links in an on-demand manner [39].

Our work optimizes the energy consumption of data center networks. One fundamental difference of data center networks from the Internet is that the network environment is controlled. We can depend on centralized SDN framework and application inputs to design smarter energy efficient solutions.

6.3 Greening of Data Centers

The energy consumption of data centers in recent years is so high that researchers are actively designing power-aware mechanisms. The consistent theme is to let the power consumption of data centers be proportional to the computation/traffic load. Lin et al. propose an online *lazy capacity provisioning* algorithm to dynamically adjust the number of active servers in data centers for a certain load, which shows great promise in energy saving by real workloads [32]. Fan et al. investigate the energy consumption of Google data centers, and suggest saving energy by both CPU voltage/frequency scaling and improving the non-peak power efficiency [3]. Abts et al. design low-power, highly-scalable data center topologies, and exploit the link's dynamic range for energy saving [33]. There are also proposals for greening the data center networks by flow scheduling, as presented in Section 6.1.

In our work, we make a special focus on the networks, by identifying the trend that the energy consumption ratio of the networking part in data centers is increasing. We do not depend on any hardware based energy saving techniques. Instead, we use software based flow scheduling to save the network energy. We design the algorithm for representative topologies in modern data centers, without introducing any new data center architectures.

7 Conclusion

In this paper, we design a flow scheduling algorithm for saving the network energy of network-limited flows in data centers. The key contribution is to consider both the number of used switches and the active running duration of the switches for network energy saving, and uses on an SDN based approach to schedule the flows. We formulate the problem by programming, and bring forward an online approximate algorithm to solve it. Simulation results show that our online approximate algorithm saves up to 60 percent network energy compared with ECMP scheduling, performing close to the offline optimization. Testbed experiment also demonstrates that the dynamic flow entry update causes negligible impact on upper-layer applications.

Acknowledgments

The work was supported by the National Key Basic Research Program of China (973 program) under Grant 2014CB347800 and 2012CB315800, the National Natural Science Foundation of China under Grant No. 61170291, No. 61133006, No. 61161140454, and the National High-tech R&D Program of China (863 program) under Grant 2013AA013303.

References

[1] U. S. Environmental Protection Agency. Data Center Report to Congress [Online]. Available: http://www.energystar.gov, 2007.

[2] J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale services," in *Proc. 4th Biennial Conf. Innovative Data Syst. Res.*, 2009, pp. 1–8.

[3] X. Fan, W. Weber, and L. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. 34th Annu. Int. Symp. Comput. Archit.*, 2007, pp. 13–23.

[4] Open Computer Project [Online]. Available: http://perspectives.mvdirona.com/2011/04/07/OpenComputeProject.aspx, 2011.

[5] L. Ye, G. Lu, S. Kumar, C. Gniady, and J. H. Hartman, "Energy-efficient storage in virtual machine environments," in *Proc. 6th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2010, pp. 75–84

[6] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, p. 17.

[7] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992*, IETF, 2000.

[8] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, p. 19.

[9] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.

[10] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[11] 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 Conf. Data Commun.*, 2009, pp. 63–74.

[12] T. Hoff. (2007, Jul.). Google architecture [Online]. Available: http://highscalability.com/google-architecture

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Oper. Syst. Des. Implementation*, 2004, p. 10.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.

[15] PUE and Total Power Usage Efficiency (tPUE). James Hamilton's Blog [Online]. Available: http://perspectives.mvdirona.com/2009/06/15/PUEAndTotalPowerUsageEfficiencyTPUE.aspx, 2009.

[16] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, and J. Wu, "Scalable and cost-effective interconnection of data-center servers using dual server ports," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 102–114, Feb. 2011.

[17] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "A power benchmarking framework for network devices," in *Proc. 8th Int. IFIP-TC 6 Netw. Conf.*, 2009, pp. 795–808,.

[18] G. Ananthanarayanan and R. Katz, "Greening the switch," in *Proc. Conf. Power Aware Comput. Syst.*, 2008, p. 7.

[19] R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 39–50.

[20] *Jeff Dean on Google Infrastructure*. James Hamilton's Blog. [Online]. Available: http://perspectives.mvdirona.com/2008/06/11/JeffDeanOnGoogleInfrastructure.aspx, 2008.

[21] M. Fleischer, "Simulated annealing: Past, present, and future," in *Proc. IEEE 27th Conf. Winter Simul.*, 1995, pp. 155–161.

[22] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[24] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proc. 16th Annu. Symp. Comput. Sci.*, 1975, pp. 184–193.

[25] M. Gupta and S. Singh, "Greening of the Internet," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2003, pp. 19–26.

[26] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proc. 5th USENIX Symp. Netw. Syst. Des. Implementation*, 2008, pp. 323–336.

[27] R. Hays. (2008). Active/Idle Toggling with Low-Power Idle [Online]. Available: http://www.ieee802.org/3/az/public/jan08/hays_01_0108.pdf

[28] Y. Gong, B. He, and D. Li, "Finding constant from change: Revisiting network performance aware optimizations on iaas clouds," in *Proc. Int. Conf. High Performance Comput., Netw., Storage Anal.*, 2014.

[29] Y. Shang, D. Li, and M. Xu, "Energy-aware routing in data center networks," in *Proc. 1st ACM SIGCOMM Workshop Green Netw*, 2010, pp. 1–8.

[30] A. Cianfrani, V. Eramo, M. Listanti, M. Marazza, and E. Vittorini, "An energy saving routing algorithm for a green OSPF protocol," in *Proc. IEEE Conf. Comput. Commun.*, 2010, pp. 1–5.

[31] M. Zhang, C. Yi, B. Liu, and B. Zhang, "GreenTE: Power-aware traffic engineering," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2010, pp. 21–30.

[32] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proc. IEEE Conf. Comput. Commun.*, 2011, pp. 1098–1106.

[33] D. Abts, M. Marty, P. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 338–347.

[34] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 68–73, Jan. 2009.

[35] Where Does Power Go?. (2008, Jan.) [Online]. Available: http://www.greendataproject.org/

[36] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of datacenter traffic: Measurements and analysis," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf.*, Nov. 2009, pp. 202–208.

[37] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 50–61, 2011.

[38] S. Ghemawat, H. Gobio, and S. Leungm, "The google file system," in *Proc. 19th ACM Symp. Oper. Syst. Principles*, 2003, pp. 29–43.

[39] N. Vasić, P. Bhurat, D. Novakovic, M. Canini, S. Shekhar, and D. Kostić, "Identifying and using energy-critical paths," in *Proc. 7th Conf. Emerging Netw. Exp. Technol.*, 2011, p. 18.

Dan Li received the PhD degree in computer science from Tsinghua University in 2007. He is currently an associate professor in the Computer Science Department of Tsinghua University, Beijing, China. His research interest includes future internet architecture and data center networking.

Yirong Yu received the BS degree from the Beijing University of Post and Telecommunication in 2008. He is currently working toward the master's degree in the Computer Science Department of Tsinghua University, China. His main research interest include computer networks, especially energy-aware networking and software defined networking.

Wu He received the BS degree from the Department of Computer Science and Technology, Beijing Normal University in 2012. Currently, he is a graduate student in Beijing Normal University and a visiting graduate student in Tsinghua University. His research interests include data center network and network virtualization.

Kai Zheng received the BA degree in electronic engineering from the Beijing University of Posts and Telecommunications, China, in 2001 and the MS and PhD degrees in computer science from Tsinghua University, China, in 2003 and 2006, respectively. He joined IBM Research in July 2006, as a research staff. His current research interests include software defined networking, cloud networking, and network security. He is a senior member of the IEEE.

Bingsheng He received the bachelor's degree from Shanghai Jiao Tong University and the PhD degree from the Hong Kong University of Science and Technology, both in computer science, in 2003 and 2008, respectively. He is currently an assistant professor in the Division of Networks and Distributed Systems, School of Computer Engineering of Nanyang Technological University, Singapore. His research interests include high-performance computing, distributed and parallel systems, and database systems.