This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

Saving Energy in Partially Deployed Software Defined Networks

Huandong Wang, Yong Li, *Member, IEEE,* Depeng Jin, Pan Hui, *Senior Member, IEEE,* and Jie Wu, *Fellow, IEEE*

Abstract—As power consumption of the Internet has been growing quickly in recent years, saving energy has become an important problem of networking research, for which the most promising solution is to find the minimum-power network subsets and shut down other unnecessary network devices and links to satisfy changing traffic loads. However, in traditional networks, it is difficult to implement a coordinated strategy among the network devices due to their distributed network control. On the other hand, the new networking paradigm – software defined network (SDN) provides us an efficient way of having a centralized controller with a global network view to control the power states. As an emerging technology, SDNs usually coexist with traditional networks at present. Therefore, we need to investigate how to save energy in partially deployed SDNs. In this paper, we formulate the optimization problem of finding minimum-power network subsets in partially deployed SDNs. After proving the problem is NP-hard, we propose a heuristic solution to approach its exact solution. Through extensive simulations, we demonstrate that our heuristic algorithm has a good performance; that is, on average we can save about 50% of total power consumption in the full SDN, having a distance less than 5% of the exact solution's power consumption. Moreover, it also achieves good performance in the partially deployed SDN, on average saving about 40% of the total power consumption when there are about 60% SDN nodes in the network. Meanwhile, it runs significantly faster than a general linear solver of this problem, by reducing the computation time of the network containing hundreds of nodes by 100x at least.

Index Terms—Saving energy, partially deployed SDN, minimum-power network, mixed integer programming.

1 Introduction

There have been increasing concerns about the energy crisis in recent years, considering that most of our major energy resources are non-renewable and are expected to be depleted in the not-too-distant future. An early study showed that depletion times for oil, coal, and gas reserves are approximately 35, 107, and 37 years old respectively [2]. Meanwhile, since energy we use today mainly comes from the burning of fossil fuel, large power consumption means releasing large amounts of Green House Gases (GHG), such as carbon dioxide, which leads to global warming and brings various negative effects. On the other hand, power consumed and GHG producted by the Internet have been growing fast in recent years. In 2007, the total footprint of the Internet was 0.83 Gton CO_2, about 2% of global GHG emissions, with a compound annual growth rate of 6% [1]. Meanwhile, energy efficiency of the Internet is very low; that is, energy consumption is high compared to the amount of carried traffic in the network. For example, if transmitting data cross–country (New York to San Diego is approximately 4480 km), the wireless link is between 2x to 3x times more efficient than that of the Internet [6]. The reason is that energy consumed by network devices such as switches is not in proportion to the utilization of its links. Generally speaking, turning on a switch consumes most of its power, and getting the utilization of a link from idle to full only consumes an extra 8% of power [5], [8]. Moreover, the link utilization of the Internet is not very high most of the time. For example, the average link utilization in backbone networks of large Internet service providers is about 30–40% [12]. In a network whose link utilization is only 40%, about 95% of the power will be consumed compared with the fully–loaded power consumption. That means we use a lot of energy to transmit only a small portion of traffic, and a huge quantity of energy is wasted. Thus, we must find a way to change the network's energy consumption, in order to make energy usage grow in proportion with traffic loads. The promising solution is to find the minimum-power network subsets and shut down other unnecessary switches and links to satisfy changing traffic loads, and thus minimize network energy cost as well. In [13], [14], the possibility of turning off switches and links under connectivity and Quality of Service (QoS) constraints is shown, and heuristic algorithms are presented. In [7], a modification of the OSPF protocol to switch off links in an IP network is proposed, showing us the potential of energy-aware routing algorithms in backbone networks. However, compared to previous works, our work focuses on saving energy in partially deployed SDNs.

In this problem, the routing decisions of a network device are determined by the power states of other devices. Paths of flows can only be composed of links belonging to the active network subsets. In turn, a device's routing decisions have an influence on other devices' power states. Network devices that a determined path goes through cannot be powered off. Because of the distributed control in traditional networks, it is difficult to coordinate between devices. We need cen-

- *H. Wang, Y. Li and D. Jin are with Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (E-mail: liyong07@tsinghua.edu.cn).*
- *P. Hui is with Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China (E-mail: panhui@cse.ust.hk). He is also affiliated with Telekom Innovation Laboratories, Berlin Germany, and Aalto University, Helsinki, Finland.*
- *J. Wu is with the Department of Computer and Information Sciences, Temple University, 1805 N. Broad Street, Philadelphia, PA 19122 (E-mail: jiewu@temple.edu).*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

tralized control to solve the problem. Software defined network (SDN) is a new network architecture, which has a logically centralized controller responsible for all the control decisions in the network. Having a global network view, the controller is able to determine the power states and routing decisions in the network. By using some protocols of power-state control, it can turn off elements – links and switches – that are unnecessary in the network to save energy. Thus, SDN provides us a better way to save energy.

Despite these advantages SDN has, rapid adoption of this new architecture remains difficult. One important reason is that the budget is limited and only a part of the network can be upgraded at a time especially for large-scale networks. In addition, as an emerging technique, SDN is not mature enough to replace the traditional network at once. It is necessary to incrementally deploy SDN to evaluate its practicality. Therefore, SDNs are usually incrementally introduced into an existing network, and the transition is likely to span several years. During the transition, SDNs should coexist with traditional networks, and network operators have to manage the hybrid networks. In addition, in the hybrid network, we can still implement centralized control of these SDN-enabled devices to save energy, though the effect in hybrid SDN is not as good as in full SDN. Besides, in the partially deployed SDN, we must operate the power states of devices to minimize the energy cost of the whole network with only a part of the network under control. It is a new problem that has not been solved. Thus, partially deployed SDNs will be our main scenarios to implement algorithms of saving energy, rather than full SDNs. Our key question becomes how to save energy in partially deployed SDNs.

However, there are several unique problems and challenges of saving energy in partially deployed SDNs. First of all, different with saving energy in fully deployed SDNs, not all elements in the network can be controlled by the controller in the partially deployed SDN. On the other hand, different with traditional networks, synchronization and cooperation among SDN controllers should be considered [28]. Moreover, the solution of saving energy must apply to the transitional networks with all different percentages of SDN and traditional switches. It also need to be able to achieve efficient performance for saving energy and be time-efficient enough to be practical. Problems of bandwidth guarantee and network update [18], [29] also need to be considered to provide services with good quality to customers. Thus, it is not an easy problem to be investigated.

In this paper, we investigate the problem of how to reduce unnecessary energy consumption in partially deployed SDNs. More specifically, we consider the problem of finding minimum-power network subsets in partially deployed SDNs. The objective of this paper is to develop a scheme that is able to dynamically adjust the active network subsets in the partially deployed SDN to satisfy changing traffic loads, as well as to reduce unnecessary energy consumption. Our novel contribution is threefold, and is summarized as follows:

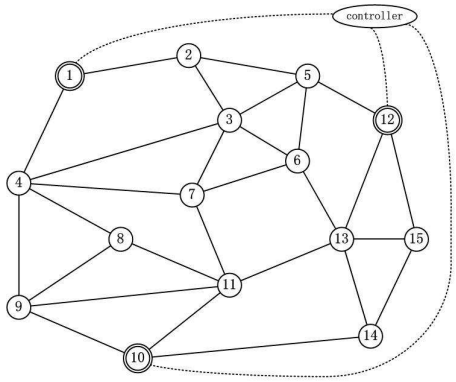


Fig. 1. Partially deployed SDN for example

- We are the first to address the problem of saving energy in partially deployed SDNs. Although saving energy in traditional networks and SDNs is an old problem, saving energy in partially deployed SDNs is a new problem that is not presented in previous works.
- We formulate the problem of finding minimum-power network subsets in partially deployed SDNs, and further theoretically prove that it is NP-hard.
- We propose an efficient heuristic algorithm to solve the formulated problem. By extensive simulations, we demonstrate that our heuristic algorithm achieves good performance of saving energy.

The rest of the paper is organized as follows. In Section 2, we formulate the problem of shutting down unnecessary network elements in partially deployed SDNs, and prove it is NP-hard. Based on the formulation, we design a heuristic algorithm in Section 3. In Section 4, we evaluate the performance of our solution through extensive simulations. We present related work in Section 5, and draw our conclusion in Section 6.

2 System Model and Problem Formulation

2.1 System Overview

We consider a network where SDN elements are only partially deployed. Take the network shown in Fig. 1 as an example. In this network, nodes 1, 10, 12 are SDN switches, which are controlled by the SDN controller. The SDN controller is responsible for computing the routing tables for all SDN switches in the network, using information it get from the network. It use a standardized protocol – Openflow – to communicate with the SDN switches. SDN switches' forwarding is implemented according to their obtained routing tables, which are determined by the SDN controller. The remaining nodes beyond the control of the SDN controller run some traditional routing protocols such as OSPF.

The SDN controller gathers network information to have a global view of the network. SDN switches carry out traffic measurement, and the result is sent to the controller. Besides, non-SDN switches use old routing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

protocols like OSPF to exchange bandwidth information on links in the network, and it is available for the SDN controller as well. Using this information from SDN switches and non-SDN switches, the controller can update routing tables at SDN switches, keeping in tune with changing traffic conditions. In addition, energy costs of switches or links can be known ahead of time by the types and configurations of switches. Thus, the SDN controller is able to compute the power states for SDN elements – SDN switches or links connected to SDN switches – and ensure a correct routing as well as good performance of energy saving. If the mechanism of saving energy is implemented in the traditional network, the same computation must be done by each device individually, which is much more complex than the computation of OSPF routing. Besides, we cannot ensure a safe update process during switching between two sets of power states of devices in the network, which may lead to problems such as routing loop, network black holes.

The controller is responsible for all the SDN elements' power states, but those non-SDN elements' power states cannot be operated by the controller. Therefore, the basic idea of our solution is to shut down as many SDN elements as possible, in the condition that the alive network subset can satisfy the traffic demand, so that we can reduce unnecessary energy consumption.

2.2 System Model

In this section, we present a simple mathematical model of forwarding in the partially deployed SDNs. Our model is a multi-commodity flow (MCF) formulation. Besides, we use binary variables to denote the power states of links and switches.

We use a directed graph to describe network's topology: assume the network is comprised of a set of nodes $\mathcal{N}$ interconnected by a set of directed links $\mathcal{A}$. Considering that actual links have no direction, if we have edge $(i,j) \in \mathcal{A}$, we also have edge $(j,i) \in \mathcal{A}$. Let $\mathcal{N}_S \subseteq \mathcal{N}$ be the set of SDN nodes and $\mathcal{A}_S \subseteq \mathcal{A}$ denote the set of links connected to SDN switches. For all $(u,v) \in \mathcal{A}$, $(u,v) \in \mathcal{A}_S$ if and only if $u \in \mathcal{N}_S$ or $v \in \mathcal{N}_S$. For each link $(i,j) \in \mathcal{A}$, $c(i,j)$ denotes its bandwidth capacity and $a(u,v)$ denotes its power cost. For each switch u, $b(u)$ denotes the switch's power cost. In addition, we collect all paths consisting of links belonging to $\mathcal{A}$ in the set $\mathcal{P}^{\mathcal{A}}$.

To describe traffic in the network, we regard flow demand from a source to a sink as a commodity. There are M commodities K_1,K_2,K_3....K_M in the network, defined by $K_m = (i_m, j_m, r_m)$, where for commodity m, i_m is the source, j_m is the sink, and r_m is the demand. The flow of commodity m along edge (i,j) is $f_m(i,j)$. In addition, we let $f_m(i,j)$ denote flow from i to j, while we let $f_m(j,i)$ denote flow from j to i. If the traffic between these two nodes is from node i to node j, we let $f_m(i,j) > 0$ while $f_m(j,i) = 0$. In turn, if the traffic between these two nodes is from node j to node i, we let $f_m(j,i) > 0$ while $f_m(i,j) = 0$. By this way, we can ensure that each $f_m(i,j)$ is always no less than 0.

TABLE 1
List of commonly used notations and variables.

Notation	*Description*
$\mathcal{N}, \mathcal{N}_S$	Set of all switches and SDN switches, respectively.
$\mathcal{A}, \mathcal{A}_S$	Set of all links and links connected to SDN switches, respectively.
$\mathcal{P}^{\mathcal{A}}$	Set of all paths consisting of links belonging to $\mathcal{A}$.
$c(u,v)$	Capacity for link (u,v).
k_m	Commodity m with its source i_m destination j_m and demand r_m.
$\mathcal{K}$	Set of all commodities.
$a(u,v)$	Power cost for link (u,v).
$b(u)$	Power cost for switch u.
$X(u,v)$	Binary decision variable indicating whether link (u,v) is powered on.
$Y(u)$	Binary decision variable indicating whether switch u is powered on.
$f_m(i,j)$	The flow of commodity m along link (i,j).

Except for these variables of normal MCF problem, we add binary variables for every link and switch. Binary variable $X(i,j)$ indicate whether link (i,j) is powered on. That is,

$$X(i,j) = \begin{cases} 1, & \text{if } (i,j) \text{ is powered on,} \\ 0, & \text{otherwise.} \end{cases}$$

Similarly, $Y(i)$ indicates whether switch i is powered on. Only SDN switches and links connected to SDN switches have binary variables to indicate their power states. The notations used by our formulation are shown in Table 1.

2.3 Problem Formulation

Because the power states of non-SDN elements cannot be operated by the controller, the power consumption of the SDN elements can represent the total power consumption in the network. Thus, the objective function can be represented as:

$$\textbf{min} \sum_{(u,v)\in\mathcal{A}_S} a(u,v)X(u,v) + \sum_{u\in\mathcal{N}_S} b(u)Y(u).$$

In all notations shown in the last section, only $X(i,j)$, Y_i and $f_m(i,j)$ are variables that are the output of our algorithm, and other notations are all known constants. At the same time, these variables and constants should satisfy a set of constraints according to some principles as follows:

1) Positivity constraints: The traffic from node i to node j is greater than zero; that is, $f_m(i,j)$ must be positive:

$$f_m(u,v) \geq 0, \quad \forall m, \ \forall (u,v) \in \mathcal{A}. \tag{1}$$

2) Capacity constraints: The total flow along each link must not exceed its capacity. Besides, a link connected to SDN switches has no traffic if we turn it off. Thus, for all commodity m, $f_m(i,j) = 0$ when $X(i,j) = 0$. The constraints for SDN links can be represented as:

$$\sum_{m=1}^{M} f_m(u,v) \leq X(u,v)C(u,v), \quad \forall (u,v) \in \mathcal{A}_S. \tag{2}$$

The capacity constraints for normal links remain unchanged, expressed by

$$\sum_{m=1}^{M} f_m(u,v) \leq C(u,v), \quad \forall (u,v), \in \mathcal{A}/\mathcal{A}_S. \tag{3}$$

3) Flow Conservation & Demand satisfaction:

$\sum_{\{v|(u,v)\in\mathcal{A}\}} f_m(u,v)$ presents the traffic created at the node v and $\sum_{\{v|(v,u)\in\mathcal{A}\}} f_m(v,u)$ presents the traffic destroyed at the node v. There are r_m units of traffic created in its source i_m and r_m units of traffic destroyed in its

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

destination j_m, and commodities are neither created nor destroyed at other nodes. Thus, $\sum_{\{v|(u,v)\in\mathcal{A}\}} f_m(u,v) - \sum_{\{v|(v,u)\in\mathcal{A}\}} f_m(v,u)$ should be zero at other nodes. The constraints are:

$$\sum_{\{v|(u,v)\in\mathcal{A}\}} f_m(u,v) - \sum_{\{v|(v,u)\in\mathcal{A}\}} f_m(v,u) = \begin{cases} r_m, & \text{if } u = i_m, \\ -r_m, & \text{if } u = j_m, \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

4) Correlate link and switch decision variable:

When a SDN switch u is powered off, all links connected to this switch are also powered off, which can be expressed as:

$$X(u,v) \le Y(u), \quad \forall u \in \mathcal{N}_S \; \forall (u,v) \in \mathcal{A}. \quad (5)$$

Similarly, when all links connecting to a SDN switch are off, the switch can be powered off. The linearized constraint is:

$$Y(u) \le \sum_{\{v|(u,v)\in\mathcal{A}\}} X(u,v), \quad \forall u \in \mathcal{N}_S. \quad (6)$$

These are all the constraints. Then, the mixed-integer linear programming (MIP) problem can be represented as:

$$\begin{aligned} &\textbf{min} \sum_{(u,v)\in\mathcal{A}_S} a(u,v)X(u,v) + \sum_{u\in\mathcal{N}_S} b(u)Y(u) \\ &\textbf{s.t.} \quad (1) \sim (6). \end{aligned} \quad (*)$$

Our problem is to find a feasible solution satisfying all these constraints, and minimizing the objective function as well.

5) Extension of the programming problem:

However, the above-formulated programming problem does not always have a solution. For example, in the peak time of traffic, the traffic demand usually cannot be satisfied even when all devices in the network are active. Take this factor into consideration, we loose constraint (4) to be:

$$\sum_{\{v|(u,v)\in\mathcal{A}\}} f_m(u,v) - \sum_{\{v|(v,u)\in\mathcal{A}\}} f_m(v,u) = \begin{cases} (1-\lambda_m)r_m, & \text{if } u = i_m, \\ -(1-\lambda_m)r_m, & \text{if } u = j_m, \\ 0, & \text{otherwise,} \end{cases} \quad (7)$$

in which λ_m represents the loss rate of traffic m, and we have $0 \le \lambda_m \le 1$. Let the maximum of the loss rate to be the metrics of reduction in Quality of Service (Qos). It can be written in the form of linear constraints as follows:

$$\begin{cases} \lambda_m \le \lambda, & \forall m, \\ C_Q = \lambda. \end{cases} \quad (8)$$

By putting C_Q in the objective function, it will automatically become the maximum of the loss rate when achieving the minimum of the objective function.

On the other hand, switching devices between different power states also leads to energy consumption and causes problems such as synchronization between switches. Thus, we should control the number of devices of which the power states are switched. Let C_S denote the cost of switching devices between different power states. We have the following equation:

$$C_S = \tau_1 \sum_{(u,v)\in\mathcal{A}_S} |X(u,v) - X^C(u,v)| + \tau_2 \sum_{u\in\mathcal{N}_S} |Y(u) - Y^C(u)|, \quad (9)$$

in which $X^C(u,v)$ and $Y^C(u)$ are binary variables indicating the current power states of devices, and τ_1, τ_2 represent the cost of switching the power state of a switch or a link, respectively.

Finally, let C_E denote the power consumption, the objective function of $(*)$, which can be represented as:

$$C_E = \sum_{(u,v)\in\mathcal{A}_S} a(u,v)X(u,v) + \sum_{u\in\mathcal{N}_S} b(u)Y(u). \quad (10)$$

Then, the extension of $(*)$, which takes both QoS and cost of switching power states of devices into account, can be represented as:

$$\begin{aligned} &\textbf{min} \; \alpha C_E + \beta C_Q + \gamma C_S \\ &\textbf{s.t.} \quad (1) \sim (3), \; (5) \sim (10), \end{aligned} \quad (*)$$

in which α, β and γ are weights of the three components to adjust their influence. Considering saving energy should not influence QoS, we have $\beta \gg \max\{\alpha, \gamma\}$.

Of course, solutions of the optimization problem are closest to optimal. However, the mixed integer programming (MIP) problem is NP-hard, of which the proof can be found in Appendix A. Thus, the time to find the optimal solution is intolerable, especially on medium and large sized problem. For example, finding the optimal solution of the optimization of a network with 30 nodes and a medium traffic load by GLPK on a Quad-Core 3.2GHz machine takes at least an hour. Therefore, we need a heuristic algorithm with a low time complexity.

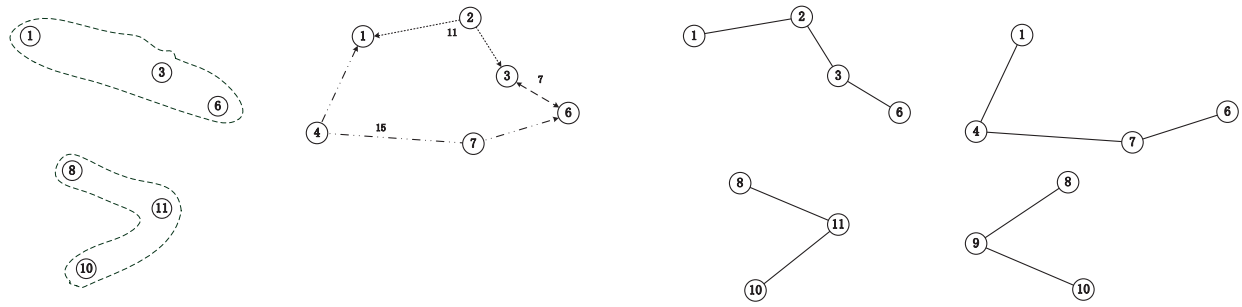
3 Heuristic Algorithm

If we want to connect all nodes and minimize the number of active links, using the topology of tree structure is an obvious solution. However, not every pair of nodes in the network has traffic to exchange. Thus, we only need to create spanning trees of some subsets of nodes. These subsets are called SCSs which will be introduced in Section 3.1. Besides, there are a number of paths between each pair of nodes. To reduce energy cost, we select paths with the smallest power consumption between nodes as edges of spanning trees. Thus, an intuitive consideration of our proposed heuristic algorithm is to use the topology of tree structure to connect nodes that have traffic to exchange, and select paths with the smallest power consumption between nodes as edges of spanning trees. To select edges with the smallest power consumption, weights are used to represent the energy consumption of network elements, and the method of modifying weights is used. More details about the modified weights will be introduced in Section 3.2. Thus, we set a path's weight according to the power states and energy consumption of the path's constituent elements. We select paths of minimum weight between nodes to form a group of spanning trees. If this group of spanning trees cannot satisfy the traffic loads, we will remove the overloaded links from the network topology, and generate other groups of spanning trees according to the current network topology. Finally, several groups of spanning trees will be used to satisfy the traffic loads, and they compose the target network subset that our algorithm outputs. Now, we describe the proposed algorithm in more details.

3.1 Smallest Closed Set

In preparation for description in detail, we define some operators above the set:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers



(a) SCSs of the given traffic demand (b) Shortest path between nodes 1,3,6 and their weight (c) First group of spanning trees (d) Second group of spanning trees

Fig. 2. SCSs and spanning trees for example.

Definition 0 (G($\mathcal{A}$), Delete and Add) If $\mathcal{A}$ is an arbitrary set and b is an arbitrary element. Assume $\mathcal{A}$ can be represented as $\{a_i : i \in \mathcal{I}\}$, where $\mathcal{I}$ is an index set. We define $G(\mathcal{A}) = \{\{a_i\} : i \in \mathcal{I}\}$, Delete$(b, \mathcal{A}) = \{a : a \neq b\ a \in \mathcal{A}\}$ and Add$(b, \mathcal{A}) = \{b\} \cup \mathcal{A}$.

For example, assume $\mathcal{A}$ is a set of natural numbers such as $\{1, 2, 3\}$. Let $b_1 = 2$ and $b_2 = 4$. Then we have $G(\mathcal{A}) = \{\{1\}, \{2\}, \{3\}\}$, Delete$(b_1, \mathcal{A}) = \{1, 3\}$, and Add$(b_2, \mathcal{A}) = \{1, 2, 3, 4\}$.

If we have a traffic $k_m = (i_m, j_m, r_m)$ of our demand, we cannot turn off the switch i_m and j_m. Besides, we must ensure there is at least one complete active path between i_m and j_m. For simplicity, we define a relation between nodes that have traffic to communicate.

Definition 1 ($\leftrightarrow$) A node i is said to communicate with node j (written $i \leftrightarrow j$), if $\exists k_m$ such that $k_m = (i, j, r)$ or $k_m = (j, i, r)$.

Then, we collect nodes that should be connected in a set named smallest closed set (hereafter referred to as SCS).

Definition 2 (closed set) A node set $\mathcal{A}$ is called a closed set, if $\forall i \in \mathcal{A}\ \forall j \in \mathcal{N}\ i \leftrightarrow j$ iff $j \in \mathcal{A}$.

Definition 3 (SCS) A closed set $\mathcal{A}$ is called a smallest closed set, if $\nexists \mathcal{B} \subset \mathcal{A}\ \mathcal{B} \neq \mathcal{A}$ such that $\mathcal{B}$ is a closed set.

Algorithm 1: Computing SCSs for traffic $\mathcal{K}$.

Input: set of all commodities of traffic $\mathcal{K}$
Initialize:
  $\mathcal{S}^{\mathcal{K}} \leftarrow \emptyset$
for *each* $k_m \in \mathcal{K}$ **do**
  if $i_m \in S_i^{\mathcal{K}}$ *and* $j_m \in S_j^{\mathcal{K}}$ $i \neq j$ **then**
    Delete$(S_i^{\mathcal{K}}, \mathcal{S}^{\mathcal{K}})$
    Delete$(S_j^{\mathcal{K}}, \mathcal{S}^{\mathcal{K}})$
    Add$(S_i^{\mathcal{K}} \cup S_j^{\mathcal{K}}, \mathcal{S}^{\mathcal{K}})$
  else if $i_m \in S_i^{\mathcal{K}}$ **then**
    Add$(j_m, S_i^{\mathcal{K}})$
  else if $j_m \in S_i^{\mathcal{K}}$ **then**
    Add$(i_m, S_i^{\mathcal{K}})$
  else
    Add$(\{i_m, j_m\}, \mathcal{S}^{\mathcal{K}})$
Output: $\mathcal{S}^{\mathcal{K}}$

A closed set is a set with the property that all nodes belonging to this set only have traffic to exchange with nodes also contained in this set. Besides, in this paper, we only consider SCS – the smallest closed sets that do not have any closed proper subset. We call the set of SCSs we have $\mathcal{S}^{\mathcal{K}} = \{S_1^{\mathcal{K}} ... S_N^{\mathcal{K}}\}$ under the taffic $\mathcal{K}$. Then, more detail about computing SCSs under a traffic K is shown in Algorithm 1. For simplicity, we refer this algorithm as $\mathcal{S}^{\mathcal{K}} = SCS(\mathcal{K})$ hereafter. After having SCSs, to form an energy-saving network subset, we only need to create a spanning tree for each SCS.

To illustrate the definition of SCS, consider an example network with topology shown in Fig. 1 and having the following traffic demand:

$$(k_1^T, k_2^T, k_3^T, k_4^T, k_5^T) = \begin{pmatrix} 1 & 3 & 3 & 8 & 8 \\ 6 & 4 & 1 & 10 & 11 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Then as shown in Fig. 2(a), we have two SCSs, the first SCS contains nodes 1, 3, 6, and the second SCS contains nodes 8, 10, 11.

3.2 Spanning Tree of SCSs

We now discuss how to create an energy-saving spanning tree of a SCS. We give each link (u, v) a weight

Algorithm 2: Computing Spanning Trees for $\{S_i^{\mathcal{K}}\}$.

Input: the set of SCSs $\mathcal{S}^{\mathcal{K}} = \{S_i^{\mathcal{K}}\}$, the set of all links $\mathcal{A}$, binary decision variables for links' and nodes' power states X, Y.
Initialize:
  $\mathcal{T} \leftarrow \emptyset$
for *each* $S_i^{\mathcal{K}} \in \mathcal{S}^{\mathcal{K}}$ **do**
  $\mathcal{G} = G(S_i^{\mathcal{K}})$
  while $card(\mathcal{G}) \neq 1$ **do**
    $(A_0, B_0) =$argmin$_{A \in \mathcal{G}\ B \in \mathcal{G}\ A \neq B}dis^{\mathcal{A}}(A, B)$
    Delete$(A_0, \mathcal{G})$
    Delete$(B_0, \mathcal{G})$
    Add$(A_0 \cup B_0, \mathcal{G})$
    $p =$path$^{\mathcal{A}}(A_0, B_0)$
    for *each* $(u, v) \in p$ **do**
      Add$((u, v), \mathcal{T})$
      $X(u, v) = Y(u) = Y(v) = 1$
Output: $\mathcal{T}$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

$a'(u,v)$ and each node a weight $b'(u)$. The weights are related to their energy cost. For elements having been determined to be used, using their residual bandwidth only consumes an extra 8% of power at most [5], [8], which can be ignored. Thus, we set their weights to be zero. Non-SDN elements' power states cannot be operated by the controller. Thus, they must be powered on, and their weights are also set to be zero. Then the sum of weights describes a path's additional energy cost when it will be used. However, in order to have a correct routing with no routing loops when all switches and links are used, we cannot let weights be exactly zero. Thus, we finally define

$$\begin{cases} a'(u,v) = a(u,v)(1 - X(u,v)) + c \cdot a''(u,v), \\ b'(u) = b(u)(1 - Y(u)), \end{cases} \tag{11}$$

where $a''(u,v)$ is related to the link's throughput, availability or reliability just as OSPF. Besides, c is a small constant, so that if $X(u,v) = 0$, $c \cdot a''(u,v)$ can be ignored. Then the sum of weights can approximately present the path's additional energy cost when it will be used. On the other hand, when all switches and links are used, these weights have only the part of $c \cdot a''(u,v)$ left, and our routing becomes an approximate OSPF routing. Thus, this kind of weight can meet the demand in all cases. In addition, if considering the cost of switching power states of devices, the weight can be extended to:

$$\begin{cases} a^E(u,v) = a'(u,v) + \tau_1(1 - X^C(u,v)), \\ b^E(u) = b'(u) + \tau_2(1 - Y^C(u)), \end{cases} \tag{12}$$

where $X^C(u,v)$ and $Y^C(u)$ are binary variables indicating the current power states of devices. The switches and links which are currently powered on have relatively smaller weight than those currently powered off. Thus, they will be used in a relatively higher probability in the next period, and they can stay active. In reverse, those currently powered off will be used in a relatively lower probability in the next period, and they can stay inactive. By using this kind of weight, the number of devices of which the power states are switched is able to be controlled. However, in order to achieve good performance, a lot of prior knowledge and repeated attempts are needed to obtain the appropriate values of τ_1 and τ_2. Thus, we only use $a'(u,v)$ and $b'(u)$ as the weight in the following analysis and discussion. Then, in order to have a more exact description, we have the following definitions:

Definition 4 (length of path) $\forall$ path $p \in \mathcal{P}^{\mathcal{A}}$, we define $\text{length}^{\mathcal{A}}(p) = \sum_{(u,v)\in p} a'(u,v) + \sum_{u \in p} b'(u)$.

Definition 5 (distance between nodes) $\forall i,j \in \mathcal{N}$, we define $\text{dis}^{\mathcal{A}}(i,j) = a$ if and only if $\forall$ path $p \in \mathcal{P}^{\mathcal{A}}$ from i to j, $\text{length}^{\mathcal{A}}(p) \geq a$ and $\exists$ path $p_0 \in \mathcal{P}^{\mathcal{A}}$ from i to j such that $\text{length}^{\mathcal{A}}(p_0) = a$. We define that $p_0 = \text{path}^{\mathcal{A}}(i,j)$, p_0 is called the shortest path between i and j under the topology $\mathcal{A}$.

Definition 6 (distance between node sets) $\forall \mathcal{A}, \mathcal{B} \subset \mathcal{N}$, under the topology $\mathcal{T}$, we define $\text{dis}^{\mathcal{T}}(A,B) = a$ if and only if $\forall i \in \mathcal{A}$ $\forall j \in \mathcal{B}$ $\forall$ path $p \in \mathcal{P}^{\mathcal{T}}$ form i to j, $\text{length}^{\mathcal{T}}(p) \geq a$ and $\exists$ path $p_0 \in \mathcal{P}^{\mathcal{T}}$ from $i_0 \in \mathcal{A}$ to $j_0 \in \mathcal{B}$ such that $\text{length}^{\mathcal{T}}(p_0) = a$. And we define $p_0 = \text{path}^{\mathcal{T}}(A,B)$ as the shortest path between $\mathcal{A}$ and $\mathcal{B}$ under the topology $\mathcal{T}$.

A path's length is the sum of weights of switches and links that this path goes through. Distance between two nodes is the minimum length of all paths between these two nodes. We use a heuristic algorithm to create the spanning tree. For all pairs of nodes in $S_i^{\mathcal{K}}$, we find the pair $i,j \in S_i^{\mathcal{K}}$ such that $\text{dis}^{\mathcal{A}}(i,j)$ is minimum. Let the pair's shortest path be an edge of $S_i^{\mathcal{K}}$'s spanning tree. Besides, we treat nodes that have been connected by selected edge as one node (node set). Thus, if we want a shortest path from another node k to this node set, we select the shortest path between node set $\{k\}$ and this set as Definition 6. It is similar when looking for the shortest path between two node sets. When we search for next edge, we should not let selected edges form a loop. It is clear that we repeat this process until all nodes of $S_i^{\mathcal{K}}$ are connected together, then we have a spanning tree of $S_i^{\mathcal{K}}$. By this process, all pairs of nodes in the same SCS are guaranteed to be connected with each other in the network subset consisting of all the spanning trees of $S_i^{\mathcal{K}} \in \mathcal{S}^{\mathcal{K}}$, while nodes in different SCSs or not in any SCS might be disconnected. Thus, the spanning trees obtained from this heuristic algorithm are different from minimum spanning tree (MST) obtained from Kruskal algorithm [26]. In addition, the concept of spanning tree of SCS is similar to the concept of SCT in [22]. However, they use SCT to solve the problem of planning and operating the hybrid network, while we use the spanning tree of the SCS to converge traffic in order to power off other unused devices. The detailed process of creating a spanning tree of $\mathcal{S}^{\mathcal{K}}$ can be shown in the Algorithm 2. Similarly, we refer to this algorithm here as $T = ST(\mathcal{S}^{\mathcal{K}}, \mathcal{A}, X, Y)$. In addition, this algorithm needs binary variables to calculate links' weight, but it does not modify them; that is, modifying $X(u,v)$ and $Y(u)$ in this algorithm does not influence $X(u,v)$ and $Y(u)$ out of this algorithm.

In the previous example of Section 3.1, for the first SCS, we have nodes 1, 3, 6. For simplicity, we define weights of all links as 1, and weights of all nodes as 3. The shortest paths between each pair of nodes are shown in Fig. 2(b). The shortest path between node 1 and node 3 is represented by the short-dashed line, and it has a length of 11. The shortest path between node 1 and node 6 is represented by the dash dot line and has a length of 15. Between node 3 and node 6, the shortest path is $(3,6)$, represented by the long-dashed line, and it has a length of 7. Thus, we will first select link $(3,6)$ as an edge of our spanning tree. Repeat this step, we will have a spanning tree with links $(1,2),(2,3),(3,6)$, which is shown in Fig. 2(c).

3.3 More Groups of Spanning Tree

When traffic loads in the network are low enough that only one group of spanning trees can satisfy the traffic demand, our algorithm can achieve excellent performance, because the tree structure uses the fewest edges,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

and edges of our spanning are selected as paths of minimum weight. However, an obvious problem is what to do when only one group of spanning trees cannot satisfy traffic demand.

We allocate traffic with shorter path in the spanning trees with a higher priority. When the load of some links exceed its capacity, we remove this link from the network topology. Then we use left traffic's closed sets and current network topology to create another group of spanning trees. We repeat this process until all traffic in the network has been allocated. Considering the design of our links' weights, unused links' weights are obviously larger than weights of used links. Therefore, when we select paths, unused links are more difficult to be selected, and then the spanning trees we create have smaller weights and more used links. Summing up the above discussion, our solution consists of the following steps. Besides, more detail is given in Algorithm 3.

1) Create spanning trees according to traffic demand and current network topology, then go to step 2.
2) Allocate traffic in the spanning trees. If some link's loads exceed its capacity, go to step 3, or if all traffic is allocated, we stop.
3) Remove the overloaded links from the network topology, and use left traffic and new network topology to repeat step 1.

Algorithm 3: Heuristic algorithm for energy saving.

Input: set of nodes $\mathcal{N}$,set of links $\mathcal{A}$, link capacity $c(u,v)$ for all link $(u,v) \in \mathcal{A}$.
Initialize:
  $X(u,v) \leftarrow 0 \ \forall (u,v) \in \mathcal{A}$
  $Y(u) \leftarrow 0 \ \forall u \in \mathcal{N}$
while $\mathcal{K} \neq \emptyset$ **do**
  $\mathcal{S}^{\mathcal{K}} = SCS(\mathcal{K})$
  $\mathcal{T} = ST(\mathcal{S}^{\mathcal{K}}, \mathcal{A}, X, Y)$.
  while $\mathcal{K} \neq \emptyset$ **do**
    $m_0 = \text{argmin}_m \text{dis}^{\mathcal{T}}(i_m, j_m)$
    $p = \text{path}^{\mathcal{T}}(i_{m_0}, j_{m_0})$
    $c_{min} = \text{min}_{(u,v)\in p} c(u,v)$
    $(u_{min}, v_{min}) = \text{argmin}_{(u,v)\in p} c(u,v)$
    if $c_{min} > r_{m_0}$ **then**
      for *each* $(u,v) \in p$ **do**
        $c(u,v) = c(u,v) - r_{m_0}$
        $f_{m_0}(u,v) = f_{m_0}(u,v) + r_{m_0}$
        $X(u,v) = Y(u) = Y(v) = 1$
      Delete$(k_{m_0}, \mathcal{K})$
    else
      for *each* $(u,v) \in p$ **do**
        $c(u,v) = c(u,v) - c_{min}$
        $f_{m_0}(u,v) = f_{m_0}(u,v) + c_{min}$
        $X(u,v) = Y(u) = Y(v) = 1$
      Delete$((u_{min}, v_{min}), \mathcal{A})$
      $r_{m_0} = r_{m_0} - c_{min}$
      break
Output: $\{X(u,v)\},\ \{Y(u)\}$ *and* $\{f_m(u,v)\}$

Still use the example of Section 3.1 and Section 3.2. Assume all links' capacity is 1. We now have the first group of spanning trees composed of links $(1,2),(2,3),(3,6)$ and $(8,11),(10,11)$, which is shown in Fig. 2(c). Because paths of flows $K_2(3,6,1), K_3(3,1,1), K_4(8,10,1)$ are shorter than those of $K_1(1,6,1), K_5(8,11,1)$ in the network, they are allocated in higher priority. However, when it comes to flows K_1 and K_5, links $(1,2),(2,3),(3,6),(8,11)$ are all overloaded. Thus, we remove links $(1,2),(2,3),(3,6),(8,11)$ from the network topology, and use the left traffic – K_1, K_5 to create the second groups of spanning trees, which are composed of links $(1,4),(4,7),(7,6)$ and $(8,9),(9,10)$ as shown in Fig. 2(d). Then all traffic demand is satisfied by these two groups of spanning trees. Only 10 switches and 10 links are used, and power consumption of the other 5 switches and 18 links is saved. Besides, in actual networks, the volume of a flow is not so big compared with the links' capacity as the example. Thus, the performance of this algorithm will be much better.

With the growth of the number of SDN elements deployed in the network, the ability of the controller to manage traffic and power consumption becomes stronger. When all elements are SDNs', our proposed solution is still feasible. In the fully deployed SND network, we only need to set all elements' weights as the function of their energy cost at the beginning, then the heuristic algorithm will work properly.

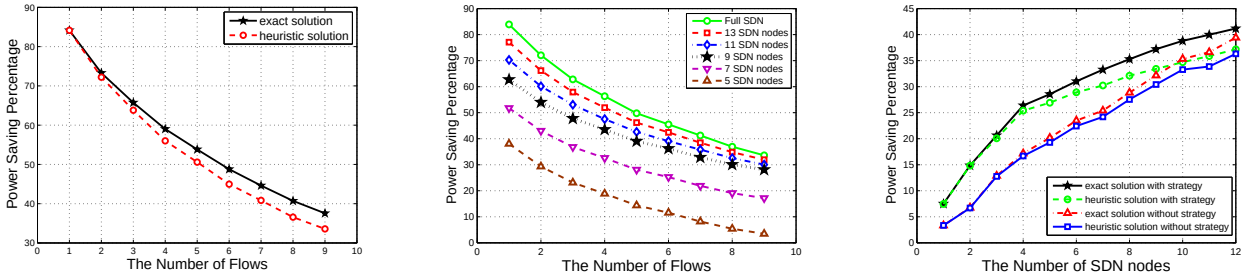
4 Performance Evaluation

4.1 Simulation System Set Up

To evaluate the performance of the heuristic algorithm in various cases, we run simulations of networks from tens of nodes to hundreds of nodes. Besides, simulations of networks with low traffic loads and high traffic loads are run, respectively. Then, we carry out a group of experiments in typical data center network to simulate the dynamic performance with varying time and traffic load of the heuristic algorithm. After that, a group of experiments are implemented by using the packet-level traces provided in [3] to evaluate the performance of the heuristic algorithm in realistic scenarios. Packet delay and loss are also measured by conducting experiments with the true packet traces using the packet-based emulator for SDN, mininet [21].

As for some other solutions to be compared with, the exact solution of the optimization problem is a good choice. Thus, we implement the MIP problem using YALMIP C a language for formulating generic optimization problems [15], and utilize the GLPK to solve the formulation [16]. However, when the network size is large, its time cost cannot be tolerable. At the same time, performance of the heuristic solution in large networks also need to be discussed. Thus, simulations of a small size network are run to compare performance of the exact solution and the heuristic solution. In large networks, we select three other algorithms to be compared with. The first algorithm (referred to as simple dijkstra algorithm)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers



(a) (b) (c)

Fig. 3. Power saving percentages (PSP) vs different parameters under the topology shown in Fig. 1

is selected to show the potential energy that can be saved by using the current routing algorithm. Specifically, it is a normal shortest path algorithm, but differently, after we finish allocation of traffic, we will close switches and links which are still idle. The second algorithm (referred to as modified dijkstra algorithm) is selected to show the profit of using tree structure by comparing with the heuristic algorithm. On the basis of the first algorithm, the method of modifying weight is used in the second algorithm, $i.e.$, the weights introduced in Section 3. By using them, this algorithm can select paths with less energy consumption, having a correct routing at same time. However, the minimum-power network subsets found by it do not have a tree structure, which is different from the heuristic algorithm. The third compared algorithm (referred to as simple heuristic algorithm) is selected to show the effect of using modified weight by comparing with the heuristic algorithm. Specifically, it is almost the same as the heuristic algorithm, expect that it does not use the modified weight. In addition, to show robustness of our proposed algorithm in fully deployed SDNs, it is also compared with an existing algorithm for saving energy in fully deployed SDNs in [5]. This algorithm (referred to as greedy algorithm) chooses the leftmost possible path for each flow. This is, within a layer of some structured topology such as fat tree, paths with sufficient capacity are chosen in a deterministic left-to-right order.

We run five groups of experiments to check the effectiveness of our algorithm in the following five cases: (1) The SDN of the 15 node topology shown in Fig. 1, with varying traffic load, (2) The partially deployed SDN of the same topology, with varying number of SDN nodes, (3) Network of large size, whose topologies are generated randomly, with varying network size, (4) Typical data center network with a Fat-Tree topology (shown in Fig 6), (5) Real data center network of a university in Mid-United States (EDU1 provided in [3]). Experiments in the first four cases are simulations of the energy consumption with varying parameters. However, in the $5th$ case, we implement the experiment by creating a SDN network and using true packet traces of a data center network. Performance in terms of packet loss and delay is also evaluated.

In addition, we define P_{total} to indicate the power consumption without any algorithm to save energy, and P_{actual} to indicate actual power consumption with a power saving algorithm. Let $P_{saved} = P_{total} - P_{actual}$ denote the energy saved by the algorithm. Then our primary metric, which is referred to as power saving percentage, denoted by PSP, can be computed as,

$$PSP = \frac{P_{saved}}{P_{total}} \times 100 = (1 - \frac{P_{actual}}{P_{total}}) \times 100.$$

This percentage gives an accurate idea of the power saved by turning off switches and links. The higher its value is, the better saving effect our solution has.

4.2 Results and Analysis

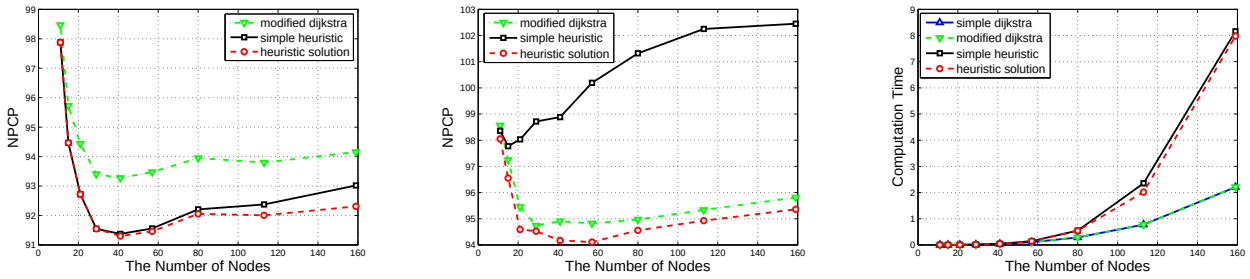
4.2.1 Performance Changes with Traffic Load

We run two groups of simulations to show the relation between the performance and traffic load. The topology of the network is shown in Fig. 1. In this group of simulations, traffic load is adjusted by changing the number of flows, while the volume of flows is set to be following a uniform distribution from 0 to 50% of the links' capacity. The result is shown in Fig. 3(a) and Fig. 3(b).

As result shown in Fig. 3(a), we run the simulation of a full-SDN network with exact solution and heuristic solution. The energy saving performance metrics are evaluated, in terms of PSP, as a function of traffic load. Here, we adjust traffic load by changing the number of flows in the network. With the increase of the traffic load in the network, we observe that our algorithm's PSP decreases: we save about 85 percent of energy with a low load, and it becomes only about 35 percent when traffic load is high. This is because larger traffic loads need to be satisfied with more network elements. Thus, elements that can be closed become less. Meanwhile, our heuristic solution has a performance very close to the exact solution: for the exact solution, the heuristic solution results in only a 5 percent drop at most. With a low traffic load, the difference cannot even be distinguished.

In the experiment shown in Fig. 3(b), we simulate the performance of the heuristic solution in the network with different numbers of SDN nodes. The curve with more SDN nodes changes more violently, and the energy saving effect is better as well. This is obvious because with more SDN elements, the ability to control traffic and elements' power state becomes stronger. Besides, the distance between curves with different numbers of SDN nodes is large when the SDN nodes' cardinal number is small. When there are enough SDN nodes, such as 60% of nodes, the performance of partially deployed SDN is very close to that of full SDN.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers



(a) NCPC at low load (b) NCPC at high load (c) computation time

Fig. 4. Normalized power consumption percentage (NPCP) and computation time vs network size under randomly generated topologies

4.2.2 Performance Changes with the Number of SDN Nodes

Except for the number of SDN nodes in the network, the locations of SDN nodes also have a strong impact on our energy saving effect. However, the optimum selection of nodes depends on the traffic pattern which will not be known ahead of time. Thus, to show the influence of the SDN nodes' locations, we use a strategy of selecting the locations of SDN nodes ignoring the traffic pattern. The principle of this strategy is to separate SDN nodes sufficiently and to have most links under the control of the SDN controller, that is, to make the number of links connected to SDN switches most at a fixed number of S-DN nodes. We select the node with the largest number of "uncontrolled" links connected to it as a SDN node each time. An "uncontrolled" link means it is not connected to selected SDN nodes. Among those nodes that have the same number of "uncontrolled" links connected to them, we select the SDN node randomly. As for topologies that are highly symmetric such as fat tree, we select the SDN node at different layers in rounds, that is, select one node at each layer in a round. Take the network with the topology shown in Fig. 1 as an example. At the beginning, nodes 3, 4, 11, 13 all have 5 links connected to them. Assume we select the node 3 as a SDN node. Then, because the node 4 is connected to the node 3, there are only 4 "uncontrolled" links left connected to it. Thus, we would choose the second SDN node from nodes 11 and 13 randomly. Repeat this process until we select enough SDN nodes, and we get the target locations of the SDN nodes.

We run two groups of simulations. In the first group, we select the locations of SDN nodes randomly, while in the second group of experiments we select the locations of SDN nodes with the strategy described above. In each group, energy saving metrics of exact solution and heuristic solution are evaluated as a function of the number of SDN nodes in the network, in terms of power saving percentage (PSP). The result is shown in Fig. 3(c). The topology is unchanged, which is shown in Fig. 1, and the volume of flows is also set to be following a uniform distribution from 0 to 50% of the links' capacity. In addition, the number of flows is set to be following a uniform distribution from 5 to 9, which is a suitable range in the network with 15 nodes according to the experiments in Section 4.2.1.

From the result, we observe that the performance of algorithms with a location selecting strategy is better than that of selecting nodes randomly, at a distance of 6 percent on average. The difference is most obvious when there are about 30 percent of SDN nodes, because of the difference in ability of controlling traffic in network, and it becomes small when the number of SDN nodes is large. Considering that when the cardinal number of SDN nodes is large, there is little part beyond the control of SDN controller, this result is pretty obvious. PSP's changing rate with the number of SDN nodes shows a similar trend: it is high with few SDN nodes, and low with a large number of SDN nodes. When we have 60 percent of SDN nodes, PSP's changing rate is low enough, and the performance is very close to full SDN.

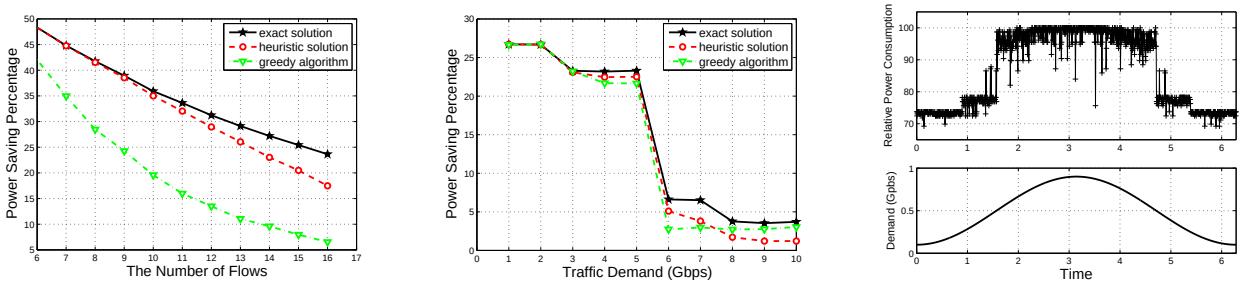
4.2.3 Performance Changes with Network Size

Performance of saving energy is evaluated in the network with low traffic loads and high traffic loads respectively in the simulations.

To simulate the system with variable network size, we use an variation on the Waxman model [25] to generate network topologies randomly, in which the probability that two nodes have a direct link is $P(u,v) = \beta \cdot e^{-d/(\alpha L)}$, where d is the distance between u and v, and L is the maximum distance between two nodes. More specifically, we locate nodes uniformly in a square area with border length $\sqrt{n}$, which means the maximum distance between two nodes is about $\sqrt{n}$. Then, we set α = 0.1 and β = 30 of the Waxman model, and get network topologies randomly generated.

In this group of simulations, the number of flows is set to be in proportion to n. Specifically, it is set to be following a uniform distribution from $\lfloor \frac{1}{12}n \rfloor$ to $\lfloor \frac{7}{12}n \rfloor$, where for any real number a, $\lfloor a \rfloor$ is the biggest integer less than a. Then, the traffic load is adjusted by changing the mean volume of flows. Since the source and sink of each flow are selected uniformly from all nodes in the network, the average number of routing hops of flows is increasing with the network size. Thus, by fixing the mean volume of flows, we can simulate the case of high traffic loads, in which the traffic load is increasing with the network size. On the other hand, the mean volume of flows is set to be $1/n$ of the links' capacity to simulate the case of low traffic load, in which the traffic load is decreasing with the network size. These parameters above are selected to make sure that the obtained traffic

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers



(a) (b) (c)

Fig. 5. Power saving percentage (PSP) vs different parameters in data center networks.

demands are feasible in most cases, where a feasible traffic demand is one that can be satisfied with traffic loads of all links under their capacity. On the other hand, the volume of each flow is set to be a relatively large value. Thus, to obtain feasible traffic demands, we cannot scale the simulations to too large number of flows in this groups of experiments.

For example, in the simulation of network with 159 nodes, the number of flows can be up to 92. However, the traffic load generated by this way is unbalanced in the small-scale network and large-scale network, which leads to a big change of PSP. To highlight the influence of the network size while reduce the influence of the unbalanced traffic loads, we normalize the power consumption with that of the simple dijkstra algorithm to represent the performance of energy saving, which is denoted by NPCP. It can be formulated as the follows:

$$NPCP(\text{algorithm}_i) = \frac{1 - PSP(\text{algorithm}_i)}{1 - PSP(\text{simple dijkstra})} \times 100,$$

in which algorithm_i can be modified dijkstra, simple heuristic or heuristic solution. Differently, this normalized power consumption percentage (NPCP) is lower when the energy saving effect is better. Besides, we ignore the partial deployment of SDN in this group of experiments in order to avoid the interference of factors such as the number and the locations of SDN nodes. At the same time, from the previous experiments, we observe that power saving percentage in networks with different numbers of SDN nodes shows a similar trend, and has a relatively fixed gap between each other. Thus, only considering fully deployed SDN in the experiment is reasonable, which shows us the potential optimal performance of the algorithm without interference of other factors.

Then we can see the $NPCP$ of the heuristic algorithm, modified dijkstra algorithm, and simple heuristic algorithm in Fig. 4(a) and Fig. 4(b), as a function of the network size. The simulations for Fig. 4(a) are set with low traffic loads, so that one or several groups of spanning trees can satisfy the traffic demand. We can see that the simple heuristic algorithm and the heuristic algorithm have a similar effect, better than modified dijkstra algorithm, because both of them use the topology of tree structure. Meanwhile, the simulations for Fig. 4(b) are set with high traffic loads. The performance of the modified dijkstra algorithm is still worse than that of that of the heuristic algorithm, but it is better than that of the simple heuristic algorithm. Moreover, without the method of modifying weight, networks with the simple heuristic algorithm consume even more power compared with networks with the simple dijkstra algorithm. It indicates that our algorithm that uses the topology of tree structure is unsuitable when traffic load is high.

In the experiments of large networks, we also simulate the time cost of algorithms. As shown in Fig. 4(c), when the number of nodes is less than 100 in the network, there is little difference of computation time between these algorithms. But when the node number exceeds 100, we can obviously see that the heuristic algorithm and simple heuristic algorithm use more time, which all use the topology of tree structure. Using the topology of tree structure can bring energy profit, but it also makes algorithms more complex and cost more time. But the time cost is not beyond ten seconds, which can be acceptable in network routing. In addition, the size of the subnet in SDN is limited by the capability of the controller. Early benchmarks on NOX, which is the first SDN controller, showed it could only handle 30,000 flow initiations per second [24]. If we assume that each flow lasts 1 second (100MB over a 1Gbps link), there are 3000 VMs at most that can be dominated by one controller. Assuming there are 20 VMs per server, the maximum number of hosts that can be dominated by one controller is about 150. As we observe from Fig. 4(c), computation time of the proposed algorithm in the network of 150 nodes is about 7 seconds, which can be further reduced by running at servers with more computing resources. Thus, the performance of our proposed algorithm is quite good in terms of computation time.

In addition, according to [8], [9], assume the additional power consumption of the server in which the controller is located by running our algorithm is 50 watts, and the static power consumption of the switch is 200 watts. Assume the computation time and the period of running our algorithm are 7s and 20s, respectively. Then, the energy saved by closing one switch is about 200×20 joules. The energy consumed by the additional computation is about 50×7 joules, which is less than 10% of the energy saved by closing one switch. In addition, it is normal to have a dozen of switches closed in a middle-scale network. Thus, the power consumption caused by the additional computation can be ignored compared with the energy saved.

4.2.4 Performance in Data Center Network

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

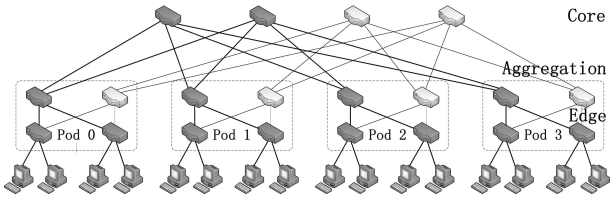


Fig. 6. Simple fat tree topology for example, in which all links' capacity is 1.0 Gbps (switches marked by gray color and links denoted by thick lines compose the spanning tree for example.)

In order to evaluate the performance of the heuristic algorithm in special scenarios of real system, we implement these groups of experiments in typical data center network with a Fat-Tree topology shown in Fig. 6.

Fig. 6 shows us a k = 4 fat tree topology. Switches marked by gray color and links denoted by thick lines in the figure compose a spanning tree for example. In a k-ary fat tree, there are k pods, and each switch also has k ports. Each pod contains two layers of $k/2$ switches. For each switch in pods, there are $k/2$ ports connected to switches of another layer and $k/2$ ports connected to hosts or core switches. An advantage of the fat-tree is that it provides full bisection bandwidth between all pairs of hosts. Specifically in our experiment, we assume no traffic comes from outside of the data center, and each link's capacity in the fat tree is 1Gpbs.

In our first group of experiments, the power saving percentage (PSP) is evaluated as a function of the number of flows in the data center network. The volumes of flows follow a uniform distribution from 0 to 1.0 Gbps. The result is shown in Fig. 5(a). From the results, we can observe that the trends of these curves are similar with Fig. 3(a), which means the performance of our proposed heuristic solution is very close to the optimal solution, indicating that our algorithm is also fit for special scenarios such as fat tree in data center networks. In addition, the performance of the heuristic algorithm is much better than that of the greedy algorithm, especially when there are medium number of flows in the network, indicating that our proposed algorithm also has comparative advantages to the existing algorithm for data center networks.

For the second group of experiments, we fix the number of flows, and vary the mean volume of flows. More specifically, we let each host communicate with only another host bidirectionally in the data center network, and thus there are totally 8 flows with the same volume of traffic between 16 hosts in the network. Fig.5(b) shows us the result. We can observe that when the traffic demand is less than 0.5 Gbps, about 25% of total power consumption is saved in the data center. But when the traffic demand exceeds 0.5 Gbps, the power saving percentages (PSP) become less than 10%. To illustrate the phenomenon, we now consider the spanning tree for example, which is composed of switches marked by gray color and links denoted by thick lines in the Fig.6. When the traffic demand is less than 0.5 Gbps, an arbitrarily

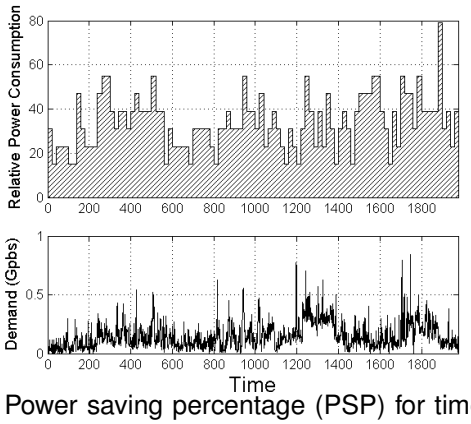


Fig. 7. Power saving percentage (PSP) for time-varying traffic demands in data center networks

spanning tree like it can satisfy the traffic demand. Thus, we ensure 30% of energy saving. However, when the traffic demand exceeds 0.5 Gbps, nearly all links and switches must be active to satisfy the traffic demand. However, in actual data center network, the volume of flows is not all the same. Thus, we can have an actual power saving percentage between 10% and 25%. In addition, as we can observe, similar to Fig. 5(a), performance of the heuristic solution is also very close to that of the exact solution, which achieves the best performance. Then, the performance of the heuristic solution is still better than that of the greedy algorithm in most cases, indicating the robustness of our proposed heuristic solution.

In order to analyze the dynamic behavior of our proposed heuristic algorithm, we simulate its performance for time-varying traffic demands in our third group of experiments. We assume the traffic demand d = 0.5-0.4cos(t) (Gbps) at time t in our experments, and the result is shown in Fig. 5(c). We can observe that the data center consumes approximately 70% of fully loaded power at begin when the traffic demand is low. Then the traffic demand goes through a peak period, leading to the increasement of the relative power consumption. When the traffic demands exceed 0.5 Gbps, almost all switches and links are active, and the relative power consumption becomes nearly 100%. Though the power consumption of the data center is not completely in proportion to the traffic demand, we only consume 86.2% of fully loaded power on average, while the average utilization of links is 50.0% in our experiments.

4.2.5 Performance in Real Data Center Network

To evaluate the performance in realistic SDN scenarios, we conduct this group of experiments using the true packet-level traces from a data center of a university in Mid-United States, EDU1 provided in [3]. There are 22 devices and 500 servers in the data center network, which has a 2-Tier architecture. The packet traces of [3] were captured using a Cisco port span, spanning 12 hours over multiple days. However, only a small part is provided, which is divided into 20 compressed pcap files. Each pcap file only spans 4 mins on average.

We first plot the curves of the relative power consumption and traffic demand with varying time in Fig.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

	Mean Delay (ms)		Mean loss rate (%)	
	heuristic	OSPF	heuristic	OSPF
Period 4	0.475	0.498	10.33	11.40
Period 37	0.161	0.143	4.80	4.16
Period 39	0.281	0.264	0.00	0.00
Period 48	0.397	0.313	3.00	4.10
Period 89	0.499	0.490	7.34	6.14
Period 95	0.492	0.484	19.47	19.01

TABLE 2
Comparision of Mean Delay and Mean loss rate of the heuristic solution and OSPF

7. We assume each link or switch can be switched on or switched off in 10 seconds, and the SDN controller dynamically adjusts active network subsets according to the mean traffic demands of the past 5 seconds by running the proposed heuristic algorithm every 20 seconds. The packet traces we use are merged from the first 9 pcap files provided by [3], ranging about 33 mins, that is, 99 periods of 20 seconds.

From the Fig. 7, we can observe that the curve shape of the relative energy consumption is in some way similar to it of the traffic demand. That is, using the proposed algorithm saves more energy in the off-peak time compared with it of the peak time. Over all, the effect of energy saving of the proposed algorithm is good. There is 60% of energy saved on average by it.

To evaluate the service quality, we use mininet [21], a network emulation platform, to create a SDN network with the topology of EDU1. Floodlight [20] is run as the controller. We use traffic matrices of 6 randomly selected periods, that is, the period 4, 37, 39, 48, 89, and 95 in the previous merged packet traces. Given the traffic matrices, Iperf is used to generate constant rate traffic flows and measure the packet loss and delay. The packet loss and delay of the proposed algorithm are evaluated compared with those of the OSPF routing. The result is shown in Table 2.

Our algorithm is designed to save energy by shutting down network devices and links, which may lead to performance degradation of packet loss and delay. However, at off-peak time, when there are plenty of residual bandwidth resources, applying our algorithm will not worsen the service quality. On the other hand, at peak time, all devices should be powered on. According to (12), the expression of weights of links and nodes, our algorithm degenerates to an approximate OSPF routing. Thus, the packet loss and delay still do not get worse compared with OSPF.

From the results shown in Table 2, we can observe that the performance degradation of packet loss and delay is much smaller compared with the energy saved. Packet loss and delay under the proposed algorithm are 8% and 2% more on average compared with these of only using OSPF routing, respectively, which indicates the overall performance is not affected much. On the other hand, the energy is saved by 60%. Thus, it is worth to utilize this algorithm in practice.

5 Related Work

Works related to our paper can be divided by three categories: saving energy in traditional networks, saving energy in SDNs, other works about partially deployed SDNs.

There have been a number of works focused on saving energy in traditional networks. Cianfrani *et al.* [7] focused on saving energy of OSPF protocol. They proposed a novel network-level strategy to save energy during low traffic periods. Their solution is based on coordination among routers, the idea of which is that only a subset of router Shortest Path Trees (SPTs) are used to select the routing paths, reducing the number of links used to route traffic. The works [5], [10]–[12] focused on saving energy of networks in other scenarios. Heller *et al.* [5] studied saving energy in a data center network by finding minimum-power network subsets. They formulated the problem of finding minimum-power network subsets, and presented a network-wide power manager, which dynamically adjusts the set of active network elements to save energy. Fisher *et al.* [12] developed and evaluated techniques that save energy in core networks by selectively powering down individual cables of large bundled links, and they developed several easy-to-implement heuristics. In addition, Correia *et al.* [10] and Marsan *et al.* [11] studied energy saving in mobile radio networks and cellular access networks, respectively, and developed their own heuristic algorithms as well.

Recently, some works studied the problem of saving energy in software-defined networks [17], [27]. Kakadia *et al.* [17] focused on saving energy based on OpenFlow. Without explicit constraints, they dynamically detected switches with minimal traffic and powered them off by consolidating the flows to other switches from them. Prete *et al.* [27] presented an OpenFlow controller that created a loop-free layer-2 topology and reduced the network energy consumption by switching off inactive interfaces.

In terms of works about partially deployed SDNs, Agarwal *et al.* [4] focused on traffic engineering in incrementally deployed SDNs. They formulated the routing optimization problem in the hybrid networks, and used the Fully Polynomial Time Approximation Scheme (FPTAS) to solve it. Their results show that the improvements of network utilization, packet losses, and delays are possible when SDN is only partially deployed in a network. Vissicchiowork *et al.* [18] studied network updates in incrementally deployed SDNs. They developed a machinery to realize anomaly-free updates of hybrid networks. Levin *et al.* [22] focused on the problem of how to incrementally introducing SDNs into an existing network. They proposed Panopticon, an architecture and methodology that integrates legacy and upgraded SDN switches and exposes an logical SDN abstraction to the control platform. Their results show that by only upgrading a few SDN switches, the transitional network is able to act as a SDN. In addition, Mukerjee *et al.* [23] focused on tradeoffs in incremental deployment of new network architectures such as SDN. They show that various deployment mechanisms only differ in how they answer four fundamental questions, *i.e.*, selecting and reaching

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

the egress and ingress, respectively. According to their theories, they proposed two intriguing approaches that were absent from existing mechanisms.

Different with the aforementioned works, our work focuses on the partially deployed SDN scenario, and tries to apply the traditional method of shutting down unnecessary network elements in partially deployed SDNs to save energy.

6 Conclusion and Future Work

In this work we analyzed the problem of saving energy in partially deployed SDNs. By formulating a mixed integer programming problem, which is proved to be NP-hard. Then, a heuristic algorithm is proposed to solve the problem. Results show that the proposed heuristic algorithm achieves good performance of saving energy and significantly reduces the computation time of the problem in both large scale and practical scenarios. Besides, the proposed algorithm achieves best performance in full SDN networks, but when there are enough SDN nodes, such as 60% of nodes, the performance of partially deployed SDN is very close to that of full SDN, that is, the performance gap is less than 10% relative to the fully deployed SDN.

Our heuristic algorithm aims at saving energy when traffic loads are not high in the network. On the other hand, saving energy may cause extra delay and loss of packets in the network, or it may increase traffic congestion during peak periods. Thus, strategies of voiding various side effects need to be considered in future works.

Appendix A

We now prove that the optimization problem is NP-hard in this appendix. We prove the complexity of the optimization problem by transforming it to a well-known NP-hard problem, i.e., 0-1 knapsack problem. Thus, some supplements about 0-1 knapsack problem are first provided in the following subsection.

A.1 0-1 knapsack problem

0-1 knapsack problem can be described as follow: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. Let v_i, w_i be the value and the weight of the item i, respectively. Let W be the limit of the total weight. Let binary variable x_i denote whether the item i is included. Then, the 0-1 knapsack problem can be represented as follows:

$$\textbf{max} \ \sum_{i=1}^{N} v_i x_i \quad \textbf{s.t.} \begin{cases} \sum_{i=1}^{N} w_i x_i \le W, \\ x_i \in \{0,1\}, \quad i = 1, ..., N. \end{cases} \tag{13}$$

This 0-1 knapsack problem is a well-known NP-hard problem [19]. We next prove that the optimization problem $(*)$ is NP-hard.

A.2 Proof of the complexity of the optimization problem

Theorem 1 (NP-hard) The optimization problem $(*)$ of finding the minimum-power network subsets described before is NP-hard.

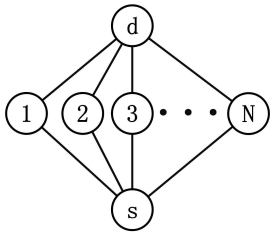


Fig. 8. Topology for example

Proof: To prove that the problem $(*)$ is NP-hard, we use the technique of reduction. Considering a simple case with a topology as shown in Fig. 8, the network is full SDN and we have traffic demand $\mathcal{K} = \{(s, d, r)\}$.

For all $i \in \{1, 2, ...N\}$, according to the flow conservation and demand satisfaction of nodes i, s, d, we have $f_1(i, s) = f_1(d, i) = 0$ and $f_1(s, i) - f_1(i, d) = 0$. Thus, we have $X(s, i) = X(i, d) = Y(i)$.

Define that

$$\begin{aligned} X_i &\triangleq X(s,i) = X(i,d) = Y(i), \\ f_i &\triangleq f_1(s,i) = f_1(i,d), \\ c_i &\triangleq min\{c(s,i), c(i,d)\}, \\ a_i &\triangleq a(s,i) + a(i,d) + b(i). \end{aligned}$$

The optimization problem becomes:

$$\textbf{min} \ \sum_{i=1}^{N} a_i X_i + b(s) + b(d) \quad \textbf{s.t.} \begin{cases} 0 \le f_i \le c_i X_i, \quad i = 1, ..., N, \\ \sum_{i=1}^{N} f_i \ge r, \\ X_i \in \{0,1\}, \quad i = 1, ..., N. \end{cases} \tag{14}$$

This problem is equivalent to the following problem by removing the intermediate variable f_i:

$$\textbf{min} \ \sum_{i=1}^{N} a_i X_i + b(s) + b(d) \quad \textbf{s.t.} \begin{cases} \sum_{i=1}^{N} c_i X_i \ge r, \\ X_i \in \{0,1\}, \quad i = 1, ..., N. \end{cases} \tag{15}$$

Then we consider the following 0-1 knapsack problem, where a_i, c_i are respectively the value and the weight of item i:

$$\textbf{max} \ \sum_{i=1}^{N} a_i Z_i \quad \textbf{s.t.} \begin{cases} \sum_{i=1}^{N} c_i Z_i \le W, \\ X_i \in \{0,1\}, \quad i = 1, ..., N. \end{cases} \tag{16}$$

Now by defining

$$\begin{aligned} Z_i &= 1 - X_i, \\ W &= \sum_{i=1}^{N} c_i - r. \end{aligned}$$

this 0-1 knapsack problem is transformed to problem (15) except for a constant $\sum_{i=1}^{N} a_i + b(s) + b(d)$ in the objective function, which can be ignored. Since the problem (15) is equivalent to our optimization problem, and the 0-1 knapsack problem is NP-hard, and the optimization problem is NP-hard as well.■

References

[1] M. Webb, "Smart 2020: Enabling the Low Carbon Economy in the Information Age," The Climate Group, tech. rep., 2008.

[2] S. Shafiee, E. Topal, "When will fossil fuel reserves be diminished?" *Energy Policy*, 2009, vol. 37, no. 1, pp. 181–189, Jan. 2009.

[3] T. Benson, A. Akella,D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild,"In *Proc. 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267–280.

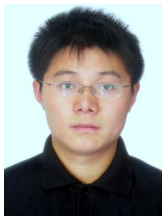
[4] S. Agarwal, M. Kodialam, T. V. Lakshman, "Traffic Engineering in Software Defined Networks," In *Proc. IEEE INFOCOM*, April 14-19, 2013, pp. 2211–2219.

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," In *Proc. of NSDI*, April 2010, pp. 249–264.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2451662, IEEE Transactions on Computers

[6] M. Gupta, S. Singh, "Greening of the internet," In *Proc. ACM SIGCOMM 03* (Karlsruhe, Germany), Aug. 2003, pp. 19-26.
[7] A. Cianfrani, V. Eramo, M. Listanti, M. Marazza, E. Vittorini, "An Energy Saving Routing Algorithm for a Green OSPF Protocol," in *Proc. IEEE INFOCOM*, Mar. 15-19, 2010, pp. 1–5.
[8] P. Mahadevan, P. Sharma, S. Banerjee, P. Ranganathan, "A Power Benchmarking Framework for Network Devices," In *Proceedings of IFIP Networking* (Aachen, Germany), May 2009, pp. 795-808.
[9] D. Economou, S. Rivoire, C. Kozyrakis, "Full-system power analysis and modeling for server environments," in *International Symposium on Computer Architecture-IEEE*, 2006.
[10] L. M. Correia, D. Zeller, O. Blume, D. Ferling, Y. Jading, I. Gódor, G. Auer, L. Van der Perre, "Challenges and enabling technologies for energy aware mobile radio networks," *IEEE Communications Magazine*, Vo. 48, no. 11, pp. 66–72, Nov. 2010.
[11] M. A. Marsan, L. Chiaraviglio, D. Ciullo, M. Meo, "Optimal Energy Savings in Cellular Access Networks," In *Proc. of GreenComm09*, June 14-18, 2009, pp. 1–5.
[12] W. Fisher, M. Suchara, and J. Rexford, "Greening Backbone Networks: Reducing Energy Consumption by Shutting Off Cables in Bundled Links," In *Proc. 1st ACM SIGCOMM Workshop on Green Networking*, Aug. 2010, pp. 29–34.
[13] L. Chiaraviglio, M. Mellia, F. Neri, "Reducing Power Consumption in Backbone Networks," In *Proc. IEEE ICC*, 2009, pp. 1–6.
[14] L. Chiaraviglio, M. Mellia, F. Neri, Energy-aware Backbone Networks: a Case Study, In *Proc. IEEE ICC*, June 14–18, 2009, pp. 1–5.
[15] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," In *Proc. Conference*, Sept. 4–4, 2004, pp. 284–289.
[16] A. Makhorin, GLPK (GNU linear programming kit), 2008.
[17] D. Kakadia and V. Varma. "Energy Efficient Data Center Networks-A SDN based approach," 2012.
[18] S. Vissicchio, L. Vanbever, L. Cittadini, G. Xie and O. Bonaventure, "Safe Updates of Hybrid SDN Networks," UCL, 2013, available at http://hdl.handle.net/2078.1/134360.
[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.
[20] Floodlight. Open Source Software for Building Software-Defined Networks. [Online]. Available: http://www.projectfloodlight.org/floodlight/
[21] Mininet. (2013, Mar). An Instant Virtual Network on your Laptop (or other PC). [Online].Available: http://mininet.org/
[22] D. Levin, M. Canini, S. Schmid, "Panopticon: Reaping the benefits of partial sdn deployment in enterprise networks." TU Berlin/T-Labs, Tech. Rep, 2013.
[23] M. K. Mukerjee, D. Han, S. Seshan, "Understanding tradeoffs in incremental deployment of new network architectures," in *Proc. of ACM CoNEXT*, 2013, pp. 271-282.
[24] A. Tavakoli, M. Casado, T. Koponen, "Applying NOX to the Datacenter," in *Proc. of HotNets*, 2009.
[25] L. Wei, D. Estrin, "The trade-offs of multicast trees and algorithms,"in *Proc. of ICCCN*, 1994, pp. 17-24.
[26] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice-Hall, Englewood Cli9s, NJ, 1993.
[27] L. Prete, F. Farina, M. Campanella and A. Biancini, "Energy efficient minimum spanning tree in OpenFlow networks," In *Proc. EWSDN*, Oct. 25-26, 2012, pp. 36–41.
[28] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." OSDI. Vol. 10. 2010.
[29] S. Ghorbani, M. Caesar, "Walk the line: consistent network updates with bandwidth guarantees", in *Proc. of HotSDN*, 2012, pp. 67-72.

Huandong Wang is a undergraduate student with the Department of Electronic Engineering of Tsinghua University, Beijing, China. His research interests include software-defined approach for green networks, network virtualization.



Yong Li (M'2009) received his B.S. degree in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2007, and his Ph.D. degree in electronic engineering from Tsinghua University, Beijing, China, in 2012.
He is now a postdoctoral researcher at Tsinghua University. He serves as a paper reviewer for international conferences of IEEE ICC, VTC, ICOIN, PIMRC, APCC and many others. His research fields include mobile delay tolerant networks, topics including forwarding polices design, buffer management design and performance evaluation; mobility modeling; and mobility management in next generation wireless IP networks, topics including Mobile IP, SIP, Proxy mobile IP, cross-layer design, multicast mobility, modeling for mobility performance evaluation, enhancing handoff performance and proposing mobility management architecture.

Depeng Jin (M'2009) received his B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1995 and 1999 respectively both in electronics engineering. Now he is an associate professor at Tsinghua University and vice chair of Department of Electronic Engineering. Dr. Jin was awarded National Scientific and Technological Innovation Prize (Second Class) in 2002. His research fields include telecommunications, high-speed networks, ASIC design and future internet architecture.

Pan Hui received his Ph.D degree from Computer Laboratory, University of Cambridge, and earned his MPhil and BEng both from the Department of Electrical and Electronic Engineering, University of Hong Kong. He is currently a faculty member of the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology where he directs the System and Media Lab. He also serves as a Distinguished Scientist of Telekom Innovation Laboratories (T-labs) Germany and an adjunct Professor of social computing and networking at Aalto University Finland. Before returning to Hong Kong, he has spent several years in T-labs and Intel Research Cambridge. He has published more than 100 research papers and has several granted and pending European patents. He has founded and chaired several IEEE/ACM conferences/workshops, and served on the technical program committee of numerous international conferences and workshops including IEEE Infocom, SECON, MASS, Globecom, WCNC, and ITC.

Jie Wu (F'09) is the chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, he was a program director at the National Science Foundation and Distinguished Professor at Florida Atlantic University. His research interests include wireless networks, mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. Dr. Wu publications include over 600 papers in scholarly journals, conference proceedings, and books. He has served on several editorial boards, including IEEE Transactions on Computers and Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS2008, and IEEE DCOSS 2009 and was the program co-chair for IEEE INFOCOMM 2011. He served as general chair for IEEE ICDCS 2013. He was an IEEE Computer Society Distinguished Visitor and the chair for the IEEE Technical Committee on Distributed Processing (TCDP). Currently, Dr. Wu is an ACM Distinguished Speaker and a Fellow of the IEEE. He is the recipient of 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.