Joint Host-Network Optimization for Energy-Efficient Data Center Networking

Hao Jin*, Tosmate Cheocherngngarn*, Dmita Levy*, Alex Smith†, Deng Pan*, Jason Liu*, and Niki Pissinou*
*Florida International University, Miami, FL
†Terra Environmental Research Institute, Miami, FL

***Abstract*—Data centers consume significant amounts of energy. As severs become more energy efficient with various energy saving techniques, the data center network (DCN) has been accounting for 20% or more of the energy consumed by the entire data center. While DCNs are typically provisioned with full bisection bandwidth, DCN traffic demonstrates fluctuating patterns. The objective of this work is to improve the energy efficiency of DCNs during off-peak traffic time by powering off idle devices. Although there exist a number of energy optimization solutions for DCNs, they consider only either the hosts or network, but not both. In this paper, we propose a joint optimization scheme that simultaneously optimizes virtual machine (VM) placement and network flow routing to maximize energy savings, and we also build an OpenFlow based prototype to experimentally demonstrate the effectiveness of our design. First, we formulate the joint optimization problem as an integer linear program, but it is not a practical solution due to high complexity. To practically and effectively combine host and network based optimization, we present a unified representation method that converts the VM placement problem to a routing problem. In addition, to accelerate processing the large number of servers and an even larger number of VMs, we describe a parallelization approach that divides the DCN into clusters for parallel processing. Further, to quickly find efficient paths for flows, we propose a fast topology oriented multipath routing algorithm that uses depth-first search to quickly traverse between hierarchical switch layers and uses the best-fit criterion to maximize flow consolidation. Finally, we have conducted extensive simulations and experiments to compare our design with existing ones. The simulation and experiment results fully demonstrate that our design outperforms existing host- or network-only optimization solutions, and well approximates the ideal linear program.**

***Index Terms*—Data center networks; virtual machine migration; multipath routing; energy efficiency.**

I. Introduction

Public and private data centers are becoming popular, since they achieve economies of scale with hundreds of thousands of servers [20], e.g. about 300,000 servers in Microsoft's Chicago data center [3]. The huge number of servers in data centers consume significant amounts of energy. It is estimated [2] that national energy consumption by data centers in 2011 was more than 100 billion kWh, representing a $7.4 billion annual electricity cost. As a result, energy efficiency of data centers has attracted significant attention in recent years [13], [22].

With the improvement of sever energy efficiency, the data center network (DCN), the other important component of a data center, has been accounting for 20% ore more [4], [25] of the energy consumed by the entire data center. With the huge number of servers in a data center, the DCN needs proportionally large bandwidth to interconnect the servers. In addition, a DCN is typically provisioned with full bisection bandwidth [5], [11], [12], [21] to support burst all-to-all communication. However, since DCN traffic demonstrates fluctuating patterns, the fully provisioned bandwidth cannot be always well utilized, resulting in resource underutilization and energy waste. For example, Figure 1 shows a 7-day traffic sample of a core router interface from a data center service provider. We can clearly see a wave pattern, with the highest instant traffic volume at about 13Gbps, and the lowest at about 2Gbps. Different colors in the figure represent different transport layer protocols, with TCP being the majority.

The key for DCNs to achieve energy conservation during off-peak traffic hours is to power off idle devices when possible. There exist a number of DCN energy saving solutions in the literature, which can be divided into two broad categories: optimizing network flow routing [14] and optimizing virtual machine (VM) placement [19]. The former consolidates flows to a smaller number of links, and thus leaves more idle links and consequently switches to be powered off. The latter consolidates VMs to physical servers in such a way that VM pairs with more traffic are placed closer, to avoid heavy flows traversing long paths.

To the best of our knowledge, existing DCN energy saving solutions consider only either the hosts or the network, but not both. In this paper, we study the joint host-network optimization problem to improve the energy efficiency of DCNs. The basic idea is to simultaneously consider VM placement and network flow routing [24], so as to create more energy saving opportunities. The simplest way to combine host and network based optimization is just to naively first determine the VM placement and then the flow routing. Unfortunately, the existing VM placement algorithm [19] is not practical, since it does not consider the bandwidth capacity constraints of links, assumes fixed VM memory sizes, and has high time complexity of $O(|V|^4)$, where V is the set of VMs.

For effective joint host-network optimization, the first challenge is how to simultaneously consider the two types of optimization problems. To address the challenge, we present a unified representation method that converts the VM placement problem as a routing problem, so that a single optimization solution can apply to both types of problems. Further, the second challenge is how to accelerate the processing of the huge number of VMs in a data center. To this end, we propose

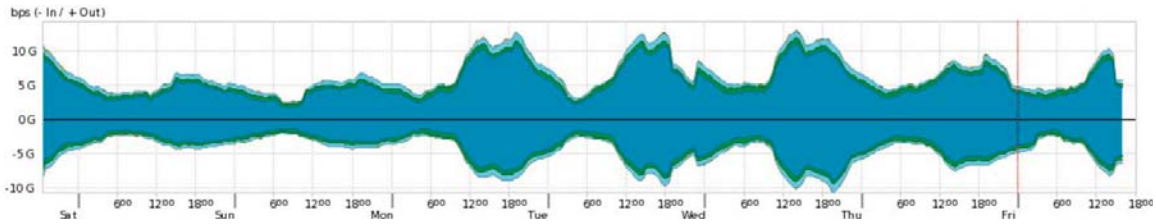


Fig. 1. Fluctuating DCN traffic pattern.

a parallelization approach that divides the DCN into clusters based on their subnet IP addresses, and processes the clusters in parallel for fast completion. Finally, the third challenge is how to quickly find efficient routing paths for the flows. To solve this problem, we propose a topology oriented fast multipath routing algorithm, which uses depth-first search to quickly traverse between the hierarchical layers in a DCN, and the best-fit criterion to maximize flow consolidation.

In this paper, we propose a joint host-network energy optimization scheme that combines VM placement and flow routing optimization, and build an OpenFlow based prototype to experimentally demonstrate the effectiveness of our design. We first formulate the problem as an integer linear program. Since integer linear programming is NP-complete and not suitable for practical deployment, we then propose a series of techniques to quickly and effectively solve the joint optimization problem. In addition, we have implemented the proposed scheme in a simulator and a prototype, and conducted extensive evaluations to compare it with existing solutions. The simulation and experiment results fully demonstrate that our scheme outperforms existing host- or network-only optimization solutions, and well approximates the ideal linear program.

The rest of the paper is organized as follows. Section II reviews the related work. Section III formulates the problem. Section IV elaborates our design guidelines. Section V presents the joint host-network energy optimization scheme. Section VI describes a prototype implementation. Section VII shows the simulation and the experiment results. Section VIII discusses the broadcast and safety margin issues. Finally, Section IX concludes the paper.

II. Related Work

In this section, we briefly review existing energy saving solutions for DCNs and more broadly wide area networks. Those solutions can be divided into two broad categories: network-side optimization and host-side optimization.

A. Network-Side Optimization

In the first category, ElasticTree [14] is a DCN power manager to find the set of switches and links that can accommodate the traffic and consume the minimum power. In addition, ElasticTree also addresses the robustness issue so that the optimized network has sufficient safety margins to prepare for traffic surges and network failures. GreenTE [27] manipulates the routing paths of wide area networks, so that the least number of routers shall be used to satisfy the performance constraints such as traffic demands and packet delays. Energy conservation can then be achieved by shutting down the idle routers and links without traffic. [10] proposes an energy saving scheme for the idle cables in bundled links. By reorganizing network traffic and powering off individual cables as well as the associated line cards in the low-utilized bundles, the scheme achieves a theoretical 79% improvement on energy efficiency for backbone networks. [4] indicates that a flattened butterfly DCN topology is more energy efficient than the folded Clos topology. [16] presents a large power profile study for the power manager Urja in an enterprise network, which saves over 30% of the network energy. [25] establishes a model of energy-aware routing in DCNs, and designs a heuristic to achieve the goal.

B. Host-Side Optimization

In the host-side optimization category, one approach is to optimize VM placement using live migrations [8], which will help consolidate VMs into fewer physical servers and traffic flows into fewer links. [19] proposes a traffic-aware VM placement scheme that localizes large traffic chunks and thus reduces loads of high layer switches. The scheme achieves energy conservation by shutting down idle servers and switches after the placement. [26] studies the VM consolidation problem in the context of dynamic bandwidth demands. The problem is formulated as a stochastic bin packing problem and proved as NP-hard. The paper then proposes an approximation algorithm, which uses fewer servers while still satisfies all the performance constraints. The second host-side optimization approach is to improve the energy proportionality on the server itself. PowerNap [18] is an energy saving scheme for servers to quickly switch between two states: a high-performance active state to transmit traffic, and an idle state with low power to save energy.

III. Problem Formulation

The joint host-network energy optimization problem is a variant of the multi-commodity problem [9], and can be formulated as a linear program. The optimization objective is to minimize the power consumption of all the servers, switches, and links in a DCN. Recent studies [14], [17], [23] indicate that power consumption of servers and switches in data centers can be roughly modeled as linear functions, which are suitable for linear programming. Even with non-linear power functions, approximation techniques can help convert them to piece-wise linear ones.

Model a DCN as a directed graph $G = (S \cup X, L)$, where a node $s \in S$ is a physical server, a node $x \in X$ is a switch, and an edge $(n_i, n_j) \in L$ is a link connecting

TABLE I
NOTATIONS FOR PROBLEM FORMULATION

Notation	Meaning
v, s, x, f	virtual machine, server, switch, or flow
(n_i, n_j)	link connecting two nodes n_i and n_j, with one node being a switch, and the other being be a switch or server
V, S, X, L, F	set of virtual machines, servers, switches, links, or flows
$DS(v)$	potential migration destination servers of VM v
$md(v)$	memory demand of VM v
$mc(s)$	memory capacity of server s
$src(f), dst(f)$	source or destination VM of flow f
$bd(f)$	bandwidth demand of flow f
$bc(n_i, n_j)$	bandwidth capacity of link (n_i, n_j)
$p(*)$	linear power function of $*$, where $*$ may be a server, switch, or link
$on(*)$	**decision variable:** 1 or 0 if $*$ is powered on or off, where $*$ may be a server, switch, or link
$host(s, v)$	**decision variable:** 1 or 0 if VM v is or not hosted on server s
$route(f, (n_i, n_j))$	**decision variable:** 1 or 0 if flow f is or not routed through link (n_i, n_j)

a switch and a server or two switches. Assume that V is the set of VMs, and a VM $v \in V$ must be hosted by a server s, denoted by $host(s, v) = 1$. When a server hosts a VM, the former provides the latter with various resources, such as memory space and CPU time, and we use memory space as a representative of such resources. Each server s has a memory capacity $mc(s)$, and each VM v has a memory demand $md(v)$. Due to constraints such as subnet IP addresses and hardware configurations, a VM v has a restricted set of migration destination servers, denoted as $DS(v) \subset S$. Use $on(*)$ to denote that a device $*$ is powered on, which may be a switch, link, or sever, and use $p(*)$ to denote the power consumption of the device $*$.

Assume that $f \in F$ is a flow in the DCN. f is defined as a triple $f = (src(f), dst(f), bd(f))$, where $src(f)$ is the source VM, $dst(f)$ is the destination VM, and $bd(f)$ is the bandwidth demand. Use $route(f, (n_i, n_j))$ to denote whether flow f is routed through link (n_i, n_j). $f_k(x_i, x_j)$ can only be either 1 or 0 to prohibit splitting a single flow among multiple paths. The reason is that, as seen in Figure 1, more than 99% of data center traffic flows are TCP ones [6], which will suffer performance degradation with out-of-order packet delivery. Note that a link $(n_i, n_j) \in L$ has a bandwidth capacity $bc(n_i, n_j)$.

With the above notations (summarized in Table I), we can thus formulate the joint optimization problem as the following linear program. Equation 1 is the objective function, simply to minimize the total power consumption of all the switches, links, and servers.

Equations 2 to 4 define the VM and server related constraints. Specifically, Equation 2 states the server-VM correlation constraint, i.e. only a powered on server can host VMs. Equation 3 states the server memory capacity constraint, i.e. the total memory demand of all the VMs hosted by a server cannot exceed its memory capacity. Equation 4 states the VM migration destination constraint, i.e. a VM can only be hosted by one of its destination servers.

Equations 5 to 10 define flow and link related constraints. Specifically, Equation 5 states the flow source/destination constraint, i.e. a flow cannot start/end at a server that is not hosting the source/destination VM. Equation 6 states the flow demand satisfaction constraint, which means that if the source and destination VMs of a flow are hosted by the same server, then the flow can be transmitted using the local bus of the sever, without traversing any switches; otherwise, the flow

$$\textbf{minimize} \sum_{x \in X} p(x) + \sum_{(n_i, n_j) \in L} p(n_i, n_j) + \sum_{s \in S} p(s) \quad (1)$$

subject to

$$\forall s \in S, \forall v \in V, host(s, v) \leq on(s) \quad (2)$$

$$\forall s \in S, \sum_{v \in V} md(v) host(s, v) \leq mc(s) on(s) \quad (3)$$

$$\forall v \in V, \sum_{s \in DS(v)} host(s, v) = 1,$$
$$\sum_{s \in S \setminus DS(v)} host(s, v) = 0 \quad (4)$$

$$\forall f \in F, \forall s \in S$$
$$\sum_{x \in X} route(f, (s, x)) \leq host(s, src(f)),$$
$$\sum_{x \in X} route(f, (x, s)) \leq host(s, dst(f)) \quad (5)$$

$$\forall f \in F, \forall s \in S$$
$$host(s, src(f)) - host(s, dst(f))$$
$$= \sum_{x \in X} route(f, (s, x)) - \sum_{x \in X} route(f, (x, s)) \quad (6)$$

$$\forall f \in F, \forall x \in X$$
$$\sum_{n \in S \cup X} route(f, (n, x)) = \sum_{n \in S \cup X} route(f, (x, n)) \quad (7)$$

$$\forall (n_i, n_j) \in L,$$
$$on(n_i, n_j) \leq on(n_i), on(n_i, n_j) \leq on(n_j) \quad (8)$$

$$\forall (n_i, n_j) \in L, on(n_i, n_j) = on(n_j, n_i) \quad (9)$$

$$\forall (n_i, n_j) \in L, \sum_{f \in F} bd(f) route(f, (n_i, n_j)))$$
$$\leq bc(n_i, n_j) on(n_i, n_j) \quad (10)$$

$$\forall x \in X, p(x) = \alpha(x) on(x) + \beta(x) \sum_{(x,n) \in L} on(x, n) \quad (11)$$

$$\forall (n_i, n_j) \in L, p(n_i, n_j) = \gamma(n_i, n_j) on(n_i, n_j) \quad (12)$$

$$\forall s \in S, p(s) = \delta(s) on(s) + \epsilon(s) \sum_{v \in V} host(s, v) \quad (13)$$

must start at the server hosting the source VM and end at the server hosting the destination VM. Equation 7 states the switch flow conservation constraint, i.e. a switch can only transmit flows but not generate or destroy any. Equation 8 states the node-link correlation constraint, i.e. only a powered on switch can have active links. Equation 9 states the bidirectional link power constraint, i.e. both directions of a link should have the same on/off status [14]. Equation 10 states the link bandwidth capacity constraint, i.e. the total bandwidth demand of all the flows through a link cannot exceed its bandwidth capacity.

Equations 11 to 13 are sample power functions for switches, links, and servers. Equation 11 defines that a powered on switch x consumes $\alpha(x)$ base power, and each active port consumes additional $\beta(x)$ power [14]. Equation 12 defines that an active link (n_i, n_j) consumes $2\gamma(n_i, n_j)$ power, or $\gamma(n_i, n_j)$ for each direction. Equation 13 defines that a server s consumes $\delta(s)$ base power, and each hosted VM consumes additional $\epsilon(s)$ power due to increased CPU usage [23].

Since integer linear programming is NP-complete, the above solution is not a practical one, but it can still be an ultimate bench mark to evaluate other approximation solutions.

IV. Design Guidelines

In this section, we elaborate our design guidelines to quickly and efficiently solve the joint optimization problem.

A. Unified Representation of Both Types of Optimization

For effective joint optimization, the first challenge is that there are two types of optimization, i.e. VM placement and flow routing, and it is not clear how to simultaneously consider them. We propose a unified representation method that converts the VM placement problem to a routing problem, so that a single solution can apply to both optimization problems. DCNs are typically organized in a multiple-layer hierarchical structure. For example, Figure 2(a) shows a fat tree based DCN with core, aggregation, and top-of-rack (ToR) three layers of switches, and an additional layer of servers.

The key observation is that the VM-server relationship is similar to that of sever-switch. A VM can select to reside on one of multiple allowed servers, and send its traffic through the physical network adapter of the hosting server. This is similar to the selection made by a server to pick one of multiple connected ToR switches to send its traffic. Inspired by the observation, we add an additional hierarchical layer of VMs. In detail, we create in the graph a new node for each VM, and use an edge to connect it with a server if it can migrate to the server. Figure 2(b) shows a simple example, where v_1, v_2, and v_3 can migrate to any server connected by the same aggregation switch, and v_4 can migrate to any server connected by the same ToR switch. In the optimization process, we search routing paths for the flows between VM pairs. If a VM has a path to a server in the optimization result, then the VM will be hosted by the server. In this way, we provide a unified view to solve both optimization problems.

The next challenge is then to determine the capacity of the newly added edges between VMs and servers. Theoretically, a server can sustain a very large amount of traffic between two hosted VMs by the local bus, and therefore the bandwidth capacity constraints are not important for VM-server edges. However, the servers do have memory capacity constraints with the VMs. To reflect such constraints, we create a dummy node for each server, and use an edge to connect them whose capacity will be the memory capacity of the server. We now let the VMs connect to the dummy node instead of the server. Specifically, if a VM can migrate to a server, the VM has an edge with infinite capacity connecting to the dummy node of the server. When search a path for a flow between the dummy node and the server or vice versa, the demand of the flow will be the memory demand of the closer end VM instead of bandwidth demand. In this way, the VMs connected to a server is constrained by the server memory capacity. Figure 2(c) shows the results after adding the dummy nodes to Figure 2(b).

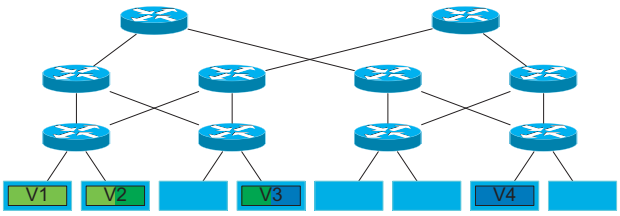
Finally, there is a difference between a VM node and a physical server node. While a server can send different flows to different ToR switches, a VM has to send all its flows through the same physical server. In other words, a VM can select only one of the links connecting to different dummy nodes. If a VM has multiple traffic flows, all of them should share the same path between the VM and the hosting server.

B. Cluster based Parallel Processing

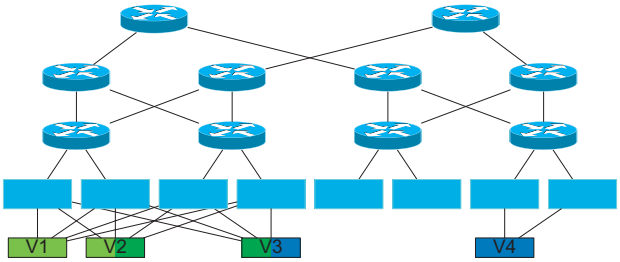
To accelerate processing the huge number of VMs in a data center, we propose a parallelization approach that divides the DCN into clusters to reduce the problem size, and processes the clusters in parallel for fast completion. The design is based on the requirement that live VM migration needs to keep ongoing network connections and therefore the current IP address [21]. Although with existing techniques such as mobile IP, it is possible for an IP address to move into a different subnet (or a foreign network in the term of mobile IP) and keep the ongoing connections, there is an expensive overhead caused by triangle routing [15]. As a result, we assume that a VM will only migrate within its own subnet [21], which consists of a fixed set of servers connecting to the same router interface by the predefined wiring. Note that a VM may not be able to migrate to every server in the subnet because of other constraints, such as different hardware configurations.

The main idea is to organize the servers and VMs in the same subnet as a cluster, and conduct intra- and inter-cluster processing separately at reduced scales. For intra-cluster processing, we find the paths for all traffic flows between the VMs in the cluster, and as a result determine the placement of such VMs. If a VM has only inter-cluster flows, i.e. to VMs in other clusters, we simply calculate its placement according to its memory and bandwidth demands. The reasoning is that the DCN topology is usually symmetric, and VM placement anywhere in the cluster is not likely to affect inter-cluster routing.

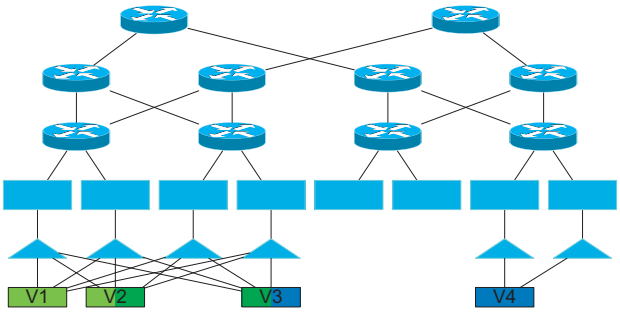
The advantages are two-fold. First, by dividing the problem into a few smaller scale ones, the parallelization approach reduces the solution search space and allows fast completion. Second, since intra-cluster processing of different clusters are



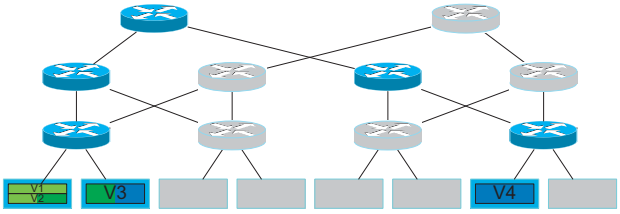
(a) Initial state.



(b) Adding hierarchical layer of VMs.



(c) Adding dummy nodes for memory capacity constraints.



(d) Optimization results.

Fig. 2. Unified representation of VM placement and flow routing.

independent, it can be done in parallel to reduce the total processing time.

V. Host-Network Joint Optimization Scheme

With the above design guidelines, we now present a fast topology oriented multipath routing algorithm to quickly search paths for the intra- and inter-cluster flows. The design utilizes the hierarchical feature of DCN topologies to conduct fast routing. The basic idea is to use depth-first search to find a sequence of best-fit links for the flow. Since a path usually includes links connecting nodes at different layers, depth first search can quickly traverse the layers. If the search has exhausted all the links in a layer and cannot proceed further, it is necessary to backtrack to the previous layer and try the next candidate. For easy description, we define the VMs to be the lowest layer, and the upstream direction to be from a lower layer node to a higher layer one.

When there are multiple available links to the next hierarchical layer, the best-fit criterion selects the one with the best matching capacity, i.e. the smallest and sufficient capacity, so as to consolidate the flows to a few links and improve energy conservation. The first-fit criterion achieves $O(\log N)$ time complexity to select from N links by conducting binary search on a sorted list. Compared with worst-fit that distributes flows to available links in a load balancing way, best-fit maximizes flow consolidation. A concern is then whether the flow consolidation by best-fit will exhaust bandwidth of a specific link, and block a future flow that has to use this link. Fortunately, a switch in a typical DCN has more than one link to switches in the neighboring layers for path redundancy, and therefore the probability for all the links to be unavailable is small. Further, exhaustive depth-first search with backtracking guarantees to explore every possible path, and we observe that all the three selection criteria have similar routing success ratios, close to 100% under reasonable traffic loads, as shown in Section VII.

As the initialization, the scheme uses the unified representation method in Section IV-A to show all the VMs, servers, and switches in a graph. The scheme first processes each cluster, which is the subgraph corresponding to the subnet, and then searches paths between the subgraphs for inter-cluster flows.

A. Intra-cluster Processing

Intra-cluster processing starts with sorting the VMs in the cluster by their memory demands in a decreasing order, for two reasons. First, VM migration consumes energy, which is proportional to the VM memory image size since VM migration copies the VM memory image between servers [8]. By sorting the VM memory demands, we intend to keep the VMs with large memory images intact and move those with small ones. Second, the scheme will use best-fit decreasing for VM placement, since it has better performance than best-fit [9]. This will result in a smaller number of hosting servers.

Next, the scheme searches paths for intra-cluster flows using the depth-first best-fit rule. The scheme picks among the VMs with intra-cluster flows the one with the largest memory demand, and processes its intra-cluster flows one by one. Initially, neither the source nor the destination VM has found a hosting server, the path will include three possible sections: VM-to-server, server-to-switch-to-server, and server-to-VM. Note that if the two VMs migrate to the same server, the path does not need to traverse any switch.

The first step of the path searching is to determine the necessary layer to connect the source and destination VMs. Hosts in DCNs usually have IP addresses corresponding to their topological locations [7]. For example, in a fat tree based DCN, hosts in the same pod usually share the same subnet address [21]. Thus, it is easy to determine by which layer the two edge switches can be connected. Since the network topology and IP address assignment are known in advance, it is appropriate to do the calculation for all IP addresses in advance and store the results, so that they can be quickly obtained during path searching. Determining the connecting

layer avoids wasting bandwidth of switches at higher layers, which will be available for future flows.

After determining the connecting layer, the scheme searches paths for intra-cluster flows using the depth-first best-fit rule. Specifically, starting from the source VM, the scheme searches upstream by selecting the best-fit link to the next higher layer. After reaching the connecting layer, the searching direction turns downstream, similarly by selecting the best-fit link to the next lower layer. For certain topologies, such as the fat tree, the downstream path to a specific server is determined after reaching the connecting layer. Since the depth-first best-fit rule does not guarantee a path on the first try, backtracking with the next candidate or to a higher layer may be necessary. However, the depth-first search guarantees $O(|N|+|E|)$ time complexity [9], where N and E are the node and edge set, respectively. For easy understanding, Table II gives the pseudo code description of the depth-first best-fit search.

Recall that for the VM-to-server section, the capacity between the dummy node represents the available memory capacity of the server and the flow demand is the memory demand of the source VM. If the source VM is connected to multiple servers and they have the same smallest but sufficient capacity, preference will be given to the current hosting server of the VM. If the servers in the cluster are homogeneous in terms of memory and bandwidth configurations, the first VM will stay on its current server, and thus we avoid migrating the VM with the largest memory demand among the ones with intra-cluster flows. Similarly, for the sever-VM section, the link capacity between the server and the dummy node is the server memory space, and the flow demand is the memory demand of the destination VM.

Once the scheme finds the path for an intra-cluster flow, the placement of the source and destination VMs are determined as well based on the VM-to-server and server-to-VM paths. The scheme will thus move the VMs to the selected servers, so that additional flows of the VMs can start from the servers instead.

After the scheme finishes processing a flow, it picks another among the remaining ones of the same VM or the VM with the next largest memory demand. The processing of the newly selected flow is similar. However, since the source and destination VMs of the flow may have been determined, the scheme searches a path between the servers instead of the VMs. The scheme continues the iterations until finishes processing the VMs with intra-cluster flows. For each of the remaining VMs with only inter-cluster flows, the scheme decides only its hosting server based on the memory and bandwidth demands by the best-fit criterion with priority given to the memory demand.

B. Inter-Cluster Processing

Inter-cluster processing searches paths for flows between different clusters, using the same depth-first best-fit rule. After intra-cluster processing is done, all the VMs have found their hosting servers and corresponding ToR switches. Similar as intra-cluster processing, inter-cluster processing first determines the necessary layer to connect the source and destination ToR switches, which is solely based on the network topology and can be calculated in advance. Then, starting from the source ToR switch, the scheme searches upstream by selecting the best-fit link. After reaching the connecting layer, the scheme turns downstream by also selecting the best-fit link. Again, backtracking may be necessary to successfully find a path. Figure 2(d) shows the example optimization results after applying the joint optimization scheme to the DCN in Figure 2(a), where a VM pair with the same color has a flow between them.

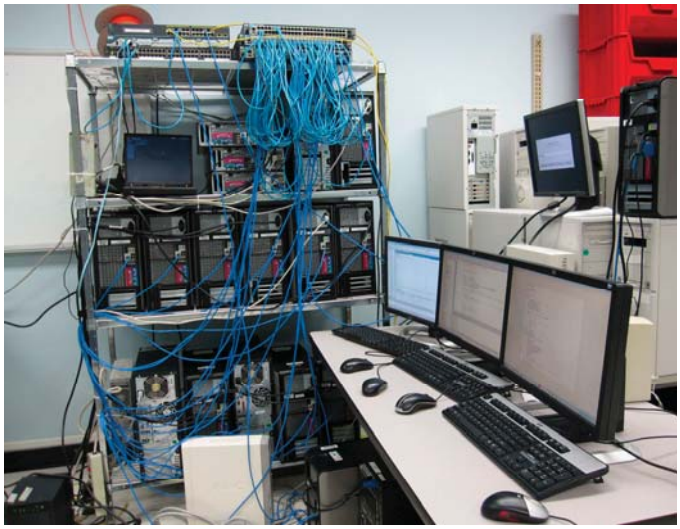


Fig. 3. Photo of our prototype.

VI. Prototype Implementation

In this section, we describe our implementation of the proposed scheme in a prototype using the Beacon OpenFlow controller, HP ProCurve OpenFlow switches, VMware vCenter server, and VMware ESXi hypervisor.

A. Hardware and Software Configuration

We have built a 4-pod and 16-host fat-tree prototype as shown in Figure 3, to demonstrate the effectiveness and practicality of our optimization algorithm in real networks. We use 2 OpenFlow enabled 48-port HP ProCurve 6600 switches running firmware version K.15.06.5008, and create 20 virtual switches. Each virtual switch is assigned 4 ports, except that the first core layer switch has 3 extra ports to allow connections to management nodes, including the VMware vCenter server, Network File System (NFS) server, and DHCP server. All switches are managed by the Beacon OpenFlow controller version 1.0.0 with a self-developed Equinox framework bundle that implements our optimization algorithm. Each host is running the VMware ESXi hypervisor version 5.0.0 to host VMs running operating system of Ubuntu Server 12.04.1 LTS 64 bit. The hosts and VMs are configured to request IP address upon start-up through the DHCP protocol. When the controller detects the DHCP discovery message sent by a host or a VM, it records the host's or the VM's MAC address and location based on which input port of which ToR switch received the

TABLE II
PSEUDO CODE DESCRIPTION OF DEPTH-FIRST BEST-FIT SEARCH.

DFS(G, a, b, d) { // G: network, a: source, b: destination, d: demand
1 $H =$ necessary-layer-to-connect(G, a, b);
2 $path = \{\}$;
3 $u = a$; // temp variable indicating current location
4 $next = 1$; // flag indicating search direction, 1: upstream, -1: downstream
5 return SEARCH($u, path, next$);
}

SEARCH($u, path, next$) {
1 $path = path + u$;
2 if ($u = b$) return $true$;
3 if (layer-of(u) $= H$) $next = -1$; // reverse search direction after reaching connecting layer
4 if ($next = -1$ && layer-of(u) $= 1$) return $false$; // failure at bottom layer
5 $links =$ links of u to layer (layer-of(u) $+ next$) and with available bandwidth $\geq d$;
6 $found = false$;
7 while ($links \neq \emptyset$ && $found = false$) {
8 $v =$ best-fit($links$); $links = links \setminus \{v\}$;
9 $found =$ SEARCH($v, path, next$);
10 };
11 return $found$;
}

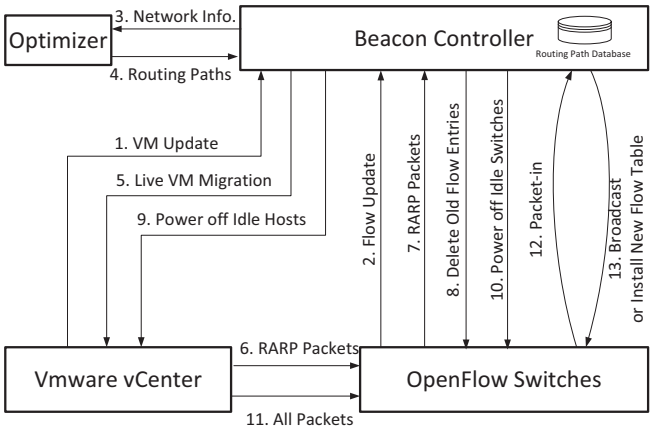


Fig. 4. Major steps of optimization.

message. The IP address of the host or VM is updated when the controller detects the DHCP offer message. All hosts and VMs are remotely managed by the VMware vCenter server version 5.0.0 and the VM's file system is provided by the NFS server.

Iperf UDP flows are employed to emulate the production traffic in data centers. The controller assigns initial routing paths to the flows. The initial routing paths are calculated by using the Shortest Path Routing algorithm. If there exist multiple routing paths from the source to the destination, the controller selects one of them randomly. For each switch on the routing path, the controller also calculates each flow's input and output ports. The controller installs the flow table entries to all the switches on the routing path by sending them `ofp_flow_mod` messages with the flow's matching information and the calculated input and output ports.

B. Optimization

Since the Beacon controller has the view of the entire network, we integrate the optimizer into the controller as one of its Equinox framework bundles. The optimizer can access all the information gathered in the initialization stage. The optimization executes cycle by cycle and each cycle follows the four steps described below. The major exchanged data in one optimization cycle are shown in Figure 4.

1) Network Status Update: Before executing the optimization algorithm, the controller takes a snapshot of the current network. The main purpose of this step is to update the network topology, the VMs information, and the flows information that might have changed between two optimization cycles.

For the switches and links, the controller utilizes the Link Layer Discovery Protocol (LLDP) implemented in the original Beacon controller. The network topology information is stored inside the controller and will be used later in the optimization stage describe in the next subsection. The memory and CPU capacities of each host are fetched from the host management node, i.e., the VMware vCenter server. Specifically, the controller sends `Get-VMHost` command with the host's IP address to the VMware vCenter through the VMware vSphere PowerCLI interface. The VMs location and capacity information can be gathered by using similar methods. For the VM information, the controller sends a `Get-VM` command to the VMware vCenter to update each VM's location, CPU demand and memory demand. For the flow information, the controller sends `ofp_stats_request` messages of type `OFPST_FLOW` to each virtual switch to request the flow statistics. From the statistic reply messages, each flow's average bandwidth demand can be estimated by the flow's duration and total traffic amount. The controller stores the updated information in its database. Note that only the bandwidth of large flows will be stored, since these flows have the major impact on the optimization result.

2) Optimal Network Scheme Calculation: Based on the network status snapshot, the optimizer executes the host-network joint optimization algorithm described in Section V.

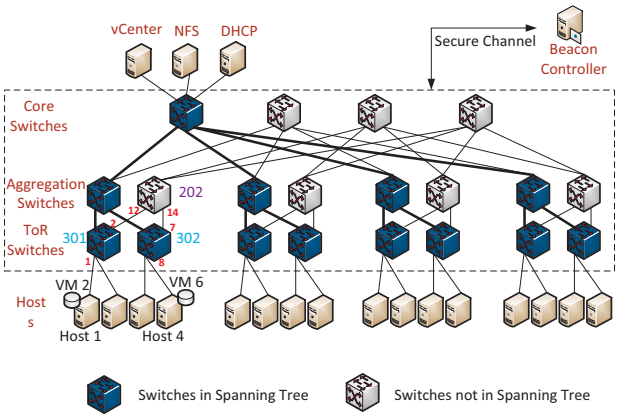


Fig. 5. Fat-tree topology of the prototype.

An optimal network scheme, including the VM's new location and the flow's new routing path, is calculated. The routing path of each large flow are stored in a HashMap dictionary in the controller. The key of each map entry is a string that uniquely identifies each individual flow. The value of each map entry contains the flow table information of each switch on the routing path, including the switch's datapath ID, and the input and output port of the flow. In our prototype, Iperf UDP flows among VMs are the large flows. We concatenate the source VM's MAC address, the destination VM's MAC address and the transport layer port number of the destination VM to form the HashMap key.

For example, one Iperf flow is from VM 2 with MAC address `00:00:00:00:00:02` to VM 6 with MAC address `00:00:00:00:00:06` on transport layer port 5001. First we transform the MAC address to **Long** type and the source and destination MAC addresses become 2 and 6, respectively. Then, the HashMap key should be the string of `"265001"`. As shown in Figure 5, we assume that the optimized locations of VM 2 and VM 6 are on host 1 and host 4, respectively. We also assume that the optimized routing path of this Iperf flow is from ToR switch 301's port 1 to port 2, then from aggregation switch 202's port 12 to port 14 and then from ToR switch 302's port 7 to port 8. Then the routing path of this flow should be `<<301, 1, 2>, <202, 12, 14>, <302, 7, 8>>`.

3) VM Location and Flow Routing Path Adjustment: In this stage, the controller passes the optimization result to the VMWare vCenter which will execute live VM migrations to adjust the VM locations. Then the controller will adjust the flow routing paths accordingly. In out prototype, the live VM migration is implemented by the VMware vMotion migration command `Move-VM`. Switches and hosts will be powered on if they are included in the optimization result.

Upon the completion of each VM's live migration, the VM will broadcast several `RARP` messages to announce its new location. When the controller detects such messages, it will delete flow table entries that are calculated based on the VM's old location. To be specific, the controller sends two messages to each switch to delete all flow entries that are related to the just migrated VM. One message is to delete flow entries whose source is the VM, while the other message is to delete flow entries whose destination is the VM. The two messages have the similar structure. They are both `ofp_flow_mod` messages of command type `OFPFC_DELETE`. The only differences are in the the flow match attribute and in the wildcards attributes. One sets the data link layer source address as the address of the migrated VM, and sets the wild card to `OFPFW_ALL^OFPFW_DL_SRC`. The other one sets the data link layer destination address as the address of the migrated VM, and sets the wild card to `OFPFW_ALL^OFPFW_DL_DST`.

4) Powering Off Idle Hosts, Switch Ports and Switches: In this stage, the controller sends commands to power off idle hosts, idle switch ports, and idle switches. A host is considered idle if it is not hosting any active VM. A switch port is considered idle if no traffic is currently running on the port, and according to the optimization result, no traffic will pass this port before the next optimization cycle. A switch is considered idle if all of its ports are idle. For the idle hosts, the controller sends out `Stop-VMHost` commands through the VMware PowerCLI interface to power them off. For the idle switch ports and idle switches, the controller sends port-down commands and switch power-off commands through the switch's command line interface to power them off, respectively.

The optimization cycle completes when all idle hosts and idle switches are powered off. At this moment, when a packet is sent to the controller to find the routing path, 1) if the packet is a broadcast packet, the controller will ask the switches in the spanning tree to flood this packet; 2) if the packet is a unicast packet and belongs to a large flow with a stored routing path, the controller will fetch the path in the HashMap database and then install flow table entries to switches on the routing path; 3) if the packet is a unicast packet but belongs to a flow without a stored routing path, the controller will calculate a random routing path based on the current topology by using the Shortest Path Routing algorithm.

VII. Simulation and Experiment Results

We have implemented the proposed joint host-network energy optimization scheme in a simulator. In this section, we present the numerical results from the simulations and experiments, to evaluate our design.

A. Simulation Results

We compare the simulation results with the network-only [14], host-only [19], and the linear programing optimization solutions. The simulation results demonstrate that our design outperforms the network- and host-only optimization solutions, and well approximates the ideal linear program.

1) Comparison with Linear Program: First, we compare our joint optimization scheme with the ideal linear program, as well as the network-only optimization solution. We use the 32-bit IBM ILOG CPLEX [1] as the linear program solver. For network-only optimization, we pick the greedy-bin packing

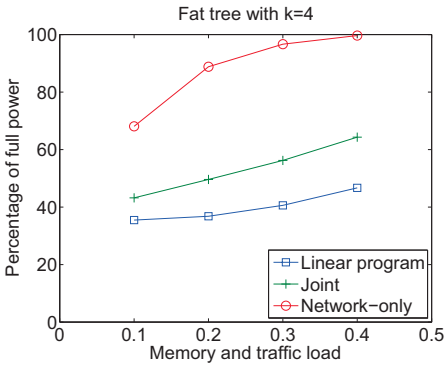


Fig. 6. Comparison with linear program and network-only optimization.

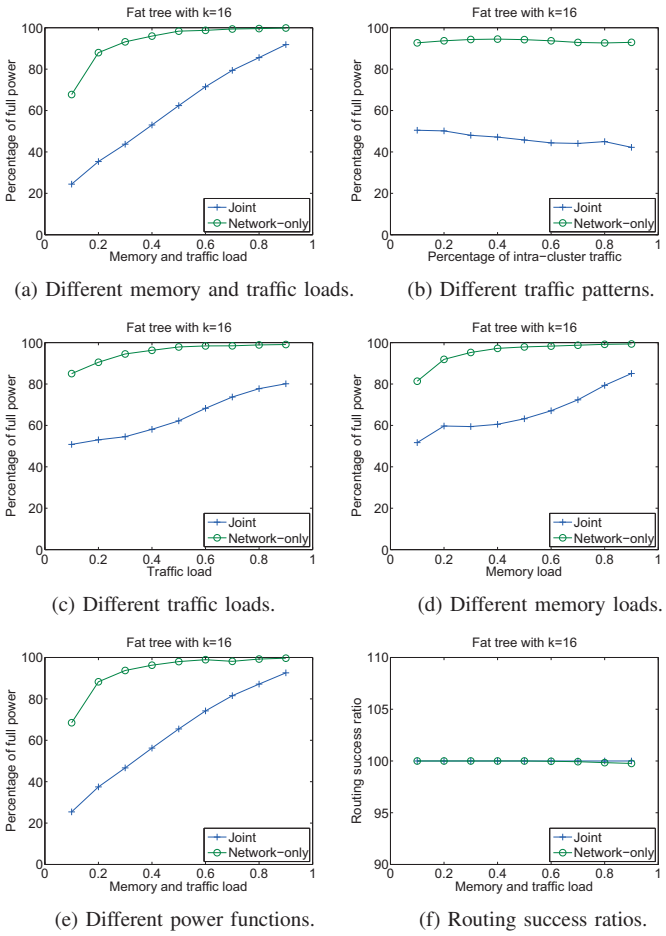
algorithm in [14], because the topology-aware heuristic in [14] uses different assumptions by splitting a flow among multiple paths. We do not include host-only optimization in this subsection, because it assumes that all the VMs have the same memory demand.

Since the linear program is NP-complete, the simulations can only be at small scales. We consider a fat tree with $k = 4$, with the numbers of servers and switches determined as in [5]. Each link has a bandwidth capacity of 1Gbps. Each server has a memory capacity of 8GB. The memory demand of a VM is a random number between 500MB to 1GB, and the number of VMs is determined by the memory load parameter. We restrict that a VM can only migrate to a server connected by the same aggregation layer switch, i.e. within the same pod [21]. Each VM has 2 flows in average with uniformly distributed destinations, and the the flow bandwidth demand is determined by the traffic load parameter. We use Equations 11 and 13 as the switch and server power functions, respectively, with $\alpha(x) = \delta(s) = 200$ and $\beta(x) = \epsilon(x) = 10$, and assume that links are powered by the switches and consume no additional energy.

In Figure 6, we let the memory load equal to the traffic load, which changes from 0.1 to 0.9, and compare the power saving percentages of the three solutions. We can see that our scheme is consistently superior over network-only optimization, up to 40% better. On the other hand, our scheme well approximates the linear program. We tried to solve the linear program with a higher load or a larger network size, but unfortunately CPLEX reported insufficient memory errors due to too many constraints.

2) Comparison with Network-only Optimization: Next, we compare the joint and network-only optimization solutions on a fat tree with $k = 16$. The simulation settings are similar to those in the previous subsection.

In Figure 7(a), we adjust the memory and traffic load, and compare the power saving percentage of the two solutions. Joint optimization consistently outperforms network-only optimization. While the power consumption of network-only optimization increases quickly with the load, joint optimization demonstrates energy proportionality. In Figure 7(b), we fix the memory and traffic load at 0.3 and adjust the percentage of intra-cluster traffic. Joint optimization still performs much



(a) Different memory and traffic loads.
(b) Different traffic patterns.
(c) Different traffic loads.
(d) Different memory loads.
(e) Different power functions.
(f) Routing success ratios.

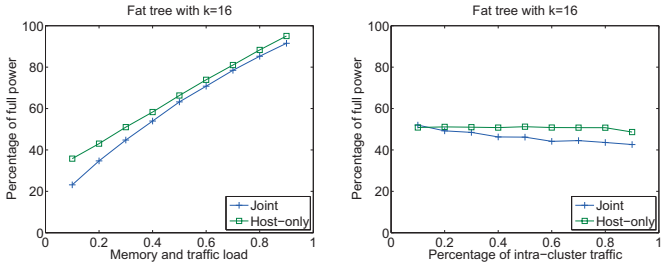
Fig. 7. Comparison with network-only optimization.

better than network-only optimization. As the percentage of intra-cluster traffic increases, we can see that the performance of joint optimization improves, because there are more optimization opportunities for intra-cluster processing. In Figure 7(c), we fix the memory load at 0.5 and adjust the traffic load. The power consumption of both solutions increases with the traffic load, but joint optimization still beats network-only optimization. In Figure 7(d), we fix the traffic load at 0.5 and adjust the memory load, and the conclusion is similar to that in Figure 7(c). In Figure 7(e), we use the following different power functions for switches and servers:

$$\text{switch power} = 200\left(1 + \frac{\text{\# of active powers}}{\text{\# of total ports}}\right) \quad (14)$$

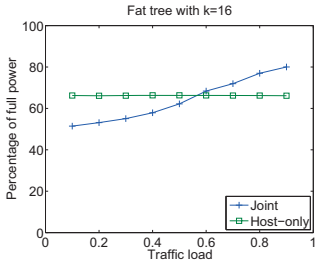
$$\text{server power} = 200\left(1 + \frac{\text{memory of hosted VMs}}{\text{total available memory}}\right) \quad (15)$$

Under the new power functions, joint optimization still performs better than network-only optimization. Finally, in Figure 7(f), we compare the routing success ratio of the two solutions. Although our scheme is not designed to work under heavy loads for a high routing success ratio, it achieves about 100% routing success under reasonably heavy loads, and so does network-only optimization.

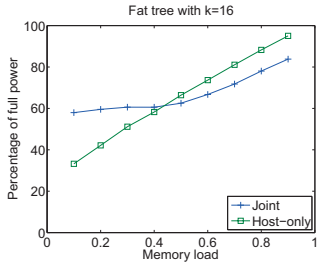


(a) Different memory and traffic loads.

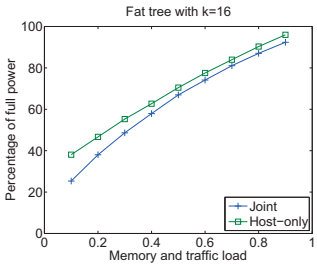
(b) Different traffic patterns.



(c) Different traffic loads.



(d) Different memory loads.



(e) Different power functions

Fig. 8. Comparison with host-only optimization.

3) Comparison with Host-only Optimization: Finally, we compare the joint and host-only optimization solutions on a fat tree with $k = 16$. Since the host-only optimization solution considers VM placement but not flow routing, we use a OSPF like ECMP multipath routing algorithm for it in the simulations. Because host-only optimization assumes fixed VM memory demands, we use a value of 0.8GB.

In Figure 8(a), we adjust the memory and traffic load, and compare the power saving percentage of the two solutions. Both solutions demonstrate similar and proportional power efficiencies, with joint optimization having a slight advantage. In Figure 8(b), we fix the memory and traffic load at 0.3 and adjust the percentage of intra-cluster traffic. Joint optimization is still better than network-only optimization. Similarly, the performance of joint optimization improves with the increase of intra-cluster traffic, because there are more optimization opportunities. In Figure 8(c), we fix the memory load at 0.5 and adjust the traffic load. Joint optimization is initially better than host-only optimization, but becomes worse after the traffic load is greater than 0.6. This implies that host-only optimization does a better job in terms of VM placement. However, we should also notice that, host-only optimization has higher time complexity as the cost. On the other hand, the performance of host-only optimization is not sensitive to the change of traffic load, because it optimizes only VM placement. In Figure 8(d), we fix the traffic load at 0.5 and adjust the memory load. Joint optimization is initially worse than host-only optimization, but becomes better after the memory load is grater than 0.5. It is interesting to note that the performance of host-only optimization is almost linear to the memory load. Finally, in Figure 8(e), we use the second set of switch and server power functions, and joint optimization still outperforms host-only optimization.

B. Experimental Results

We have conducted experiments in our prototype to evaluate the performance of the optimization scheme. In this subsection, we first describe the experimental setup, and then present the results to demonstrate that the optimization scheme is effective and practical.

1) Experiment Configuration: Two experiments are conducted. In the fist experiment, the average memory and traffic load of the prototype is set to 15%, while in the second experiment, the load is set to 30%. VMs and network flows are generated according to the load and the following rules. Each VM's memory is randomly selected between 250 MB and 500 MB. The initial location of each VM is randomly selected among the hosts. We adjust the number of VMs to meet the host utilization goal of each experiment. Each VM is configured to send out one Iperf UDP flow in average. The normal size of the flows are randomly selected between 20 Mbps and 250 Mbps and we adjust the flow size to meet the network utilization goal of each experiment if necessary. For the experiment with 15% memory and traffic load, 21 VMs and 24 Iperf UDP flows are generated. For the second experiment with 30% memory and traffic load, 42 VMs and 49 Iperf UDP flows are generated.

Both experiments run the following steps. Initially all switches, hosts and VMs are powered on and all flows are started. We measure and record the current power consumption. Then, we run one full optimization cycle. After the optimization completes, we measure the power consumption again and compare it with the original value. Note that only the power consumption of the hosts are measured both before and after the optimization. This is because that the physical switch has large power consumption overhead, and the power consumption of each virtual switch cannot be accurately measured. We use the Kill-A-Watt power meter model P4320 to measure the power consumption.

2) Experimental Results - Routing Path Control: In order to verify whether the optimization is able to adjust flow's routing paths correctly, we study the traffic amount of the hosts to see if they are as same as expected. Due to space limitations, we only present the result of Host 4 in this paper. The results of other hosts follow the similar pattern and lead to the same conclusion.

Table III(a) and (b) give the detailed configuration of the flows and traffic amount of Host 4, before and after the optimization, respectively. The VMs with bold name are the ones hosted by Host 4. Thus, before the optimization, only VM 7 is hosted by Host 4; after the optimization, 3 additional

TABLE III
FLOW CONFIGURATION AND TRAFFIC AMOUNT OF HOST 4

Flow Direction	Flow Bandwidth (Mbps)	Outgoing Traffic Amount of Host 4 (Mbps)	Incoming Traffic Amount of Host 5 (Mbps)
VM 4 → **VM 7**	83.7	-	83.7
VM 7 → VM 19	90.1	90.1	-
Total	**173.8**	**90.1**	**83.7**

(a) Before Optimization

Flow Direction	Flow Bandwidth (Mbps)	Outgoing Traffic Amount of Host 4 (Mbps)	Incoming Traffic Amount of Host 5 (Mbps)
VM 4 → **VM 7**	83.7	-	-
VM 4 → VM 17	27.8	27.8	-
VM 7 → **VM 19**	90.1	-	-
VM 9 → VM 3	26.8	26.8	-
VM 11 → **VM 9**	22.7	-	-
VM 11 → VM 20	56.0	56.0	-
VM 19 → VM 3	119.1	119.1	-
Total	**396.2**	**229.7**	**0**

(b) After Optimization

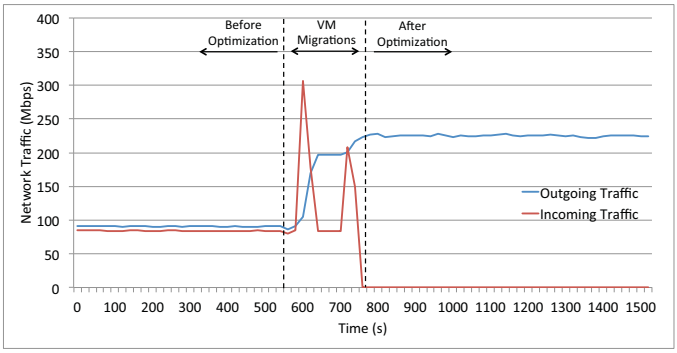
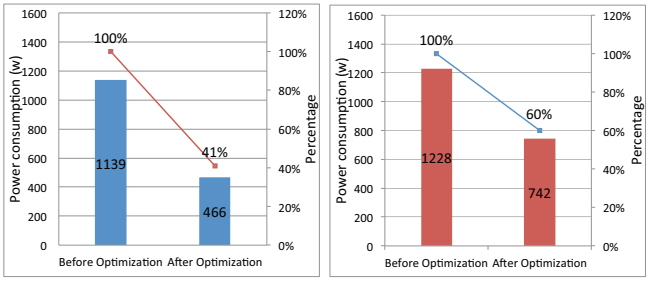


Fig. 9. Measured incoming and outgoing traffic of Host 4.



(a) Memory and traffic load = 15%. (b) Memory and traffic load = 30%.

Fig. 10. Power consumption comparison before and after optimization.

VMs are hosted by Host 4, including VM 4, VM 11, and VM 19. If either a flow's source or destination VM is on Host 4, this flow accounts Host 4's network traffic. The traffic's direction and amount is the same as the flow's direction and bandwidth, respectively. For example, after the optimization, the flow from VM 4 to VM 17 accounts 27.8 Mbps of Host 4's total outgoing traffic. If both the source and destination VMs of a flow are on Host 4, the flow does not account any of Host 4's traffic. One example is the flow from VM 7 to VM 19 after the optimization. The total outgoing traffic amount and total incoming traffic amount of Host 4 should be the summation of each flow's outgoing traffic amount and the summation of each flow's incoming traffic amount, respectively.

Figure 9 shows the measured total outgoing and the measured total incoming traffic of Host 4 before and after the optimization. We can find that, before the optimization and the VM migrations, the amount of outgoing traffic and the amount of incoming traffic of Host 4 are the same as the calculated amount shown in Table III(a). During the optimization, Host 4's incoming traffic amount changes dramatically. This is due to the VM migration traffics that has Host 4 as the destination. After the optimization, the total outgoing traffic of Host 4 stabilizes at around 225 Mbps which has less than 2% difference as the calculated amount shown in Table III(b). The incoming traffic amount of Host 4 after the optimization drops to zero which is the same as calculated in Table III(b). In summary, by comparing the measured traffic amount with the calculated traffic amount, we show that the optimization can adjust the flow's routing paths correctly.

3) Experimental Results - Power Consumption Reduction: Figure 10(a) and 10(b) illustrates the comparison of the power consumptions before and after the optimization when system load is equal to 15% and 30%, respectively. Both figures show that the power consumption after the optimization is lower than before the optimization. Specifically, Figure 10(a) shows that when system load is equal to 15%, the total power consumption drops from 1139W to 466W or from 100% to 41%. In other words, the optimization saves 59% of the original power consumption. Figure 10(b) shows that when system load is equal to 30%, the total power consumption drops from 1228W to 742W or from 100% to 60%. In other words, the optimization saves 40% of the original power consumption. It is worth noting that the optimization yields larger reduction of the power consumption when system load is lighter. This is because that, when the system load is lighter, VMs will be consolidated into fewer number of hosts and thus more idle hosts can be powered off to further reduce the power consumption. The experiment results have demonstrated that our joint host-network optimization is an effective and practical method to improve the data center's energy efficiency.

VIII. DISCUSSIONS

In this section, we discuss some practical issues in the joint host-network optimization implementation.

A. Broadcast in OpenFlow Networks

Traditional switches flood broadcast packets to certain ports, while other ports are blocked by the spanning tree protocol such that no loop exists in the network. But since the spanning tree protocol may conflict with the controller, it is disabled on OpenFlow switches by default. Thus it is challenging to enable broadcast in networks that have loops. To solve such problem, we construct a spanning tree and put it in the controller instead of on the switches. When the controller detects broadcast packets, it asks only the switches in the spanning tree to flood the packets.

B. Energy Consumed by VM Migration

It should be noted that while we try to reduce energy consumption by migrating VMs, the operation of VM migration itself also consumes energy, but it is fortunately a one-time overhead. Since VM migration is essentially to copy the VM memory image between servers, its energy consumption can be estimated by by the size of the VM memory image.

C. Safety Margins

While our design uses the best-fit criterion to maximize flow consolidation, it is necessary to leave certain safety margins for redundancy and traffic burst. This can be done by leaving a certain percentage of the link capacity untouched in the optimization process, and it will be our future work to determine a reasonable percentage value.

IX. Conclusions

The data center network (DCN) has become a major fraction of the energy consumption of a data center. In this paper, we propose a joint host-network optimization scheme to improve the energy efficiency of DCNs. First, we present a unified representation method to convert the virtual machine placement problem to a routing problem, so that a single solution can apply to both types of optimization. Next, we describe a parallelization approach that divides the DCN into clusters based on subnet IP addresses, and processes the clusters in parallel for fast completion. Further, we propose a fast topology oriented mutipath routing algorithm that can quickly find paths by using depth-first search to traverse between the hierarchical layers, and maximize energy conservation by using the best-fit criterion to consolidate flows. Finally, we build an OpenFlow based prototype, and have conducted extensive simulations and experiments to evaluate the performance of our design. The simulation and experiment results demonstrate that our design is effective and practical, and it is also superior over existing host- or network-only optimization solutions, and well approximates the ideal linear program.

References

[1] "IBM ILOG CPLEX Optimizer," http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[2] "U.S. Environmental Protection Agencys Data Center Report to Congress," http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.

[3] "Who Has the Most Web Servers?" http://www.datacenterknowledge.com/archives/2009/10/13/facebook-now-has-30000-servers/.

[4] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *ACM ISCA*, Saint-Malo, France, Jun. 2010.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[7] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Generic and automatic address configuration for data center networks," in *ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *USENIX NSDI*, Berkeley, CA, Apr. 2005.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[10] W. Fisher, M. Suchara, and J. Rexford, "Greening backbone networks: reducing energy consumption by shutting off cables in bundled links," in *ACM SIGCOMM Workshop on Green Networking*, New Delhi, India, Aug. 2010.

[11] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[12] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: scalability and commoditization," in *ACM SIGCOMM PRESTO*, Seattle, WA, Aug. 2008.

[13] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey, III, and M. Neufeld, "Policies for dynamic clock scheduling," in *USENIX OSDI*, San Diego, CA, Oct. 2000.

[14] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. Mckeown, "Elastictree: saving energy in data center networks," in *USENIX NSDI*, San Josa, CA, Apr. 2010.

[15] J. Kurose and K. Ross, *Computer networking: a top-down approach (4th Edition)*, 4th ed. Addison Wesley, 2007.

[16] P. Mahadevan, S. Banerjee, and P. Sharma, "Energy proportionality of an enterprise network," in *ACM SIGCOMM Workshop on Green Networking*, New Delhi, India, Aug. 2010.

[17] P. Mahadevan, A. Shah, and C. Bash, "Reducing lifecycle energy use of network switches," in *ISSST*, 2010.

[18] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ASPLOS*, Washington, DC, Mar. 2009.

[19] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *IEEE INFOCOM*, San Diego, CA, Mar. 2010.

[20] J. Mudigonda and P. Yalagandula, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," in *USENIX NSDI*, San Josa, CA, Apr. 2010.

[21] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer2 data center network fabric," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[22] C. Patel, C. Bash, R. Sharma, M. Beitelmam, and R. Friedrich, "Smart cooling of data centers," in *InterPack*, Maui, HI, Jul. 2003.

[23] S. Pelley, D. Meisner, T. F. Wenish, and J. W. VanGilder, "Understanding and abstracting total data center power," in *ISCA 2009: Workshop on Energy Efficient Design (WEED)*, Jun. 2009.

[24] G. Schaffrath, S. Schmid, and A. Feldmann, "Optimizing long-lived cloudnets with migrations," in *Proceedings of 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2012.

[25] Y. Shang, D. Li, and M. Xu, "Energy-aware routing in data center network," in *ACM SIGCOMM Workshop on Green Networking*, New Delhi, India, Aug. 2010.

[26] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *INFOCOM*, Shanghai, China, Apr. 2011.

[27] M. Zhang, C. Yi, B. Liu, and B. Zhang, "Greente: Power-aware traffic engineering," in *IEEE ICNP*, Koyoto, Japan, Oct. 2010.