Joint flow routing-scheduling for energy efficient software defined data center networks

A prototype of energy-aware network management platform

Zhu, H.; Liao, X.; de Laat, C.; Grosso, P.

DOI
10.1016/j.jnca.2015.10.017
Publication date
2016
Document Version
Final published version
Published in
Journal of Network and Computer Applications
License
Article 25fa Dutch Copyright Act

Link to publication

Journal of Network and Computer Applications 63 (2016) 110–124

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Joint flow routing-scheduling for energy efficient software defined data center networks

A prototype of energy-aware network management platform

Hao Zhu [a,*], Xiangke Liao [b], Cees de Laat [a], Paola Grosso [a]

[a] System and Network Engineering, University of Amsterdam, The Netherlands
[b] Department of Computer, National University of Defense Technology, Changsha, China

ARTICLE INFO

Article history:
Received 20 March 2015
Received in revised form
3 August 2015
Accepted 5 October 2015
Available online 9 February 2016

Keywords:
OpenNaaS
Energy monitoring
Energy-aware routing
SDN

ABSTRACT

Data centers are a cost-effective infrastructure for hosting Cloud and Grid applications, but they do incur tremendous energy cost and CO_2 emissions. Today's data center network architectures such as Fat-tree and BCube are over-provisioned to guarantee large network capacity and meet peak performance requirement. Networks suffer from inefficient power usage when data center traffic is not high. A solution to this problem is the adoption of network management platform such as OpenNaaS, which can be augmented with energy-aware capabilities. We developed a component for energy monitoring and routing in OpenNaaS. Energy-aware OpenNaaS can support different types of OpenFlow controller; it inherits and enhances network management capabilities, e.g. dynamically obtaining power and topology.

In this paper we also discuss the evaluation and selection of energy-aware routing strategies based on an initial prototype of energy-aware OpenNaaS. The target strategies are fine-grained as they combine flow routing algorithms that make routing decisions for the flows and flow scheduling algorithms that schedule the flows on the same link. Differently from previous routing work which focuses on power-minimization problem in data center networks, we aim to optimize energy consumption. Our simulation shows that the combination of priority-based shortest routing and exclusive flow scheduling achieves about 5%–35% higher energy efficiency without performance degradation.

1. Introduction

Data centers are a cost-effective infrastructure for hosting Cloud and Grid applications but they do incur tremendous energy cost and CO_2 emissions. In the total energy footprint of data centers, the network component has rarely been considered the most relevant component for optimization, given its lower contribution (only 10–20% of the total power Greenberg et al., 2008) in respect to other components such as servers and cooling. However, the proportion of energy consumed by the network components can be up to 50% if optimal power management techniques are used on the server-side (Abts et al., 2010). This occurs particularly in scientific data centers where large volumes of data are frequently transported. It is therefore crucial to reduce the energy consumption of networks in data centers.

Advanced network architectures in data centers such as Fat-tree (Al-Fares et al., 2008) and BCube (Guo et al., 2009) are usually over-provisioned, with full-connected topologies and multi-path routing to guarantee large network capacity and high robustness. A large number of network resources are used to meet performance requirements at peak times. However, these resources are usually underused and rarely work at the peak performance. Unfortunately, networks in a data center are not power proportional – networks with low loads still consume more than 90% of the power used during the busiest hours (Heller et al., 2010). They effectively suffer from inefficient power usage when traffic is not heavy.

Energy-aware routing techniques are effective approaches that can fix this problem. They are in essence strategies which focus on the energy state of the network, e.g. energy consumption or CO2 emission rate. They make routing decisions to aggregate traffic over a subset of links and devices in over-provisioned networks and switch off unused network components.

There are two ways to implement energy-aware routing: one is in IP networks, the other modality is to focus on data centers using Software Defined Networking (SDN). In IP networks, we can for example observe the development of a Green OSPF protocol (Cianfrani et al., 2010). In SDN, in particular when looking at

* Corresponding author. Tel.: +86 17077430374.
E-mail addresses: h.zhu@uva.nl, haozhu@nudt.edu.cn (H. Zhu).

networks adopting OpenFlow (McKeown et al., 2008), control plane (routing decision) decoupled from data plane (data forwarding) is moved to a centralized controller. Energy-aware routing could be easily implemented in the control plane. We work on data centers using Software Defined Networking in this paper. The question is what the most effective manner is to achieve this.

Previous studies, which we will further discuss in Section 3, implemented energy-aware routing using various OpenFlow controllers, such as NOX (NOX, 2014) and Beacon (Ericsson, 2013). These implementation has two drawbacks. First, neither implementation can easily be migrated between SDN networks with different types of SDN controllers because of incompatible structure and APIs. Second, energy-aware routing techniques rely on precise information about current topology and traffic, but the capabilities of the OpenFlow controllers are usually limited in their ability to obtain this information, e.g. NOX can only discover the network topology, and most of the controllers cannot obtain topology or traffic statistics. Existing network management platforms can support multiple types of controller, and this is quite useful for multi-domain environment. They have no compatibility problem when integrating new energy-optimizer modules into them. They can provide the capabilities missing from the controllers. Considering these factors as well as the extensible structure and powerful network management capabilities of OpenNaaS, we decided to concentrate on OpenNaaS as a suitable platform for the implementation of energy-aware routing in SDN.

We used the features of OpenNaaS to provide the capability to monitor energy usage and make energy-aware routing decisions. Monitoring capabilities provide the data basis for the decision-making of energy-aware routing as well as other power management techniques in networks, e.g. adaptive link rate (ALR) (Gunaratne et al., 2005). A previous information model is used to define data elements and their structure for energy monitoring. Green routing capabilities can calculate energy-aware paths and push the paths to OpenFlow switches. We implemented an initial prototype of energy-aware OpenNaaS and validated its functionality using greedy routing algorithm.

Same as greedy routing, previous energy-aware routing algorithms solve the subset of networks for saving power consumption. Considering that energy consumption equals to power consumption multiplied by time, they could not be energy efficient because they neglect the scheduling of flows on the same link in their consolidated networks and the overall time of transferring traffic flows. For the future version of energy-aware OpenNaaS, we will implement a better routing strategy which can save more energy than these algorithms. Before that, we aim to determine this optimized strategy by evaluating the effect of different routing strategies on energy consumption and network performance by theoretical analysis and simulation experiments.

Our target routing strategies which are more fine-grained integrate the routing algorithms and scheduling algorithms. They are the solutions to optimize energy consumption. Specifically, the major contributions of this paper are as follows:

- To easily implement energy-aware routing across different types of OpenFlow controller, we developed an energy-aware component in an open-sourced network management platform – OpenNaaS. Energy-aware OpenNaaS can be used for energy monitoring and routing, and it has powerful network management function.
- To study the effect of different scheduling algorithms on energy optimization, we theoretically prove the exclusive scheduling of flows on the same link is the most energy efficient.
- To further optimize energy consumption of DCNs, we discussed and evaluated a set of routing strategies which integrate common routing algorithms and scheduling algorithms. Our simulation experiments show that the combination of priority-based shortest routing and exclusive flow scheduling can achieve 5%–35% higher energy efficiency than using other common routing strategies without performance degradation when traffic consists of large-sized (5 GB) of flows.

The structure of this chapter is as follows: Section 2 presents the energy-aware routing problem; we follow with a section detailing the related work on energy-aware routing techniques and software management platforms (Section 3). Section 4 describes the OpenNaaS framework, while Section 5 introduces the design of energy-aware OpenNaaS. Section 6 provides a practical usecase of energy-aware OpenNaaS. After that, Section 7 discusses the design and selection of energy-aware strategies and Section 8 evaluates the strategies by simulation. Sections 9 and 10 present discussion and conclusions.

2. The energy-aware routing problem

The power consumption of a switch is the sum of static power and dynamic power, according to the power benchmarking results of various network devices in Mahadevan et al. (2009) and Zhu et al. (2015). Static power is constant and it includes power consumed by chassis, fabric, fans, etc. Dynamic power is consumed by device interfaces and is related to the rate of traffic across the switch.

If we denote $P(u)$ as the power consumption of a switch, this will depend on the set of enabled interfaces and the load of each one of them:

$$P(u) = P_{base} + \sum_{j=1}^{N} a_j \cdot p_j(u_j) \tag{1}$$

$$u_j = \sum_{k=1}^{K} u_{j,k}, \quad 0 \le u_j \le C \tag{2}$$

where P_{base} is the static power; N is the number of links on the switch; $p(u)$ is the power consumption of a switch interface at utilization u; a_j is the binary decision variable indicating whether the interface j is powered on. Given K traffic flows $f_1, f_2, \ldots, f_k, \ldots, f_K$, the utilization of f_k along the link j is $u_{j,k}$; while the capacity of the link is C.

Multipath routing algorithms (Lin et al., 2014), such as equal-cost multipath (ECMP), use multiple paths while sending flows, but do suffer from longer delay and additional control messages. Splitting flows is typically undesirable due to TCP packet reordering effects (Kandula et al., 2007). In our study we assumed that no flow is split onto multiple paths, denoted by:

$$\forall k, \text{ if } u_{j',k} > 0, \quad \sum_{j=1, j \ne j'}^{N} u_{j,k} = 0 \tag{3}$$

Energy-aware routing is an effective solution to save energy by aggregating the traffic over a subset of network links or network devices in over-provisioned networks. Solving the energy-aware routing problem is equivalent to minimizing energy consumption of a network:

$$\text{Min} \sum_{i=1}^{M} b^i \cdot P^i(u) \cdot t^i(u) \tag{4}$$

where $P^i(u)$ denotes the power consumption of switch i at utilization u; M is the number of switches in the network; b^i is the binary decision variable indicating whether switch i is powered on; $t^i(u)$ denotes the time used for transferring the traffic by the switch i at utilization u.

Most previous energy-aware routing studies assume the completion time $t^i(u)$ is constant. In fact, they simplify the energy-aware routing problem as Eq. (5) to find a power-minimized network subset for traffic:

$$Min \sum_{i=1}^{M} b^i \cdot P^i(u) \tag{5}$$

For a given traffic, the completion time $t^i(u)$ is variable for different subsets of a network. It is obvious that the solution to the power-minimization problem could be not energy efficient. Even so, finding the optimal flow routing for power-minimization is an NP-hard problem (Heller et al., 2010; Xu et al., 2013). It is difficult to solve this problem using a formulation, in particular for large scale networks. The universal solutions are to compute a path for each flow and generate a subset of network by combining all the result paths. We will discuss the state of the art for energy-aware routing next section.

3. Related work

3.1. Energy-aware routing

Cianfrani et al. (2010) proposed a Green OSPF protocol. Its energy-aware algorithm only uses a subset of Shortest Path Trees to select the routing, reducing the number of links used to route traffic. Bianzino et al. (2012) designed a link-state protocol with a fully distributed solution to save the power consumption of links. The above studies focus on the implementation of energy-aware routing in IP networks.

Existing work that implements energy-aware routing algorithms for data center networks (DCNs) using SDN is mostly theoretical, and only a few cases exist of actual implementations.

Several authors have tackled the problem theoretically. Wang et al. (2012) analyzed the correlation between flows, and they found that traffic flows within a data center are usually loosely correlated together. The flows usually do not reach peak capacity at the same time. Their correlation-aware routing algorithm greedily consolidates as many weak-correlation flows as possible onto a path. They further adapted the data rate of each link to save power as links may not be fully utilized in the routing algorithm. Their approach is not suitable for data-intensive data centers which have highly correlated flows. They did not discuss the trade-off between energy optimization and network quality of service (QoS). Power can be saved by migrating Virtual machines (VMs) to a smaller set of servers and powering off unused servers in data centers (Ahmad et al., 2015). Zheng et al. (2014) combined Wang's work with VM consolidation for further power saving.

Fang et al. (2012) proposed an energy-aware scheme with multipath routing. The scheme first selects a minimal subset of network elements by using the Steiner tree framework, and then uses a multipath routing algorithm to find multiple routes for each flow. Finally, it combines the routes that need to activate as few unselected elements as possible along with the initial subset to generate the final network subset. Their multipath routing for each flow can improve network throughput. But the heuristic algorithm they used for the Steiner tree problem needs long computation time. Their work is lack of a mechanism to achieve trade-off between energy optimization and QoS.

Xu et al. (2013) and Shang et al. (2012) proposed a power-aware routing algorithm for reducing power consumption of high-density data center networks while meeting the overall throughput requirement. The idea of the algorithm is to compute a basic routing path set for all flows and then to remove switches and links from the path set without violating the demand of overall network throughput. Its power saving is over 55% than that of using ElasticTree under the 30% network load. But it does not consider the transition time between sleep and wake-up for switches.

Wang et al. (2014) use multi-objective evolutionary algorithm for route selection in dynamic optical networks. Compared to Xu et al. (2013) and Shang et al. (2012), their approach supports multiple QoS requirements in terms of network performance, e.g. throughput, delay and blocking rate. It further improves the energy performance of high priority traffic without degrading QoS by taking energy saving as the secondary objective. But it cannot support network reliability in the QoS requirements.

The decision-making for energy-aware routing and control in SDN are centralized, and this mechanism is not scalable for large networks. Liu et al. (2013) proposed a distributed strategy in each point of delivery (PoD) of a DCN to find the routing path for elephant flows and make tradeoff between energy consumption and network performance. They observed that elephant flows easily lead to network congestion, so their elephant flow routing greedily selects the path with the least network utilization caused by the running elephant flows to improve network performance. Their algorithm powers off the unused network components but its route selection is not power-aware and it saves less power.

Others have provided actual implementations and prototypes. Heller et al. (2010) presented ElasticTree for adapting the power usage in a Fat Tree data center with OpenFlow switches. ElasticTree uses NOX that is an original OpenFlow controller to pull traffic data and push computed flow routes to each switch; it employs an optimizer to compute power-minimization routes which meet current traffic conditions. ElasticTree proposed and evaluated two algorithms: (1) Formal model solves power-minimization problem directly; (2) Greedy bin-packing evaluates possible paths and chooses the leftmost one with sufficient capacity for each flow. Unfortunately, formal model is not scalable for medium or large networks and greedy bin-packing saves less power. Mahadevan et al. (2011) combined the routing algorithm used in Elastic Tree and the algorithm of server load consolidation that migrates jobs to use fewer servers for data center networks. They found that 74% percent of total network power can be saved.

Thanh et al. (2012) presented a platform to measure and analyze the power consumption of software-defined DCNs using energy-aware topology optimization and routing. Topology optimization first calculates the number of active links and switches based on traffic statistics. Then the platform leverages the same routing algorithm with ElasticTree in POX controllers based on the power profiling of NetFPGA switches. Thanh et al. (2013) improved the previous routing algorithm to enable an adaptive link rate technique. In order to further save power in data centers, Jin et al. (2013) converted the VM placement problem into a routing problem, which combined hosts and network based power optimization. They use depth-first search to quickly traverse the hierarchical layers between VM pairs in a DCN, and employ a best-fit criterion in terms of memory demand of VMs to determine the link between any two layers. Their prototype integrates the depth-first best fit rule into the Beacon OpenFlow controller. This work ignores the cost of VM migration. None of the three work here considers the impact of the proposed approaches on network QoS.

A summary of the discussed energy-aware routing algorithms is demonstrated in Table 1. Our work is different from these previous work in terms of algorithms and implementations. The studies above focus on power minimization problem (See Eq. (5)). They neglect the scheduling of flows on the same link in consolidated networks. Li et al. (2014) studied an energy efficient scheduling algorithm, and observed that exclusively transferring flows on the link is more energy efficient than fair share scheduling, however it has no analysis of different routing algorithms.

Table 1
Summary of energy-aware routing algorithms.

Work	Prototype	Description	Pros & Cons
Wang et al. (2012)	–	CARPO consolidates as many weak-correlation flows as possible onto a path; It leverages ALR.	It focuses on different flows and saves 17.7% more energy than ElasticTree. It is not suitable for data centers with highly-correlated flows; it is lack of a mechanism to achieve trade-off between energy optimization and QoS.
Fang et al. (2012)	–	It computes a minimal node set by using the Steiner tree framework; then it uses a multipath routing algorithm to find multiple routes for each flow; finally it combines the routes that need to activate as few unselected elements as possible along with the initial subset to generate the final network subset.	Multipath routing for each flow can improve network throughput. An heuristic algorithm for the Steiner tree problem needs long computation time; the algorithm is lack of a mechanism to achieve trade-off between energy optimization and QoS.
Xu et al. (2013), Shang et al. (2012)	–	It computes a basic routing path set for all flows and then removes switches and links from the path set without violating the demand of overall network throughput.	It guarantees the network throughput; its power saving is over 55% than that of using ElasticTree under the 30% network load. It does not consider the transition time between sleep and wake-up for switches.
Wang et al. (2014)	–	It uses multi-objective evolutionary algorithm for route selection in dynamic optical networks	It supports multiple QoS requirements in terms of network performance; it further improves the energy performance of high priority traffic. It does not consider network reliability in the QoS requirements.
Liu et al. (2013)	–	Distributed schedulers interact a set of switches and decides the next hop to forward a flow; it greedily select the path with the least network utilization for elephant flows.	Distributed control can reduce the network delay and have high scalability. Its route selection is not power-aware and it saves less power.
Heller et al. (2010)	NOX	Two approaches in ElasticTree: (1) Formal model solves power-minimization problem directly; (2) Greedy bin-packing evaluates possible paths and chooses the leftmost one with sufficient capacity for each flow.	It proposes and evaluates multiple approaches. Formal model is not scalable for medium or large networks and greedy bin-packing saves less power.
Thanh et al. (2012), Thanh et al. (2013)	POX	It calculates the number of active links and switches based on traffic statistics; then it implements the same routing algorithm with ElasticTree but enables an ALR technique.	It focuses on different flows; it has a fine-grained optimization of energy consumption. It doesn't consider the impact of the proposed approach on network QoS.
Jin et al. (2013)	Beacon	It uses depth-first search to quickly traverse between hierarchical switch layers and then it employs a best-fit criterion in terms of memory demand of VMs to determine the link between any two layers.	It focuses on both host and network optimization for energy efficient DCNs. It ignores the cost of VM migration; it doesn't consider the impact of the proposed approach on network QoS.

The energy-aware routing strategies we evaluated are complete by integrating routing algorithms and scheduling algorithms. Our evaluation targets at the original energy-aware routing problem concerning energy consumption (See Eq. (4)).

Besides, we implemented a prototype based on OpenNaaS. Our energy-aware component is implemented in this network management platform rather than only based on SDN controllers. So energy-aware OpenNaas is highly-compatible because it supports multiple types of SDN controller. Previous systems, for example ElasticTree, cannot dynamically discover the topology and monitor the power consumption of switches and they are blind about the changing of networks; they assume that the topology and power are always invariable once they are manually input. Energy-aware OpenNaaS inherits and enhances powerful network management capabilities, e.g. dynamically obtaining power and topology information.

3.2. Cloud/network management platform

Given that OpenNaaS is effectively a full network platform, it is fair to compare it with existing cloud/network management platforms that also support SDN technologies.

OpenNebula and OpenStack are both open-source cloud computing platforms for public and private clouds. Cloud operators can control computing, storage and network resource in clouds through their APIs. Their network capabilities are limited to IP, vLANs and SDN currently. OpenNaaS has more sufficient network capabilities, e.g. Bandwidth on Demand (BoD), topology discovery, and Generic Routing Encapsulation (GRE) and support more use-cases, e.g. virtual Customer Premises Equipment (vCPE) than OpenNebula and OpenStack. We can implement more energy-aware capabilities like energy-aware BoD by combining energy monitoring with existing network capabilities in OpenNaaS. Besides, OpenNaaS has a well-organized structure to enable the abstraction of underlying network technologies and resources, easily extended to implement new network technologies.

Nuage Networks (NUAGE, 2014) and the Tail-f Network Control System (TAIL-F, 2014) both provide general SDN capabilities and vCPE solutions for data center networks. They offer a virtualization environment for a data center network, but these software-based products are not open source, and as such cannot be easily extended.

RouteFlow (Nascimento et al., 2011) provides remote IP routing services based on a set of open-source software. RouteFlow does not make a routing decision, which depends on a virtualized IP routing engine, Quagga, that provides the implementation of routing protocols, e.g. OSPF and BGP (Nascimento et al., 2010). RouteFlow only focuses on routing services and its structure is not easily extensible for new network functionality.

4. OpenNaaS framework

OpenNaaS (OPENAAS, 2014) is the outcome of the European Community Mantychore FP7 project. It is as an open source platform for the GÉANT network operations center (NOC), National Research and Education Networks (NRENs) and other infrastructure providers of network and computing resources, which has been proposed to provide NaaS-based services. The main contributors include Juniper, HEAnet and i2CAT. OpenNaaS is proposed as a common network management and service orchestration platform, capable of providing and managing network in a flexible and efficient way. OpenNaaS can take advantage of SDN and Network Functions Virtualization (NFV) technologies.

OpenNaaS abstracts aside physical resources, enabling physical topology and vendor-specific details to be decoupled from their control and management features. The fundamental unit that OpenNaaS uses to accomplish this is the *Resource*. A Resource

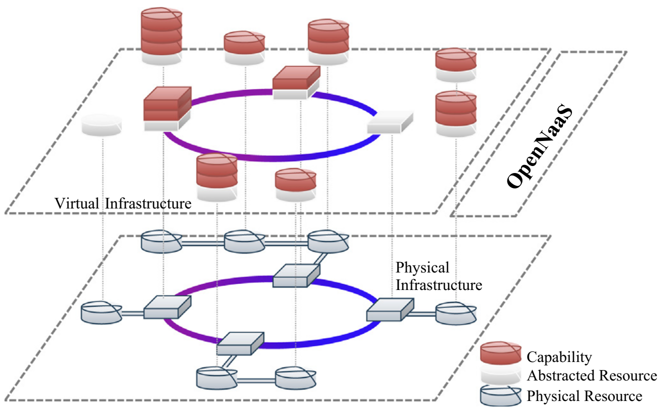


Fig. 1. OpenNaaS abstraction view: resources and capabilities.

models a device and represents a manageable unit inside the NaaS concept, e.g., a switch, a router, a link, a logical router, or a network. The basic resource considered is the Physical Resource (PR); Virtual Resources (VRs) are then created by manipulating the PRs. *Capabilities* shape resource functionality and provide an interface onto given resource functionality, e.g. OSPF, IPv6, the ability to create/manage logical routers, etc for a router. Fig. 1 shows this OpenNaaS vision in which physical devices are abstracted into resources and capabilities whose management can be delegated to upper application layers. The OpenNaaS model aims to be flexible enough to support different designs and orientations, but fixed enough so common tools can be built and reused across plugins.

The implementation of OpenNaaS consists of two distinctive parts: the core and the extensions. The core can be understood as a provider of basic functionality, e.g. resource management, which can then be used by the extensions. The extensions provide functionality for a specific aspect of networking, e.g. configuration of routers, by defining capabilities the resources have. The structure of OpenNaaS allows developers to easily implement more functionality by creating capabilities in a new extension bundle. We exploited this feature to create an energy-aware bundle (see Section 5.2).

OpenNaaS has a complete view of the entire network and interacts directly with the data plane through its SDN capabilities. It separates the control and data planes and its architecture follows the SDN paradigm. The OpenNaaS framework is widely used as an enabler for SDN technologies. OpenNaaS interworks several SDN platforms OpenDaylight (OPENDAYLIGHT, 2014), RYU (RYU, 2014) and Floodlight controllers (FLOODLIGHT, 2014) to orchestrate network services on top of SDN-based infrastructures and to enable new SDN applications for different stakeholders.

OpenNaaS defines an OpenFlow resource model for OpenFlow switches and create capabilities for OpenFlow resources in the *OpenFlow bundle*. The OpenFlow bundle works as an OpenFlow driver, which accesses the REST APIs of OpenFlow controllers for port statistics and flow forwarding (allowing developers to create, remove and get forwarding rules).

5. Design of energy-aware openNaaS

We integrated an energy-aware bundle in OpenNaaS to allow energy monitoring and green routing capabilities for OpenFlow networks.

5.1. Architecture

Fig. 2 shows the architecture of the energy-aware OpenNaaS we developed. The OpenNaaS server runs on top of an OpenFlow

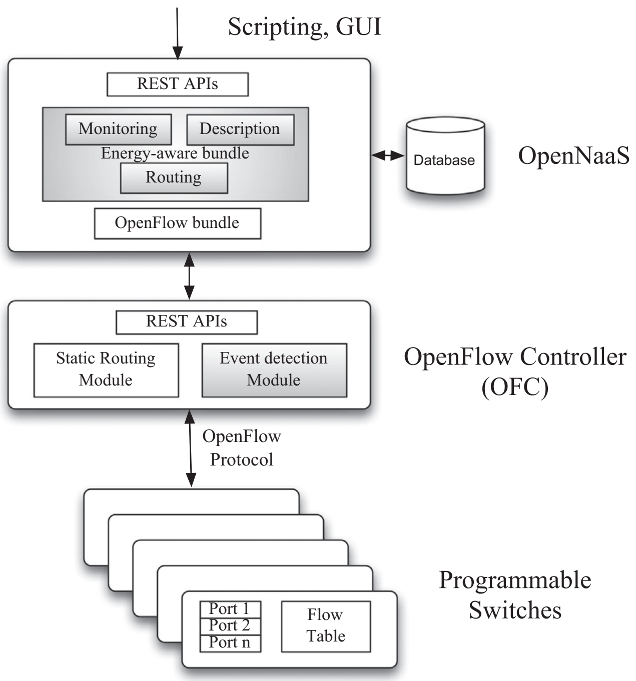


Fig. 2. Energy-aware OpenNaaS architecture.

controller (OFC), and the OpenFlow-enabled switches are connected with the OFC. There are two modes to invoke green routing functionality using REST APIs in OpenNaaS (we will describe the REST APIs in OpenNaaS in Section 6):

- Network providers directly send green routing requests to OpenNaaS. OpenNaaS provides virtual routing function as an NaaS service. The network providers can precreate forwarding rules for incoming flows when deployment to avoid the delay of routing control.
- A routing request is sent to the OpenNaaS server by an OFC for a switch. Different from the first mode, this mode calculates the path and creates the flow forwarding rules when a new flow arrives. It does not need the accurate prediction of flow arrivals.

The *energy-aware bundle* in OpenNaaS receives and handles routing requests. This bundle is capable of making a routing decision by calculating a green path. The energy-aware bundle, which communicates with an OFC through the *OpenFlow bundle*, calls the defined REST APIs in the OFC to add static flow rules for the path. The *static routing module* in the OFC executes the add operations by inserting flow entries into the flow tables of the switches. The *event detection module* in the OFC detects packet-in messages from switches, and then sends routing requests to OpenNaaS. We describe these components in detail in the following sections.

5.2. Energy-aware bundle

The *Energy-aware bundle* at the core of our design implements energy-aware description, monitoring and routing capabilities for OpenFlow resources. Energy-aware routing depends on topology information, traffic information and energy usage information to determine the configuration of the network when scheduling traffic. OpenNaaS can provide the first two pieces of information. OpenNaaS employs the Link Layer Discovery Protocol (LLDP) to obtain network topology and uses OpenDayLight to sample current traffic flows. There are also other tools and protocols that are being supported by OpenNaaS for the bandwidth and traffic monitoring purpose, such as SNMP and NetFlow (Cisco, 2012).

However energy monitoring for networks is challenging due to the dynamics of infrastructure information and special measurement mechanisms. Various network resources have drastically different energy-related attributes, and these attributes keep changing. For example, network devices can switch between renewable energy and brown energy. Some devices support IEEE 802.3az (802.3az, 2010) while others do not. These information could impact the method and effect of energy management. Besides, energy monitoring needs extra instrumentation devices to measure power consumption. The power is measured on the instrumentation devices instead of directly on resources. It is essential to capture the relationship between network resources and instrumentation devices. Therefore, a complete energy-aware information model is important for OpenNaaS to describe all the measurement information and meta information for monitoring and to exchange the energy information between software components. Based on the analysis above, we implemented both energy monitoring capabilities and energy description capabilities:

Energy monitoring capabilities obtain and provide energy usage information from power meters for the observed metrics: (power, energy, CO2 emission rate, and electricity price) and from data statistics for the calculated metrics: (total electricity, CO2 emission, and energy efficiency). The measurement data is saved in an OpenNaaS database, as shown in Fig. 2. The capabilities are responsible for communication with power meters. We have created power meter drivers for SNMP access to different power meter vendors, e.g. Rackitivity and APC PDUs. Only numeric Object Identifiers (OIDs) in the drivers are different for different PDUs. Each OID identifies a variable that can be read or set via SNMP. The capabilities not only measure the power usage of a single device, but also monitor the power usage of a network path. Using SNMP, the capabilities can obtain and control the power state of switches and ports.

Energy description capabilities describe and create meta information about energy source, power meters, green metric, power state, etc. in OpenNaaS. We developed Energy Description Language (EDL) ontology, which reuses Infrastructure and Network Description Language (INDL) ontology to describe the resources and network infrastructure that connects these resources (Zhu et al., 2014; Ghijsen et al., 2012). INDL can describe a set of network elements such as topology, ports, links, paths, and so on. EDL itself focuses on the knowledge representation in the domain of energy monitoring. With EDL, OpenNaaS instantiates energy information using a common vocabulary and makes information understandable between software components. An important piece of information that EDL can describe is the relationship of power meters and remote network devices, so that it is understandable which measured energy usage information from the port of a power meter belongs to which remote device.

The third set of capabilities we developed is the *Green routing capabilities*, needed to calculate the green routing path. Three green metric options are available for routing: power consumption, electricity cost and CO2 emission, as these metrics are provided by the monitoring capabilities. In our implementation we adopted a greedy routing algorithm. Once an OpenNaaS user selects the green metric, e.g. electricity cost to optimize upon, the algorithm traverses all the possible network routes between the original host and destination host of the flow. It uses power models and electricity price of switches to estimate the overall electricity costs, and chooses the path with the minimized value.

OpenNaaS includes a model for the description of OpenFlow route tables and network paths. The switch ports are identified by number, and a route table is defined by: *IP source*, *IP destination*, *Source switch identifier, also named Datapath identifier (DPID)*, *Input port of the switch* and *Output port of the switch*. The output port identifies which switch the traffic is sent to on the next hop. A routing path is a list of route tables.

The calculated path is converted to a list of OpenFlow flow rules in the specific structure by the *OpenFlow bundle* according to the type of OFC. Each flow rule represents a hop of the path. Then the OpenFlow bundle sends each rule to the OFC. The following is an example of a flow rule in FloodLight:

{ *"switch": "00:00:00:00:00:00:00:01", "name":"flowmod1", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":" output=2"*}.

The identifier of an switch indicates where the rule will be configured and the output port of this switch represents the forwarding direction. So that the controller can notify the switch about where they should forward the flow and inserts a new matching entry in the Flow Table of the switch.

OpenNaaS has knowledge of the network topology that also depicts the connections between OFCs and switches. So even if there are multiple OFCs in a network, the OpenFlow bundle knows which switch the flow belongs to and which OFC controls this switch. OpenNaaS can add the flow rules to the correct OFC.

5.3. *OpenFlow controllers*

To support the second mode for invoking green routing functionality, we created the *event detection module* in the controller to send a routing request to OpenNaaS when a new flow arrives. OpenFlow protocol defines a packet-in messages. Each time a switch receives a packet from a new flow, it tries to match the packet in a Flow Table. In the case that it does not match for any entry, the packet is sent to the OFC using the packet-in message. When the event detection module in the OFC detect this message, it creates and sends a routing request to OpenNaaS. The arguments in the request contain the source and destination IP of the packet, the DPID of the switch and the input port where the packet enters the switch.

In our design, after the path is calculated OFCs insert flow forwarding rules for the path in a proactive way. The *static routing module* pushes all the flow entries to the switches before traffic arrives, to save the time of processing routing requests from all the switches. We did not change the static routing module and its REST APIs in the OFCs. The OpenFlow bundle in the OpenNaaS server calls different APIs according to the type of an OFC: Static Flow Pusher APIs in Floodlight and Static Routing APIs in OpenDaylight.

6. Prototype

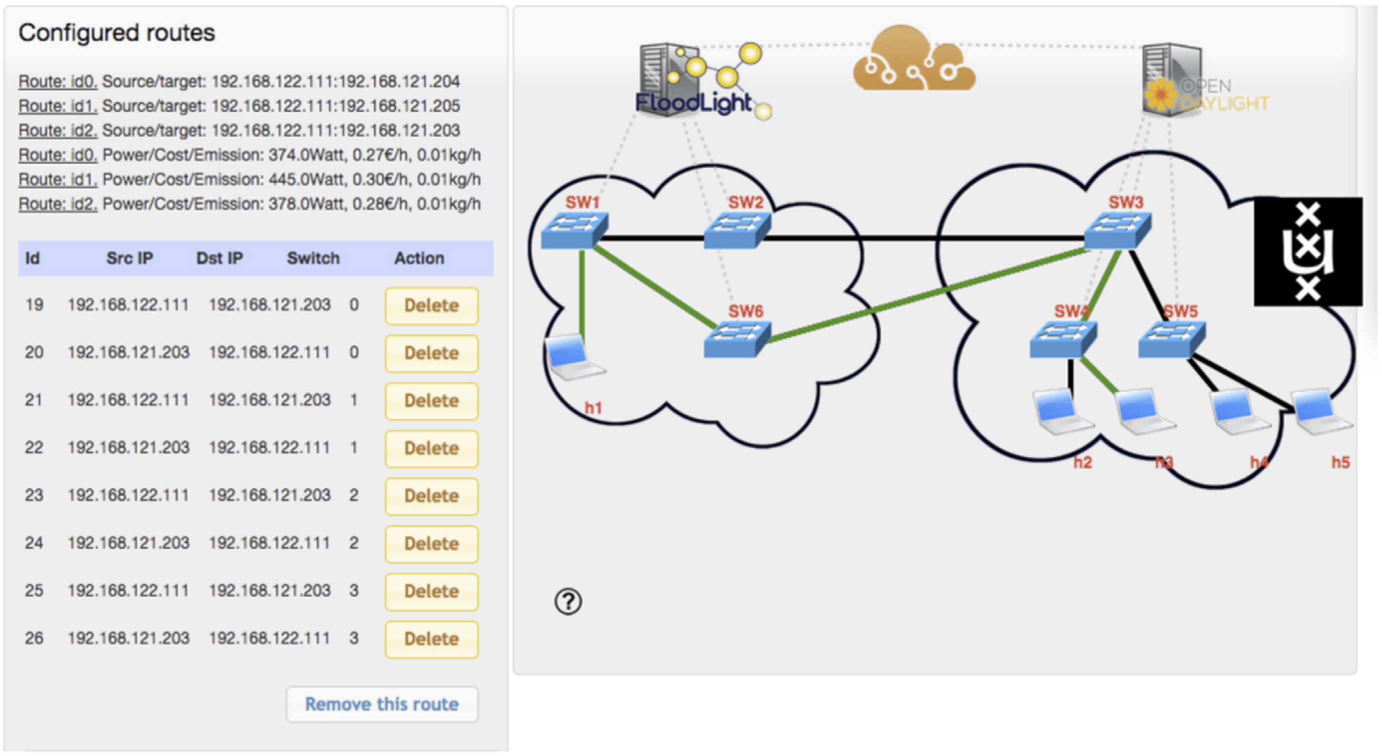
We developed a prototype to show the functionality of energy-aware OpenNaaS[1]. The prototype includes a web client GUI, through which a network provider invokes green routing functionality of OpenNaaS server using the first invoking mode. We list part of the REST APIs in energy-aware OpenNaaS in Table 2.

We emulated a Mininet network with a topology of 6 OpenFlow switches and 5 hosts, shown in Fig. 3. The switches have the same network capacity and are divided into two groups, controlled by Floodlight and OpenDaylight controllers, respectively. To simulate different CO2 emission rate (van der Veldt et al., 2014), we assumed the two groups consume solar and thermal energy, respectively. We ran the OpenNaaS server in the same machine.

[1] The source codes of energy-aware OpenNaaS and its demo are shown here: https://bitbucket.org/uva-sne/green-routing-demo

Table 2
Part of the REST APIs in energy-aware OpenNaaS.

Type	Description	URI (/opennaas/…)	Arguments
Energy description	Set the energy source	edlnode/{nodename}/edl_node_setup /energysource/{es_id}	**nodename**:SW1, **es_id**:solar1
	Set the electricity price of energy source	edlnode/{nodename}/edl_node_setup /energysource/{es_id}/{price}	**es_id**:solar1, **price**:0.42
	Set the emission rate of energy source	edlnode/{nodename}/edl_node_setup /energysource/{es_id}/{emission}	**es_id** solar1, **emission**:0.015
	Set the power meter	edlnode/{nodename}/edl_node_setup /powermeter?pduname= &numOfModule=&numOfPort	**pduname**:rpdu-vu01, **numOf-Module**:3, **numOfPort**:8
	Set the address of power meter	edlnode/{nodename}/edl_node_setup /powermeter/driver/{UDPaddress}	**UDPaddress**: localhost/16120
	Set the outlet	edlnode/{nodename}/edl_node_setup /outlet?moduleNum=&portNum=	**moduleNum**:1, **portNum**:7
Energy monitoring	Get the information about energy source	edlnode/{nodename}/edl_node_powermonitor /energysource	
	Get the current monitor log	edlnode/{nodename}/edl_node_powermonitor /observedgmetric	
	Get the monitor log for a period of time	edlnode/{nodename}/edl_node_powermonitor /gmetric/{period}/{interval}	**period**: 60, **interval**: 5
Green routing	Set a green metric option	vrf/routing/greenRouteMetric/{metric}	Power
	Get a green routing path	vrf/routing/route /{ipSrc}/{ipDst}/{DPID}/ {inPort}	**ipSrc**:192.168.1.1, **ipDst**:192.168.2.51, **DPID**: 00:00:64:87:88:58:f6:57, **inPort**: 2
	Get the configured route table	openflowswitch/{nodename} /offorwarding/ getOFForwardingRules	**nodename**: SW1


Configured routes

Route: id0. Source/target: 192.168.122.111:192.168.121.204
Route: id1. Source/target: 192.168.122.111:192.168.121.205
Route: id2. Source/target: 192.168.122.111:192.168.121.203
Route: id0. Power/Cost/Emission: 374.0Watt, 0.27€/h, 0.01kg/h
Route: id1. Power/Cost/Emission: 445.0Watt, 0.30€/h, 0.01kg/h
Route: id2. Power/Cost/Emission: 378.0Watt, 0.28€/h, 0.01kg/h

Id	Src IP	Dst IP	Switch	Action
19	192.168.122.111	192.168.121.203	0	Delete
20	192.168.121.203	192.168.122.111	0	Delete
21	192.168.122.111	192.168.121.203	1	Delete
22	192.168.121.203	192.168.122.111	1	Delete
23	192.168.122.111	192.168.121.203	2	Delete
24	192.168.121.203	192.168.122.111	2	Delete
25	192.168.122.111	192.168.121.203	3	Delete
26	192.168.121.203	192.168.122.111	3	Delete

Fig. 3. A screen shot of the topology and the configured routes in the emulated network environment.

Fig. 4 presents a flowchart of the usecase in our emulated environment. A provider creates a set of abstract OpenFlow resources and loads the OpenFlow capabilities and energy-aware capabilities for the resources using OpenNaaS (*Step 1*).

After creating the resources, the provider sets the energy source information for the network, as well as the connections between switches and the outlets of power meters according to APIs for energy description in Table 2 (*Step 2*). For our prototype, we interfaced OpenNaaS with an actual power meter to provide the power readings of emulated resources. At this point the provider sends the monitoring request with the specific metric (*Step 3*). OpenNaaS translates this request and forwards an SNMP sampling command to the power meter (*Step 3.1*) and reads the measurement data out (*Step 3.2*). The data is described by EDL, and

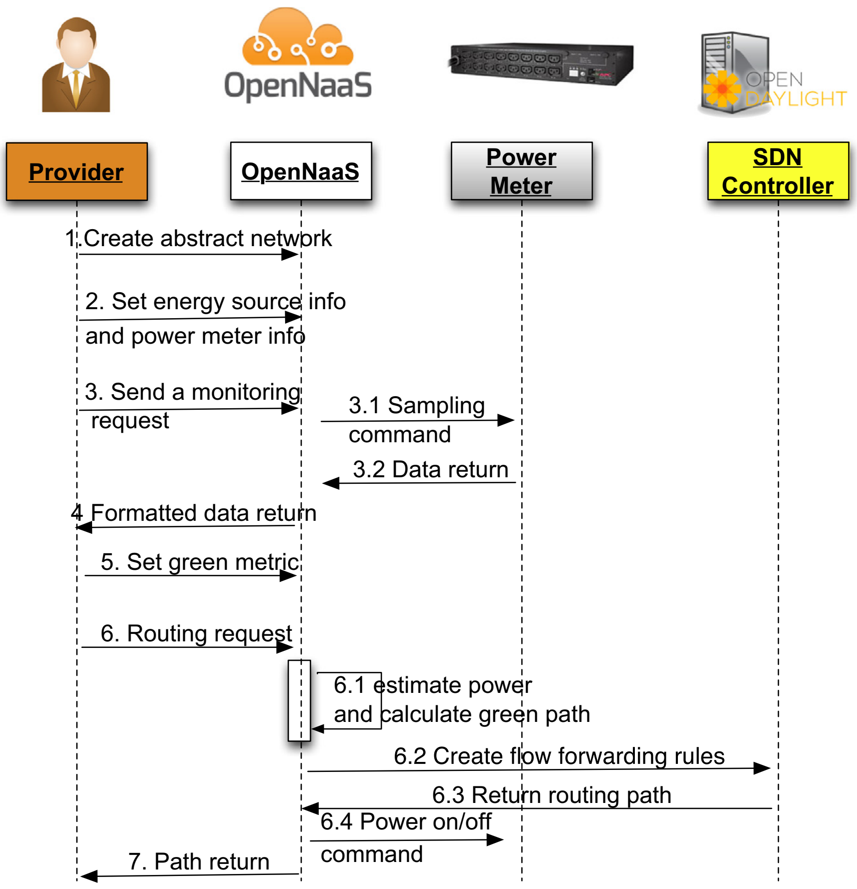


Fig. 4. The flowchart of a general usecase of energy-aware OpenNaaS (Steps 1–Step 7).

Table 3
An example of switch (SW1) information monitored by OpenNaaS.

DPID	00:00:64:87:88:58:f6:57
Controller IP	controllersVM
Controller Port	8080
Power Consumption	70.0 Watt
Energy Source	
ID	Solar1
Price	0.42 Euros/kWh
CO2 emission rate	0.015 kg/kWh

data is saved in a database and is sent back to the provider (*Step 4*). Table 3 shows the energy information for switch SW1 at a certain moment. The measurement data is to collect once when the deployment of a switch. Providers conduct experiments to make a switch run at idle state and fully-used state when deployment. In this case, the static and dynamical power consumption are known; all the interfaces are powered on at full utilization. So the power consumption of an interface can be known. A linear power model of the switch (See Eq. (1)) can be derived.

Our prototype also proves that OpenNaaS can measure the energy usage information of a specified routing path. Fig. 3 lists all the configured routes in the network and presents the total power consumption, electricity rate and emission rate of each path.

At this point the provider decides on a green optimization metric using the first API for green routing in Table 2 (*Step 5*) and submits a request to OpenNaaS for a path between a source and a destination (*Step 6*) using the second API for green routing. OpenNaaS cannot support the routing request with bandwidth demand. We will implement this in future by using existing BoD capabilities. OpenNaaS estimates the energy information of all the possible network routes between the end points based on power models (*Step 6.1*). OpenNaaS greedily selects the path and interacts with the OFCs to create flow forwarding rules in the switches (*Step 6.2*). We will discuss the details of the power-greedy algorithm in Section 7.1. At the end, the formatted path information is returned to the provider if the flow rules are successfully created (*Step 6.3*). According to the result path, OpenNaaS sends power on/off command to the power meter via SNMP and then the power meter changes the state of switches and links (*Step 6.4*). Fig. 3 shows a routing path we found between host1 and host3 using energy-aware OpenNaaS.

The benefits for adopting energy-aware OpenNaaS are manyfold. First, OpenNaaS has lightweight management costs. Network users and providers can configure their network through unified capabilities published in OpenNaaS that are transparent to specific technologies and hardware details. The control centralized in the OpenNaaS server manages the network as a whole rather than as a number of individual devices. Second, energy-aware OpenNaaS allows network users and providers to understand their energy usage information. They can profile the energy information for their network for further analysis, e.g. troubleshoot power shortage problems in data centers and prepare for the design a new network infrastructure. Network providers can even implement their own power management methods with the profiling information.

Our prototype implemented a simple usecase where OpenNaaS creates a path with a greedy routing algorithm when receiving a routing request from a network provider. Similar to previous energy routing work, the greedy algorithm is for the power minimization problem. For the next version of energy-aware

OpenNaaS we decided to focus on a routing strategy for the original energy-aware routing problem; before implementing it, we evaluated a set of algorithms and found out a suitable one to replace the greedy algorithm next section.

7. Energy-aware routing strategies

Energy-aware routing is a bin-packing problem, and it is not possible to find an optimal solution, especially for large scale networks. Therefore, rather than directly solving the energy-aware routing problem, we analyze the energy efficiency of common routing algorithms that already exist. We use the routing algorithms to find routes, and the switches and links on unused routes will be powered off.

Traffic aggregation in these routing algorithms could make flows run on the same link. Different flow scheduling algorithms can influence the time of transferring the flows on the same path. So we will also analyze the energy efficiency of flow scheduling algorithms as post of optimized energy-aware routing strategy.

7.1. Flow routing

As discussed in Section 2, solutions to the power minimization problem rely on the assumption that the completion time of traffic is constant. They find a routing path for each flow and combine all the result paths to generate a subset of the network topology. *Power-greedy routing* is a simple solution to the power minimization problem (See Algorithm 1). For each flow, power-greedy routing find out all possible paths with sufficient capacity (lines 1–2); it traverses the paths and selects a routing path with the least increase of total network power consumption (lines 5–13). The estimation of power consumption using a path depends on the power models derived from historical measurement data (line 9).

Algorithm 1. PowerGreedyRouting(G,f,rb).

```
Input:
    $G(V,E,U)$: data center network topology and link
    utilization;
    $f$: the target flow in a request
    $rb$: required bandwidth for flow f.
Output:
    $p$: the path for flow $f$.
1:  $PathsX \leftarrow Get\_all\_simple\_paths(G,f)$
2:  $PathsY \leftarrow Get\_available\_paths(PathsX,rb)$
3:  if $PathsY$ is empty then
4:      return $None$
5:  else
6:      for$pathy$ in $PathsX$
7:          $i \leftarrow 0$
8:          $G' \leftarrow Add\_network\_utilization(G,pathy,rb)$
9:          $power[i] \leftarrow Get\_power(G')$
10:         $i \leftarrow i+1$
11:     end for
12:     $index \leftarrow Get\_Index(argmin(power))$
13:     return $PathsX[index]$
14: end if
```

Shortest path routing pursues low-delay and high throughput for data transmission. It selects a routing path with the least number of lengths or weights. Shortest path routing is not power-aware, but it could lead to low completion time.

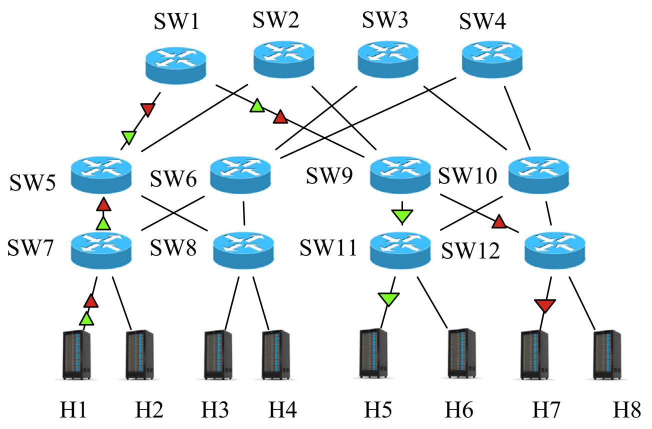


Fig. 5. An example of partial Fat Tree network (Li et al. 2014).

In data center networks, networks are over-provisioned. Multiple paths with similar length between two hosts are likely to exist, so their performance is similar, e.g. there are multiple parallel paths between any two hosts in a BCube network and some of paths have the same length (Guo et al., 2009). The same applies to a Fat Tree network. If we observe Fig. 5, which is an example of partial Fat Tree network, we see that 4 shortest paths exist between any two hosts. For example, the shortest paths between *Host1* and *Host5* are as follows:

$$path1 : SW7 \rightarrow SW5 \rightarrow SW1 \rightarrow SW9 \rightarrow SW11$$
$$path2 : SW7 \rightarrow SW5 \rightarrow SW2 \rightarrow SW9 \rightarrow SW11$$
$$path3 : SW7 \rightarrow SW6 \rightarrow SW3 \rightarrow SW10 \rightarrow SW11$$
$$path4 : SW7 \rightarrow SW6 \rightarrow SW4 \rightarrow SW10 \rightarrow SW11$$

Algorithm 2. PriorityRouting(G,f,rb).

```
Input:
    $G$: data center network topology
    $f$: the target flow in a request
    $rb$: required bandwidth for flow $f$.
Output:
    $p$: the path for flow $f$.
1:  $PathsX \leftarrow Get\_all\_shortest\_paths(G,f)$
2:  $PathsY \leftarrow Get\_available\_paths(PathsX,rb)$
3:  if $PathsY$ is empty then
4:      return $None$
5:  else
6:      for$pathy$ in $PathsX$
7:          $i \leftarrow 0$
8:          $active\_SW[i] \leftarrow Get\_number\_of\_active\_switches(G,pathy)$
9:          $i \leftarrow i+1$
10:     end for
11:     $index \leftarrow Get\_Index(argmin(active\_SW))$
12:     return $PathsX[index]$
13: end if
```

It is reasonable to select an energy efficient path among multiple shortest paths. Thus, we propose *priority-based shortest routing* (See Algorithm 2). It first finds all the shortest paths with sufficient capacity (lines 1–2). The priority of a routing path is differentiated by the use of switches in the path. For each path, its priority is measured by the number of currently used switches (line 8). Priority-based shortest routing selects the one with the highest priority from all the shortest paths (lines 11–12). For example, if *SW6* and *SW4* are currently working or not being powered off, *path4* has the highest priority and should be selected.

The basic reasoning is to use as few network resources as possible to achieve the same network performance. *Random routing*, which randomly selects a routing path from possible paths, is the baseline for the three other routing algorithms.

7.2. Flow scheduling

Traffic aggregation in flow routing might make multiple flows run on the same link and share the overall bandwidth. Exclusive flow scheduling (EXR) is an alternative to link sharing; EXR transfers the flows one by one and only allows each flow exclusive use of the overall link bandwidth. We can prove that EXR is a more efficient scheduling algorithm comparing to link-shared scheduling. This is well illustrated with an example based on Fig. 5. This fact is also correct for other network topologies. Let us assume that two flows are transferred at the same time in this Fat Tree network, namely “flow1: H1 to H5” and “flow2: H1 to H7” with size of s_1 and s_2, respectively. For simplicity, we assume that only a set of switches (*SW7, SW5, SW1, SW9, SW11 and SW12*) and only the links with data transmission are currently active. The completion time of two flows are t_1 and t_2. Let α be the proportion of bandwidth used by *flow1* and $1-\alpha$ be the proportion used by *flow2*. We assume the flow1 is finished first (The result can be proved accordingly if the flow2 is finished first), so the range of α is as follows:

$$\frac{s_1}{s_1+s_2} < \alpha \le 1 \tag{6}$$

Since most data center networks employ a homogeneous set of switches to interconnect servers, we assume all the switches have same static power P_{base} and all the links of the switch have the same power model $p(u)$. The network energy E of transmitting flow1 and flow2 can be calculated as follows:

$$\begin{aligned} E &= E_{SW7}+E_{SW5}+E_{SW1}+E_{SW9}+E_{SW11}+E_{SW12} \\ &= (P_{base}+p(\alpha C))\cdot t_1+5P_{base}\cdot t_2+4p(C)\cdot t_2 \\ &\quad +p(C))\cdot(t_2-t_1)+p((1-\alpha)\cdot C)\cdot t_1 \\ &= (P_{base}+p(\alpha C))\cdot\frac{s_1}{\alpha C}+5P_{base}\cdot\frac{s_1+s_2}{C}+4p(C)\cdot\frac{s_1+s_2}{C} \\ &\quad +p(C))\cdot\left(\frac{s_1+s_2}{C}-\frac{s_1}{\alpha C}\right)+p((1-\alpha)\cdot C)\cdot\frac{s_1}{\alpha C}=\frac{P_{base}}{C}\left(5s_1+\frac{s_1}{\alpha C}+5s_2\right) \\ &\quad +\frac{p(\alpha C)}{\alpha C}s_1+\frac{p(C)}{c}\left(5s_1-\frac{s_1}{\alpha}+5s_2\right)+\frac{p((1-\alpha)\cdot C)}{\alpha C}s_1 \end{aligned} \tag{7}$$

According to the power profiling of networking devices in previous experiments (Zhu et al., 2015), the power consumed by the link is small compared to the static power consumption; the power consumption of an idle link is quite close to that of a full load . We can assume $p((1-\alpha)\cdot C)$, $p(\alpha C)$ and $p(C)$ are equal. Therefore, Eq. (7) can be simplified as follows:

$$E=\frac{P_{base}}{C}\left(5s_1+\frac{s_1}{\alpha}+5s_2\right)+\frac{p(C)}{C}\left(5s_1+\frac{s_1}{\alpha}+5s_2\right) \tag{8}$$

When α equals to 1, Eq. (8) reaches a minimized value. So, the network has the least energy consumption when a flow is transferred exclusively on a link.

Based on the above analysis, the combination of priority routing and EXR scheduling (See Algorithm 3) could be more energy efficient. This combined strategy defines two flow lists (F_{active} and $F_{suspend}$) to monitor active flows and suspended flows. This strategy first calculates the routing path for any new incoming flow using Algorithm 2 (lines 1–2). But the Algorithm 2 needs a modification. The line 2 of the Algorithm 2 is replaced by *Get_idle_paths* to guarantee that no path is shared by flows. Then, this strategy needs to determine the scheduling state of the flow. If no idle path is found, this strategy must put the flow into the suspended list (lines 3–4). If an idle path exists, this strategy adds the flow into the active list, inserts the flow and its path to a set, and updates the state of the network (lines 5–8). When an active flow is finished, the flow is removed and the network state is updated (lines 10–12). This strategy finds a suspended flow which has an idle path by traversing the suspended list (lines 13–17), and makes the flow active (lines 18–22).

Algorithm 3. Combination of EXR scheduling and priority routing.

```
Input:
      $G(V,E,U)$: data center network topology and link
      utilization
      $F_{active}$: list of active flows
      $F_{suspend}$: list of suspended flows
      $cp$: capacity of a link.
Output:
      PS$=\{\langle f,p\rangle\ \forall \text{flow } f \in F_{active}\}$, $p$ is a path.
1:  if a new flow $f$ arrives then
2:     $p \leftarrow PriorityRouting(G,f,cp)$
3:     if $p$ is empty then
4:        $F_{Suspend} \leftarrow F_{Suspend}+f$
5:     else
6:        $F_{active} \leftarrow F_{active}+f$
7:        insert $<f,p>$ into $PS$
8:        $G \leftarrow Add\_network\_utilization(G,f,cp)$
9:     end if
10: else if a flow $f$ in $F_{active}$ finishes
11:    $F_{active} \leftarrow F_{active}-f$
12:    $G \leftarrow Reduce\_network\_utilization(G,f,cp)$
13:    for $f_{suspend}$ in $F_{suspend}$ do
14:       $p \leftarrow PriorityRouting(G,f_{suspend},cp)$
15:       if $p$ is empty then
16:          continue
17:       else
18:          $F_{Suspend} \leftarrow F_{Suspend}-f$
19:          $F_{active} \leftarrow F_{active}+f$
20:          insert $<f,p>$ into $PS$
21:          $G \leftarrow Add\_network\_utilization(G,f_{suspend},cp)$
22:          break
23:       end if
24:    end for
25: end if
26: return $PS$
```

In the next section we will validate our analysis by conducting comparative studies by simulation.

8. Evaluation

In our simulation we use two common data center topologies – Fat Tree using 4-port switches (Fat Tree(4)) and 3-level BCube using 2-port switch (BCube(2,3)). The capacity of each link in the simulated topologies is 1 Gbps; the total number of server is 16 servers. There are 32 switches in BCube(2,3) and 20 switches in Fat Tree(4). We present two groups of flows with totally different flow size to study their potential in extreme situations. We simulate the flows with a typical size (64 MB) in Hadoop data centers and a large size (5 GB) in scientific data centers. In each group of flows, the size of each flow changes in a pretty small range (less than 1%). The source and destination servers of each flow are random. The arrival rate of flows follows the Poisson distribution. We simulate the flows coming in one minute and the maximum number of flows are about 3000. The average throughput of flows is 500 MB/s.

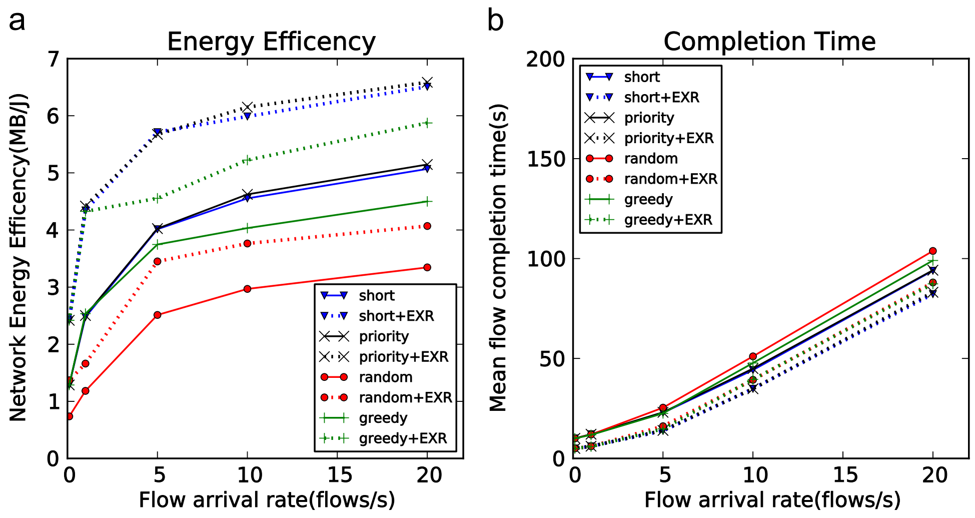


Fig. 6. Network energy efficiency and mean flow completion time against the arrival rate of large flows (5 GB) in the BCube network.

All the switches in the same network are homogeneous. According to the power profiling of networking devices in our experiments (Zhu et al., 2015) and from other studies in Heller et al. (2010), we assume the dynamic power consumption accounts for less than 10% of total power consumption. We use a linear model to capture the relation between dynamic power consumption and network load.

For the simplicity of our simulation, we focus on small-sized networks. This is because power-greedy routing is not scalable, we have to calculate all possible paths (without loops) and compare their power consumption. The algorithm to find all the paths has a computational complexity of $O(V!)$ (Sedgewick, 2001). V represents the number of switches in the network. Priority-based shortest routing and shortest routing are both fundamentally scalable. Using the Dijkstra algorithm to find a shortest routing has a computational complexity of $O(V^2)$. Priority-based shortest routing finds n shortest routing paths, and n is a small integer usually. Its computational complexity is $O(n^3 \cdot V(E+V \log V))$, where E is the number of links in the network.

We explore how much energy to consume for data transmission with different routing strategies that include the combination of routing algorithms and flow scheduling algorithms. The flow scheduling algorithms are differentiated by whether flows are allowed to share link bandwidth. We define the *Energy Efficiency* metric as the amount of data transmission per unit of energy consumption. The *Mean Flow Completion Time* is the mean time interval between arrival and completion of flows, used to qualify network performance.

Fig. 6(a) and (b) shows network energy efficiency and mean flow completion time by the different routing algorithms against the big flow arrival rate in the BCube network. The solid lines represent the strategies with shared flow scheduling and the dotted lines describe the strategies with EXR. In Fig. 6(a), we observe that the BCube network's energy efficiency rises with the growth of the arrival rate. It proves higher network utilization makes the network more energy efficient. The strategies with EXR offer significant energy saving compared to the strategies with link sharing. For example, priority-based routing with EXR is about 35% higher energy efficiency over priority-based routing with link sharing. In regard to the same scheduling algorithm, shortest routing has a growing advantage (at most 20%) on the energy efficiency than greedy routing when the flow arrival rate increases. The energy efficiency of priority-based routing is about 5% higher than that of shortest routing when the arrival rate of flows is over 10.

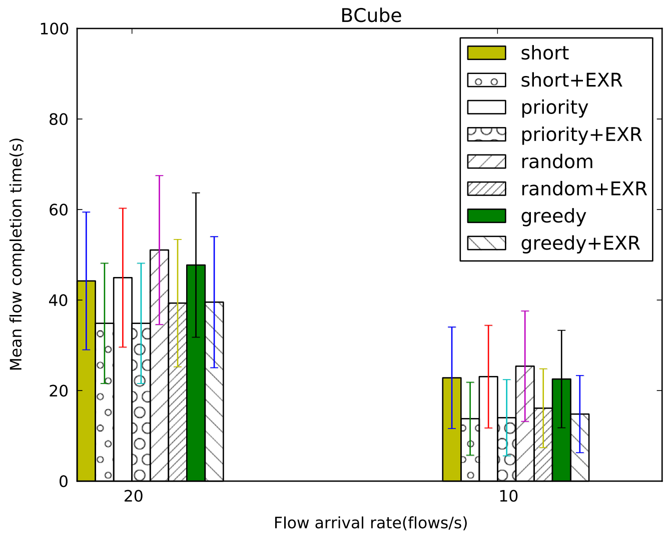


Fig. 7. Mean flow completion time with standard error against the arrival rate of large flows (5 GB) in the BCube network.

In Fig. 6(b), we observe that the strategies with link sharing have longer mean flow completion time than the strategies with EXR. Priority-based and shortest routing algorithms have similar performance, which are better than greedy routing. During exclusive scheduling, the performance of greedy routing is as bad as random routing.

Fig. 7 shows the standard error of completion time for all the flows. The completion time of the strategies with link sharing change in a similar range as the strategies with EXR.

Fig. 8(a) illustrates that network energy efficiency can improve 2.5 times when the big flow arrival rate increases from 0.1 to 20 in the Fat-Tree network. The strategies with EXR still offers an obvious improvement of energy efficiency over the strategies with link sharing. Although priority-based shortest routing is the most energy efficient most of the time, the advantage of the shortest and greedy routing algorithms in the Fat Tree network is slight compared to that of the BCube network.

Fig. 8(b) shows the mean flow completion time against the big flow arrival rate in the Fat-Tree network. The strategies with link sharing are worse than the strategies with EXR. The mean flow completion time using the priority-based algorithm and using the shortest routing algorithm are nearly the same; they are smaller than that of the other two routing algorithms.

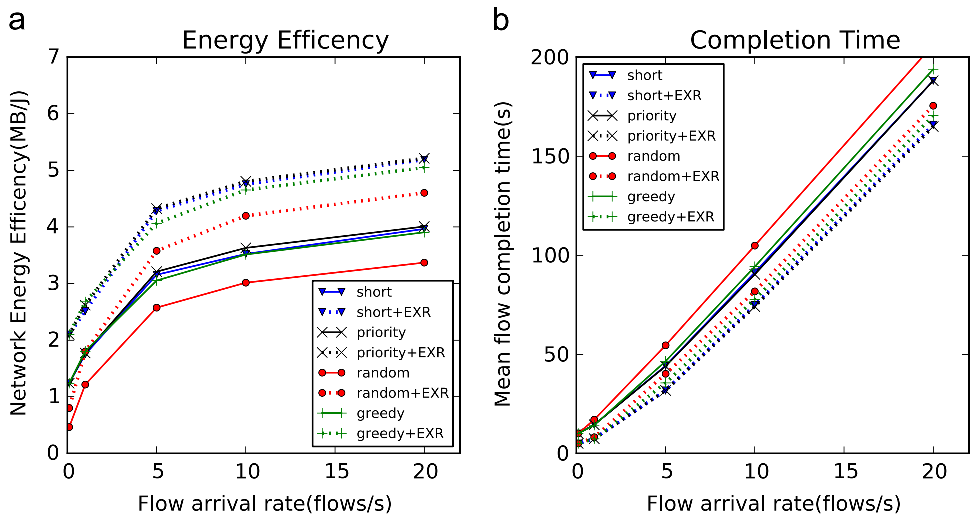


Fig. 8. Network energy efficiency and mean flow completion time against the arrival rate of large flows (5 GB) in the Fat Tree network.

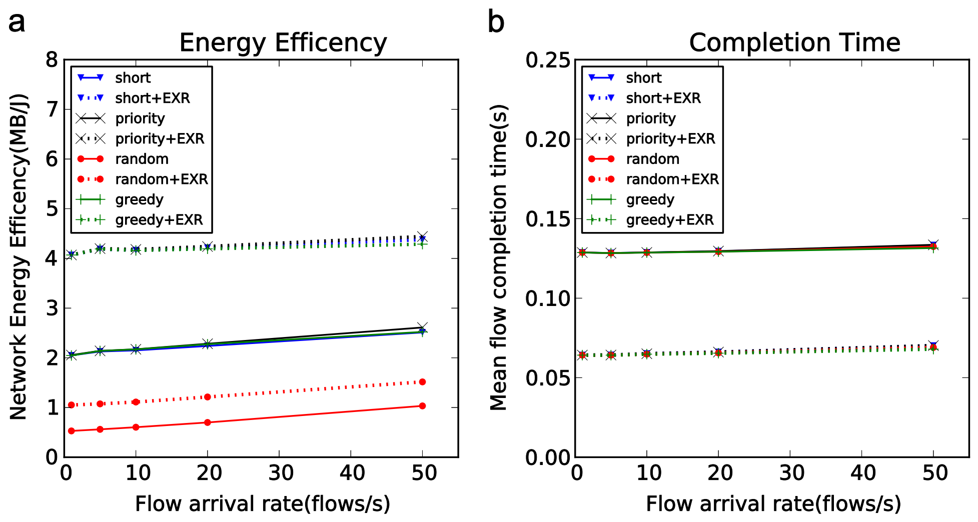


Fig. 9. Network energy efficiency and mean flow completion time against the arrival rate of small flows (64 MB) in the BCube network.

Fig. 9(a) and (b) shows network efficiency against the small flow arrival rate in the BCube network. The two figures reiterate that the strategies with link sharing are less energy efficient and have longer completion time than the strategies with EXR.

This is also observed in Fig. 10(a) and (b) for the BCube network. Compared to the experiments using traffic with big flows, the energy efficiency of the BCube and Fat Tree networks improves less (at most 1 time) when the small flow arrival rate increases; the energy efficiency of the networks for small flow transmission are lower. Energy efficiency of the shortest, priority-based and greedy routing algorithms are similar, and the mean flow completion time of these routing algorithms are the same. Although these routing algorithms could compute different routing paths for the same flow, the difference of route selection impacts the network power consumption quite lightly for small flows because they are completed in very short time.

In the above experiments we assume no transition time for switches. But in a real network environment, the switches usually take a few seconds to power on/off after receiving off/on signals (Nedevschi et al., 2008). In order to simulate the vendors with different transition times, we study network energy efficiency against the transition time. A switch does not goes off if the arrival interval between two flows along it is less than the transition time. The flow arrival rate is fixed at 10 flows per second and the size of all the flows is 5 GB. The result of Fat-Tree and BCube networks is shown in Fig. 11(a) and (b). We can find that network energy efficiency declines slowly and remains stable when the transition time is large enough. In regard to the same scheduling algorithm, we can observe that the energy efficiency of priority-based routing is about 5% higher than that of shortest routing, while shortest routing has about 5–20% higher energy efficiency than greedy routing.

We conclude all the experiments as follows. Power-greedy routing is a power-minimization algorithm, but it could not find an optimal routing for all the flows because its result is in essence a local optimization. Besides, power-greedy routing degrades the completion time of flows. That is the reason that our experiments indicate that the power-greedy algorithm is less energy efficient. Exclusive flow scheduling is much more energy efficient than scheduling with link sharing regardless of which routing algorithm is used. The strategy we highlight is the combination of priority-based shortest routing and exclusive flow scheduling. It has the obvious improvement of energy efficiency for big file transmissions in the BCube network. In this case, the priority-based routing with EXR is about 5–35% higher energy efficiency than other common strategies. When the transition time of

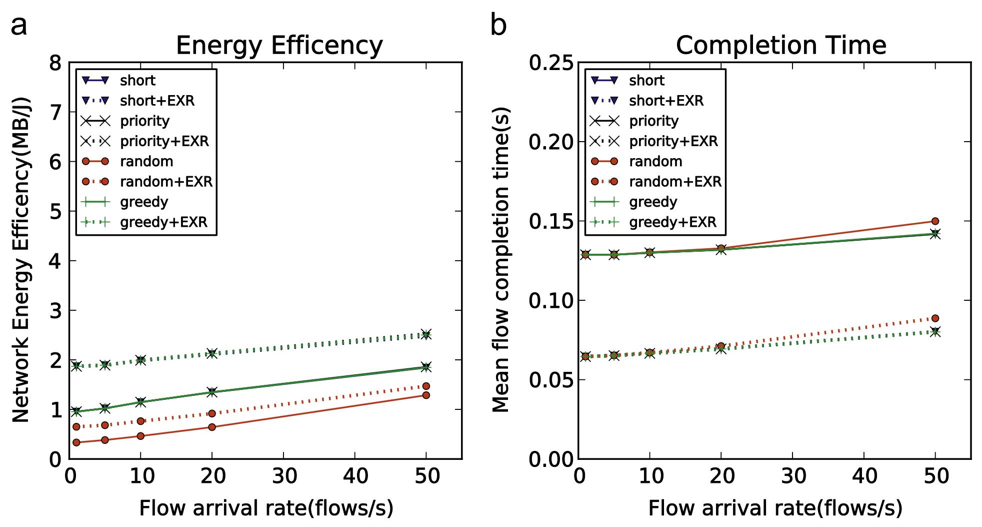


Fig. 10. Network energy efficiency and mean flow completion time against the arrival rate of small flows (64 MB) in the Fat Tree network.

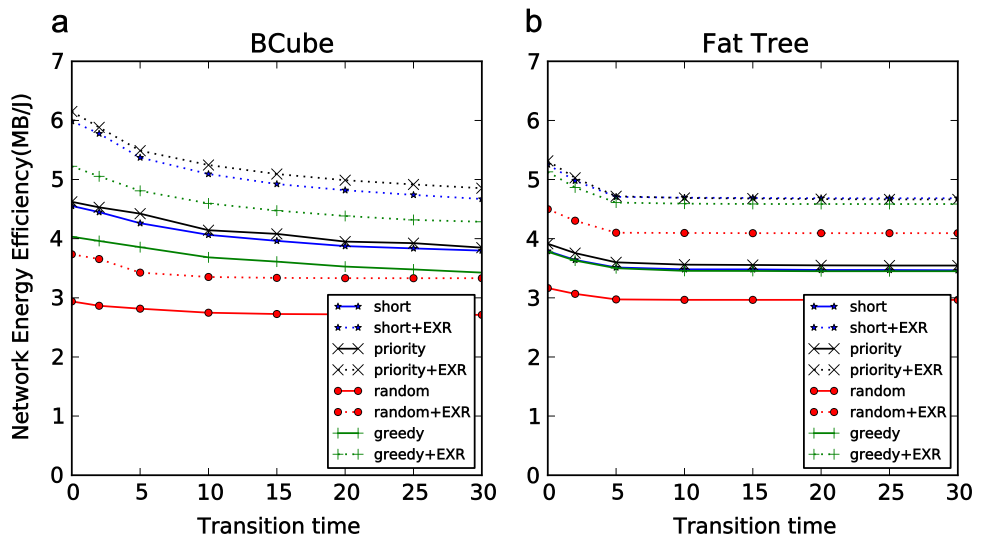


Fig. 11. Network energy efficiency against transition time of switches under large flows (5 GB) in Fat Tree and BCube networks.

switches is considered, it is still more energy efficient, compared to either the routing algorithms or the scheduling algorithms used. The network performance of this strategy does not degrade. Its mean flow completion time is nearly same as the strategies with shortest routing. Considering that the size of traffic is fixed, its mean network throughput is nearly the same as the strategies with shortest routing.

9. Discussion

We will implement the exclusive flow scheduling algorithm in the prototype in future. OpenNaaS and OFCs needs to be modified to enable exclusive flow scheduling. OpenNaaS has to define two flow lists to maintain active flows and suspended flows. OpenNaaS asks the OFCs to suspend or permit the flow. Some general features of EXR should also be carefully considered. Exclusive flow scheduling can improve on the mean flow completion time but could increase this value for small flows, compared to shared flow scheduling. Therefore, for the applications which have strict QoS, e.g. response time for small flows, exclusive flow scheduling is not the best option even if it does save energy consumption. NRENs in Europe are the important users of OpenNaaS, and the majority of applications hosted by NRENs are scientific computing that are tolerant of execution time. So exclusive flow scheduling can meet their requirements. Also, the algorithm for fetching the suspended flow could be first-in-first-out, smallest or biggest flow first, etc. We can choose one according to QoS requirements of future implementations. Large-sized flows easily lead to network congestion (Liu et al., 2013), so giving large flows higher priority can further improve the mean network performance, while giving small flows higher priority can reduce the delay in completing them.

In the first mode of invoking green routing in energy-aware OpenNaaS, we assume that network providers have complete knowledge of incoming traffic about the origin node and destination node, bandwidth requirements and the amount of flows in order to plan ahead for expected traffic peaks and reduce the delay of flow control. Such an assumption is not realistic now. The possible way is to predict the incoming traffic for network providers based on historical traffic. There exists some work about workload prediction (Khotanzad and Sadek, 2003; Di et al., 2012) we can use in future.

Energy-aware routing strategies power off unused network components and they impact the oversubscription rate of date center networks. If a burst of traffic arrives, the unused network components require some time to become available. Reliability is an important factor in QoS for some data centers, and a certain oversubscription ratio is required to guarantee the reliability of the DCN. In the process of traffic consolidation, we can guarantee the network availability by checking whether selecting a routing path which powers off some unused switches or links would reduce the requested oversubscription ratio. As a result, the tradeoff between energy savings and network reliability can be achieved. Enabling energy-aware routing strategies only at the non-peak time can lower this negative impact on reliability. During this period, the tolerance for link failure is also higher than at peak time.

10. Conclusion

OpenNaaS is a network management platform that includes SDN support and it interworks several SDN platforms to orchestrate network services; this makes it a suitable management platform for adoption in data centers moving to SDN. The energy-aware OpenNaaS we developed for energy monitoring and routing builds on the consolidated OpenNaaS framework. Our prototype shows we can also measure the energy, cost and sustainability information of networks for providers or users. Based on the prototype, we discuss the combination of flow routing and flow scheduling algorithms. The priority-based shortest routing finds the most power efficient paths from multiple shortest paths. The exclusive scheduling speeds up all the flows on the same link. The simulation results show that this combined strategy can effectively improve network energy efficiency, in particular when traffic contains big size of flows. Meanwhile, this strategy achieves nearly the same mean network throughput with the common shortest routing strategy.

Acknowledgements

This work was supported by NWO through the GreenClouds project, by RAAK-MKB through the Greening the Cloud project, by the Dutch national program COMMIT and by the European Community Seventh Framework Programme under grant agreement no. 605243 (GN3plus). The author thanks Jose Aznar and Isart Canyameres from i2CAT for their providing technical details about OpenNaaS.

References

802.3az, 2010. IEEE Std 802.3az-2010 (Amendment to IEEE Std 802.3-2008).

Abts D, Marty MR, Wells PM, Klausler P, Liu H. Energy proportional datacenter networks. In: Proceedings of the 37th annual international symposium on Computer architecture (ISCA). New York, New York, USA; 2010. pp. 338.

Ahmad RW, Gani A, Hamid SHA, Shiraz M, Yousafzai A, Xia F. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. J Netw Comput Appl 2015;52(0):11–25.

Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication. SIGCOMM '08. ACM, New York, NY, USA; 2008. pp. 63–74.

Bianzino AP, Chiaraviglio L, Mellia M. Distributed algorithms for green IP networks. In: Proceedings – IEEE INFOCOM; 2012. pp. 121–26.

Cianfrani A, Eramo V, Listanti M, Marazza M, Vittorini E. An energy saving routing algorithm for a green ospf protocol. In: INFOCOM IEEE Conference on Computer Communications Workshops, March 2010. pp. 1–5.

Cisco, Introduction to Cisco IOS NetFlow – A Technical Overview. Cisco White Paper; 2012.

Di S, Kondo D, Cirne W. Host load prediction in a google compute cloud with a Bayesian model. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for November 2012. pp. 1–11.

Erickson D. The Beacon OpenFlow Controller; 2013.

Fang W, Liang X, Sun Y, Vasilakos AV. Network element scheduling for achieving energy-aware data center networks. Int J Comput Commun Control 2012;7(2):241–51.

FLOODLIGHT, 2014. Floodlight. ⟨http://www.projectfloodlight.org/⟩, visited February 2015.

Ghijsen M, van der Ham J, Grosso P, de Laat C. Towards an Infrastructure Description Language for Modeling Computing Infrastructures. In: 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), Madrid; 2012. pp. 207–14.

Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. SIGCOMM Comput Commun Rev 2008;39(December (1)):68–73.

Gunaratne C, Christensen K, Nordman B. Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed. Int J Netw Manag 2005;15(September (5)):297–310.

Guo C, Lu G, Li D, Wu H, Zhang X, Shi Y, Tian C, Zhang Y, Lu S, Bcube: a high performance, server-centric network architecture for modular data centers. In: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication. SIGCOMM '09, ACM, New York, NY, USA; 2009. pp. 63–74.

Heller B, Seetharaman S, Mahadevan P. ElasticTree: Saving Energy in Data Center Networks. In: The 7th USENIX Conference on Network Systems Design and Implementation (NSDI); 2010. pp. 2–17.

Jin H, Cheocherngngarn T, Levy D, Smith A, Pan D, Liu J, Pissinou N. Joint host-network optimization for energy-efficient data center networking. In: Proceedings – IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013; 2013. pp. 623–634.

Kandula S, Katabi D, Sinha S, Berger A. Dynamic load balancing without packet reordering. SIGCOMM Comput Commun Rev 2007;37(March (2)):51–62.

Khotanzad A, Sadek N. Multi-scale high-speed network traffic prediction using combination of neural networks. In: Proceedings of the International Joint Conference on Neural Networks, 2003, vol. 2; July 2003. pp. 1071–1075.

Li D, Shang Y, Chen C. Software defined green data center network with exclusive routing. In: IEEE INFOCOM 2014 – IEEE Conference on Computer Communications, April 2014. pp. 1743–51.

Lin G, Soh S, Chin K-W, Lazarescu M. Energy aware two disjoint paths routing. J Netw Comput Appl 2014;43(0):27–41.

Liu R, Gu H, Yu X, Nian X. Distributed flow scheduling in energy-aware data center networks. IEEE Commun Lett 2013;17(4):801–4.

Mahadevan P, Banerjee S, Sharma P, Shah a, Ranganathan P. On energy efficiency for enterprise and data center networks. In: Communications Magazine, IEEE 49 (August); 2011. 94–100.

Mahadevan P, Sharma P, Banerjee S. A Power Benchmarking Framework for Network Devices. In: Proceedings of the 8th International IFIP-TC 6 Networking Conference; 2009. pp. 795–808.

McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. Openflow: enabling innovation in campus networks. SIGCOMM Comput Commun Rev 2008;38(March (2)):69–74.

Nascimento MR, Rothenberg CE, Salvador MR, Corrêa CN, de Lucena SC, Magalhães MF. Virtual routers as a service: the routeflow approach leveraging software-defined networks. In: Proceedings of the 6th International Conference on Future Internet Technologies, ACM; 2011. pp. 34–7.

Nascimento MR, Rothenberg CE, Salvador MR, Magalhães MF. Quagflow: partnering quagga with openflow. In: ACM SIGCOMM Computer Communication Review, vol. 40, ACM; 2010. pp. 441–2.

Nedevschi S, Popa L, Iannaccone G, Ratnasamy S, Wetherall D. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation; 2008. pp. 323–36.

NOX, 2014. NOX and POX OpenFlow controller. ⟨http://www.noxrepo.org/⟩, visited February 2015.

NUAGE, 2014. Nauge network. ⟨http://www.nuagenetworks.net/⟩, visited December 2014.

OPENAAS, 2014. Open platform for networks as a service. ⟨http://www.opennaas.org⟩, visited December 2014.

OPENDAYLIGHT, 2014. OpenDaylight. ⟨http://www.opendaylight.org/⟩, visited February 2015.

RYU, 2014. Ryu SDN framework. ⟨http://osrg.github.io/ryu/⟩, visited February 2015.

Sedgewick R. Algorithms in C third edition Part 5 Graph Algorithms. Addison-Wesley Professional; 2001.

Shang Y, Li D, Xu M. A Comparison Study of Energy Proportionality of Data Center Network Architectures. In: 32nd International Conference on Distributed Computing Systems Workshops; June 2012. pp. 1–7.

TAIL-F, 2014. Tail-f network control system. ⟨http://www.tail-f.com/network-control-system/⟩, visited December 2014.

Thanh N, Nam P, Truong T, Hung N, Doanh L, Rastin P. Enabling experiments for energy-efficient data center networks on OpenFlow-based platform. In: 2012 4th International Conference on Communications and Electronics, ICCE 2012 (2), 2012. pp. 239–44.

Thanh NH, Nam PN, Huong TT, Thuan TN, Duong NM, Van GN, Hung NT, Thu NQ, David H, Christian S. Modeling and experimenting combined smart sleep and power scaling algorithms in energy-aware data center networks. Simul Model Pract Theory 2013;39(0):20–40 s.I.Energy efficiency in grids and clouds.

van der Veldt K, Monga I, Dugan J, de Laat C, Grosso P. Carbon-aware path provisioning for NRENs. In: 2014 International Green Computing Conference (IGCC). Dallas TX USA; 2014.

Wang J, Chen X, Phillips C, Yan Y. Energy efficiency with QoS control in dynamic optical networks with SDN enabled integrated control plane. Comput Netw 2014;78:57–67.

Wang X, Yao Y, Wang X, Lu K, Cao Q. CARPO: Correlation-aware power optimization in data center networks. In: Proceedings IEEE INFOCOM; March 2012. pp. 1125–33.

Xu M, Shang Y, Li D, Wang X. Greening data center networks with throughput-guaranteed power-aware routing. Comput Netw 2013;57(October (15)):2880–99.

Zheng K, Wang X, Li L, Wang X. Joint Power Optimization of Data Center Network and Servers with Correlation Analysis. In: IEEE INFOCOM 2014 - IEEE Conference on Computer Communications; 2014.

Zhu H, van der Veldt K, Grosso P, de Laat C. EDL: an energy-aware semantic model for large-scale infrastructures. Technical report uva-sne, no. 2014-02, University of Amsterdam; February 2014.

Zhu H, van der Veldt K, Zhao Z, Grosso P, Pavlov D, Soeurt J, Liao X, de Laat C. A semantic enhanced power budget calculator for distributed computing using ieee 802.3az. Clust Comput 2015;18(1):61–77.