Green Routing in Software-Defined Data Center Networks Based on OpenNaaS

Hao Zhu
System and Network Engineering group, University of Amsterdam, The Netherlands
h.zhu@uva.nl

Jose Ignacio Aznar Baranda
Fundaci i2CAT, Barcelona, Spain
jose.aznar@i2cat.net

Cees de Laat, Paola Grosso
System and Network Engineering group, University of Amsterdam, The Netherlands
delaat@uva.nl, p.grosso@uva.nl

Keywords: NaaS, OpenNaaS, Energy Monitoring, Green routing, SDN.

Abstract: Today's data center networks such as Fat-tree and BCube, are over-provisioned for multi-path routing; and they suffer from inefficient power usage when traffic is not heavy. Green routing technologies are effective approaches that can fix this problem. In this paper we present a new solution to implementing green routing: we integrated energy-aware routing capabilities in OpenNaaS to make a centralized routing decision of scheduling traffic, which leverage SDN technologies to decouple data forwarding from routing decision. This method is efficient due to managing the network as a whole rather than as a number of individual devices. Based on measured power information, energy-aware OpenNaaS can calculate a green routing path and configure the forwarding rules for the path. In the end, we present a practical use case to demonstrate its functionality and evaluate its effect on power saving by simulation.

1 INTRODUCTION

Network architectures in data centers such as Fat-tree (Al-Fares et al., 2008) and BCube (Guo et al., 2009) are 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 the performance requirement at peak time. However, these resources are usually underused and rarely work at the peak performance. Unfortunately, networks in a data center at the low load still consume more than 90% of power used at the busy-hour load (Heller and Mahadevan, 2010), and effectively suffer from inefficient power usage when traffic is not heavy.

Green routing technologies are effective approaches that can fix this problem. They are in essence strategies which focus on the energy state of network, e.g. energy consumption or CO2 emission rate. They make a routing decision to aggregate traffic over a subset of links and devices in over-provision networks and switch off unused network components. There are two ways to implement green routing. One is in IP networks. For instance, a Green OSPF protocol has been proposed (Cianfrani et al., 2010) and its implementation of energy-aware routing is based on the exclusive use of the topological information exchanged among routers via the OSPF protocol.

The other modality is to focus on Software Defined Networks (SDNs), in particular OpenFlow networks. In the OpenFlow protocol, the control plane (routing decision) decoupled from data plane (data forwarding) is moved to a centralized controller. This controller provides a centralized view of the entire network state such as traffic and topology. Green routing techniques do rely on the precise traffic and topology information, so green routing is easily implemented in the controllers of OpenFlow networks. However, the current implementation of the OpenFlow controllers are usually limited to obtain these information e.g. NOX (NOX, 2014) can only discover network topology, and most of the controllers can't obtain topology or traffic statistics. We decided to concentrate on an existing framework, OpenNaaS.

OpenNaaS (OPENNAAS, 2014) is the outcome of the European Community Mantychore FP7 project.

The main contributors include Juniper, HEAnet, i2CAT etc. 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 technologies.

In this paper, our contribution is that the utilization of the OpenNaaS features to provide green routing capabilities. We integrated an energy-aware bundle in OpenNaaS for monitoring energy and making routing decision. Energy-aware OpenNaaS provides green routing services in a centralized way, as a consequence of effectively decoupling the forwarding and control planes. With our energy-aware OpenNaaS, network users and providers can understand energy usage information of the networks. To be more important, network providers could easily achieve network routing in terms of power consumption, electricity cost and CO2 emission metrics.

The structure of this paper is as follows: Sec. 2 presents related work on green routing technologies and software management platforms. Sec. 3 describes the framework of OpenNaaS and Sec. 4 introduces the design of energy-aware OpenNaaS. Sec. 5 and Sec. 6 provides the practical of energy-aware OpenNaaS and the evaluation of its effect on power saving respectively. Finally, Sec. 7 discusses our conclusions.

2 RELATED WORK

Green routing is an effective solution to save energy and CO2 by aggregating the traffic over a subset of network links or network devices in over-provisioned networks. (Xu et al., 2013) discussed an algorithm for reducing power consumption of high-density data center networks from the routing perspective while meeting the total throughput requirement. (Chabarek et al., 2008) studied how to save energy of aggregate traffic through optimal route selection and configuration in wide-area networks. This work is implemented in traditional (e.g. IP) networks, while our green routing solution focuses on SDNs in data centers. (Heller and Mahadevan, 2010) presented ElasticTree for adapting the energy 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 energy-minimization routes which meet current traffic condition. But ElasticTree can't discover the topology and monitor the power consumption of switches, so it assumes that the topology and power are always invariable once they are input. Our solution has wide applicable range as it is based on OpenNaaS that can dynamically obtain these information and support multiple types of SDN controllers.

Some cloud/network management platforms are supporting SDN technologies, which have efficient network management mechanisms. OpenNebula and OpenStack are both open-source cloud computing platforms for public and private clouds. Cloud administrator can control computing, storage and network resource in clouds through APIs of them. Their network capabilities are limited at IP, vLANs and SDN currently. OpenNaaS has richer network capabilities e.g. Bandwidth on Demand (BoD), topology discovery than OpenNebula and OpenStack. We can implement more energy-aware capabilities like energy-aware BoD by combining energy monitoring or management 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.

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

3 OpenNaaS FRAMEWORK

OpenNaaS is a management platform that enables the abstraction of underlying network technologies and offers NaaS-based services. It provides the capabilities to configure and deploy novel network services, enabling administrators to manage any part of the infrastructure or to deploy any type of application on top of it.

OpenNaaS abstracts the physical resources enabling to decouple physical topology and vendor-specific details from their control and management features to be offered to the tenants. The fundamental unit that OpenNaaS uses to accomplish this is the *Resource*. A Resource models a device and represents a manageable unit inside the NaaS concept e.g., switch, a router, a link, a logical router, a network. *Capabilities* shape resource functionality and provide an interface to given resource functionality, e.g. for a router: OSPF, IPv6, create/manage logical routers, etc. Fig. 1 shows this OpenNaaS vision in which physical devices are abstracted into resources and capabilities

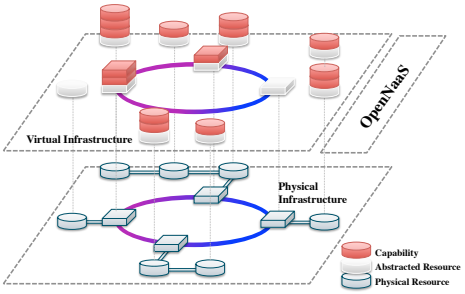

Figure 1: OpenNaaS abstraction view: resources and capabilities.

whose management can be delegated to upper application layers. 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 extensions. 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 easily implement more functionality by creating capabilities in a new extension bundle. We exploited this feature to create an energy-aware bundle (see section 4.2).

OpenNaaS has a complete view of the entire network and interacts directly with the data plane through its SDN capabilities. OpenNaaS interworks several SDN platforms (OpenDaylight (OPENDAYLIGHT, 2014), RYU (RYU, 2014) and Floodlight (FLOODLIGHT, 2014) controllers) to orchestrate network services on top of SDN-based infrastructures and to enable new SDN applications to 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 REST APIs of OpenFlow controllers for port statistics and flow forwarding (create, remove and get forwarding rules).

4 DESIGN OF ENERGY-AWARE OpenNaaS

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

4.1 Architecture

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

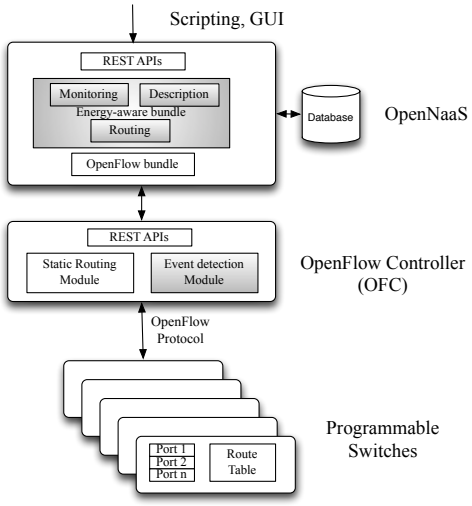

Figure 2: Energy-aware OpenNaaS architecture.

on top of an OpenFlow controller (OFC), and the OpenFlow-enabled switches are connected with the OFC. There are two methods to invoke green routing functionality in OpenNaaS: 1) A network provider or user directly sends a green routing request through scripts or GUI to OpenNaaS; 2) The OFC detects a packet-in event in a switch and then sends a routing request to the OpenNaaS server. The *energy-aware bundle* in OpenNaaS receives and handles the request. This bundle is capable of calculating a green route and making a routing decision. The energy-aware bundle, which communicates with the OFC through the *OpenFlow bundle* calls the defined REST APIs in the OFC to add static flow rules for the route. The *static routing module* in the OFC adds flow rules by inserting flow entries in flow tables of the switches.

4.2 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 description and monitoring are essential in energy management for networks. Energy management mechanisms usually depend on topology information, traffic information and energy usage information to determine the configuration of network when scheduling of traffic. OpenNaaS can already provide the first two information given that it maintains an abstract overview of whole network.

We implemented both energy monitoring capabilities and energy description capabilities:

Energy monitoring capabilities obtain and provide energy usage information from power meters in the observed metrics: (power, energy, CO2 emission

rate, electricity price) and from data statistics in the calculated metrics: (total Electricity, CO2 emission, energy efficiency). The measurement data is saved in a database of OpenNaaS, as shown in Fig. 2. The capabilities are responsible for the communication with power meters. We have created the power meter drivers for the Simple Network Management Protocol (SNMP) access to different power meter vendors e.g. Rackitivity and APC Power Distribution Units (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 route.

Energy description capabilities describe and create meta information about energy source, power meters, green metric, power state etc. in OpenNaaS. We employ Energy Description Language (EDL) ontology (Zhu et al., 2014), which reuses Infrastructure and Network Description Language (INDL) ontology to describe the resources and network infrastructure that connects these resources. 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 information that EDL can describe is the relationship of power meters and remote network devices, so that we know that the measured energy usage information from a 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. We adopt a greedy routing algorithm in the initial prototype. Once a OpenNaaS user selects the green metric to optimize upon, the algorithm traverses all the possible network routes between the original host and destination host of the flow and it chooses the route with the lowest value of the metric. The calculated route is converted to a list of the OpenFlow flows in the Json format according to specific OFCs, and then the OpenFlow bundle sends the list to the OFCs for creating flow forwarding rules.

OpenNaaS extends the INDL model for the description of OpenFlow route table and network route, which are used by green routing capabilities. The switch ports are identified by numbers, and the 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 next hop. A route or a routing path is a list of route tables.

Similar to route tables in OpenNaaS, OpenFlow flows also include the identifier of switches and the output ports of the switches. OpenNaaS has the knowledge of network topology that also depicts the connection 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 the switch. OpenNaaS can add the flow rules in the correct OFC.

4.3 OpenFlow Controllers

In our design, OFCs insert flow forwarding rules 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 didn't change the static routing module and its REST APIs in OFCs. The OpenFlow bundle in the OpenNaaS server calls different APIs according to the type of OFCs: Static Flow Pusher APIs in Floodlight and Static Routing APIs in OpenDaylight.

To support the second mode of invocation, we created the *event detection module* in the controllers to forward routing requests from switches to OpenNaaS. The module detects a packet-in event, and then sends a REST routing request to the energy-aware bundle. The message contains the source and destination IP of the packet, the DPID of the switch and the input port where the packet enters the switch.

5 PROTOTYPE

We developed a prototype to show the functionality of energy-aware OpenNaaS. The source codes are available online[1]. The prototype includes a web client GUI, which communicates with the capabilities of OpenNaaS through REST APIs.

We emulated a Mininet network with a topology of 6 OpenFlow switches and 5 hosts, shown in Fig. 3. The switches had the same network capacity and were divided into two groups, controlled by Floodlight and OpenDaylight controllers respectively. To have different CO2 emission rates, we assumed the

[1] https://bitbucket.org/uva-sne/greennet-demo

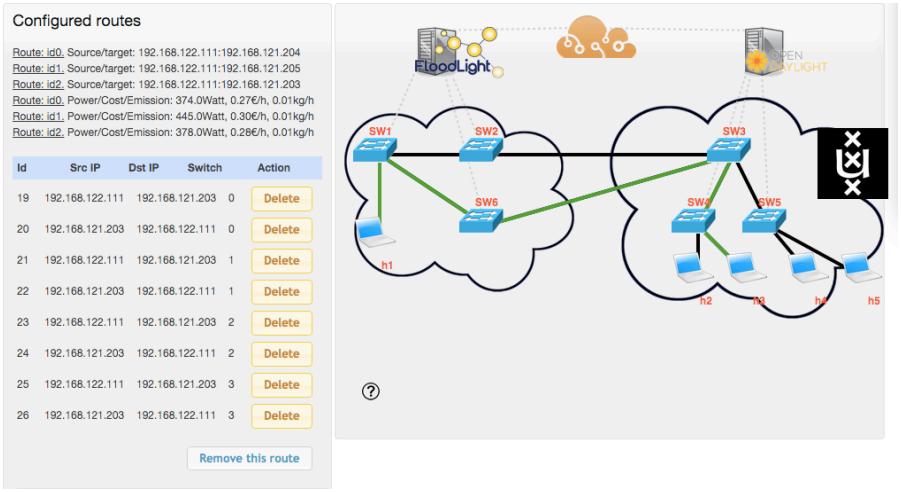

Figure 3: The screen shot of the topology and the configured route in emulated network environment.

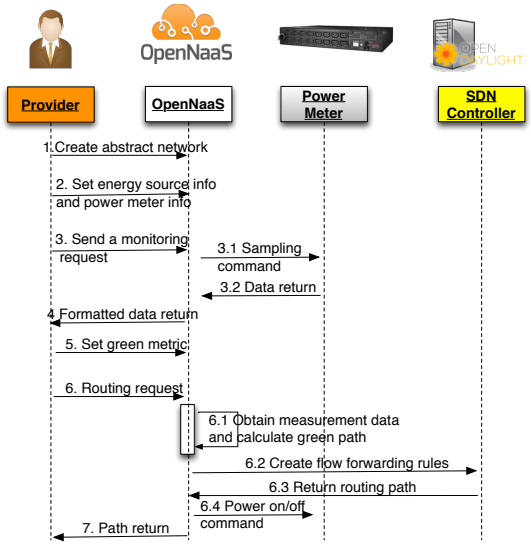

Figure 4: The flowchart of a general usecase of energy-aware OpenNaaS (Step 1 to Step 7).

two groups consumed solar and thermal energy respectively. We ran the OpenNaaS server in the same machine.

Fig. 4 presents the flowchart of 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 (*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 request to the power meter (*Step 3.1*) and reads the measurement data out (*Step 3.2*). The data is instantiated by EDL, and data is saved in a database or returned to the provider (*Step 4*).

Then the provider decides on a green optimization metric (*Step 5*) and submits a request to OpenNaaS for a path between a source and a destination (*Step 6*). OpenNaaS obtains the energy information of all the possible network routes between the end points; if the information is not available in the database, it will send a monitoring request immediately to obtain real-time energy information (*Step 6.1*). OpenNaaS selects a route with the least value and it interacts with the OFCs to create flow forwarding rules in the switches (*Step 6.2*). In the end, the formatted route information returns to the provider if the flow rules are successfully created (*Step 6.3*). According to the result route, 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 the output of our prototype when three routes are configured: id0, id1 and id2. For each one of the routes, our prototype provides the value of the total power, cost and CO2 emission metrics.

6 EVALUATION

We explore how much power to consume for data transmission with different routing algorithms. In the beginning, we employ the shortest routing algorithm and keep all the switches and links always awake. We

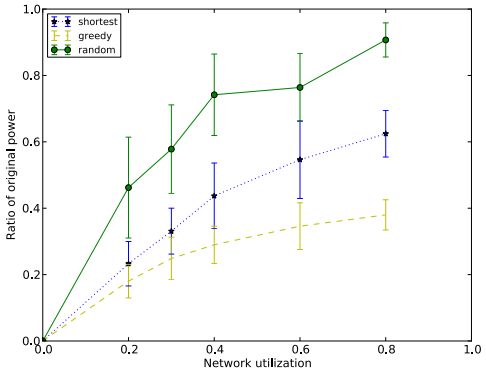

Figure 5: Network power saving against the network utilization in the BCube network.

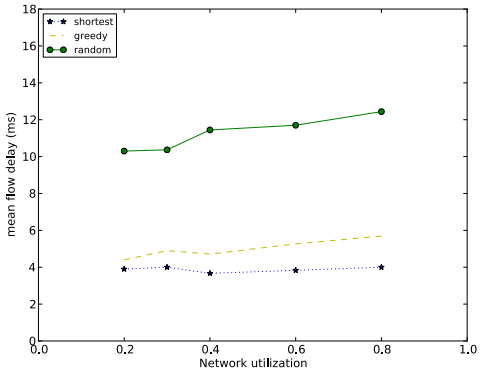

Figure 6: Mean flow delay time against the network utilization in the BCube network.

define the network power consumption at this moment as the original power. Then we evaluate green routing algorithms that enable idle switches and links to go to sleep. We compare the power consumption and average flow delay time of network when using power-greedy, the shortest and random routing algorithms. Power-greedy that is used in our prototype finds a routing path for current flow, which increases the least power consumption for the overall network.

In our prototype, the OpenFlow resources are emulated but their power consumption is from an actual power meter for real devices. So we can't obtain the actual relation between network traffic and network power consumption. We perform our study of the routing algorithms by simulation. In our simulation we use BCube(2,3) data center topology. There are 8 8-port switches in total. The capacity of each link in the simulated topology is 1Gbps; each link has 1 millisecond (ms) delay. We take the maximum of the total number of flows on all servers as 100% network load and vary the percentage of the number of flows in the topology to simulate different network utilization. The source and destination of each flow are randomly chosen from leaf switches. The speed of each flow is uniform distribution, but it has to guarantee that the largest link throughput in the network is less 1 Gbps.

Fig. 5 shows the power saving of three green routing algorithms against the network utilization in the BCube network. The y-axis here depicts the power consumption of the algorithms is how much ratio of the original power consumption. We can find that the power saving becomes less when lower network utilization. As the less networking components are idle when the traffic becomes more intensive. When network utilization grows from 20% to 80%, the power-greedy routing only uses less that 40% of the original power. And it can save at most 20% power than the shortest routing.

Fig. 6 shows mean flow delay time of three green routing algorithms against the network utilization in the BCube network. We can find the mean flow delay of greedy routing increases from 4 ms to 5.6 ms while that of shortest routing is nearly stable at 4 ms. It tells us that greedy routing finds power efficient paths although that have more hops.

Our evaluation results show that the greedy routing algorithm is effective at saving power, in particular when traffic is not heavy. It can be used at non-peak time or in scenarios with tolerant performance requirement. In future, we expect to study the trade-off between power consumption and performance of networks for routing algorithms.

7 CONCLUSIONS

OpenNaaS is an efficient network management platform for infrastructure providers. OpenNaaS 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 builds on the consolidated OpenNaaS framework, and it's the first implementation of this kind. Our system can measure the energy, cost and sustainability information of networks for providers or users. It can calculate and create a green-greedy routing path based on these information for them. Our experiments show that our prototype with power-greedy routing algorithm can effectively save power.

The scalability of energy-aware OpenNaaS for large-scale network depends on two factors: one is the scalability of calculating routing paths in green routing algorithms; another is the scalability of flow configuration in the SDN control plane. The later prob-

lem that extends SDN to large-scale networks is still challenging (McCauley et al., 2013) (Fu et al., 2014). We will discuss the scalability of energy-aware OpenNaaS in future.

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). We also thank Isart Canyameres from i2CAT for his comments on the system implementation.

REFERENCES

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

Chabarek, J., Sommers, J., Barford, P., Estan, C., Tsiang, D., and Wright, S. (2008). Power Awareness in Network Design and Routing. In *IEEE INFOCOM*, pages 457–465.

Cianfrani, A., Eramo, V., Listanti, M., Marazza, M., and Vittorini, E. (2010). An energy saving routing algorithm for a green ospf protocol. In *INFOCOM*, pages 1–5.

FLOODLIGHT (2014). Floodlight. `http://www.projectfloodlight.org/`. Visited Feb. 2015.

Fu, Y., Bi, J., Gao, K., Chen, Z., Wu, J., and Hao, B. (2014). Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 569–576.

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). Bcube: A high performance, server-centric network architecture for modular data centers. SIGCOMM '09, pages 63–74, New York, NY, USA. ACM.

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

McCauley, J., Panda, A., Casado, M., Koponen, T., and Shenker, S. (2013). Extending sdn to large-scale networks. *Open Networking Summit*, pages 1–2.

Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., Corrêa, C. N., de Lucena, S. C., and Magalhães, M. F. (2011). Virtual routers as a service: the routeflow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, pages 34–37. ACM.

Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., and Magalhães, M. F. (2010). Quagflow: partnering quagga with openflow. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 441–442. ACM.

NOX (2014). NOX and POX OpenFlow controller. http://www.noxrepo.org/. Visited Feb. 2015.

OPENDAYLIGHT (2014). OpenDaylight. `http://www.opendaylight.org/`. Visited Feb. 2015.

OPENNAAS (2014). Open platform for networks as a service. `http://www.opennaas.org`. Visited Dec. 2014.

RYU (2014). Ryu SDN framework. `http://osrg.github.io/ryu/`. Visited Feb. 2015.

Xu, M., Shang, Y., Li, D., and Wang, X. (2013). Greening data center networks with throughput-guaranteed power-aware routing. *Computer Networks*, 57(15):2880–2899.

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