SEEDS International Conference

Development of An Ecology-oriented SDN Framework

Chandra Satriana[1], Oleg Sadov[2], Vladimir Grudinin[2]

[1] Pervasive Computing and Communication for Sustainable Development (PERCCOM), Universite de Lorraine, 34 Cours Léopold, 54000 Nancy, France

[2] Saint Petersburg State University of Information Technologies, Mechanics and Optics (ITMO University), Birzhevaya liniya, 14, 199034 Saint-Petersburg, Russia

Keywords: Software-Defined Network, Energy, Green IT, OpenFlow.

Abstract

ICT contributed to about 0.83 GtCO2 emissions where the 37% comes from the telecoms infrastructures. At the same time, the increasing cost of energy has been hindering the industry in providing more affordable services for the users. One of the sources of these problems is said to be the rigidity of the current network infrastructures which limits innovations in the network. SDN (Software Defined Network) has emerged as one of the prominent solutions with its idea of abstraction, visibility, and programmability in the network. Nevertheless, there are still significant efforts needed to actually utilize it to create a more energy and environmentally friendly network. In this paper, we suggested and developed a platform for developing ecology-related SDN applications. The main approach we take in realizing this goal is by maximizing the abstractions provided by OpenFlow and to expose RESTful interfaces to modules which enable energy saving in the network. While OpenFlow is made to be the standard for SDN protocol, there are still some mechanisms not defined in its specification, especially related to energy saving. To solve this, we created REST interfaces for setting of QoS (Quality of Service) in the switches which can maximize network utilization. We also created the interfaces for minimizing the required network resources in delivering packets across the network. This is achieved by utilizing redundant links when it is needed, but disabling them when the load in the network decreases. While the amount of power saved by disabling a port on a switch is not so much, this can be a huge saving when it is applied to real and bigger network infrastructures.

INTRODUCTION

ICT currently contributed to about 0.83 GtCO2 emissions where the 37% of it comes from the telecoms infrastructures [28]. Similarly, the network infrastructure in a data center contributes to about 20% of power consumption [29]. While this number does not seem huge, 3 billion kWh was consumed by the networking elements in the data centers in the United States [21]. Innovation in this area can certainly contribute to not only saving the environment but also improve business by reducing energy costs.

Unfortunately, the rigidity of the current network infrastructures has been said to limit innovations in the network. Nevertheless, Software-Defined Network (SDN) has emerged as one of the prominent solutions with its idea of abstraction, visibility, and programmability in the network. Basically, it is achieved via the separation between the control plane and the forwarding plane of a network switch. The forwarding plane itself is programmable via the interfaces specified by OpenFlow protocol. OpenFlow also specifies the mechanisms needed to exist in SDN capable switches and how these switches can be programmed.

However, if the goal is to utilize SDN to create a more energy and environmentally friendly network, then significant efforts are needed to actually realize it. Currently there is not yet a framework in which users, specifically developers, can make use of to create ecology-related applications. To solve this problem, we propose an ecology-oriented SDN framework in which users can easily use to save energy in their network.

The developed Ecology SDN Framework is designed with idea of maximizing the abstraction feature of SDN and to integrate researched mechanisms to save energy in a network. The framework has some modules including REST API for setting of QoS in OpenFlow switch, Adaptive Link Rate to accustom interface rate to network utilization, and network optimization to increase performance while at the same time saving energy.

This framework is developed as a master thesis for a master degree of Pervasive Computing and Communication for Sustainable Development (PERCCOM). PERCCOM is an Erasmus Mundus Joint Master Degree (EMJMD) which focuses on building a green ICT system.

LITERATURE REVIEW

Software-Defined Network

According to [3] SDN is defined as the physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices. In other words, SDN allows the network to be programmable.

The logical architecture of SDN is shown in Fig.1. On the bottom layer, the network of physical switches are abstracted and centrally managed through an SDN controller at the contol layer. Other than managing the current state of the underlying network, the controller also provides Application Programming Interface (API) which can be used by SDN applications to provide network services such as routing, traffic engineering, energy usage, quality of service, and security.

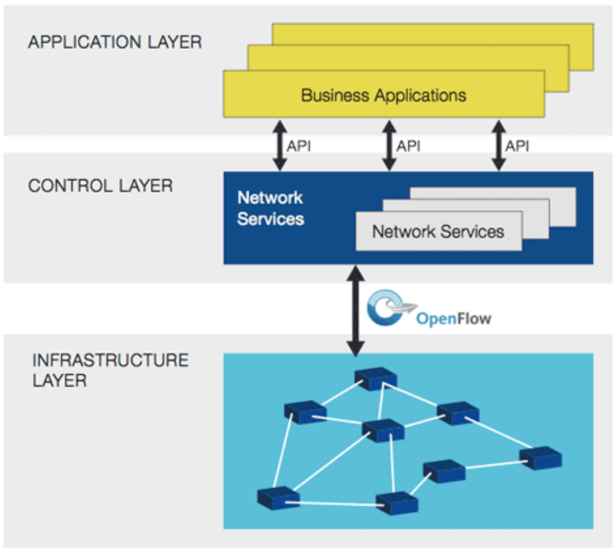
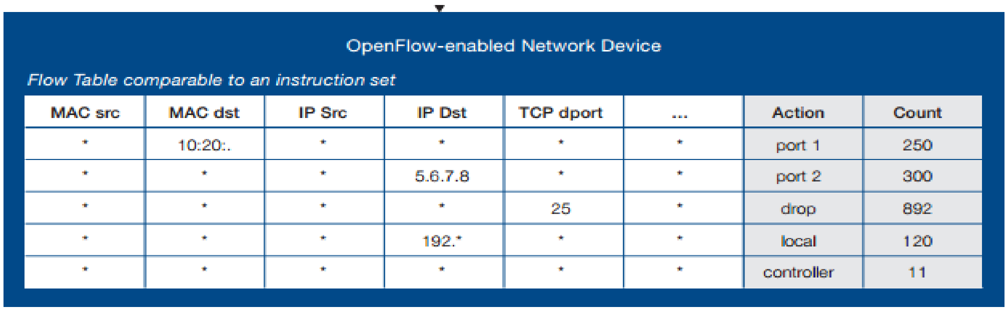


Figure 1. Software-Defined Network Architecture

One particular attention on the architecture, is the protocol used for communication between the infrastructure layer and control layer. Currently there is no standard protocol, but the most used one is the OpenFlow. Basically it specifies the instructions or commands which can be executed by the SDN controllers to modify the forwarding tables of the underlying infrastructure layer (phisical or virtual switches). Fig.2 is an example of instruction which can be set on the OpenFlow-enabled switches. Depending on the MAC destination address, MAC source address IP address, and TCP port, a certain packet which match those fields will be forwarded to port 1 of the switch or forwarded to the controller, based on the value on the action field.



OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figure 2. Example of OpenFlow Instruction Set [ ]

Representational State Transfer (REST)

Representational State Transfer (REST) is a coordinated set of architectural constraints that attempts to miimize latency and network communication while at the same time maximizing the independence and scalibility of component implementations (Roy fielding). REST also enables the caching and reuse of interactions, dynamic substitutability of components, and processing of actions by intermediaries, thereby meeting the needs of an Internet-scale distributed hypermedia system.

According to (REST Api design patters for SDN), adopting REST for SDN northbound API has some benefits such as: decentralized management of dynamic resources, heterogeneous clients, service composition, localized migration, and scalability.

RESEARCH REVIEW AND METHODOLOGY

The research approach adopted in this work is formulative research. This approach is suggested by Morrison and George (1995) in which they also suggested other research approaches including evaluative research, descriptive research, and developmental research. Formulative research involves development and refinement of theories, models, or frameworks that govern research activities, and support scientific progress through paradigm shifts. Also, most of formulative work involves synthesizing and integrating information and then developing guidelines, models, or frameworks.

Agile software development approach is followed in conducting this research. Pekka (2002) explains that this development approach has some important characteristics such as modularity on development process level, iterative with short cycles enabling fast verifications and corrections, adaptive with possible emergent new risks, incremental process approach that allows functioning application building in small steps, and collaborative and communicative working style.

There are different agile software development methodologies, such as feature driven development (FDD), scrum, rational unified process (RUP), and adaptive software development. Out of these, RUP is chosen as it is more appropriate for iterative development in object-oriented approach. RUP's project lifespan consists of four phases: inception, elaboration, construction, and transition, as depicted in figure 3.

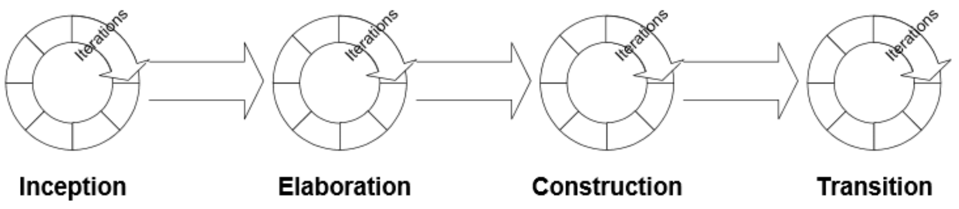


Figure 3. RUP Phases

ECOLOGY SDN FRAMEWORK DESIGN AND IMPLEMENTATION

Inception

Inception phase which comprises of requirement gathering and analysis is the first step in RUP. In our work, we analysed some of the important requirements:

1. Develop a solution that makes the network infrastructure to be ecology-friendly
2. The solution should be implemented in a form of framework which can be utilized by other users or other developers in which to build upon more green solutions
3. It should use emerging technologies in SDN which provides abstraction, visibility, and programmability in a network

4. It should have good quality requirements such as modularity, composability, and scalability.

Elaboration

In this step, high level as well as detailed design and implementation is laid out.

There are two main ideas we apply in designing the framework is by maximizing the abstraction provided in SDN through OpenFlow and to implement researched mechanisms to save energy in a network.

Figure 3 shows the high level view of the framework. This framework sits on top of Ryu SDN controller. The main idea is that this framework extends the capability of Ryu which communicates with the underlying physical or virtual network.

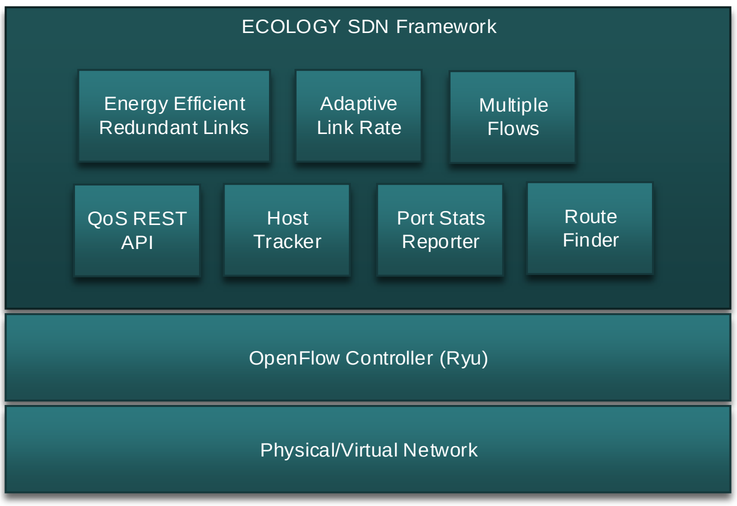


Figure 4. Ecology SDN Framework Architecture

The idea of creating separate modules which serves their own individual functions is to support composability in the framework. This is also an idea borrowed from Service-oriented Architecture which supports in creating small functional services which can collaborate in achieving bigger functionalities. This approach is also good for scalability because in the system, any new application or functionality can get certain data from an already running module instead of implementing its own mechanism.

The individual modules are described in the following subsections.

QoS REST API

OpenFlow defines the mechanisms to access the forwarding plane of a switch and the features needed to be implemented in it through OpenFlow specification [2]. But it does not specify the mechanisms for queue settings which can be useful in guaranteeing Quality of Service (QoS) and in implementing the adaptive link rate feature. This certainly becomes a problem when the network consists of switches from different vendors, physical or virtual switch, where each has their own way to configure the queue, decreasing the abstraction nature of SDN itself. To overcome this, we create a RESTful API to set the queue settings in CPqD OpenFlow 1.2 and 1.3 compatible software switch. REST (Representational State Transfer) is chosen because it has some benefits such as: service composition and localized migration [3]. The API we have implemented is described in Table 1.

Table 1. QoS REST API Endpoints

Endpoint	Description
PUT /v1.0/conf/switches/{SWITCH_ID}	Set switch address
POST /qos/queue/{SWITCH_ID}	Set QoS settings with data : port-name, queues: min-rate:, max-rate:
GET /qos/queue/{SWITCH_ID}	Get all queues settings in the switch
DELETE /qos/queue/{SWITCH_ID}/{PORT}/{QUEUE_ID}	Delete a specific queue

Port Stats Reporter

Port stats reporter is the component in the SDN framework which reports port statistics of ports in OpenFlow switch. The statistics consists of data such the number of packets received/transmitted, received/transmitted bytes, packets dropped and received/transmit errors.

This data then will be used by other applications to monitor the bandwidth utilization of the ports of switches. To communicate this data in the application, the application that wants to use it only need to register to receive the event. In the implementation, this is achieved by adding a decorator of ofp_event.EventOFPPortStatsReply to the python method inside the application.

The application which receives the data then will calculate the utilization. The calculation of the utilization is carried out with the following formula:

$$\frac{(\Delta Tx + \Delta Rx) \times 8 \, x \, 100}{Tmeasurement \times ifBw}$$

where Tx and Rx is the number of packets received and transmitted on that interface respectively, Tmeasurement is duration since the previous measurement and ifBw is the interface bandwidth or the maximum capacity of the interface.

Energy Saving in Redundant Links

Redundant links in a network are usually used either to increase performance by utilizing both links, or as a backup mechanism when the normal link is down. In this framework we created a sample application which utilized the redundant links when the load of the network require more capacity and turn the redundant link off when such link capacity is not needed.

Figure 2 shows the example network we setup in Mininet. To avoid ARP broadcast storm, we set flow rules so that only one of the ports is used to transmit these broadcast packets

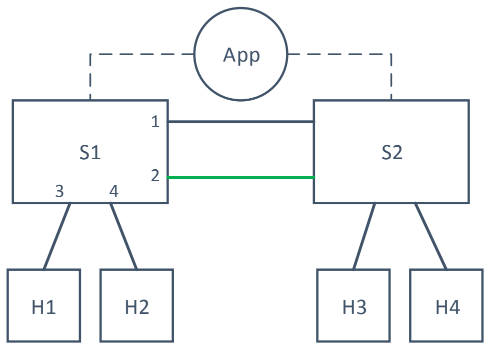


Figure 5. Redundant Link Example Scenario

This is achieved by installing flow rules which will drop packets coming in from port 2 and have Ethernet destination address of broadcast address. Then, to use both links we decided to load balance the traffic. A simple selection mechanism is implemented. Hosts connected to odd port number will go to port 1 and hosts connected to even port number will go to port 2, that is when their destination is a host in another switch. At the same time, the Port Stats module reports the number of transmitted and received bytes on the switch's ports.

Then the module applies the rules depending on the policy specified, such as to disable port 2 when the utilization of port 1 + utilization port 2 is under 90%. The disabling of the port is achieved by sending ofppc_port_down message to the port. Host Tracker module is used to get information of the mac addresses connected to the switches and used in installing balancing flows.

Adaptive Link Rate

According to [4], 1 Gbps Ethernet Links consume 4 W more than 100 Mbps links, while both idle and fully utilized Ethernet links consume same amount of power. The suggested mechanism to save the power is to use Adaptive Link Rate (ALR), which is by adaptively varying the link rate based on the load or utilization in the network. In this framework we provide a module which will decrease the link rate of the switches, when the utilization of the network is under 10% or certain defined value. To implement this, firstly we setup several queue settings in the switch. Using the REST QoS API we developed, queue 1 and 2 are set up, where each has a rate of 10 Mbps and 100 Mbps respectively. When the utilization on a port is low, a qos rule is installed to flow table 0 in which traffic flows which have the destination of that host will be queued to queue id 1, and also to be forwarded to flow table 1 which has the exact information on which port to output this flow.

Multiple Flows

A trivial yet working approach in saving energy is by increasing the performance of network itself during high utilization. By increasing the network performance, the required time to do the work-- in this case transferring the data-- can be achieved faster, and the resources involved may rest earlier once the work is done.

In our work we test the working of the module by sending data from H1 and H2 using BBCP tool. BBCP is a tool to securely and quickly (approaching line speeds) from source to target. The network architecture is depicted in figure 6.

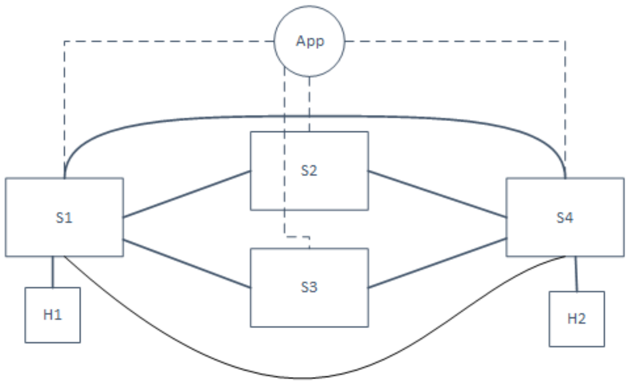


Figure 6. Multiple Paths Network Example

A BBCP streams should be sent from H1 to H2 through the available paths. It is assumed that the available paths are known to the SDN application. These are the paths from H1 to H2:

1. H1 – (Port 3)S2(Port 15) – (Port 15)S5(Port 4)
2. H1 – (Port 3)S2(Port 14) – (Port 14)S3(Port 15) – (Port 13)S5(Port 4)
3. H1 – (Port 3)S2(Port 14) – (Port 13)S4(Port 15) – (Port 14)S5(Port 4)
4. H1 – (Port 3)S2(Port 16) – (Port 16)S4(Port 4)

To achieve the load balancing, BBCP streams from H1 are routed through the different paths. Each BBCP stream can be identified by its TCP source port because each stream has the same TCP destination port (5031), but different TCP source port. The activity diagram below shows the logic used in the application to load balance the BBCP streams:

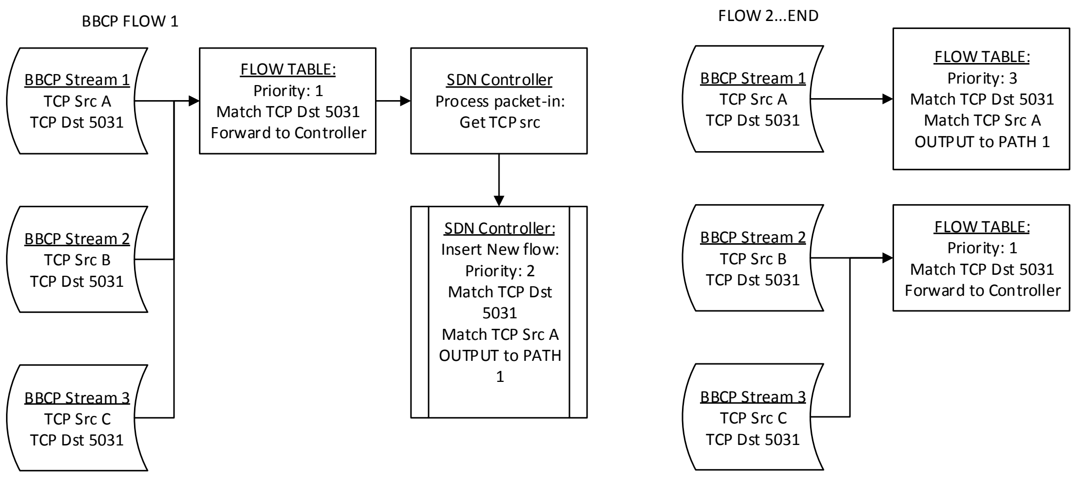

BBCP FLOW 1
BBCP Stream 1 TCP Src A TCP Dst 5031
BBCP Stream 2 TCP Src B TCP Dst 5031
BBCP Stream 3 TCP Src C TCP Dst 5031
FLOW TABLE: Priority: 1 Match TCP Dst 5031 Forward to Controller
SDN Controller Process packet-in: Get TCP src
SDN Controller: Insert New flow: Priority: 2 Match TCP Dst 5031 Match TCP Src A OUTPUT to PATH 1
FLOW 2...END
BBCP Stream 1 TCP Src A TCP Dst 5031
BBCP Stream 2 TCP Src B TCP Dst 5031
BBCP Stream 3 TCP Src C TCP Dst 5031
FLOW TABLE: Priority: 3 Match TCP Dst 5031 Match TCP Src A OUTPUT to PATH 1
FLOW TABLE: Priority: 1 Match TCP Dst 5031 Forward to Controller

Figure 7. Multipathing Module Flow Diagram

The first incoming BBCP stream from H1 for example will be matched to the flow rule: match tcp_dst 5031. This flow is then forwarded to controller where further processing takes place. This process is to read the TCP source port of the flow and to insert a new flow rule matching the TCP source port and TCP destination port. The next flow coming from this same BBCP stream will then be matched to this new flow rule, for example will be output to path 1, instead of matching the previous rule which will forward it to the controller, because the new rule has higher priority.

Host Tracker

The application keep tracks of the hosts connected to the switches. The data that is stored is MAC address of the host, IP address of the host, DPID of the switch and port number in which the host is connected to, timestamp to store the time the mac address is stored to ensure the freshness of the data,

Route Finder

The route finder finds all the available paths from node 1 to node 2. For example it is used in multipath modules to find the routes from host 1 to host 2.

ARP Handler

ARP handler is used to prevent ARP broadcast storms in the network as well as helping in answering ARP requests.

RESEARCH RESULTS

Identify the relevant question that is being answered and then provide the data and experimental results that relate to the key topic. Figures, tables and graphs should be clearly presented. Each time data is introduced there should be an opening sentences that informs the reader the exact nature of the data and its relevance

DISCUSSION

The research results agree with the literature study.

CONCLUSION

The Ecology SDN Framework we developed is able to enforce some of the known mechanisms to save energy in a network infrastructure. The programmability nature of SDN has been important in making this possible. This has shown the potential of SDN in supporting innovation in the networking areas to save the environment. The framework’s code is open sourced and can be accessed at (Satriana, 2015)

Further direction on this research will be to conduct detailed measurement on the amount of energy saved when using this framework. While the framework is made by implementing the researched mechanisms to optimize energy usage in a network, it will encourage more people to develop the framework when such measurements are available.

REFERENCES

The Climate Group, Global eSustainability Initiative (GeSI). (2008) “Smart 2020: Enabling low carbon economy in the information age”

Open Networking Foundation. (2012) OpenFlow Switch Specification version 1.3.3.

Wei Zhou; Li Li; Min Luo; Wu Chou. (2014) "REST API Design Patterns for SDN Northbound API," WAINA, 2014 28th International Conference on , vol., no., pp.358,365.

Gunaratne, C.; Christensen, K.; Nordman, B.; Suen, S.. (2008)"Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR)," Computers, IEEE Transactions on , vol.57, no.4, pp.448,461

Satriana, C. (2015) Open Source Code repository of EcoSDN. https://github.com/itmo-infocom/EcoSDN

Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. 2010. ElasticTree: saving energy in data center networks. InProceedings of the 7th USENIX conference on Networked systems design and implementation(NSDI'10). USENIX Association, Berkeley, CA, USA, 17-17.

Khondoker, R.; Zaalouk, A.; Marx, R.; Bayarou, K., "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," Computer Applications and Information Systems (WCCAIS), 2014 World Congress on , vol., no., pp.1,7, 17-19 Jan. 2014

Shiraki, O.; Nakagawa, Y.; Hyoudou, K.; Kobayashi, S.; Shimizu, T., "Managing storage flows with SDN approach in I/O converged networks," Globecom Workshops (GC Wkshps), 2013 IEEE , vol., no., pp.890,895, 9-13 Dec. 2013

M. Said Seddiki, Muhammad Shahbaz, Sean Donovan, Sarthak Grover, Miseon Park, Nick Feamster, and Ye-Qiong Song. 2014. FlowQoS: QoS for the rest of us. In Proceedings of the third workshop on Hot topics in software defined networking (HotSDN '14). ACM, New York, NY, USA, 207-208

Roy T. Fielding and Richard N. Taylor. 2002. Principled design of the modern Web architecture. ACM Trans. Internet Technol. 2, 2 (May 2002), 115-150. DOI=10.1145/514183.514185 http://doi.acm.org/10.1145/514183.514185

Peter M. Vitousek 1994. Beyond Global Warming: Ecology and Global Change. Ecology 75:1861–1876. http://dx.doi.org/10.2307/1941591

Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. 2008. The cost of a cloud: research problems in data center networks. SIGCOMM Comput. Commun. Rev. 39, 1 (December 2008), 68-73. DOI=10.1145/1496091.1496103

Andy Hanushevsky. BBCP https://www.slac.stanford.edu/~abh/bbcp/.