Saving Energy in OpenFlow Computer Networks

Marco Schaap
5871352

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. Paola Grosso
Fahimeh Alizadeh Moghaddam

System and Network Engineering Group
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 26th, 2015

Contents

List of Figures

List of Tables

Abstract

New innovations in the information and communication sector have ensured that more people have access to the Internet and the prospect is that this number will even grow further. With this also the energy consumption, and thus the CO_2 will increase. It is thus important to look for cleaver solutions to reduce the energy consumption. This thesis analyses the influence of an efficient scheduling algorithm on the energy consumption of computer networks. We compare the influence of two different scheduling algorithm in a Software-Defined Network which deploys OpenFlow. We present a linear programming algorithm that reduces the number of active switches and hereby reduces the energy consumption of the network. While the shortest path algorithm functions as a baseline for comparison. For the simulation a realistic network environment is setup in which we run extensive experiments to collect data for analysis. The algorithm proposed in this project reduces the energy consumption of the network with 30% and still manages to keep the bandwidth at an acceptable level.

1 Introduction

Innovations in the field of information and communications technology (ICT) have led to a huge increase in Internet users. In a span of only 20 years the number of Internet users rose roughly by a factor of 200. In 1995 there where around 16 million Internet users (0.4% of the world population). In 2014 the estimation is that around 42.4% of the world population has access to the Internet, meaning 3,079 million users. [3]

However, this growth has a downside: ICT systems are responsible for the same amount of CO_2 emissions as global air travel. [7] And if this growth continues it will not only cost a lot of money it will also worsen the climate problem. It is thus important to look for ways to make ICT services more energy efficient and hereby reducing the CO_2 emission.

For this project we are especially interested in data centers, because they have become common and essential to the functioning of business, communication, academic, and governmental systems. Due to increasing demands for computer resources, data centers are growing in numbers and they consume huge amounts of energy that have various impacts. Such as increasing energy costs, extra emission of greenhouse gasses, and extra load on the existing power grid. [4]

Data centres consume huge amounts of energy to provide a reliable and scalable computing infrastructure for various Internet services. The energy consumption of data centers in the year 2013 was 91 billion kWh in the United States alone and the prospect is this will increase to 138 billion kWh in the year 2020 (see figure 1). This costs the U.S 9 billion dollars a year. It is thus important to find solutions to reduce the energy consumption of data centers.

The biggest part of the energy consumption in data centres comes from servers and cooling, which is responsible for 70% of energy consumption. The main focus of this research is the energy consumption of the networking component in data centers, which is responsible for 10-20% of the energy consumption. In this research we present a linear scheduling algorithm that reduces the energy consumption of the networking component of computer networks.

With the use of an energy efficient scheduling algorithm, a path for data flows is calculated that reduces the energy consumption of the network in a Software-Defined Network(SDN). SDNs are best suited for this task because, compared to traditional networks, it is possible to program the behaviour of the network from the software layer.[14] This feature makes it different to traditional networks, in which each device makes independent decisions. We set out to answer the following research question:

To what extent can linear programming scheduling algorithms improve energy efficiency of the networking component in Software-Defined data center Networks?

To validate the results we set out the following sub-question:

How does the linear programming scheduling algorithm compare to the shortest-path algorithm in terms of reliability, bandwidth and consumed time?

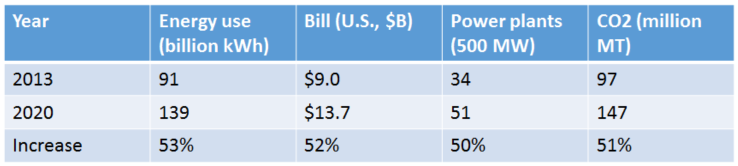
In an attempt to answer these questions we have developed the following approach which consists of three phases. In the first phase we set up a simulation environment to model a Software-Defined Network. The simulation environment runs on top of Mininet, which makes it possible to create a realistic virtual network, on a single machine. Next we set up two scheduling algorithms. The scheduling algorithms provide the controller with the routing tables. The first algorithm is the shortest-path algorithm which functions as a base line. The second algorithm is the linear programming scheduling algorithm which is designed to reduce energy consumption.

In the second phase we run extensive experiments to collect data from the simulation set up. We use a FatTree architecture network with sixteen hosts and twenty switches. For the simulation of traffic we use Iperf for sending data across the network. Different settings for the linear programming scheduling algorithm, such as minimum bandwidth, are examined and compared to the shortest-path algorithm.

In the final phase we analyse the results of the experiments and compare the linear programming scheduling with the shortest-path algorithm in terms of energy consumption, reliability, and performance.

The algorithm presented in this presentation reduces the energy consumption of the network devices in a Software-Defined Network with 30% compared to the shortest path algorithm while maintaining almost the same bandwidth speed.

The remainder of this thesis is organized as follows: in the second chapter we discuss the state of the art regarding the research subject. In the third chapter we describe the simulation environment and our approach in more detail. In chapter four we clarify the experiments to get the results which we evaluated in chapter five. And finally, in chapter six we conclude the thesis and in chapter seven discus the project and future work.



Year	Energy use (billion kWh)	Bill (U.S., $B)	Power plants (500 MW)	CO2 (million MT)
2013	91	$9.0	34	97
2020	139	$13.7	51	147
Increase	53%	52%	50%	51%

Figure 1: Energy consumption of data centers in the U.S. [1]

2 Theoretical foundation

There have been some studies on scheduling algorithms in order to reduce the power consumption of the networking component. The work in [9] reports about a network-wide power manager, ElasticTree, which dynamically adjusts the set of active network elements to respond to the changing traffic loads with the use of a mixed-integer linear program algorithm.

ElasticTree introduces a Topology-aware heuristic algorithm, which can not handle traffic spikes due to multi-minute switch boot duration. The linear programming scheduling algorithm we developed puts switches in sleeping mode when being idle instead of turning them off in order to reduce the overhead.

The authors of [20] have proposed to use a correlation aware power optimization algorithm, CARPO, that dynamically consolidates traffic flows onto a small set of links and switches in a data center network. This algorithm, that also uses linear programming, is even more effective than the ElasticTree algorithm, because it integrates correlation-aware traffic consolidations.

The CARPO algorithm uses a Mixed Integer Linear Programming (MILP) model with the objective of minimizing the energy consumption of all active switches. Therefore, every 10 minutes based on their framework, unused switches and ports will be put off. Differently we dynamically schedule incoming flow requests in order to minimize the energy consumption of all the network and maximize other quality requirements such as throughput. However, CARPO gets an initial traffic matrix as an input, while our system manages incoming flow requests in run time.

The authors of [11] focus on the smart placement of virtual machines(VM) in order to efficiently exploit the computing resources in data centers. The proposed MILP algorithm not only reduced the energy consumption it also achieved better fairness. Though they do not focus on SDNs and energy efficiency in networking devices.

In [19] the authors present a method to achieve energy efficiency in data centers by reducing the amount of traffic via assigning virtual machines to servers and to generate favorable conditions for energy-efficient routing. They reduce the number of active switches and balance traffic flows, but they do not use SDNs and this makes it very hard to apply changes to the network.

Finally the authors in [10] propose the Rate Adaptive-Aware Heuristic algorithm that combines sleeping of the switches with power scaling for OpenFlow switches. In case of high utilization the energy efficiency is remarkably improved by the algorithm.

For this project we analyse the results of a linear programming scheduling algorithm in a software defined network, because little is known about the effects of a energy efficient scheduling algorithms in SDNs.

3 Research method

3.1 Software-Defined Networks

For this project we attempt to reduce the energy consumption of the networking component. For this we can not use traditional IP networks, because they are complex and very hard to manage.[5] Traditional computer networks can be divided into three planes, the data, control and management plane. The data plane consists of the networking devices and is responsible for forwarding the packets according to the decisions made by the control plane. The control plane decides how to handle traffic and provides the data plane with the routing tables. The management plane describes the software that is used to monitor and configure the control functionality.[14] Traditional IP networks are vertically integrated, meaning that the control and data planes are bundled together. As a result it is difficult to configure the network because all switches and routers need to operate according to the same control and transport network protocols. Changing some behaviour means all the network devices need to be updated by a network administrator which takes a lot of time. This makes it difficult to reconfigure the network to respond to faults, and load changes which are common in data centers.

A way to manage with these difficulties are Software-Defined Networks(SDNs) which are networks that break with vertical integration by separating the control logic from the routers and switches to a logical centralized point. A SDN can be divided into three planes: the application, control, and data plane (see figure 2). The data plane consists of the forwarding devices which have the task to forward packets into the network according to their flow tables. In the control plane resides the network controller, also called the Network Operating System(NOS), which acts as an interface connecting the applications running op top of it with the hardware(forwarding devices) beneath it. On top of the controller is the application plane where applications can direct scheduling, security, management and other specific functions through the controller. New features can be added to the controller without the need to update the network devices, allowing the network to evolve at software speed.

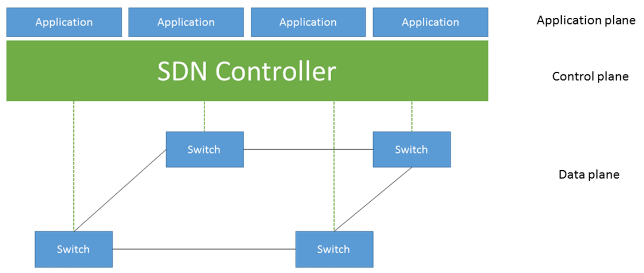


Figure 2: A Software-Defined Network

By separating the control and data plane the control logic can be moved to a logical centralized point, the controller, introducing the ability to program the network. This makes it much easier to enforce new policies and make new network (re)configurations.[13] This way it is much easier to create and introduce new abstractions in the network, simplifying network management and facilitating network evolution.

The authors of [14] define an SDN as a network architecture with four pillars.

- The control and data plane are decoupled. Control functionality is removed from switches that will become simple forwarding elements.
- Forwarding decisions are flow-based instead of destination based.
- Control logic is moved to an external entity, the so called controller or Network Operating System(NOS).
- The network is programmable through software applications that run on top of the NOS and interact with the underlying data plane devices.

The controller is connected to the network devices via an application programming interface (API), the most common API for this task is the network protocol OpenFlow.[17] This protocol makes it possible for the control plane to communicate with the OpenFlow switches in the data plane. An OpenFlow switch has one or more tables of packet-handling rules(flow tables). Each rule matches a subset of traffic and performs certain actions (dropping, forwarding, modifying) to the packets.

3.2 Simulation environment

To simulate our Software-Defined Network we use Mininet.[15] This is a system for rapidly prototyping large networks on the constrained resources of a single laptop. There are other options for simulation of networks, such as ns-2 or

Opnet, but these systems lack realism and the code developed in the simulation environment is not the same code you can use in a real network. With Mininet it is possible to simulate a real network environment which can run on a simple laptop. A user can implement new network features or architectures, and test it on large topologies with realistic traffic simulation. The developed code can then also be used in a real production network.

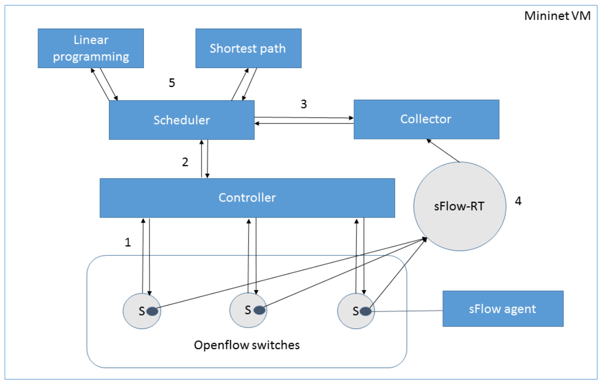


Figure 3: The software architecture running inside Mininet

A simplified version of the architecture used in this project can be found in figure 3. As said the SDN runs inside Mininet. Beneath the controller we can see the OpenFlow switches. On each switch runs an sFlow agent. The sFlow agents sends data about the switches via the sFlow-RT protocol to the collector. These statistics, that are sent in real time, consists of sFlow datagrams that are converted into actionable metrics that can be used to collect data of the energy consumption by the switches. To estimate the energy consumption an estimation model of the OpenFlow-based hardware switch (NEC PF5240) is used which has an error of less than 1%.[12]

On top of the controller runs the scheduler, which is responsible for determining the paths between hosts. The scheduler calculates the path according to the specified scheduling algorithm. The scheduler and the scheduling algorithms will be further explained in section 3.3.

The deployed network architecture for this project is the FatTree architecture, because this is the most common architecture in data centers.[18] A FatTree consists of three layers of switches: the core, aggregation and Top of Rack (ToR) layer. In a FatTree architecture, if n shows the number of ports in a switch, there will be $(n/2)^2$ core switches with n ports, n pods with n switches having n ports ($n/2$ aggregation switches + $n/2$ ToR switches).[8] For the experiments of this project we use a FatTree network architecture with $n = 4$, which can be seen in figure 4. It consists of 16 hosts and 20 switches.

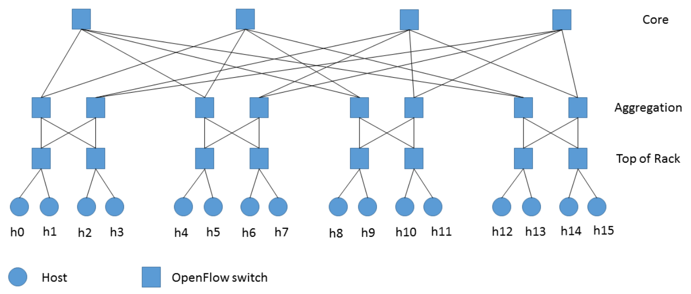


Figure 4: The FatTree used for this project.

3.3 Scheduling

The SDN controller is responsible for providing the switches with the right routing tables. Finding the right path, which is dependent of the selected algorithm, from point A to point B is called scheduling. In this project we use two different scheduling algorithms. The first algorithm is the shortest path algorithm which will function as a baseline to compare the results of the second linear programming scheduling algorithm that needs to reduce the energy consumption of the network. To optimize the energy consumption we use linear programming which is an optimization method for problems that can be expressed as a linear problem.

3.3.1 Shortest Path

The shortest path algorithm simply finds the shortest path between two nodes in the network topology. It does so such that the sum of the weights of the intermediate nodes is minimized. The links connecting the nodes are weighted by the length of the links. The shortest path algorithm used in this project is based on Dijkstra's algorithm, conceived by Edsger W. Dijkstra in 1959.[6]

When a new flow needs to be set the controller ask the scheduler where to send the packets. It sends the scheduler the topology of the network and the sending and receiving hosts with corresponding port numbers. For an example see figure 5. The shortest path algorithm starts at the sending node (node1) and finds all adjacent nodes (nodes 2 and 3). The adjacent nodes and their lengths are stored in a list (2:1, 3:10). Next the algorithm finds the adjacent nodes of the node with the lowest length value, and stores the sum of their lengths in the list (4:2). This goes on until the algorithm reaches the final node (node 6). As a results the algorithm selects the path that minimizes the sum of the lengths (1,2,4,6) and the shortest path is found. The algorithm does not care about the energy consumption of the network but only about the the shortest path.

When the algorithm is finished the scheduler sends the path with the right port numbers to use to the controller which, in turn, updates the forwarding devices in the network with the right routing tables. Data between the two hosts can now be send.

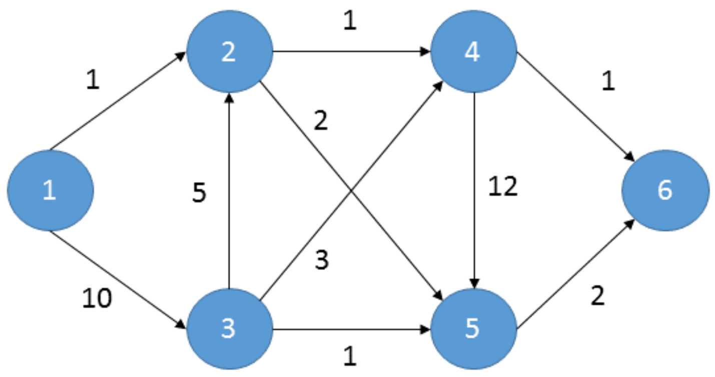


Figure 5: Small network with the lengths of the arcs.

3.3.2 Linear programming scheduling

We designed our linear programming scheduling algorithm in order to reduce the energy consumption of the networking component. For finding a path between the sending node and the receiving node we want to find the path that minimizes the amount of energy used by the network. This means the algorithm has to make a trade off between performance and energy consumption. The algorithm needs to decide if it wants to share the bandwidth with other flows such that some switches can stay in sleep mode and hereby reducing the energy consumption of the whole network. To optimize this objective we use linear programming which is a method to solve optimization problems, called linear programming problems(LP-problems).

The purpose of linear programming is to optimize an objective function (minimizing energy consumption) given a set of constraints (time, bandwidth) and decision variables (available paths and corresponding energy cost). The constraints define a polyhedron in geometric terms, that is called the candidate solution space. Because the objective function is a linear one, all local optima are automatic global optima. This implies that the optimal solution lies on only one border point of the candidate solution space. The mathematical formulation of the problem is as follows:

$$min \sum_{p \in A} e_p x_p (\text{total energy consumption of the network}) \tag{1}$$

$$\sum_{p \in A} x_p = 1 \tag{2}$$

$$b_w \geq M_b \tag{3}$$

where:

A = Set of all possible paths

M_b = Minimum bandwidth requirement

e_p = The energy consumption of the network when using path p

$$x_t = \begin{cases} 1 & \text{if path t is used} \\ 0 & \text{otherwise} \end{cases}$$

What this means is that we want one path to be selected that reduces the energy consumption of the network but satisfies the minimum bandwidth requirement. The mathematical formulation of the problem is exported to a modeller called PuLP, which is written in Python. This modeller makes it easier to formulate your problem, that is then converted by PuLP into an LP problem that can be used by the solver. This way we can focus on modelling instead of developing a solver. As solver the optimisation software Gurobi is chosen, because it is the fastest solver for linear programming problems and has the most accurate results and is free of charge for academic use.[2]

With PuLP you set some functions to transform your problem to an LP problem. We specify our objective function, constraints, and decision variables. Given a sending host and a receiving host the algorithm first finds all possible paths that connect the two hosts. In PuLp we first define our decision variables via the LpVariable function. In this function you specify the name of your variable, the low and up bound values your variables can take. And finally the category of you variable. As the variables of our problem we define all the possible paths between the sending and receiving nodes. Each possible path can take the binary value zero or one indicating if the path is selected or not. We add the LpVariables to a list in python.

```
for demand in possiblePaths:
    self.x[demand] = pulp.LpVariable.dicts('path', possiblePaths[demand], \
    lowBound = 0, upBound = 1, cat = pulp.LpInteger)
```

Secondly, we formulate our problem. Because we want to minimize the energy consumption we specify a minimization problem.

```
path_model = pulp.LpProblem("Switching Flows Model", pulp.LpMinimize)
```

Next we add our constraints to the model. A constraint is added to the problem by adding $'=='$, $'<='$, or $'>='$ to the model. For the first constraint we have to check for each path if the bandwidth is greater then the minimum bandwidth.

```
path_model += min(self.calculateBW().values()) >= minBW
```

The second constraint is that one and only one path can be selected. This is done for all the paths in the following way:

```
for demand in self.possiblePaths:
path_model += self.checkIfSelected(self.x[demand]) == 1, \
    "Only_one_selected_path_for_%s_demand"%demand
```

The checkIfSelected() is a function that returns a binary value indicating if the path is selected or not.

Thirdly we specify our linear objective function to the model.

```
path_model += self.E_Consumption()
```

The objective is to reduce the energy consumption of the network. The E_Consumption() function calculates the energy consumption of the whole network. Turning on switches that are in sleep mode has extra cost in terms of energy consumption and transition time, such that it is sometimes better to use a path that is already in use and thus the bandwidth has to be shared. Not much energy can be saved by turning of unused ports. Ports use at maximum $1 * 0.5295W$ for the highest link rate of 1Gbps. On the contrary by turning off switches that are not used energy can be saved. Because switches consume the most amount of energy when they are turned on. The added energy consumption for sending data accounts only for a small amount. [12]

Finally we call the solver to find the path that minimizes the energy consumption of the network. When the path is found the scheduler provides the controller with the path which, in turn, updates the flow tables of the network devices. In figure 6 a flowchart of the whole process can be found.

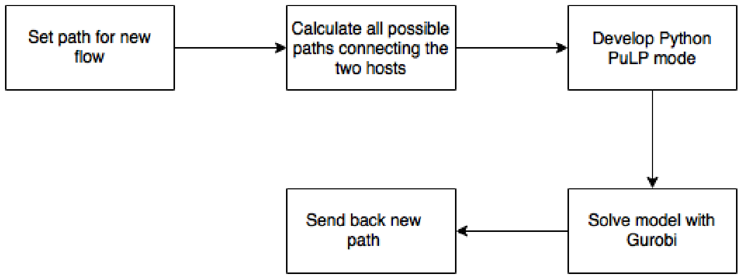


Figure 6: Flow chart linear programming scheduling algorithm.

4 Experiments

In the second phase of this research we perform the experiments. We want to compare the effect of the two described algorithms on bandwidth, time, and energy consumption. For this we use the network architecture FatTree, described above with 16 hosts and 20 switches with four ports each(see image 4).

The experiments consists of two phases. In the first phase of the simulation the scheduler needs to set the flows for the hosts we want to send data between, this is done by sending a ping messages back and forth between the specified hosts. We measure the time it takes for the algorithms to set these flows for comparison. We set flows between the following nodes:

- h0-h5
- h1-h6
- h2-h15
- h3-h8
- h4-h10
- h7-h13
- h9-h14
- h11-12

See figure 4 for more details.

After the flows are set we go to the second phase of the experiments and use the Iperf application to send data between the hosts. Iperf is a network testing tool that can send Transmission Control Protocol(TCP) data streams and measures the bandwidth speed of the data being send. One of the host acts as a server while the other host serves as a client. The hosts then send data to each other for two minutes. The data is stored to an external file for further analysis.

For this project the idea was to collect statistics of the energy consumption by the network via the sFlow-RT protocol described earlier. The sFlow agents sends data about traffic and stats of the switch to the collector which is used to measure the total energy consumption of the network. The power consumption is estimated based on the model from [12] for the NEC PF5240 OpenFlow switch.

$$P_{switch} = P_{base} + P_{config} + P_{control} \tag{4}$$

Where:
P_{base} = base power. The base power is 1183W.

P_{config} = power consumption of the configuration (i.e. number of active ports, configured line speed).

$$P_{config} = \sum_{i}^{N_{activePorts}} s_i * P_{port} \tag{5}$$

Where $N_{activePorts}$ is the number of active ports, s_i is the relative power consumption of the configured speed of the port and $P_{port} = 0.5295W/port$ is the power consumption of the port at full speed.

In SDN switches also get messages from the controller, therefore the power consumption model for SDN switches is different from traditional switches. These extra messages are processed in the following formula.

$P_{control}$ = power consumption of the control traffic

$$P_{control} = r_{PacketIn} * E_{PacketIn} + r_{FlowMod} * E_{FlowMod} \tag{6}$$

Where the rate of the PacketIn and Flowmod messages is multiplied with their respective energy consumption per packet.

Because we encountered some problems with the sFlow-RT protocol it was not possible to use this model. So we used the model from [16] of an L2 switch which is very similar to the switches we us, to still make an realistic estimation of the energy consumption.

$$Power_{switch} = Power_{chassis} + \sum_{i}^{N_{activePorts}} utilizationFactor * Power_{configs_i} \tag{7}$$

Where $Power_{chassis}$ is the energy consumption of the switch in idle state, which is 150W. $Power_{configs_i}$ = the power for a port running at line rate speed of 1Gbps and the *utilizationFactor* is the scaling factor to account for the utilization of each port. Because in the FatTree architecture the switches have 4 ports each we set this value to be 4W because the utilizationFactor is usually very low. To calculate the energy consumption of the network we multiply the number of active switches times 154W.

5 Results and evaluation

The 2 different flow sets used in the experiments are the sets in table 1. Each sending hosts sends data to a host in another pod. This way we make sure that traffic interferes with each other.

Flows	Connected hosts
6	h0-h5, h1-h6, h2-h15, h3-h8, h4-h10, h7-h13
8	h0-h5, h1-h6, h2-h15, h3-h8, h4-h10, h7-h13, h9-h14, h11-h12

Table 1: The flows set between hosts (see figure 4).

The results of the experiments are stored in table 2. We can see from this table that the linear programming scheduling algorithm uses less switches to send the data and hereby reduces the energy consumption of the network. Although it takes a longer time for the linear programming scheduling algorithm to set the paths.

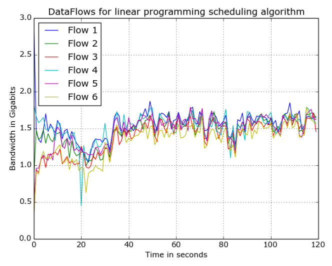
For the first flow set, with six flows, the energy consumption is 2618W for the shortest path algorithm which uses 17 switches. For the linear programming scheduling algorithm this the consumption is 1848W which uses 12 switches. This is 30% less energy consumption.

Linear programming scheduling algorithm		
Number of flows	Number of used switches	Total energy consumption
6	12	1848W
8	13	2002W
Shortest path algorithm		
Number of flows	Number of used switches	Total energy consumption
6	17	2618W
8	18	2772W

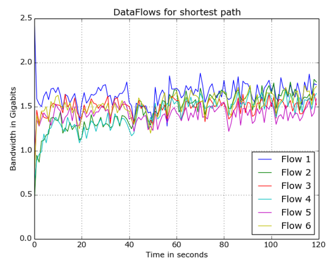
Table 2: Results of the experiments for the two tested algorithms.

In figure 7 and 8 we can see the bandwidth used by each flow set from the Iperf application. Because the linear programming scheduling algorithm is using less switches, it follows that the bandwidth has to be shared with other flows. But in the results we can see that the bandwidth for the linear programming scheduling is only a bit lower. For example in figure 7 the value of bandwidth for shortest path is more between 1.5 and 2 Gbps while for the linear programming scheduling algorithm this is between 1 and 1.5 Gbps. What this means is that that the developed algorithm for this project reduces the energy consumption with 30% while still providing an acceptable bandwidth rate. However, depending on the end users requirements the minimum bandwidth can be adjusted and results accordingly will vary.

The minimum amount of switches needed for this network architecture to function for these flow sets is 10. If more tests could be conducted it is maybe possible to reduces the energy consumption even more. But as a consequence this means more of the bandwidth has to be shared. Obviously this means that more test have to be conducted to test the network in terms of performance, reliability, throughput, and error rate.

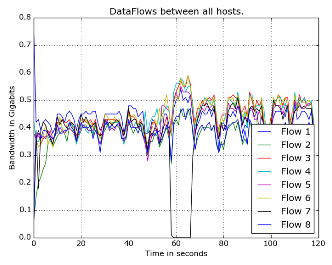


(a) Linear programming scheduling

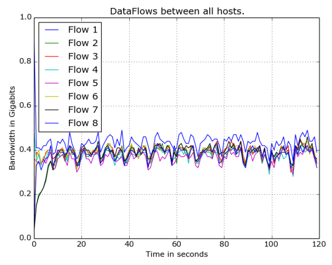


(b) Shortest path

Figure 7: Bandwidth usage with 6 flows.



(a) Linear programming scheduling



(b) Shortest path

Figure 8: Bandwidth usage with 8 flows.

6 Conclusion

In this project we tested the influence of an energy efficient scheduling algorithm. The energy reduction is only worthwhile if the performance of the network is still decent. Turning off switches that are in idle state is only desirable if the active switches can still manage to handle load. And only if sharing the links with multiple flows does not result in more errors. We set out to answer the question to what extend we can reduce the energy consumption in Software-Defined computer networks. In an attempt to answer this question we first set up a realistic simulation environment for which we use Mininet. Secondly, we implemented the linear programming scheduling and shortest path algorithm. Thirdly we ran extensive experiments to collect data from the network which we used to make an estimation of the power consumption by the networking devices. Finally, we analyzed the results from the experiments. In this project we presented a scheduling algorithm that can reduce the energy consumption with 30% compared to the shortest path while still keeping the performance requirements to an acceptable level. Although we did not have enough time to do more experiments to answer the sub-question completely, these first results are promising but more tests need to be conducted to, for example, test the reliability of the network.

7 Future work

More experiments need to be run in order to study several aspects of the system. Because of lack of time and facing implementation challenges we were not able to do so. By tweaking the trade-offs in the scheduling algorithm, such as minimum bandwidth and the cost of turning on switches, it is likely that more energy can be saved but with a cost of less bandwidth. But to what extend this can be lowered and still provide a reliable service still needs to be examined.

The algorithm opens up a lot of question. Such as, how does the algorithm performs in terms of throughput? If for example we want to send a file across the network. How much longer would it take for the file to arrive compared to the shortest path algorithm? Also an interesting question is, how would the algorithm perform on other network architectures?

It would also be interesting to see how the algorithms would work on larger scales of network architectures. For the experiments we used a 4port FatTree, but in real life data centers switches have a lot more ports. The NEC PF5240 switch used for the energy consumption model for example has 48 ports, but a switch with 200 ports is not rare.

References

[1] America's data centers consuming and wasting growing amounts of energy. http://www.nrdc.org/energy/data-center-efficiency-assessment.asp. Accessed: 06-25-2015.

[2] Gurobi 5.5 performance benchmarks. http://www.sat-ag.com/Benchmarks Accessed: 06-03-2015.

[3] Internet growth statistics - the global village online. http://www.internetworldstats.com/emarketing.htm. Accessed: 06-20-2015.

[4] U.s. environmental protection agency's data center report to congress. http://tinyurl.com/2jz3ft. Accessed: 06-03-2015.

[5] Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the complexity of network management. In *NSDI*, pages 335–348, 2009.

[6] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[7] Gerhard Fettweis and Ernesto Zimmermann. Ict energy consumption-trends and challenges. In *Proceedings of the 11th International Symposium on Wireless Personal Multimedia Communications*, volume 2, page 6, 2008.

[8] László Gyarmati and Tuan Anh Trinh. How can architecture help to reduce energy consumption in data center networking? In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 183–186. ACM, 2010.

[9] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. In *NSDI*, volume 10, pages 249–264, 2010.

[10] Thanh Nguyen Huu, Nam Pham Ngoc, Huong Truong Thu, Thuan Tran Ngoc, Duong Nguyen Minh, Hung Nguyen Tai, Thu Ngo Quynh, David Hock, Christian Schwartz, et al. Modeling and experimenting combined smart sleep and power scaling algorithms in energy-aware data center networks. *Simulation Modelling Practice and Theory*, 39:20–40, 2013.

[11] Burak Kantarci, Luca Foschini, Antonio Corradi, and Hussein T Mouftah. Inter-and-intra data center vm-placement for energy-efficient large-scale cloud systems. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 708–713. IEEE, 2012.

[12] Fabian Kaup, Sergej Melnikowitsch, and David Hausheer. Measuring and modeling the power consumption of openflow switches. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 181–186. IEEE, 2014.

[13] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, 2013.

[14] Diego Kreutz, Fernando MV Ramos, PE Verissimo, C Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *proceedings of the IEEE*, 103(1):14–76, 2015.

[15] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[16] Priya Mahadevan, Puneet Sharma, Sujata Banerjee, and Parthasarathy Ranganathan. A power benchmarking framework for network devices. In *NETWORKING 2009*, pages 795–808. Springer, 2009.

[17] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[18] Fahimeh Alizadeh Moghaddam, Patricia Lago, and Paola Grosso. Energy-efficient networking solutions in cloud-based environments: A systematic literature review. *ACM Computing Surveys (CSUR)*, 47(4):64, 2015.

[19] Lin Wang, Fa Zhang, Jordi Arjona Aroca, Athanasios V Vasilakos, Kai Zheng, Chenying Hou, Dan Li, and Zhiyong Liu. A general framework for achieving energy efficiency in data center networks. *arXiv preprint arXiv:1304.3519*, 2013.

[20] Xiaodong Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu, and Qing Cao. Carpo: Correlation-aware power optimization in data center networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 1125–1133. IEEE, 2012.