Green SDN: A Power Optimization Approach

Gokcan Cantali, *2016700027,* and Cem Ersoy

***Abstract*—The energy consumption problem is getting bigger as people keep depending on energy exploiter products and services. With the continuous development of the Internet, data center networks have become one of such energy consuming products. This project focuses on data center networks that use Software-Defined Networking (SDN), and proposes an approach to optimize the energy consumption in such environment.**

I. Introduction

As the need to access the Internet keeps increasing every day, organizations and companies are searching better ways to provide their clients services through the Web. As the number of their clients scale up, such organizations face difficulties regarding the network configuration and traffic congestion in their data centers. With the emergence of Software-Defined Networking (SDN)[1], programmable network devices along with decoupling of data plane and control plane begin to attract data center owners. Therefore, many network designers have started to replace their traditional networking architecture with an SDN architecture for such easiness [2].

Although SDN can provide more manageability and flexibility for controlling forwarding devices (e.g. routers, switches etc.), there still exists an important issue that is available in any type of network design: energy efficiency. Nowadays, energy consumption is becoming concern of everyone since more energy consumption leads to more emissions of CO_2, increasing the effect of global warming and triggering climate change. Furthermore, increasing amounts of energy consumption will eventually lead to depletion of non-renewable energy resources. Moreover, more energy consumption also results in more maintenance and operation cost, leading to an unpleasant situation for the organization. Therefore, many companies are now trying to reduce their power consumption[1] by taking measures. One of such measures is to reduce the power consumed by the operational network and its devices. The approach of minimizing energy consumption of the network as much as possible is also called Green Networking.

In this project, we[2] try to find an optimum solution to the problem of power minimization of a network in SDN environment. Since data centers use huge amounts of energy to provide services [3], they are main focus of the project. Thus, we build a model representing the network of a data center and based on the model, we implement an algorithm to find the optimal (or near optimal in some cases) design that minimizes energy consumption.

[1]Since such companies keep their services active all the time, power reduction implies energy reduction in the context. Hence, 'power' will be used interchangeably with 'energy' hereafter

[2]For the sake of formalism, 'we' is used instead of 'I' even though only one person is working on this project right now

II. Background and Related Work

Many researchers have shown interest for constructing a Green SDN environment by proposing energy-efficient frameworks which do not have much performance degradation. Based on the study in [4], energy consumption optimization approaches can be divided into main four category: traffic-aware solutions, TCAM (Ternary Content Addressable Memory) compacting solutions, rule placement solutions and end host aware solutions. Since the most popular one among these categories is the traffic-aware energy efficiency approach, we focus on the related work regarding such solutions.

One of the studies regarding power consumption minimization is conducted by Heller et al. in [5]. The scheme called ElasticTree shuts down the network devices which are not used. They propose three different scheduling algorithms. The first one is a formal model and uses Mixed Integer Programming (MIP). The second is Greedy Bin-Packing algorithm. The third is a topology-aware heuristic. Even though the MIP solution provides the best value, its complexity is not low enough to scale up to big data center networks. The heuristic solution has linear complexity and gives sub-optimal values for the objective function. Nevertheless, it fails at some peak points where traffic increases.

Another solution to the energy efficiency problem is given in the study [6]. The authors propose a correlation-aware power optimization framework, called CARPO, which adjusts the flows on the links and shuts down the unnecessary devices. They develop a heuristic based on the greedy bin-package algorithms. The approach of shutting down the unnecessary network components is similar to the one used in ElasticTree. However, CARPO also considers the correlation among the flows to save more energy. Moreover, CARPO adopts link rate adaptation to reduce the power consumption even further. As a consequence, CARPO manages to outperform ElasticTree by 19.6%.

Bolla et al. [7] proposes a different approach, compared to ElasticTree and CARPO, for optimizing energy consumption in a data center network. The idea in the study is simply as follows: First, the algorithm pre-computes the all the possible paths between each pair of nodes. During the process, the nodes in the paths are stored in the system, along with how many times they appear in a path. The algorithm then retrieve nodes from the list one by one, and decide whether they can handle the traffic requirements under consolidation. Consolidation is then applied to these selected nodes, and test results show that energy savings can reach to around 80%.

An SDN specific energy-efficient networking solution is provided by Wang et al. in the study [8]. They build an Integer Linear Programming model to formulate their objective of minimizing power consumption. Based on the model, they develop two greedy energy-efficient routing algorithms which saves energy by turning off line cards: Global Greedy Algorithm (GGA) and Alternative Greedy Algorithm (AGA). The authors also implement a CPLEX solution to compare their results. Based on the simulation results, AGA has the least execution time, while GGA has the best energy-aware solution.

The problem of optimizing power consumption has even attracted the undergraduate students. In the bachelor thesis [9], Schaap et al. build a model to represent a data center network in SDN environment. They formulate the network problem based on linear programming and implement a scheduling algorithm which uses Gurobi. They also use an Open Shortest Path First (OSPF)[10] algorithm for routing as a baseline to compare their linear programming scheduling algorithm. Based on the results, the linear programming solution saves up to 30% compared to the OSPF algorithm. However, since the linear programming complexity is not low enough to apply to real world applications, the scalability of the solution remains as a question.

III. Problem Formulation

Even though there are various different network architectures, one of the most popular among data centers is the Fat-Tree topology (refer to [11] and [12] for more details). Many organizations, especially the ones adopt SDN, use FatTree network architecture in their data centers. With the intention of running a simulation as close as to the real applications, we build our model on a SDN environment with FatTree topology. On top of our model, we implement a network design tool which takes the given workload parameters as input, calculates the minimum possible energy consumption and finds out how to achieve this objective without violating the constraints.

A. Workload Parameters

Workload parameters are the input variables given to our network design tool. The tool accepts the following workload parameters:

- k: The number of ports in a switch.[3]
- $[r_{ij}]_{NxN}$: The traffic requirement matrix that shows which host will be sending how many packet per second to another host.
- C_l: The bandwidth capacity of the links. For simplicity, we assume that every link has the same capacity.

B. Objective Function

Objective function is what we are trying to achieve in our model. In order to achieve a "greener" state regarding the network devices, our design tool aims to reduce the power consumption as much as possible:

$$min\ P_{total}$$

In our model, P_{total} is the sum of the consumed power by the switches:

[3]The number of switches in a FatTree architecture depends on the number of ports in a switch. Therefore, there does not exist another variable that represents the number of switches. The same applies to the number of hosts.

$$P_{total} = \sum_{i=1}^{N_{activeswitches}} P_i^{switch} \tag{1}$$

Where $N_{activeswitches}$ is the number of active switches in the network and P_i^{switch} is the power consumed by only the switch i.

In on our model, the power consumption of a switch arises from its base power and the power consumed by its ports. The base power is the power used by a switch when it is online, even when it does not relay any packets. The port power depends on the rate of the flow on the port. Therefore, we can estimate the power consumption of a single switch as follows:

$$P^{switch} = P_{base} + \sum_{i=1}^{N_{ports}} P_i^{port} \tag{2}$$

Where N_ports is the number of ports in a switch, P_{base} is the power consumed by a switch when it is online and P_i^{port} is the power consumption by the port i in the switch. We calculate P_i^{port} by

$$P_i^{port} = P_{full}^{port} * \rho_i^{port} \tag{3}$$

Where P_{full}^{port} is the power consumed by a port when it uses all the bandwidth capacity of the link the port is tied to. ρ^{port} is the utilization of the port, i.e. the amount of bandwidth it is using compared to the capacity of the link.

Combining the equations (1), (2) and (3), we arrive at our objective function:

$$min \sum_{i=1}^{N_{activeswitches}} (P_{base} + \sum_{j=1}^{N_{ports}} P_{full}^{port} * \rho_i^{port}) \tag{4}$$

C. Decision Variables

Decision variables are the variables each of which will be given a value by our network design tool in order to achieve the optimum value of the objective functions. In other words, the value of decision variables are decided by the optimizer tool such that it can find a more suitable value for the objective function.

The decision variables in our network design tool are the followings:

- $C_{i,j}$: Whether or not the core switch in jth place on the ith group is active. (Binary Variable)

$$C_{i,j} = \begin{cases} 1 & \text{If the } jth \text{ core switch in } ith \text{ group is on} \\ 0 & \text{Otherwise} \end{cases}$$

- $A_{i,j}$: Whether or not the aggregate switch in jth place on the ith pod is active. (Binary Variable)

$$A_{i,j} = \begin{cases} 1 & \text{If the } jth \text{ aggregate switch in } ith \text{ pod is on} \\ 0 & \text{Otherwise} \end{cases}$$

- $E_{i,j}$: Whether or not the edge switch in jth place on the ith pod is active. (Binary Variable)

$$E_{i,j} = \begin{cases} 1 & \text{If the } jth \text{ edge switch in } ith \text{ pod is on} \\ 0 & \text{Otherwise} \end{cases}$$

- $f_{m,n}$: The flow from port m of a switch (or from host m) to port n of another switch (or to host n), based on number of packets per second. (Integer Variable)

D. Constraints

Constraints are the bounds for our decision variables. They are the set of equations and inequalities that limits the solution space of the problem. Our network design tool will try to find a minimum power consumption value, without going out of the borders set by the constraints.

The constraints in our model are the followings:

1) *Link Capacity Constraint*: The flow on a link from m to n, i.e. flow between two switches or between a host and a switch, cannot be more than the capacity of the link:

$$f_{m,n} \leq C_l \quad \forall m, n \in HS \cup PS \tag{5}$$

Where HS is the set of all hosts and PS is the set of all ports (of all switches) in the network.

2) *Flow Constraint*: The flow on the links must satisfy the network requirement matrix. To put it more simpler, the amount of flow going into a switch should be the same amount of the flow going out of the same switch. Before focusing on the constraint, we first define the following notation:

$$L = HS \cup PS \tag{6}$$

Where HS and PS have the same definition as the previous constraint. Therefore, L is considered the set of all the endpoints of the links in the system. Now, we can write a more formal expression of the constraint as the following:

$$\sum_{n \in PS_i} \sum_{x \in L} f_{x,n} = \sum_{m \in PS_i} \sum_{y \in L} f_{m,y} \quad \forall i \in SS \tag{7}$$

Where PS_i is the set of all ports belonging to the switch i and SS is the set of all switches in the network.

E. Given Constants

In the project, we assume that certain constants are given and they cannot be changes. The list of such constants is as follows:

- $Topology$: Our model dictates that the topology of the network must be FatTree. Therefore, we say that the topology is a given constant.
- P_{base}: We assume that the base power of an active switch equals to 1183W, as stated in [9].
- P_{full}^{port}: The power of a fully utilized port is approximated to 100W in the project.

IV. Solution Approach

As we planned, we successfully implemented a Simulated Annealing (SA) algorithm to find a near-optimal solution to our energy minimization problem. Since SA is a heuristic method, our solution does not always give the optimum value. However, it provides promising results considering that we have achieved around 40% energy saving compared to fully functional network, in our example case.

Our solution method makes decisions regarding two variables: i) which switches should be turned on/off and ii) which path a flow should follow. Therefore, our SA approach decides either to change a network element's state, or to change the path of a flow. Such decision is made based on a probability which we feed into the system as a parameter. Moreover, after deactivating a switch, the algorithm checks whether a feasible solution exists with current configuration. In other words, our method makes sure that even after deactivating a network element, there is at least a flow path from each source to destination. When there are not enough flow paths to satisfy the traffic requirement matrix, our algorithm randomly activates switches one by one until a feasible solution is found.

V. Implemented Algorithm

We have implemented our SA approach, using native Python, based on numerous parameters, neighborhood structure and stochastic decisions[4]. Although the latter one is mostly a natural result of SA methods, the fact that we have two different ways to move into a neighbour state forces us to depend on probability even more. We now present list of the parameters, explain how we constructed the neighborhood structure and give an overview of the way our algorithm works.

A. Parameters

The parameters used in our system are as follows:

- $T_0 => InitialTemperature$: The initial value of the temperature (T) in our algorithm. The temperature will slowly decrease throughout the simulation, starting from T_0.
- $\alpha => CoolingRate$: The ratio that describes how fast the temperature decreases during the simulation time. In each iteration, T decreases by the following formula:

$$T = T * (1 - \alpha)$$

- $T_{final} => FinalTemperature$: The final temperature that can be achieved without the termination of the simulation. Whenever temperature T reaches T_{final}, the simulation will be terminated. Hence, the following inequality is the stopping condition:

$$T <= T_{final}$$

- $\beta => SwitchTogglingProbability$: Though this parameter is an important part of our system, it does not

[4]You can find our code in the following link: https://github.com/gokcantali/GreenSDN

come from the SA approach itself. Since we have two roads between neighbour states, one is turning on/off switches and the other is assigning different paths to flows, we use β to describe the probability of choosing to toggle switches. If β is high, the algorithm will mostly try to deactivate switches to reduce the power consumption. On the other hand, if β is low, instead of messing with network elements, the algorithm will generally try to decrease power consumption by re-assigning flow paths.

B. Neighborhood Structure

Neighborhood structure is a critical component in SA. In fact, one of the key challenge and affecting factor of such approach is to decide which states are neighbour. In our methodology, two states are considered neighbour if one of the following two conditions is satisfied:

1) The number of active switches in the two states differ only by one. Naturally, this also implies that the difference of number of inactive switches between the two states is one.
2) The two states have the same number of active switches, but one of the flows have different paths for each of the states.

During the simulation part of our algorithm, we move from one neighborhood state to another, trying to find an optimum (or near-optimum) solution.

C. Overview of Algorithm

We now explain how our algorithm works, without going into much details. An overview of our method can be seen as a flowchart in Fig 1. In the figure, T_final means T_{final} which is one of our mentioned parameters. In addition, R and R' are instances of uniform random variates that are used to make decisions regarding whether to accept a worse solution and how to move into a neighbour state, respectively. The working mechanism of our solution is simply as follows:

At first, the inputs are given to our network design tool. Our inputs, which were explained expressively in the first progress report, are the followings: number of ports in a switch, number of hosts, link capacities and traffic requirement matrix. Then, our tool starts the SA algorithm by generating a random initial feasible solution to the problem and initializing the temperature. Then, the initial configuration and power consumption of such configuration is calculated and stored as the best solution so far. After that, a random number, R, between 0 and 1 is generated and compared with switch toggling probability, β. If R is greater than β, the algorithm only assigns random paths to flows, not changing the states of switches. On the other hand, if R is smaller than β, the algorithm chooses a switch randomly and toggles[5] it. Then, our algorithm makes sure that there is a feasible solution by checking if flows can find a path from source to destination. If there is not a feasible solution, our algorithm randomly chooses inactive switches and activates them one by one until a solution exists with the current configuration.

[5]In the context of this paper, toggling means turning on an inactive switch or turning off an active switch

The rest of the algorithm follows the classical SA logic. The temperature gets updated based on the cooling rate and the power consumption of the current system is evaluated. Then, the power consumption of the current solution is compared to the best solution so far. If the current solution provides a better solutions, the best solution configuration is replaced with the new one. Otherwise, the algorithm generates a random number, R', between 0 and 1. Then, R' is compared with the acceptance probability, which is:

$$e^{(E_{best}-E_{current})/T}$$

where E_{best} is the power consumption of the best solution so far, $E_{current}$ is the power consumption of the current solution, and T is the current temperature. If R' is smaller than the acceptance probability, the solution is accepted as the new best solution even though it provides worse result than the current best solution. However, if R' is larger, then the current solution is not chosen as the best solution. After that, the algorithm checks whether or not the stopping condition is satisfied. In other words, the algorithm compares T with T_{final} and if T is smaller, the program terminates. Nevertheless, if T is greater than T_{final}, the next iteration starts and a new solution is generated.

VI. Demonstrative Cases

As we have already mentioned in the previous report, the FatTree architecture ([11], [12]) is used in our demo cases. The port number of each switch, k, is chosen as 4. Therefore, each core switch is connected to 4 aggregation switch. The host number, N, is chosen as 16, which is the maximum value for the FatTree configuration with $k = 4$. The topology which we used can be seen in Fig 2.

We enumerated each of the switches using the concatenation of their types and their order from left to right, based on 0-indexing. For example, the leftmost edge switch has the identity "Edge#0" while the rightmost core switch has the identity "Core#3". Hosts are enumerated in a similar way, such as the leftmost host is called "Host#0". Moreover, the links capacities were set equally at 10 Mbps and random flows between hosts are generated between 1000-2000 packets per second.

Flow #	Source	Destination	Flow Rate
1	Host#13	Host#7	1358
2	Host#14	Host#15	1472

TABLE I: Inputs of Demo Case 1

In the first demo case, two flows are generated randomly from Host#13 to Host#7 and Host#14 to Host#15 as can be seen in Table I. Similarly, three flows are created in the second example case, as presented in Table II. Our algorithm assigned paths to these flows and turned off unnecessary switches. The results of the two demo cases can be seen in Table III and Table IV, respectively.

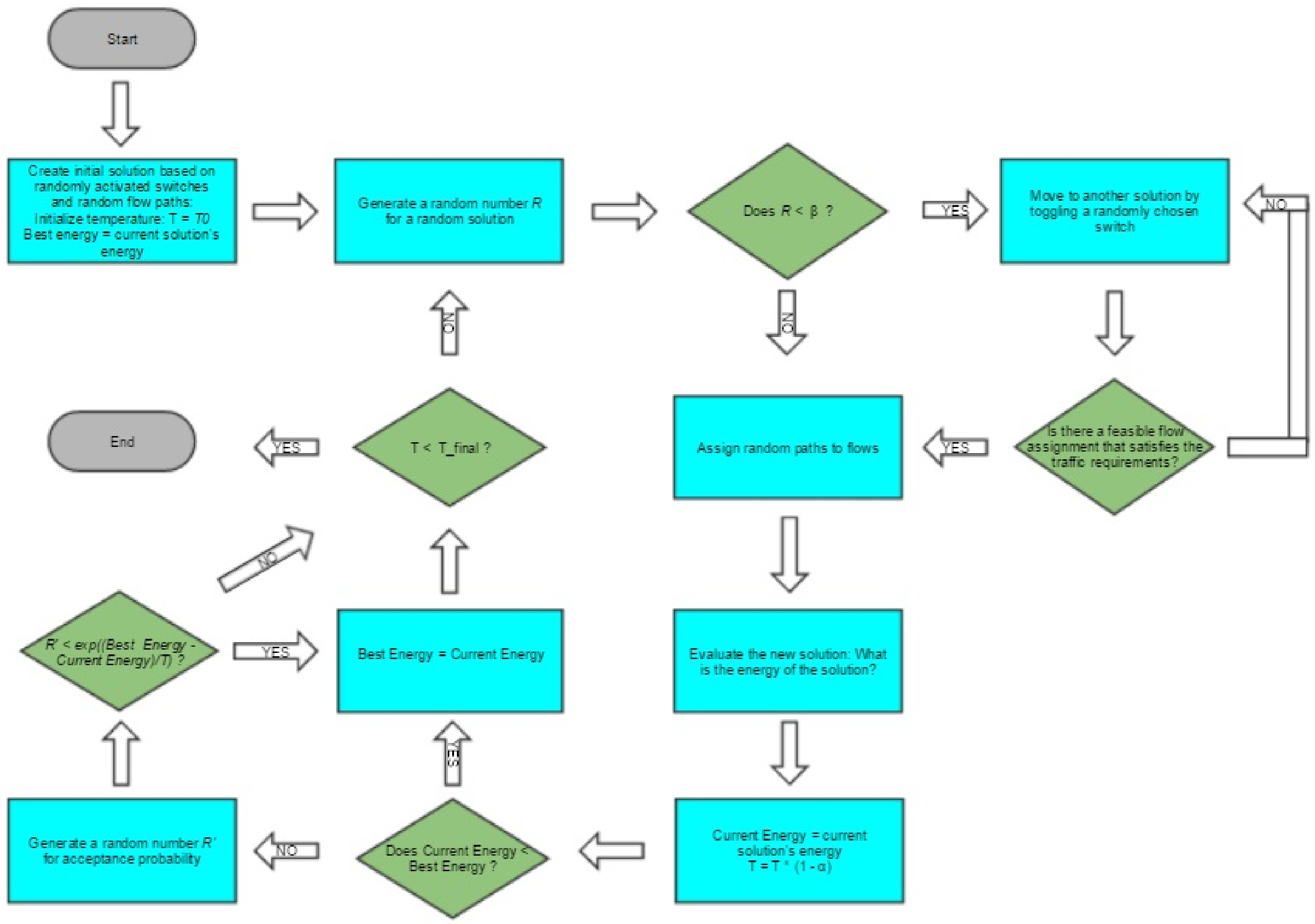

Start
Create initial solution based on randomly activated switches and random flow paths: Initialize temperature: T = T0 Best energy = current solution's energy
Generate a random number R for a random solution
Does R < β ?
YES
NO
Move to another solution by toggling a randomly chosen switch
NO
Is there a feasible flow assignment that satisfies the traffic requirements?
YES
Assign random paths to flows
Evaluate the new solution: What is the energy of the solution?
Current Energy = current solution's energy T = T * (1 - α)
Does Current Energy < Best Energy ?
YES
NO
Generate a random number R' for acceptance probability
R' < exp((Best Energy - Current Energy)/T) ?
YES
NO
Best Energy = Current Energy
T < T_final ?
YES
NO
End

Fig. 1: The Flowchart of Our Algorithm

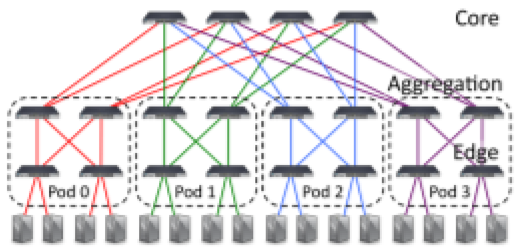


Fig. 2: The Topology used in our Demo Case: Figure from [5]

Flow #	Source	Destination	Flow Rate
1	Host#12	Host#7	1109
2	Host#12	Host#13	1840
3	Host#13	Host#6	1353

TABLE II: Inputs of Demo Case 2

Flow Path 1	H#13-E#6-A#7-C#2-A#3-E#3-H#7
Flow Path 2	H#14-E#7-H#15
# of Open Switches	9
Total Power Consumption	10,805.63 W

TABLE III: Results of Demo Case 1

VII. Evaluating the Quality of Our Solution

The problem we focus on is to reduce the power consumption of a data center network as much as possible. Hence, in order to evaluate how good our approach works, we initially look at how much energy our algorithm saves compared to the fully functional network.

For example, in our demo cases, there were a total of 20 switches in the topology. We accept that the base power of a switch equals to 1183W and the full base power of a port is

Flow Path 1	H#12-E#6-A#7-C#2-A#3-E#3-H#7
Flow Path 2	H#12-E#6-H#13
Flow Path 3	H#13-E#6-A#7-C#2-A#3-E#3-H#6
# of Open Switches	10
Total Power Consumption	12,048.96 W

TABLE IV: Results of Demo Case 2

100W, as stated in Section III-E. Under such conditions, the total power consumption of a fully functional network would be at least 23,360W (1183W * 20), without considering the port utilization. Therefore, the difference in consumed power between our demo cases and the fully functional one is at least 12,554.37 and 11,311.04, respectively. Dividing these values by the total power consumption of the fully functional network, we find that the power saving ratio is approximately 53% for the first case, and 48% for the second case.

Even though comparing our results to fully functional networks gives us a rough idea about how good the proposed solution performs, we need a more formal approach to evaluate our algorithm. Therefore, for the experiment part, we adopt two different methods to evaluate the quality of our solution.

- **Comparison with Open Shortest Path First (OPSF) algorithm**. We run both our algorithm and OPSF algorithm on each test case to see how much the results of our solution differ from the results of OSPF.
- **Statistical Quality Measurement**. We generate a large number of random solutions for each test case, plot the histogram of their results and try to fit the histogram into a normal distribution. Then, we extract mean and standard deviation of the distribution to examine how far our solutions are from the mean of random solutions.

VIII. Experiments

We build our own simulation environment for the experiments of our energy efficient approach. We use Python to develop our experiments, the same language we use for our algorithm itself. As the experiment design, we adopt the *Two Phase Experiment* method for our tests. As the name suggests, this method has two different phases which are as follows:

1) *Phase I*: In this phase, there are more parameters to be tested, but only few levels for each parameter.
2) *Phase II*: In this phase, only few number of important parameters are tested, but with more levels compared to *Phase I*.

We now provide the setup of both Phase I and Phase II of our experiments separately. Moreover, we give the results of each phase and discuss the performance of our tool in detail.

A. Phase I

In this phase, we test seven parameters (three workload parameters + four system parameters) at two levels each. The list of the tested parameters and their tested levels can be seen in Table V.

Furthermore, in order to obtain variance control, we repeat each test case for ten times. Therefore, we have:

Parameters	**Levels**
k	4 & 6
μ	0.05 & 0.20
C_l	100 Mbps & 1 Gbps
α	0.01 & 0.05
T_0	10000 & 5000
T_{final}	1000 & 100
β	0.75 & 0.90

TABLE V: Phase I parameter setup

$$2^n * r = 2^7 * 10 = 1280 \tag{8}$$

total number of executions in *Phase I*. We now briefly explain each of the parameters in the setup.

- k: The number of ports in a switch. We assume switches are identical in terms of port number and processing capacity. k also defines the number of hosts since we use FatTree architecture. Thus, the number of hosts becomes $k^3/4$ when the number of ports in a switch is k.
- μ: The communication density. Instead of providing hard-coded traffic requirement matrix into our tool, we feed a probability (μ) which we call communication density. μ defines the number of flows between host pairs. For instance, when μ is 0.05, approximately 5% of the host pairs communicate. Therefore, if the number of hosts is 16 (which implies 120 host pair) in such case, there would be approximately 6 flows in the system.
- C_l: The link capacities of switches. For the sake of simplicity, we assume all the links are homogeneous in terms of capacity. In other words, they all have the same capacity.
- α: The cooling rate of the algorithm. It determines how fast the simulation converges.[6]
- T_0: The initial temperature of the Simulated Annealing algorithm.[6]
- T_{final}: The final temperature of the Simulated Annealing algorithm. The simulation stops when the current temperature becomes less than or equal to final temperature.[6]
- β: The probability of toggling a switch at each step of the algorithm. Refer to Section V-A for more details.[6]

All of the mentioned parameters have different effects on the outcome of the simulation instances. For instance, while k gets higher, both the simulation time and the optimum power consumption calculated by our tool increases. Naturally, communication density, μ, has a similar effect since more flows usually cause a need for more power consumption. On the other hand, slower cooling rate, i.e. small values of α, causes the simulation time to increase and the calculated optimum power consumption value to decrease. Such outcome is completely expected considering the nature of Simulated Annealing. Because, slower convergence usually results in better solutions.

[6] Since these system parameters have been explained in depth before, we do not give much details here. Refer to Section V-A for more detail.

Parameters		Our solution		OSPF		Statistical Quality Measurement	
K=4 (N=16)	Uniform & sparse r_{ij} ($\mu = 0.05$)	Max	16709.2050	Max	16709.2050	Mean	23855.1266
		Avg	16088.1890	Avg	16088.1890	Standard Deviation	1071.63956
		Min	13516.8000	Min	13516.8000		
	Uniform & dense r_{ij} ($\mu = 0.20$)	Max	17986.2550	Max	17986.2550	Mean	26575.3548
		Avg	17800.2375	Avg	17800.2375	Standard Deviation	1398.94453
		Min	17572.5300	Min	17572.5300		
K=6 (N=54)	Uniform & sparse r_{ij} ($\mu = 0.05$)	Max	35590.6400	Max	35929.9450	Mean	53791.9853
		Avg	34653.8370	Avg	35718.5370	Standard Deviation	2071.68987
		Min	34236.7450	Min	35419.7450		
	Uniform & dense r_{ij} ($\mu = 0.20$)	Max	36026.6150	Max	37209.6150	Mean	55239.9109
		Avg	35712.1240	Avg	36895.1240	Standard Deviation	2702.93754
		Min	35479.3100	Min	36662.3100		

TABLE VI: The results of the Phase I experiments when C_l = 100 Mbps

Parameters		Our solution		OSPF		Statistical Quality Measurement	
K=4 (N=16)	Uniform & sparse r_{ij} ($\mu = 0.05$)	Max	15396.2056	Max	15396.2056	Mean	22422.8743
		Avg	14917.0499	Avg	14917.0499	Standard Deviation	1344.71398
		Min	14202.8375	Min	14202.8375		
	Uniform & dense r_{ij} ($\mu = 0.20$)	Max	15430.3976	Max	15430.3976	Mean	22809.6553
		Avg	15421.5266	Avg	15421.5266	Standard Deviation	1108.34388
		Min	15411.1318	Min	15411.1318		
K=6 (N=54)	Uniform & sparse r_{ij} ($\mu = 0.05$)	Max	29732.9015	Max	30915.9015	Mean	52001.4047
		Avg	29705.6982	Avg	30888.6982	Standard Deviation	1715.73225
		Min	29683.2677	Min	30866.2677		
	Uniform & dense r_{ij} ($\mu = 0.20$)	Max	30159.4239	Max	31342.4239	Mean	52253.6880
		Avg	30104.4104	Avg	31287.4104	Standard Deviation	1726.90373
		Min	30063.1541	Min	31246.1541		

TABLE VII: The results of the Phase I experiments when C_l = 1 Gbps

We provide the results of the *Phase I* experiments in two different tables. In the Table VI, the results have been obtained when all the links in the system have 100 Mbps bandwidth. Similarly, the simulation results when the capacity of the links is 1 Gbps are displayed in Table VII.

The results of *Phase I* of the experiments give hope about our solution. As it can be seen from Table VI, the results provided by our tool are exactly the same as the outcome of the OSPF algorithm in the $k = 4$, $C_l = 100Mbps$ case. Moreover, our approach seems to work really well based on the statistical quality measurement. Even the maximum energy reported by our tool in $k = 4$ case is around 5 σ away from the mean of the random solutions. Furthermore, for the $k = 6$ case in *Phase I*, our algorithm performs even better. In fact, our Simulated Annealing method works approximately 3% better than OSPF in this case. In addition, once again our approach is much efficient based on the statistical quality measurement. The average of results provided by our tool is around 7-8 σ away from the random solution set. The communication density (μ) does not affect the gap between OSPF and our solution. However, it affects the overall energy consumption results such that higher values of μ generally results in more energy consumption.

The case when $C_l = 1Gbps$ is not much different than the 100 Mbps case. The main difference is that overall power consumption values are lower in 1 Gbps configuration compared to the 100 Mbps configuration. The reason is that since link capacities are higher, the utilization rate of links are generally lower, which cause less energy consumption. As it can be seen from Table VII, the OSPF approach and our method do not differ at all for $k = 4$ configuration. Also, our algorithm provides results that are 5 σ lower than than the mean of random solutions. Similarly, for the $k = 6$ case, our proposed solution have results that are approximately 4% better than OSPF. Furthermore, the calculated power consumption results of our network design tool are almost 10 σ away from the mean of random solution set. In overall, *Phase I* results indicate that our solution has potential.

At the end of *Phase I*, we decide that certain parameters are more important regarding the results calculated by our method. Such parameters are: k, μ and α. We test these parameters with more levels, while fixing the other parameters of the system.

B. Phase II

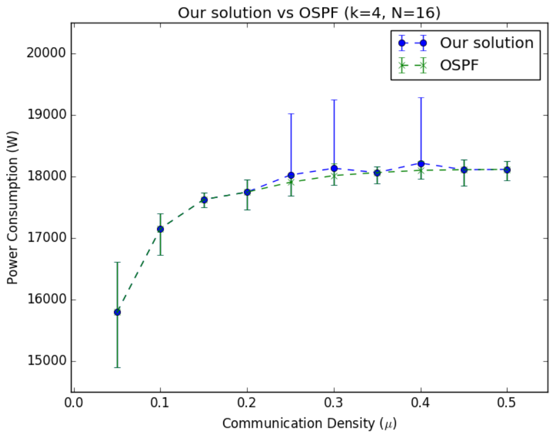
In this phase, we test three parameters (three workload parameters + four system parameters) at four, ten, ten levels respectively. The list of the tested parameters and their tested levels can be seen in Table VIII.

Parameters	Levels
k	[4 & 6 & 8 & 10]
μ	[0.05 & 0.10 & 0.15 & 0.20 & 0.25 & 0.30 & 0.35 & 0.40 & 0.45 & 0.50]
α	[0.05 & 0.15 & 0.25 & 0.35 & 0.45 & 0.55 & 0.65 & 0.75 & 0.85 & 0.95]

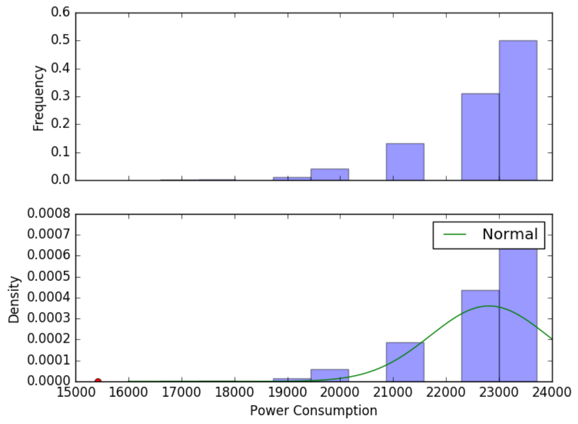
TABLE VIII: Phase II parameter setup

Furthermore, in order to obtain variance control, we repeat each test case for ten times. Therefore, we have:

$$4 * 10 * 10 * 10 = 4000 \tag{9}$$

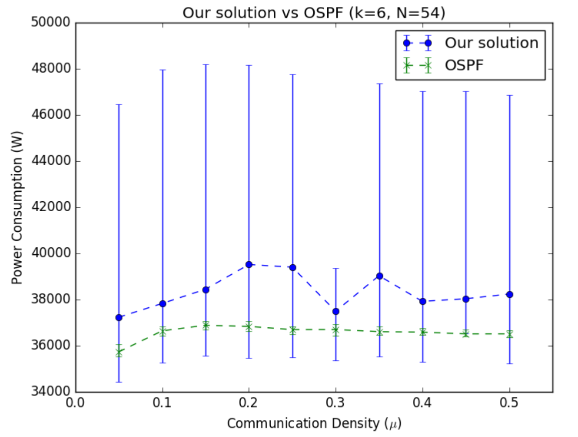


(a) Our approach vs OSPF

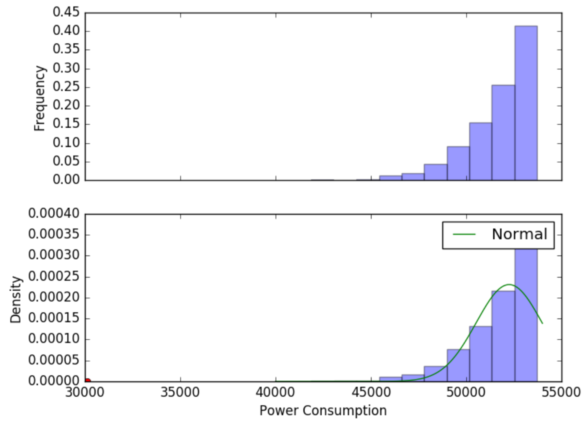


(b) Our approach in Statistical Quality Measurement

Fig. 3: The results of our tool compared to results of OSPF in (a), and compared to the results of random solutions in (b). Here, $k = 4$ and $\alpha = 0.05$. Also, μ is fixed to 0.05 in (b)



(a) Our approach vs OSPF



(b) Our approach in Statistical Quality Measurement

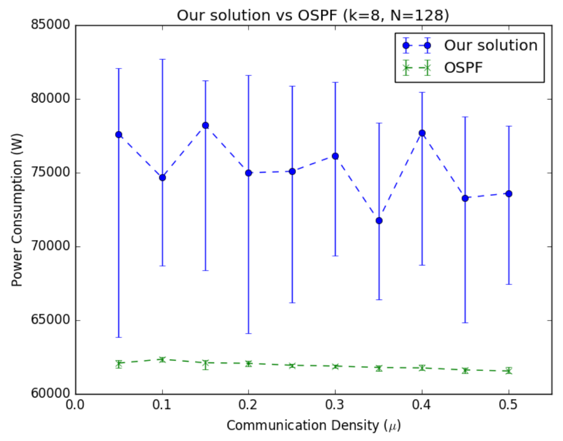
Fig. 4: The results of our tool compared to results of OSPF in (a), and compared to the results of random solutions in (b). Here, $k = 6$ and $\alpha = 0.05$. Also, μ is fixed to 0.05 in (b)

total number of executions in *Phase II*. Since we have already discussed the effects of the variable, we don't mention them again. In this phase, we experiment on the values of communication density, number of switches and cooling rate. The other parameters are fixed as follows: T_0 = 10000, T_{final} = 100, β = 0.90 and C_l = 1 Gbps.

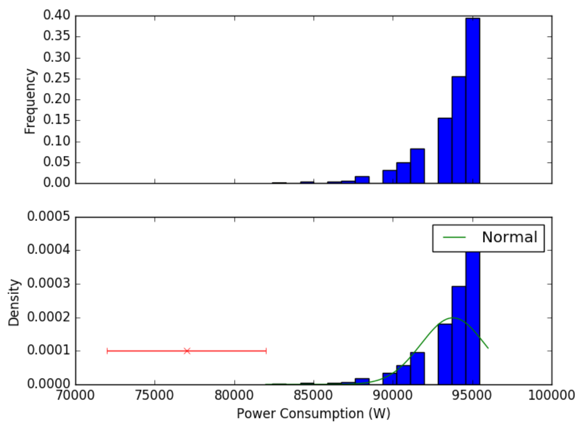
Similar to the results in *Phase I*, our algorithm competes head to head with OSPF in $k = 4$ case. As can be seen on Figure 3a, the power consumption results of our approach and OSPF are the same except for a few different levels of μ. Moreover, statistical quality measurement in Figure 3b shows that the result of our algorithm is approximately 5 σ away from the average result calculated by random solutions.

In the Figure 3b, the upper part shows what percent of the random solutions provide power consumption result in each range. For example, according to the figure, 50% of the solutions give a result between 23000W and 24000W while 30% of the solutions give a result between 22000W and 23000W. The lower part of the Figure 3b shows the random solution results as a probability distribution. Therefore, the areas under the bars are summed up to 1. The solutions are fitted into a normal distribution, whose graph is represented by the green curve. The red dot represents the average value of our solutions for this case.

Unfortunately, the case where there are six switches ($k = 6$) does not give as optimistic results as in *Phase I*. In fact, in each of the 10 levels of μ, our average result is worse than the solution provided by OSPF. Based on the Figure 4a, our algorithm works 10% worse than OSPF in overall. Moreover, our solutions have significant variance such that the difference between the maximum and minimum of the found solutions is 12000W at some cases (μ=0.1 for example). Nevertheless, our method is still in a good shape based on the statistical quality measurement. Figure 4b indicates that the mean of the random

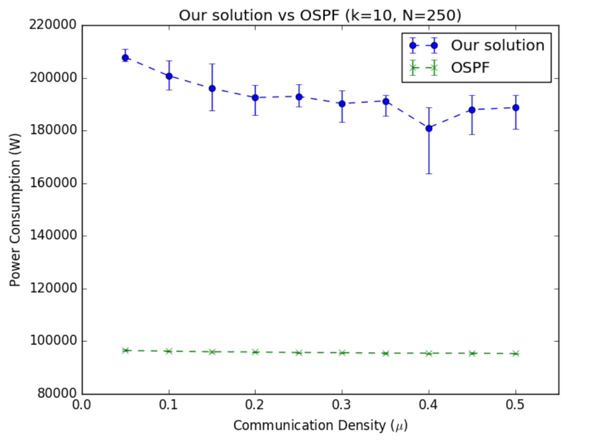


(a) Our approach vs OSPF

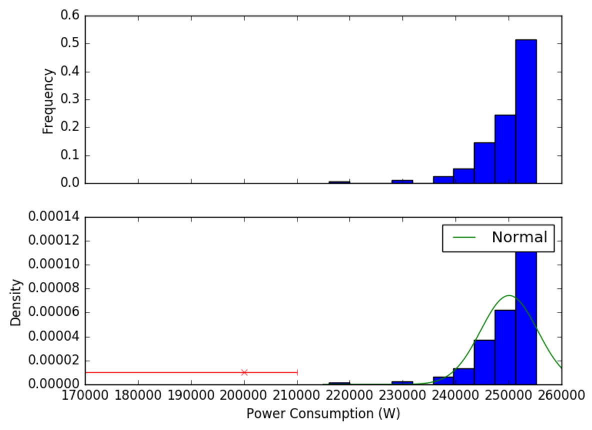


(b) Our approach in Statistical Quality Measurement

Fig. 5: The results of our tool compared to results of OSPF in (a), and compared to the results of random solutions in (b). Here, $k = 8$ and $\alpha = 0.05$. Also, μ is fixed to 0.05 in (b)



(a) Our approach vs OSPF



(b) Our approach in Statistical Quality Measurement

Fig. 6: The results of our tool compared to results of OSPF in (a), and compared to the results of random solutions in (b). Here, $k = 10$ and $\alpha = 0.55$. Also, μ is fixed to 0.20 in (b)

solutions is around 6-7 σ away from our average solutions. Therefore, we still consider our approach acceptable.

Unexpectedly, the gap between our method and OSPF gets even bigger when $k = 8$. In such case, even our minimum solutions could not reach the maximum results of OSPF, as can be seen in Figure 5a. In overall, our algorithm works 20% worse than OSPF and due to high variance nature of our approach, the gap between each of our solutions gets bigger. However, comparing our results to randomly generated solution set yields less pessimistic outcomes. According to Figure 5b, the average of our solutions is approximately 4 σ away from the mean of the random solutions. Even the maximum of our results is almost 3 σ away from the random solution mean. Hence, we say that our approach is much better than just generating random solutions in $k = 8$ case as well.

Disappointingly, but not surprisingly anymore, the results get even worse for the $k = 10$ case. Our algorithm works literally twice as bad as the OPSF approch, as can been seen in Figure 6a. The most possible reason for such amount of gap in this case, is the fact that we used 0.55 as the value of α instead of 0.05. We have used a greater value for α for this case because the simulation with low values of α was taking more than we could accept. Therefore, we had to sacrifice some of the accuracy in favor of faster convergence. At this point, we admit defeat in front of OSPF since our method could not even come close the performance of OSPF for large networks. Nevertheless, the statistical results in Figure 6b reveals that our approach is still better than random solutions. From the figure, we see that the average result provided by our tool is almost 6 σ away from the mean result of random solutions. Moreover, even the maximum value of our solutions is approximately 4 σ away from the random solution mean.

To sum up, even though our method could not match the efficiency of OPSF approach in large networks, it could at least compete head to head with OSPF for small networks. In addition, we show that our algorithm is definitely preferable to using randomly generated solutions. One of the most important drawbacks regarding our approach is the high variance. The high variance is most likely a consequence of using a final temperature for the stopping condition in our simulation. Because, generally in Simulated Annealing methods, the iterations stop when there is no progress for a certain number of steps (For example ϵ iterations). Therefore, switching to such approach might have resulted in better outcomes. Moreover, instead of randomly assigning flows, we could have adopted a specialized routing method.

IX. Conclusion and Future Work

In this work, we have proposed an approach for power efficiency in SDN environment. We have built a model representing the network, where the power consumption value if mainly decided by the number of active switches and link utilizations. We have developed a Simulated Annealing approach to find optimum (or near-optimum) configuration that consumes minimum amount of energy while preserving the flow requirements.

Based on the experiment results, we see that our algorithm performs well compared to random solutions. Moreover, the results of our method are almost head-to-head with OSPF in small networks where there are around 20-50 hosts and 20-40 switches. However, when the network size increases, OSPF algorithm performs significantly better than ours. Furthermore, the variance of our results gets higher as the number of nodes and switches increases. Hence, we do not recommend to use our approach in big network with more than 100 nodes. Nevertheless, it can be a candidate solution in small networks.

One important aspect of this work is the ability to be extendability. Even though we specifically focus on SDN environment and FatTree architecture, our work can easily be extended for the use of non-SDN networks, since the logic behind our approach would stay the same.

We have left several items as future work. For instance, we have assumed links are homogeneous in the networks. That is, all links are assumed to have the same bandwidth capacity. Extending our work for networks with heterogeneous links can be implemented in the future.

The main drawback of our algorithm is its proactive nature. For simplicity, we have assumed that the flow rates do not change throughout the simulation. Such assumption helped us to compute the flow paths at the beginning and let the simulation run without changing these paths. Nevertheless, static flow assumption is not realistic as most data center networks have dynamic traffic requirement. Therefore, handling fluctuating flows would be a seriously important improvement and highly encouraged.

Another trivial future work is to extend our algorithm into different types of topologies. Since many data centers use FatTree architecture, we have adopted it, too, and tested our approach only in such topology. Nevertheless, it would be interesting to see how our algorithm performs in other topologies. We leave this improvement as another future work.

References

[1] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *CoRR*, vol. abs/1406.0440, 2014. [Online]. Available: http://arxiv.org/abs/1406.0440

[2] “Data Center SDN growing 65% this year,” http://www.networkworld.com/article/2833619/cisco-subnet/data-center-sdn-growing-65-this-year.html, accessed: 2016-12-15.

[3] “US Data Center Energy Consumption In the Past and Prediction for the Future,” https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy, accessed: 2016-10-18.

[4] B. G. Assefa and O. Ozkasap, “State-of-the-art energy efficiency approaches in software defined networking,” in *ICN 2015 : The Fourteenth International Conference on Networks*, ser. ACM-ICN ’15. New York, NY, USA: ACM, 2015, pp. 256–260.

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, “Elastictree: Saving energy in data center networks,” in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 17–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855728

[6] X. Wang, X. Wang, K. Zheng, Y. Yao, and Q. Cao, “Correlation-aware traffic consolidation for power optimization of data center networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 992–1006, 2016. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2015.2421492

[7] R. Bolla, R. Bruschi, F. Davoli, and C. Lombardo, “Fine-grained energy-efficient consolidation in sdn networks and devices,” *Network and Service Management, IEEE Transactions on*, vol. 12, no. 2, pp. 132–145, 2015.

[8] R. Wang, Z. Jiang, S. Gao, W. Yang, Y. Xia, and M. Zhu, “Energy-aware routing algorithms in software-defined networks,” in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2014, Sydney, Australia, June 19, 2014*. IEEE Computer Society, 2014, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/WoWMoM.2014.6918982

[9] M. Schaap, “Saving energy in openflow computer networks,” B.S. Thesis, University of Amsterdam, 1098 XH Amsterdam, Holland, 6 2015.

[10] “OSPF (Open Shortest Path First),” http://searchenterprisewan.techtarget.com/definition/OSPF, accessed: 2016-12-15.

[11] C. E. Leiserson, “Fat-trees: Universal networks for hardware-efficient supercomputing,” *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985. [Online]. Available: http://dl.acm.org/citation.cfm?id=4492.4495

[12] K. S. Solnushkin, “Automated design of two-layer fat-tree networks,” http://arxiv.org/abs/1301.6179, January 2013, arXiv:1301.6179 [cs.DC].