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. 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.

II. Implemented Algorithm

We have implemented our SA approach, using native Python, based on numerous parameters, neighborhood structure and stochastic decisions. 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 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:

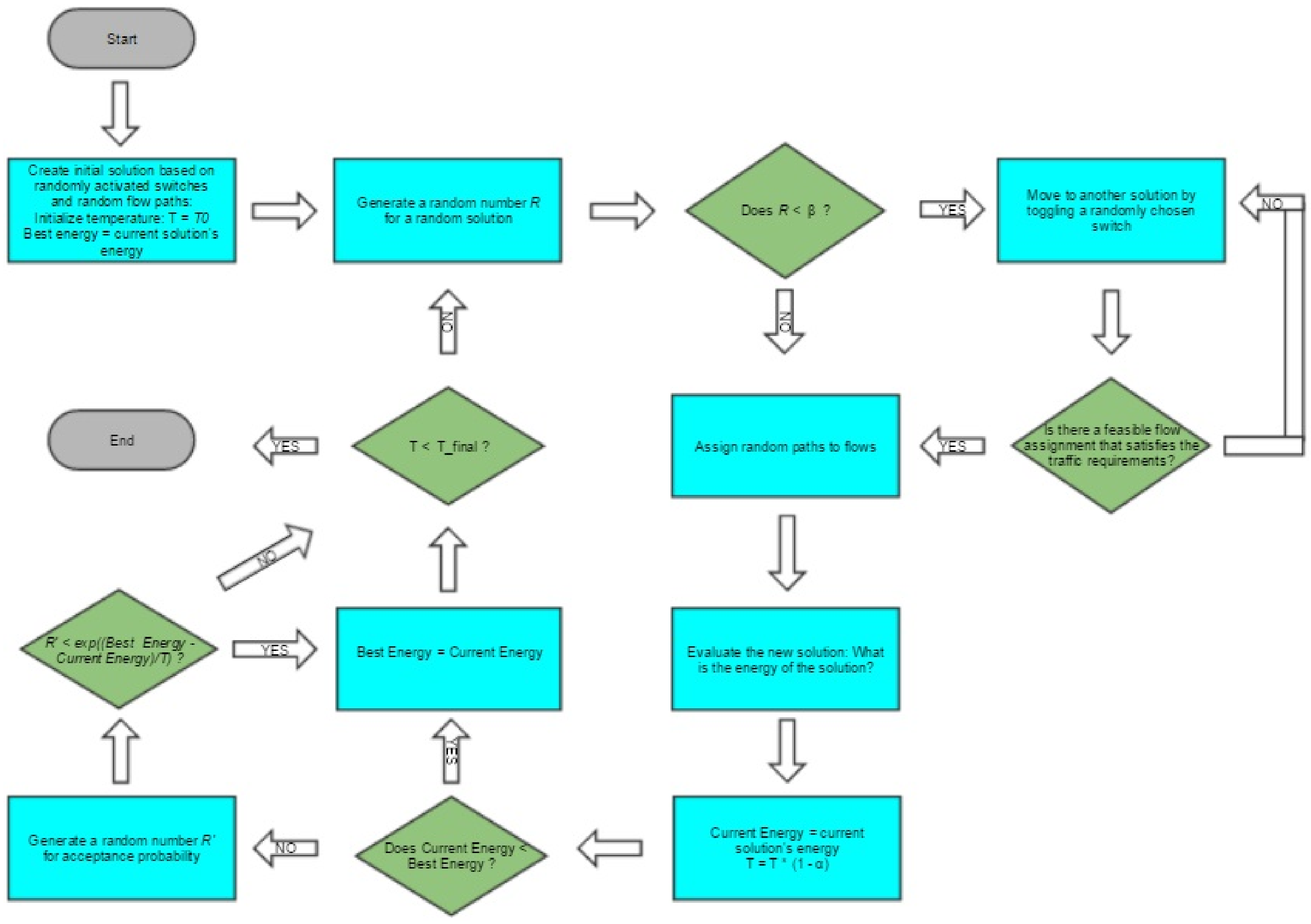
Start
Create initial solution based on randomly activated switches and random flow paths: Initialize temperature: T = T0 Best energy = current solution's energy
Generate a random number R for a random solution
Does R < β ?
YES
Move to another solution by toggling a randomly chosen switch
NO
NO
NO
Is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

Fig. 1: The Flowchart of Our Algorithm

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[1] 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.

[1]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.

III. Demonstrative Cases

As we have already mentioned in the previous report, the FatTree architecture ([1], [2]) 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.

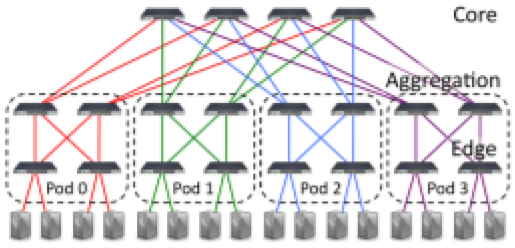

Fig. 2: The Topology used in our Demo Case: Figure from [3]

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

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

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.

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

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

IV. 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 look at how much energy[2] 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 as stated in [4]. We also approximate the full base power of a port to 100W. 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 the results look promising, we should understand that these cases are run only once. In order to eliminate variance effect, we need to setup an experiment environment and make multiple repetitions of the same cases to retrieve the mean and variance of the results.

V. Planned Experiments

In the experiment phase, we are planning to make use of the Mininet[5] simulation environment, because Mininet makes the process of simulating a SDN network much easier than many other tools. Moreover, it also provides interfaces to monitor the data flow and states of the network elements.

While doing the experiments, we are planning to factorize some of the parameters. For example, we will be testing

[2]We use power consumption and energy consumption interchangeable in the context of this paper, as stated in the previous report

our design tool with different port number (k) such as 6, 8, 10 etc. In addition, different number of hosts and various traffic requirement matrices will be provided to test our system further.

In addition to the workload parameters, we are also planning to factorize the system parameters, such as cooling rate α. Trying different values for such parameters is especially important since the results of Simulated Annealing depends highly on them. For instance, low values of α makes the process more sensitive and might result in better solutions with some performance trade-off. Similarly, the initial temperature T_0 and final temperature T_{final} parameters also affect the performance and accuracy of the algorithm. Therefore, these parameters will be tested, too.

Furthermore, after deciding which parameters are more critical for the system, we might fix the others and focus on testing the crucial ones. We might provide a wider range of possible values for such parameters to see under what conditions our algorithm can reach its best configuration. Moreover, we will be making several repetitions of the important cases in order to decrease the variance. Following this way, we will have better idea about the true quality of our approach.

REFERENCES

[1] 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

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

[3] 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

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

[5] "Mininet," http://mininet.org/, accessed: 2016-11-15.