Efficient Switch Migration for Controller Load Balancing in Software Defined Networking

(Invited Paper)

1st Rajorshi Biswas
dept. computer and information sciences
temple university
Philadelphia, United States
rajorshi@temple.edu

2nd Jie Wu
dept. computer and information sciences
temple university
Philadelphia, United States
jiewu@temple.edu

***Abstract*—The number of multi-controller datacenters is increasing with the increasing size of software-defined networking (SDN) datacenters. The performance of an SDN datacenter depends largely on the delay of response from the controller. The delay of response depends on the controller load and the distance from the SDN switch. The load of a controller depends on the number of requests it receives from the switches it controls. Therefore, a good switch-controller assignment is very important for load balancing the controller and the performance of an SDN datacenter. In this paper, we consider multiple controllers and formulate problems for initial and incremental load balancing. The initial assignment process is executed at the beginning of the network deployment. After initial deployment, the incremental assignment process is executed periodically. The incremental process migrates some of the switches to another controller to improve the performance of the network. We propose greedy and clustering-based solutions for initial switch-controller assignment. We also propose a greedy solution for incremental assignment. Our proposed solutions are evaluated using both synthetic and real datasets, and the parameters are driven by experiments at a data center.**

***Index Terms*—controller assignment, controller load balancing, software defined networking, minimization**

I. Introduction

Software-defined networking (SDN) technology is an approach to manage networks dynamically and programmatically. Unlike the traditional network system where a human (network administrator) manages the network components including switches and routers, a software called a controller manages the network components. The SDN system also enables the feasibility of decoupling the controller network and data network. An in-band controller system enables the SDN switches to communicate with the controller via the data network. On the other hand, the out-band controller system uses the control network to communicate with the controller.

The forwarding of packets in an SDN switch is defined by the rules installed in it. There are two types of rules: wildcard and re-active rules [1]. When a packet arrives at any ports of an SDN switch, it first tries to use the wildcard rules. A wildcard rule is stored in the SDN switch and its usage for forwarding packets is not reported to the controller. When there exist no rules to match the packet, the SDN switch asks for a forwarding decision from the controller. The controller replies with the decision (output to one or multiple ports, drop, broadcast, packet modifications, etc.). The requests from a switch leave some workload on the controller. For example, when the first SDN switch asks for a forwarding rule for the packet of a flow, the controller needs to construct the routing path for the flow, create the corresponding rules, and reply to the rule for forwarding the packet. We represent this type of request as a path construction request. This process takes longer than processing other types of requests. For example, when an SDN switch asks for a forwarding rule for a packet of the flow, the controller does not need to re-calculate the path. It replies to the rules already created for that flow. We call this type of request as an intermediate query request. Therefore, for different types of requests, the controller load is different. If the number of switches and the flows in an SDN network is large, then one controller cannot handle all of the requests. In this situation, the network is divided into domains and a controller is assigned to each domain. The network needs to be divided wisely to balance the load of the controllers.

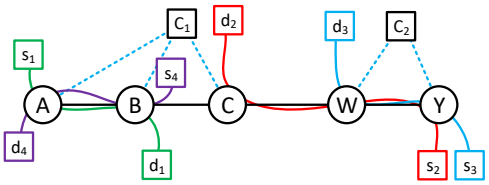


Fig. 1: An Example of switch migration.

For example, in Fig. 1, there are five SDN switches (A, B, C, X, and Y) and four flows ($f_1 : s_1 \rightarrow d_1$, $f_2 : s_2 \rightarrow d_2$, $f_3 : s_3 \rightarrow d_3$, and $f_4 : s_4 \rightarrow d_4$) in the network. Let us consider that switches A, B, and C are controlled by controller C_1 and switches W and Y are controlled by controller C_2. In this assignment, C_1 becomes overloaded with three path construction and two intermediate query requests. On the other hand, C_2 is underutilized because of two path construction and two intermediate query requests. If we migrate the switch B

ISBN 978-3-903176-43-0 © 2021 IFIP
Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

to controller C_2, then it will increase one more intermediate query to C_2, but C_1 will receive one less path construction request. Usually, path constructions induce almost ten times the load than intermediate query requests do. Therefore, C_1 is no longer overloaded and C_2 gets a slightly higher load than earlier. Switch migration is challenging for several reasons. Firstly, to the best of our knowledge none of the commercially available SDN switches support migration. However, it is possible to implement the migration feature by using SSH and Linux commands provided by some of the switches. The downside of this type of migration is that it will block packet forwarding for a period.

According to the SDN architecture, the controllers know the links, their usage, and SDN switches. Each of the controllers knows a portion of the topology and they report their knowledge to the assignment manager (AM) server. The AM gets the global view of the topology and uses that information to assign switches to the controllers so that the load is balanced. As we know, the topology and flows are subject to change over time. Therefore, the AM needs to adjust the assignment based on new topology and flow information changes. Therefore, we needed to consider two types of switch-controller assignment processes: initial assignment and incremental assignment. The initial assignment process is executed at the beginning of the network deployment and the incremental assignment process migrates some switches from one controller to another. These problems of the initial and incremental assignment are NP-hard and we provide greedy and clustering-based solutions. The main contributions of our paper are:

1) We study a switch-controller assignment problem to minimize network overhead and propose a solution for initial deployment.
2) We investigate incremental switch-controller assignment problems and provide greedy solutions.
3) We conduct extensive simulations and experimental results and compare them with existing approaches.

The remainder of this paper is arranged as follows. Section II presents some related work. In Section III, we present the system and the cost model. In Section IV, we present the problem of switch controller assignment and propose two solutions. Section V contains the problem of incremental switch-controller assignment and the proposed greedy solution. Section VI and Section VI present the simulation and experimental and results. Finally, Section VII concludes our paper.

II. Related Work

There exists several works on switch-controller assignment and load balancing. In [2], authors formulate the dynamic controller assignment problem as an online optimization to minimize the total cost. The cost includes response time and maintenance on the cluster of controllers. They propose a two-phase algorithm that integrates key concepts from both matching theory and coalitional games to solve it. In [3], authors propose an effective switch-controller mapping scheme for distributed controllers and distributes flow setup requests among them in order to minimize flow setup time, and obtain resilience. In [4], authors show that dynamic mapping between switches and controllers improves efficiency in traffic load balancing. They propose two balanced controllers (BalCon) and BalConPlus that are switch migration schemes to achieve load balance among SDN controllers with minimum migration cost. BalCon is limited to the scenarios where the network can process switch requests out of order, otherwise BalConPlus is more suitable.

In [5], authors propose a switch migration-based decision-making (SMDM) scheme that is aware of the load imbalance. They consider the tradeoff between migration costs and the load balance rate for producing a switch migration trigger. They propose a greedy method that utilizes the migration efficiency to produce possible migration actions. A game theory based decision mechanism for swith migration is proposed in [6]. Their system dynamically migrates switches from heavily loaded controllers to lightly loaded controllers based on a centralized available resource utilization maximization problem. The system takes controllers and switches as game player, and how to migrate switches among the control plane is considered as the player policy. In [7], a framework for deploying multiple controllers within a WAN is proposed. The framework dynamically adjusts the number of active controllers. This system only considers the flow setup latency involved in path construction in controllers, formulates an optimization problem, and solves using Integer Linear Program. Other works that uses ILP for solving controller load balancing includes [8–11]. In [12], authors propose a dynamic load balancing method based on switch migration mechanism for clustered controllers. The mechanism can also handle controller failover without switch disconnection. In [13], authors propose a mechanism to maximize resource utilization. It detects imbalance in load and the controller randomly selects a switch for migrating. It also maintains integrity by broadcasting the migration activity to the controller's neighbors.

Some other works that propose distributed controller system include [14–21]. However these systems do not consider the cost of migration and other performance metrics including packet forwarding delays while load balancing. Most of the works discussed here consider controller load as the main load balancing/switch migrating parameters. None of them consider the control plane communication overhead and intermediate queries for deciding switch migration that are important parameters for network performance. Therefore, a system that considers controller load and control plane communication overhead for deciding switch migration to obtain a load balanced state of the controller is necessary.

III. Network Model

Our network is composed of SDN switches, controllers, sources, destinations, and flows. We assume that the controller knows the links, their usage, and SDN switches. Each of the controllers knows a portion of the topology. The controllers report their knowledge to the assignment manager (AM)

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

server. Therefore, the AM has a global view of the topology and the flows in the network. The AM also has knowledge about the control plane and thus can infer the number of hops and delays between switches and controllers.

The AM is responsible for the assignment of switches to the controllers. By analyzing the topology and the flows, it finds out the switch-controller assignment that minimizes the overall transmission network delay. As we know, the topology and flows are subject to change over time. Therefore, the AM needs to adjust the assignment based on new topology and flow information. So, there are two types of switch-controller assignment processes: initial assignment and incremental assignment. The initial assignment process is executed at the beginning of the network deployment. After initial deployment, the AM executes the incremental assignment process. The incremental process migrates some of the switches to another controller to improve the performance of the network.

A switch migration refers to changing the controller of a switch. The switch migration is a complex process that includes removing the existing controller, adding the new controller, and initialization of the switch with the controller. Switch migration is challenging for several reasons. Firstly, to the best of our knowledge, none of the commercially available SDN switches support migration. However, it is possible to implement the migration feature by using SSH and Linux commands provided by some of the switches. The downside of this type of migration is that it will block packet forwarding for a period. The replacement of the controller can be done in two ways: direct replacement and switch to the backup. Direct replacement refers to the process of deleting the primary active controller and adding a new controller. The switch to backup approach refers to adding the new controller as a backup controller, followed by the removal of the primary controller.

Therefore, our system is a two phase system. In the first phase, the switches are assigned according to the initial deployment methods. After that, the system enters into the second phase which is basically adjusting the assignment with the dynamic changes in flows. In the second phase, the adjustments are done periodically using the incremental deployment approach. The period of adjustment is determined based on the dynamic nature of the SDN network.

A. Cost Model

Unlike the cost metric in a single controller system [22] where only the number of rules are considered, the cost of a switch-controller assignment is incurred by the three parameters: the number of path construction requests, the number of intermediate query request, and the number of hops from the controller to the switches. The response delay from a controller depends on the number of hops from the controller to the switches. $P(A, c)$ denotes the number of path construction requests at controller c for A assignment. $A(v)$ represents the assigned controller for SDN switch v. Therefore, $P(A, c)$ can be expressed as the following:

$$P(A, c) = \sum_{f \in F} | \bigcup_{v \in P_f, A(v)=c} A(v)| \tag{1}$$

$Q(A, c)$ denotes the number of intermediate query requests at controller c for A assignment. $Q(A, c)$ is the number of all forwarding nodes that are controlled by c on the path of all flows minus the number of path construction requests. Therefore, $Q(A, c)$ can be expressed as the following:

$$Q(A, c) = \sum_{f \in F} |P_f, A(v) = c| - P(A, c) \tag{2}$$

$D(A, c)$ denotes the total number of hops traveled (in control plane) by all requests at controller c for A assignment. That is why we multiply the distance from the controller to the nodes ($d_{v,A(v)}$) with the number of requests. Therefore, $D(A, c)$ can be expressed as the following:

$$D(A, c) = \sum_{f \in F} \sum_{v \in P_f, A(v)=c} d_{v,A(v)} \times (P(A, c) + Q(A, c)) \tag{3}$$

We define the cost of a switch-filter assignment as a weighted summation of the above three metrics. Therefore the cost of assignment A can be expressed as the following:

$$C(A, c) = \omega_1 P(A, c) + \omega_2 Q(A, c) + (1 - \omega_1 - \omega_2) D(A, c) \tag{4}$$

Here, ω_1 and ω_2 are the weights that correspond to the amount of workload induced at the controller by a path construction request and an intermediate query request, respectively.

In the next sections, we formulate problems to minimize this cost. This cost represents the overall transmission delays in an SDN network. This is because, when the number of path construction requests and intermediate query requests are high, the controller needs a long time to produce a decision and respond to the requests. The delay to receive the responses at a switch also depends on the number of hops to reach the controller. As a result, when the number of hops to the controller is high, a packet faces delay on each intermediate node. The SDN switches usually catche the forwarding rules locally in their limited memory which makes the forwarding fast. In this case, only the first packet of the flow encounters delays on each hop. When the number of rules in a switch is very high, then the catching cannot provide the best performance. In that case, not only the first packets but also the following packets encounter delays.

IV. Initial Deployment

In this section, we formulate the problem of switch assignment so that the maximum cost for all flows is minimal.

Problem I: *Find switch to controller mapping so that the maximum delay of all flows is minimum.*

Let the topology be $G = (V, E)$ where $V = \{v_1, v_2, ..., v_N\}$ is the set of N SDN switches and E is the set of links. Let the set of controllers $C = \{c_1, c_2, ...c_M\}$ contain M

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Find a switch assignment

Input: Topology G, set of controller C, set of flows F.
Output: A set of links to block from L_c.

Procedure: FIND-ASSIGNMENT(G)
  $\forall c \in C$ $B[c] \leftarrow$ initial switch.
  for $c \in C$ **do**
    $Candidate \leftarrow$ NEI(B[C])
    for $s \in Candidate$ **do**
      $cost[s] \leftarrow$ COST(s, B, F, G)
    $B[c] = B[c] \cup$ ARGMIN$(cost)$
    $A($ARGMIN(COST)$) \leftarrow c$
  return A

controllers. The set of flows in the network is denoted by $F = \{f_1, f_2, ..., f_K\}$. Therefore, the problem can be expressed as the following:

$$\begin{aligned} \text{minimize} \quad & \max_{\forall c \in C} C(A, c) \\ \text{subject to} \quad & \forall_{v \in V} \, d_{vA(v)} \leq Th_1 \end{aligned} \tag{5}$$

Here, Th_1 is the threshold, the maximum allowable distance from a switch to a controller.

A. Solution Approaches

In this subsection, we will present several proposed solutions for problem I. The problem is NP-Hard and its NP-Hardness can be proved by converting the problem into the famous graph partitioning problem. We propose three approaches based on greedy and clustering methods.

B. Greedy Controller Assignment

In this approach, we assign a controller to each switch in a greedy way. We consider that each controller is a bucket and each switch is an element that will eventually be put in a bucket. The process is composed of two parts: initialization of bucket and incremental growth of bucket. The initialization of a bucket is very important because it directs the rest of the assignments. If the initially assigned switches are very close to each other, then there will be fewer options to grow each bucket. Therefore, we select a switch for each bucket in such a way that they are at least a certain number of hops away from each other. We also prioritize the distance from the controller to the switch.

After the initialization of the buckets, we consider some candidates for the extension of each bucket. The candidate switches for extension of a bucket include the neighbors of each switch in that bucket. We pick the switch that adds a minimum amount of cost. The process continues until all of the switches are covered. The complete algorithm is shown in Alg. 1.

C. An Example to Greedy Controller Assignment

Let us consider the topology and flows in Fig. 2. There are eight SDN switches ($A, B, C, D, W, X, Y,$ and Z) and two controllers ($C_1,$ and C_2). There are four flows

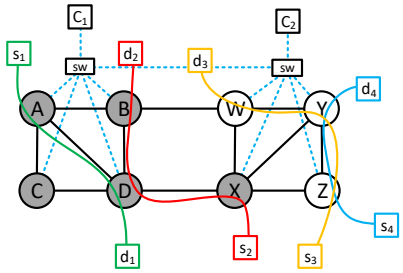


Fig. 2: Example for initial switch migration.

($f_1, f_2, f_3,$ and f_4) originating at $s_1, s_2, s_3,$, and s_4 and destined to $d_1, d_2, d_3,$, and d_4, respectively. Distances from C_1 (or C_2) to the switches $A, B, C,$ and , D is two (or three) hops. Distances from C_1 (or C_2) to the switches $W, X, Y,$ and , Z is three (or two) hops.

According to Alg. 1, we first need to initialize the buckets for the controllers. Let the minimum distance among the initial items in the bucket be 2. We consider $\omega_1 = 0.08$ and $\omega_2 = 0.8$ There are several options for choosing the initial items of the buckets. Let us choose $\{\{A\}, \{W\}\}$ where the first and second elements correspond to the bucket related to C_1 and C_2, respectively.

Now, for the first and second buckets, we have three candidates for extension $\{B, C, D\}$ and $\{B, X, Y\}$, respectively. Now we need to calculate the cost of extension for each candidate element. The cost of adding C to the first bucket is $0.8 \times 0 + 0.08 \times 0 + 0.12 \times 2 = 0.24$. This is because there is no new path construction requests or intermediate query requests. The cost of adding B to the first bucket is $0.8 \times 1 + 0.08 \times 0 + 0.12 \times 2 = 1.04$. This is because there is one new path construction requests and no intermediate query requests. The cost of adding D to the first bucket is $0.8 \times 1 + 0.08 \times 2 + 0.12 \times 2 = 1.2$. This is because there is one new path construction requests and two intermediate query requests.

Next, we need to calculate the cost of extension for each candidate element of the second bucket. The cost of adding B to the first bucket is $0.8 \times 1 + 0.08 \times 0 + 0.12 \times 3 = .26$. This is because there is no new path construction requests or intermediate query requests. The cost of adding W to the first bucket is $0.8 \times 1 + 0.08 \times 0 + 0.12 \times 2 = 1.04$. This is because there is one new path construction requests and no intermediate query requests. The cost of adding Y to the first bucket is $0.8 \times 2 + 0.08 \times 0 + 0.12 \times 2 = 1.2$. This is because there is one new path construction requests and two intermediate query requests. The cost of adding X to the first bucket is $0.8 \times 1 + 0.08 \times 2 + 0.12 \times 2 = 1.8$. This is because there is one new path construction requests and two intermediate query requests.

Therefore, adding switch C to the first bucket produces the lowest cost increase. Therefore, the new buckets are $\{\{A, C\}, \{W\}\}$. Similarly, we calculate the costs for each candidate in the candidate sets and pick the one that produces the lowest cost. Finally, the buckets are $\{\{A, C, D, X\}, \{W, Y, Z\}\}$ which represent that switches A,

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

	A	B	C	D	W	X	Y	Z	C_1	C_2
A	0	1	1	1	2	2	3	3	2	3
B	1	0	2	1	1	2	2	3	2	3
C	1	2	0	1	3	2	3	3	2	3
D	1	1	1	0	2	1	2	2	2	3
W	2	1	3	2	0	1	1	2	3	2
X	2	2	2	1	1	0	1	1	3	2
Y	3	2	3	2	1	1	0	1	3	2
Z	3	3	3	2	2	1	1	0	3	2
C_1	2	2	2	2	3	3	3	3	0	3
C_2	3	3	3	3	2	2	2	2	3	0

Fig. 3: Distance matrix for clustering.

B, C, D, and X will be assigned to controller C_1 and switches W,Y, and Z will be assigned to controller C_2. The total cost for this assignment is 24.6.

Theorem 1. *The complexity of Alg. 1 is* $O(|C|(|V|^2 + |V||F|)))$.

Proof. To calculate the complexity of Alg. 1, we need to calculate complexity of $\textsc{Cost}(s, B, F, G)$. If we pre-compute the number of path construction requests and intermediate queries, then $\textsc{Cost}(s, B, F, G)$ can be obtained in constant time. The pre-computation takes $O(|V||F|)$. Step 5 to 8 takes $O((|V|^2))$ and the loop at Step 4 runs for $|C|$ times. Therefore, the Alg. 3 takes $O(|C|(|V|^2 + |V||F|)))$. □

D. Hierarchical Clustering

We group the switches using the hierarchical clustering algorithm. We formulate the distance matrix from the distance between two nodes in topology. The distance values between a pair of nodes are normalized by dividing with the maximum distance. The normalised distances are used for hierarchical clustering. The distance matrix is represented as follows:

$$D[u, v] = \frac{d_{u,v}}{\max_{u,v \in V} d_{u,v}} \tag{6}$$

Next, we use D to cluster the nodes into $|C|$ groups. We use the hierarchical clustering algorithm to partition the nodes. We set the maximum class to be the number of controllers ($|C|$). This clustering will group the most similar nodes in a cluster. We pick a cluster and calculate the average number of hops to each controller. We assign the closest controller to the nodes in that cluster. Then, we continue the process with the unassigned clusters and controllers. The complete approach is shown in Algorithm 2.

E. An Example of Hierarchical Clustering Solution

We are going to use the topology in Fig. 3 to explain this solution. We group the nodes using the hierarchical clustering algorithm. The similarity values between a pair of nodes are subtracted from the maximum of the similarity values (except the similarity value of a node with itself) to get dissimilarities. The dissimilarities are used as distances for hierarchical clustering. The distance matrix is represented as follows:

Algorithm 2 Find a switch assignment

Input: Topology G, set of controller C, set of flows F.
Output: A set of links to block from L_c.

1: **Procedure:** FIND-ASSIGNMENT(G)
2: $\forall_{u,v \in V \cup C} D[u, v] \leftarrow d_{u,v} / \max_{u,v \in V} d_{u,v}$
3: $CL \leftarrow \textsc{Hierarchical}(D)$
4: **for** all $c \in CL$ **do**
5: $min \leftarrow \textsc{Closest}(c, C)$
6: **for** all $v \in c$ **do**
7: $A(v) \leftarrow min$
8: $C \leftarrow C \setminus min$
9: **return** A

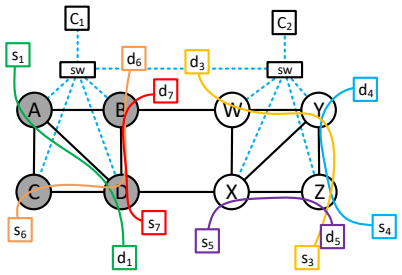


Fig. 4: Example for incremental switch migration.

Using Equation 6, we calculate $D'[A, B]$. Therefore, $D[A, B] = d_{A,B}/4 = 0.25$. Similarly, we calculate that $D'[B, C] = 2$ and $D'[B, D] = 1$. So, we can see that nodes 1 and 2 are more similar than nodes 1 and 4 or 2 and 7. The similarity score between 1 and 3 is 0. This is because they are different in type and they do not have any similar neighbors. Finally, by using hierarchical clustering we find that the clusters are $\{\{A, C, D, X, C_1\}, \{W, Y, Z, C_2\}\}$ which represent that switches A, B, C, D, and X will be assigned to controller C_1 and switches W,Y, and Z will be assigned to controller C_2. The total cost for this assignment is also 24.6.

Theorem 2. *The complexity of Alg. 2 is* $O(|V|^3))$.

Proof. To calculate the distance matrix we need $O(|V|^2)$. The the rest of the part is determined by the complexity of the hierarchical clustering. The standard algorithm for hierarchical agglomerative clustering has a time complexity of $O(|V|^3)$. In Step 5, to find the closest controller it take $O(|V|)$ for each clusters. Therefore, for all clusters, it will take at most $O(|C||V|)$ time. Therefore, the total complexity is $O(|V|^3 + |V|^2 + |C||V|)$. In the worst case, the maximum number of controllers can be at most $|V|$. Therefore, the complexity of Alg. 2 is $O(|V|^3)$. □

V. Incremental Deployment

In this section, we formulate the problem of switch assignment so that the maximum cost for all flows is minimal.

Problem II: *Find a switch to controller mapping so that the maximum delay of all flows is the minimum by ensuring limited changes.*

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3 Find a switch assignment

Input: Topology G, set of controller C, set of flows F, current Assignment A.
Output: A set of links to block from L_c.

Procedure: FIND-ASSIGNMENT(G, C, F, A)
  while DIFF(A, A') $< K$ **do**
    Divide C into C_o and C_u
    for all $c \in C_o$ **do**
      for $v, u \in V : A(v) = c$ & $A(u) \in$ NEI(c) **do**
        $B[u, v] \leftarrow$ BEN(V,U)
    $u, v \leftarrow$ ARGMAX(B)
    $A'(v) \leftarrow A(u)$
  return A'

Let the topology be $G = (V, E)$ where $V = \{v_1, v_2, ..., v_N\}$ is the set of N SDN switches and E is the set of links. Let the set of controller $C = \{c_1, c_2, ...c_M\}$ contains M controllers. The set of flows in the network is denoted by $F = \{f_1, f_2, ..., f_K\}$. Therefore, the problem can be expressed as the following:

$$\begin{aligned} \text{minimize} \quad & \max_{\forall c \in C} C(A, c) \\ \text{subject to} \quad & \forall_{v \in V},\ d_{vA(v)} \leq Th_1, \\ & \text{DIFF}(A, A') \leq Th_2 \end{aligned} \tag{7}$$

Here, Th_2 is the threshold of changes in an assignment. DIFF(A,A') is the number or changes between the current assignment A' and the next assignment A. Therefore, DIFF(A,A') can be expressed as the following:

$$\begin{aligned} \text{DIFF(A,A')} &= \sum_{v \in V} x(A(v), A'(v)) \\ x(a, b) &= \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

A. Solution Approaches

In this subsection, we present our proposed solutions for problem II. Though we have added one more constraints, the problem is still NP-Hard We propose three approaches based on greedy methods.

B. Greedy Incremental Deployment

To find the new assignment, we first calculate the load of each controller. By using the equation 3, we calculate the load of the controllers in C. Then, we divide the C into two parts C_o and C_u ($C = C_o \cup C_u$) based on the load of the controller. If a load of a controller is higher (or lower) than a threshold, then C_o (or C_u) will contain it. Therefore, C_o represents the set of the overloaded controllers and C_u represents the set of the depleted controllers. The part of the topology that is controlled by an overloaded controller is called the overloaded domain and the part of the topology that is controlled by a depleted controller is called the depleted domain.

Next, for each overloaded domain, we consider the depleted neighboring domains as the candidate domains. Then, we need to find a switch that will be migrated to the neighboring domain. Let $A(v) \in C_o$ and $A(u) \in C_u$ such that there is a link between u and v. In this case, v may be considered to migrate to $A(u)$. We consider a parameter called migration benefit for determining the priority of migration in this greedy approach. The migration benefit is the ratio of increased total load and the amount of transferred load. Let A be the current assignment and A' is the new assignment after migrating v to the neighboring domain ($A'(v) = A(u)$).

$$\begin{aligned} &\text{BEN}(v, u) = \\ &C(A, A(v)) + C(A, A(u)) - C(A', A'(v)) - C(A', A'(u)) \end{aligned} \tag{9}$$

Among all eligible migrations we pick the pair that provides the maximum benefit. The process continues until the number of migrations is equal to K.

C. An Example of Incremental Assignment

Let us consider the current topology and flows in Fig. 4 and the previous assignment $\{A, B, C, D, X\}$ to C_1 and $\{W, Y, Z\}$ to C_2. Because of three new flows (f_5, f_6 and f_7) and an expired flow f_4 in Fig. 4, controller C_1 gets overwhelmed. Therefore, the $C_o\{C_1\}$ and $C_u\{C_2\}$. There are two possible migration options here: migrate B to C_2 or migrate X to C_2. Next, we will calculate the benefit of migration for both options.

For the first option (migrate B to C_2) the benefit is $\text{BEN}(v, u) = (4.48 + 3.36 - 3.84 - 4.96) = -0.96$. If we migrate X to C_2 then the benefit will be $\text{BEN}(v, u) = (4.48 + 3.36 - 3.68 - 3.68) = 0.48$. Therefore, we get a higher benefit from migrating the switch X to C_2. Fig. 4 shows the new assignment after migration.

Theorem 3. *The complexity of Alg. 3 is* $O(|F||V|K))$.

Proof. To calculate the complexity of Alg. 3, we need to calculate complexity of BEN(u, v). According to the Eq. 4, it takes $O(|F||V|))$ to calculate cost. Therefore, the complexity of BEN(u, v) is $O(|F||V|))$. The nested loops in Step 4-8 take $O(|V||C|)$. The loop at Step 2 executes at most K times. Therefore, the Alg. 3 takes $O(|F||V|K)$. □

VI. SIMULATION AND EXPERIMENTS

In this section, we present our experimental and simulation results comparing some existing works.

A. Experimental Settings

We conduct the experiments in our data center with fifteen SDN switches. We use the regular switches in the control plane. The partial topology of the data center is shown in Fig. 5. Nodes 2 to 16 are Pica 8 (P-3297) SDN switches. The numbers at the ends of a link denote the port number where it is plugged. There are four servers (101-104) that are connected at the leaf level switches. We create three flows $\{101 \rightarrow 102, 101 \rightarrow 103, 101 \rightarrow 104\}$ and record the ping delays (round trip delay). Initially, switches between 2 and 12

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

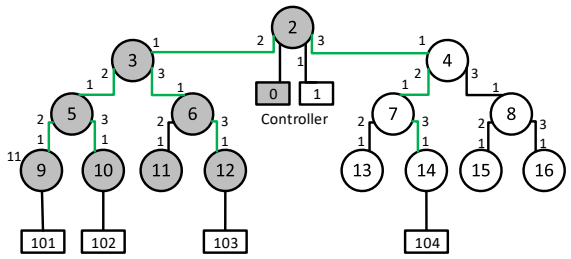


Fig. 5: Datacenter topology (partial).

are controlled by controller 0, and switches between 4 and 16 are controlled by controller 1.

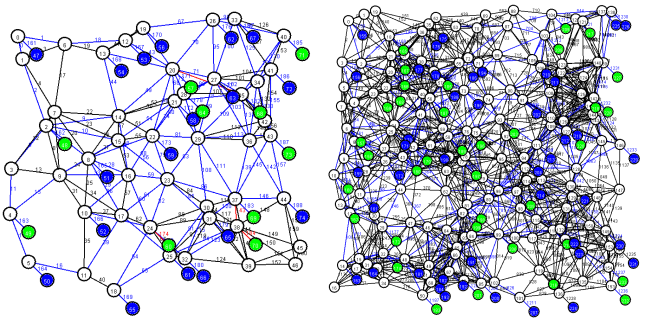
We use ONOS (2.2.2) as the controller software. We develop a Java program that can connect the switches through SSH and change the controller of any switch. The program does not need to run on the same machine as the controller because it communicates with the controller through SSH. We also develop a flow generator program that keeps generating flows to random destinations. The controller modifier program first uses the direct replacement method and observes the delay to restore the communication of $101 \rightarrow 104$ flow. We observe an average delay of 5.2 seconds for the direct replacement of controllers. The controller modifier program is used to apply the switch to backup methods. We observe an average delay of 5.6 seconds for this method. Therefore, from these experiments, we conclude that the direct replacement method works better than the switch to backup method.

After that we run some experiments to obtain the values of the weights (ω_1, and ω_2). To do so, we needed to change the control topology to make the different number of hops reach the controller. We change the number of hops between switches and the controllers from 2 to 5 and observe the round trip time of the three flows mentioned above. We observe the first round trip time is higher than the following round trip times. This is because of the path establishment overhead of the controller. This delay is used to calculate the weight of path construction cost (ω_1). From the delays, we calculate the delay of the response from the controller for different hops (ω_2). We use these values in the simulations presented in the next subsections.

B. Simulation Settings

We conducted all the experiments with a custom-built Java simulator. The main reason for using a custom-built simulator is its scalability and fast execution time. We do not need to analyze transmission time or natural packet drop issues rather to calculate cost of switch controller assignment, migration costs, and benefits. The network topologies we considered contain $100 - 300$ switches. Based on our experience, using NS3 or other similar simulators for this kind of simulation would take several days. That is why we built our own Java multi-threaded simulator to get the results quickly.

We generate random topologies by dividing an area of 500×500 square unit into 50×50 blocks. In each block, a certain number of nodes are placed randomly to make an uniform distribution of nodes. We randomly select some nodes and



(a) Topology T_1. (b) Topology T_2.

Fig. 6: Randomly generated topologies.

attach a source or destination. We set the link capacities as 100 Mbps for simplicity and ease of comparison. After that, we generate the desired number of flows by randomly selecting a source and a destination. We set the data rate randomly. The initial routing of the flows are done using shortest path routing. Fig. 6 shows the randomly generated topologies. Topology T_1 is small and sparse and contains 75 nodes. Topology T_2 is large and dense and contains 233 nodes.

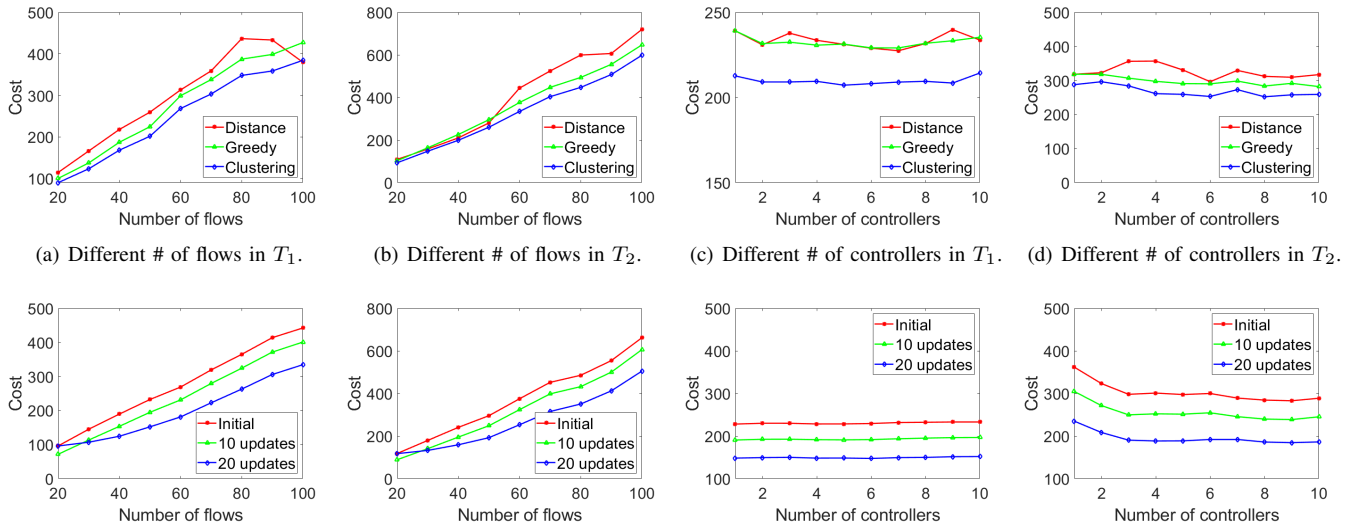
We measure the cost and compare with different approaches. In the distance based approach, each switch is assigned to the closest available controller. All of the results in the plot are the average of 1000 runs.

C. Simulation Results

Firstly, we observe the cost of controller assignment by changing different parameters, such as the number of rules and number of controllers. Fig. 7(a) shows the cost of assignment for different number of flows in T_1. We vary the number of flows from 20 to 100 and keep the maximum data rate as 10 Mbps. We set the number of controllers as 5. For all assignment methods, the cost increases with the increase of the number of flows. This is because when the number of flows is higher, the controllers get a higher number of requests from the switches. The distance-based clustering-based method produces the highest and the lowest amount of costs for all numbers of flows, respectively. The cost produced by greedy assignment method remains in between them. When the number of flows is 20 and distance-based method is used, the average cost is 114.78. When the greedy method is used, the average cost is 100.42. When the clustering-based method is used, the average cost is 90.14. The cost decreased by about 12% and 21% in greedy and clustering methods. When the number of flows is 80 and distance-based method is used, the average cost is 407.34. When the greedy method is used, the average cost is 366.25. When the clustering-based method is used, the average cost is 328.95. The cost decreased by about 10% and 19% in greedy and clustering methods.

Fig. 7(b) shows the cost of assignment for different number of flows in T_2. We keep the same settings as the previous simulation. We observe a higher overall cost in T_2 than T_1. This is because the topology is large and the average number of hops between the sources and the destinations is also higher

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.



(a) Different # of flows in T_1. (b) Different # of flows in T_2. (c) Different # of controllers in T_1. (d) Different # of controllers in T_2.

(e) Different # of flows in T_1. (f) Different # of flows in T_2. (g) Different # of controllers in T_1. (h) Different # of controllers in T_2.

Fig. 7: Simulation results.

in T_2. The distance-based clustering-based method produces the highest and the lowest amount of costs for all numbers of flows, respectively. The cost produced by greedy assignment method remains in between them. When the number of flows is 20 and distance-based method is used, the average cost is 109.39. When the greedy method is used, the average cost is 103.28. When the clustering-based method is used, the average cost is 1.54. The cost decreased by about 5.5% and 14.5% in greedy and clustering methods. When the number of flows is 100 and distance-based method is used, the average cost is 719.63. When the greedy method is used, the average cost is 647.03. When the clustering-based method is used, the average cost is 1.54. The cost decreased by about 10% and 17% in greedy and clustering methods.

Next, we observe the cost of controller assignment by changing the number of controllers. Fig. 7(c) shows the cost of assignment for different number of controllers in T_1. We vary the number of controllers from 2 to 10 and keep the number of flows as 50. Other settings are kept the same as the previous simulation. For all assignment methods, the cost decreases slightly with the increase of number of controllers. This is because when the number of controllers is higher, there are more options of assignment and the average distance between switches and controllers reduces. The distance-based clustering-based method also produces the highest and the lowest amount of costs for all numbers of controllers, respectively. The cost produced by greedy assignment method is lightly lower than the distance based method. When the number of controllers is 2 and distance-based method is used, the average cost is 230.92. When the greedy method is used, the average cost is 231.63. When the clustering-based method is used, the average cost is 209.10. The cost decreased by about 0.3% and 9% in greedy and clustering methods. When the number of controllers is 10 and distance-based method is used, the average cost is 233.70. When the greedy method is used, the average cost is 235.36. When the clustering-based method is used, the average cost is 214.37. The cost increased by about 0.85% and 8.15% in greedy and clustering methods.

Fig. 7(d) shows the cost of assignment for different number of controllers in T_2. We keep the same settings as the previous simulation. We also observe a higher overall cost in T_2 than T_1. For all assignment methods, the cost decreases slightly with the increase of number of the controllers. We also observe similar behavior as in T_1. When the number of controllers is 2 and distance-based method is used, the average cost is 322.72. When the greedy method is used, the average cost is 317.95. When the clustering-based method is used, the average cost is 296.22. The cost decreased by about 1.5% and 8% in greedy and clustering methods. When the number of controllers is 10 and distance-based method is used, the average cost is 317.12. When the greedy method is used, the average cost is 281.97. When the clustering-based method is used, the average cost is 259.08. The cost increased by about 11% and 18% in greedy and clustering methods.

Next, we observe cost of controller assignment by using the incremental switch assignment. Fig. 7(e) shows the cost of assignment for different number of controllers in T_1. We vary the number of flows from 20 to 100 and keep the number of controllers at 5. We first assign the switches by using the clustering-based method, then we randomly delete 10 (or 20) flows and add 10 (or 20) flows. For all changes, the cost increases with the increase of number of the flows. When the number of flows is 20, we observe similar cost for all changes. When the number of flows is 100, the cost decreases by about 10% and 24% for 10 and 20 flow changes, respectively. We also observe similar behavior in T_2 as shown in 7(f). When the number of flows is 20, we observe similar cost for all changes. When the number of flows is 100, the cost decreases by about 8% and 23% for 10 and 20 flow changes, respectively.

Figs. 7(g) and 7(h) show the cost of assignment for different

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.

number of controllers in T_1. We vary the number of controllers from 2 to 10 and keep the number of flows at 50. For all changes, the cost decreases slightly with the increase of the number of controllers. For all changes, the cost decreases slightly with the increase of the number of controllers. When the number of controllers is 2, we observe that the cost decreases by about 16% and 35% for 10 and 20 flow changes, respectively. We also observe similar behavior in T_2. When the number of controllers is 10, we observe similar cost difference as 2 controllers. Therefore, from the above simulations we can conclude that the clustering based approach works better than the other two approaches.

VII. Conclusion

Controller load balancing for large-scale SDN datacenters is very important for ensuring good performance. To balance the load of the controllers, we need to migrate SDN switches from overloaded controller to relatively lightly loaded controller. Currently, switch migration functionality is not available in commercially available switches. Therefore, we investigate the possibility of switch migration by accessing the system using ssh. We have formulated two problems for switch-controller assignments that consider initial deployment and incremental modification for load balancing. We proposed several solutions to the problems. Our extensive simulations by using parameters driven from experiments, show great support for the solutions.

Acknowledgement

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128.

References

[1] J.-P. Sheu and Y.-C. Chuo, "Wildcard rules caching and cache replacement algorithms in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 19–29, 2016.

[2] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic sdn controller assignment in data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.

[3] V. Sridharan, M. Gurusamy, and T. Truong-Huu, "On multiple controller mapping in software defined networks with resilience constraints," *IEEE Communications Letters*, vol. 21, no. 8, pp. 1763–1766, 2017.

[4] Y. Xu, M. Cello, I.-C. Wang, A. Walid, G. Wilfong, C. H.-P. Wen, M. Marchese, and H. J. Chao, "Dynamic switch migration in distributed software-defined networks to achieve controller load balance," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 515–529, 2019.

[5] C. Wang, B. Hu, S. Chen, D. Li, and B. Liu, "A switch migration-based decision-making scheme for balancing load in sdn," *IEEE Access*, vol. 5, pp. 4537–4544, 2017.

[6] G. Cheng, H. Chen, H. Hu, and J. Lan, "Dynamic switch migration towards a scalable sdn control plane," *Int. J. Commun. Syst.*, vol. 29, no. 9, p. 1482–1499, Jun. 2016.

[7] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management*, 2013, pp. 18–25.

[8] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, 2017.

[9] F. He and E. Oki, "Load balancing model against multiple controller failures in software defined networks," in *ICC 2020-2020 IEEE International Conference on Communications*, 2020, pp. 1–6.

[10] X. Zhang, L. Li, and C.-b. Yan, "Robust controller placement based on load balancing in software defined networks," in *2020 IEEE International Conference on Networking, Sensing and Control*, pp. 1–6.

[11] L. Li, N. Du, H. Liu, R. Zhang, and C. Yan, "Towards robust controller placement in software-defined networks against links failure," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management*, 2019, pp. 216–223.

[12] C. Liang, R. Kawashima, and H. Matsuo, "Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers," in *2014 Second International Symposium on Computing and Networking*, 2014, pp. 171–177.

[13] G. Cheng, H. Chen, Z. Wang, and S. Chen, "Dha: Distributed decisions on the switch migration toward a scalable sdn control plane," in *2015 IFIP Networking Conference*, 2015, pp. 1–9.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *9th USENIX Symposium on Operating Systems Design and Implementation*. Vancouver, BC: USENIX Association, Oct. 2010.

[15] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow." USA: USENIX Association, 2010, p. 3.

[16] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: Association for Computing Machinery, 2012, p. 19–24.

[17] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[18] J. C. Mogul and P. Congdon, "Hey, you darned counters! get off my asic!" in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: Association for Computing Machinery, 2012, p. 25–30.

[19] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Apr. 2012.

[20] D. Erickson, "The beacon openflow controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: Association for Computing Machinery, 2013, p. 13–18.

[21] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*. New York, NY, USA: Association for Computing Machinery, 2011, p. 254–265.

[22] R. Biswas and J. Wu, "Traffic engineering to minimize the number of rules in sdn datacenters," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2021.

Authorized licensed use limited to: Temple University. Downloaded on January 26,2023 at 21:57:29 UTC from IEEE Xplore. Restrictions apply.