On the Transition of Legacy Networks to SDN - An Analysis on the Impact of Deployment Time, Number, and Location of Controllers

Diogo Ferreira Thé Pontes
Department of Computer Science
University of Brasília (UnB)
Brasília, Brazil
diogo.the@aluno.unb.br

Marcos Fagundes Caetano
Department of Computer Science
University of Brasília (UnB)
Brasília, Brazil
mfcaetano@unb.br

Geraldo Pereira Rocha Filho
Department of Computer Science
University of Brasília (UnB)
Brasília, Brazil
geraldof@unb.br

Lisandro Zambenedetti Granville
Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
granville@inf.ufrgs.br

Marcelo Antonio Marotta
Department of Computer Science
University of Brasília (UnB)
Brasília, Brazil
marcelo.marotta@unb.br

***Abstract*—SDN has emerged as an alternative networking paradigm separating the control plane from the data plane. In real-world scenarios, the transition from a traditional network to an SDN one is usually done in incremental steps, due to the costs and bureaucracy of new hardware's acquisition and deployment in companies. This transition involves the deployment of SDN-ready switches and hardware to run the software responsible for managing the control plane, i.e., the controller. Many studies on how to properly place and reduce the number of controllers, known as the Controller Placement Problem (CPP), have been conducted, but only a few of them exploited the CPP on hybrid (traditional/SDN) networks. In hybrid networks, we identified a gap in the minimum number of controllers required for a hybrid SDN to operate, considering a different number of transition steps and the heuristic to place these controllers within the network. Therefore, in this paper, we propose three heuristic approaches based on graph invariant for choosing the node to place a new controller reducing its number on a network that starts legacy, becomes hybrid, and ends fully SDN-enabled at the end of a finite transition step horizon. The proposed heuristics are compared against an optimized model and have been proven useful in scenarios where the network administrator does not have full control over the network evolution. Also, we found that the optimal multi-step network upgrade requires, on average, the same amount of controllers of a one-step migration, but it has to have full knowledge on each transition change of the hybrid network. Finally, we show that the performance of the heuristic approaches is related to the network topology and characteristics, such as traffic and physical switches distribution.**

***Index Terms*—controller placement problem, software-defined networking, network planning**

I. Introduction

Software-Defined Networking (SDN) provides a logically centralized network provisioning, programmability, and low cost of operations, emerging as an alternative to meet today's stringent network requirements [1] [2]. SDN requires a new type of software component usually deployed in dedicated processing hosts: the controllers. These controllers are responsible for managing network flows and providing the network state bringing benefits such as optimized routing and enhanced network performance [3], contributing to meet costumers' Quality of Experience (QoE).

The deployment of SDN in an organization is a process that does not happen at once. Instead, the legacy network gradually evolves towards a fully-capable SDN [4]. In the meantime, several intermediate versions of a hybrid legacy/SDN network exist. In this evolution, each new version of the network evolves from the previous version influenced by diverse factors, such as budget, device capabilities, network policies, management strategies, and administrative domains. In this paper, we are interested in observing the gradual adoption of SDN in networks operated by different and independent management teams. In such a network, each team can take SDN deployment decisions independently from each other. For example, this happens at some universities, where each department or institute has the flexibility to make local improvements in its internal network with its own budget. In the enterprise world, this can be observed on a company where each branch controls when to upgrade their subnetwork that connects to the headquarters.

The SDN controller [5] is a critical component of an SDN network. Naturally, the SDN design's decisions regarding this matter have a direct impact on network performance, load, and costs [6]. Considering this, we highlight three cornerstone decisions about SDN controllers: (*i*) the minimum amount of installed controllers to reduce costs without harming the network performance; (*ii*) the controllers location, taking into account the network demand; and (*iii*) when a new controller should be deployed, following the network evolution timeline [7]. Deploying an SDN considering these design decisions is not simple, mainly when it should consider them all together, leading to the so-called Controller Placement Problem (CPP). Decisions (*i*) and (*ii*) have been extensively addressed in the literature. Decision (*iii*), however, is largely overlooked, but we argued that an organization cannot properly deploy SDN ignoring an evolutionary approach. Following an instantaneous deployment, as usually assumed in the literature, is not realistic. There are many studies considering the Controller Placement Problem [8]. However, the majority focus to tackle CPP as a single-objective optimization problem, working with features like cost, resilience, and latency [9]. Others try to model hybrid networks using hardware that is not produced yet [10]. Nevertheless, decisions regarding the minimum amount, the best location, and timing to place a controller have been analyzed separately without considering their relationship. To the best of our knowledge, this is the first analysis of the CPP addressing the relationship among these mentioned cornerstone decisions in a hybrid network.

In this paper, we address the controller placement problem's issue, considering the minimal number of controllers required to transition a traditional network to a hybrid SDN until it becomes a fully deployed SDN. Our solution makes decisions regarding the time, placement, and minimal amount of SDN controllers required to meet latency and load constraints. The main contributions of this paper are listed as follow:

- A characterization of the controller placement problem regarding timing, number, and location of controllers in Hybrid SDN;
- An analytical model to evaluate the tradeoff limits related to the number, location, and timing to deploy controllers in a Hybrid SDN; and
- An experimental analysis of the tradeoff of timing, number, and controllers' location in Hybrid SDN.

The remaining of this paper is organized as follows. In Sec. II, we summarize and discuss related work. In Sec. III, we present our system model. In Sec. IV, we define our optimization model. In Sec. V, an experimental model is presented. Afterward, in Sec. VI, we present and discuss results from our experiments. Finally, in Sec. VII, we conclude our work by presenting conclusions and future work.

II. Related Work

This section presents the most relevant related works to address the controller placement problem's issue. Amir Hossein Fakhteh, Vahid Sattari-Naeini, and Hamid Reza Naji [11] work focus on an algorithm for migrating from a traditional to a software defined network, focusing on maximizing network control ability and flexibility while minimizing the cost of this transition. The authors use closeness-centrality and neighbor eccentricity invariant from graph theory to heuristically select the best nodes to upgrade, then place the controller based on the cluster's position weighted by the number of upgraded switches in the clusters. Like our work, in [11] exploits the CPP on hybrid SDN but they only deal with one controller's location, and it assumes the network transition is predictable and controlled by the network manager. Differently, our work deals with multiple controllers on networks where the manager does not have centralized control over the switches upgrade, creating randomized scenarios to evaluate the average case. Since the main feature of an SDN is the capability to reprogram the whole control plane according to the manager's own will at any time, we aim for the minimization of the number of controllers that produce long-term improvement optimizing the network planning instead of its performance.

Levente Csikor *et al.* [12] virtualizes the SDN switches, developing new hardware coupled to legacy switches to make them SDN. This solution proved to cost less and have better data plane performance for small businesses compared to existing SDN deployment, but it might not be a definitive long-term solution. It provides one way to upgrade a legacy to an SDN network. Our work aims towards a fully SDN as the final result with higher biasing on the network's topology despite small to large businesses. Also, our results are generalized to achieve the average case not specific to a single-use case.

Damu Ding *et al.* [13] proposes a solution for composing and monitoring a network with legacy and SDN switches. The authors aim to incrementally deploy SDN switches in an ISP network to maximize the number of distinct flows monitored. To achieve this objective, the authors proposed an algorithm and a novel network-wide heavy-hitter detection that worked well on networks with few legacy switches. Both the heavy-hitter detection and the incremental deployment algorithm outperformed existing approaches considering the network update's hierarchical plan. Similarly, we define a finite horizon of incremental discrete steps to model our transitioning from legacy to full deployed SDN network. Nevertheless, we do a step further by characterizing the tradeoff of placement, number, and time to deploy SDN controllers for the average case, randomizing the legacy switches that will become SDN enabled.

Tamal Das and Mohan Gurusamy [7], [9] have two works that introduce CPP to add controllers incrementally in a network over a period of time. In [7], they evaluate their solution in terms of various performance metrics, such as the number of controllers, cost savings, and latency. The proposed model focuses on optimizing placement costs and switch-controller latency, considering rising network traffic and falling costs of SDN hardware and maintenance. This is done by first minimize the control placement cost, assuming a latency threshold, and then, with the output of the first stage, they reduce the worst-case control latency. In [9], they formulate CPP on hybrid SDN, aiming to maximize the control channel resilience. Unlike this work, we consider optimizing controller-switch latency an odd objective, given that the network performance is related to demand, and there are no guarantees we're minimizing the signaling round-trip-time between controller and switches to enhance network operation. Thus, we evaluate CPP to the average case of a given topology modeling the controller-switch latency as a constraint.

Considering the literature presented, most of the work focus on optimizing one objective, such as latency or resilience, but they do not discuss the tradeoffs involving time, number and location of controllers creating a research gap: quantify the tradeoff involving timing, number and location of controllers in the transition of a legacy network to SDN taking into account switch-controller latency and controller load constraints. This paper covers this gap, considering the average case where the network evolution is not predictable. This result is then used as a baseline to compare three heuristics that we propose for choosing the controller located in the same hybrid SDN scenario, as presented in the next sections.

III. System Model

This section presents the system model and further definitions for the controller placement problem used in this work. Table I gathers all the notation used in this section. Our model starts through the legacy network definition, which is represented by the set of $\mathcal{N} = \{1, 2, ..., N\}$ nodes, *i.e.,* switches. Each pair of nodes (n, m) constitutes a connected edge when $\{(n, m) \in \mathcal{N} \mid n \neq m;\ \varepsilon(n, m) = 1\}$, where $\varepsilon(n, m)$ stands for the edge function and assumes the value of 1 only when both nodes n and m have a physical connection between them, and 0 otherwise. The edge function can be iterated considering all nodes' permutation $\mathcal{N}$ to generate the binary matrix of the set edges $V = \{0, 1\}_{N\times N}$. Finally, a network topology can be defined by the directed graph $G(\mathcal{N}, V)$.

The upgrade of a legacy network $G(\mathcal{N}, V)$ to SDN enabled is accomplished in a finite horizon of $\mathcal{T} = \{1, 2, ..., T\}$ discrete transitions $t :\rightarrow \mathbb{N}$. Each transition step $t \in \mathcal{T}$ is determined as an upgrade of at least one or more legacy nodes to SDN. A binary matrix $X = \{\forall t \in \mathcal{T}, \forall n \in \mathcal{N} \mid X_{tn} \in \{0, 1\}\}$ is used to track which nodes are becoming SDN, where X_{tn} is only equal to 1 when node n at transition step t is SDN enabled. After a node n becomes SDN enabled, it will never turn back into legacy equipment again. Hence $X_{(t+1)n} = \{1 \mid X_{tn} = 1)\}$ property holds for any configuration of X.

TABLE I: Notation used in the paper.

Notation	Parameter
$G(\mathcal{N}, V)$	Network graph
σ	Maximum load in one controller
K_n	Load (number of packets) of node n
L	Matrix of latencies from the set of nodes $\mathcal{N}$
L_{nm}	Shortest path latency between n and m nodes
δ	Maximum switch-controller latency threshold
$\mathcal{N}$	Set of nodes
$\mathcal{T}$	Finite horizon of transition steps
T	Cardinal of the set of transition steps
V	Set of edges
X	Binary matrix
X_{tn}	Defines if a node n is SDN at transition step t
$\varepsilon(n, m)$	Connection representation between n and m nodes

Both legacy and SDN enabled nodes receive traffic as a networking load, represented by the vector $K = \{\forall n \in \mathcal{N} \mid K_n \in \mathbf{Z}^+\}$. Each element K_n is a positive integer value of absolute packets number received at node n. Since the control of legacy equipment is held locally, only SDN enabled nodes generate signaling workloads. Although the signaling workload should be presented in $packets\_in$, available data sets from the literature give only network workload values. To tackle this issue, we are using the representation of the signaling workload as a direct function of the $packets$ received per node as $f(K_n) \in \mathbf{Z}^+\ packet\_in$. A deployed controller will process a maximum workload of $\sigma \in \mathbf{Z}^+\ packet\_in$ summed from all assigned SDN nodes.

The network defined in $G(\mathcal{N}, V)$ has its latency denoted by the matrix $L = \{\forall(n, m) \in \mathcal{N} | L_{nm} \in \mathbb{R}^+\}$, where each element L_{nm} is equal to a positive real number representing the aggregated latency resulting from the shortest path problem solution between nodes n and m, otherwise 0 when $n = m$. Also, all SDN control signaling communication latency must be kept below a threshold of δ to meet QoS requirements, referred to as maximum switch-controller propagation latency [8].

The system model described was designed to represent most real network scenarios with general topologies, considering key performance indicators, such as traffic and latency. Moreover, a finite horizon comprising transition steps was also introduced to map a legacy network's upgrade into a fully SDN-enabled network. The proposed model can also represent most of the literature network data sets, such as those considered in our work, NSFNet, GEANT2, and SYNTH50 [14]. Next, considering the system model proposed, we define the CPP problem.

IV. Problem definition

In this section, we describe the CPP problem definition. We start by defining the main decision variables, followed by the constraints and objectives of a CPP. Finally, a Binary Integer Linear Problem (BILP) definition is proposed discussing differences to the models found in the literature.

The main objective of the CPP is to minimize the number of controllers. Our model's different aspect is to reduce the number of controllers considering the transitioning of a legacy network until it becomes full SDN enabled. As a consequence, decisions regarding the controller placement are described by the binary variables y_{tn}, where

$$y_{tn} = \begin{cases} 1; & Controller\ in\ node\ n\ is\ in\ transition\ step\ t; \\ 0; & Otherwise. \end{cases}$$

Similarly, every assignment decision of an SDN switch to one controller can be mapped using binary variables z_{tnm}, where

$$z_{tnm} = \begin{cases} 1; & Node\ n\ assigned\ to\ controller\ m\ in\ step\ t; \\ 0; & Otherwise. \end{cases}$$

Considering each binary variable at step t, we can decide when a node n starts to host an SDN controller with y_{tn}, and which nodes m it is serving with z_{tnm}. Each decision must be made considering different constraints, such as presented below.

1) Constraints:

- The controller needs either to run in new hardware or be virtualized in existing hardware. Hence it is more convenient, due to existing infrastructure, to place a controller where we already have a switch. Thus, this constraint allows a controller to be placed ($y_{tn} = 1$) only if node n at step t is SDN ($X_{tn} = 1$).

$$y_{tn} \leq X_{tn}, \forall n \in \mathcal{N}, \forall t \in \mathcal{T}. \quad (1)$$

- The SDN switches send $packet\_in$ messages to the controllers every time a packet arrives with an unknown header, *i.e.,* with fields that do not fit in any forwarding rule already installed. Therefore, the latency (L_{nm}) in seconds between controller at the node m and an SDN node n, when assigned ($z_{tnm} = 1$) must be below a delay budget of δ seconds to keep QoS requirements.

$$z_{tnm} \times L_{nm} \leq \delta, \forall t \in \mathcal{T}, \forall n \in \mathcal{N}, \forall m \in \mathcal{N}. \quad (2)$$

- Every SDN switch requires to be assigned to a controller to operate, receiving forwarding rules. For the t^{th} transition step, an SDN enabled ($X_{tn} = 1$) node n must be assigned to one and only one controller m ($\sum_{m=1}^{\mathcal{N}} z_{tnm} = 1$).

$$\sum_{m=1}^{\mathcal{N}} z_{tnm} = X_{tn}, \forall t \in \mathcal{T}, \forall n \in \mathcal{N}. \quad (3)$$

- Since the latency for the paired SDN node and the controller will be negligible, it is natural to enforce that the hosting switch is connected to the hosted controller. Therefore, the node n hosting a controller ($y_{tn} = 1$) at step t will always be assigned to its hosted controller ($z_{tnn} = 1$).

$$z_{tnn} = y_{tn}, \forall t \in \mathcal{T}, \forall n \in \mathcal{N}. \quad (4)$$

- An SDN node must be assigned to a controller that must exist in another node. Thus, this constraint ensures that an SDN switch n will only be assigned ($z_{tnm} = 1$) to a node with a controller ($y_{tn} = 1$) at step t.

$$y_{tn} - z_{tnm} \leq 1, \forall t \in \mathcal{T}, \forall n \in \mathcal{N}, \forall m \in \mathcal{N}. \quad (5)$$

- Due to hardware limitations, controllers can only handle a certain workload of $packet\_in$ messages per second. This constraint ensures that the summation of $packet\_in$ received (K_n) from all assigned nodes ($\sum_{n=1}^{N} z_{tnm}$) must be lesser equal to the controller workload capacity σ.

$$\sum_{n=1}^{N} z_{tnm} \times K_n \leq \sigma, \forall t \in \mathcal{T}, \forall n \in \mathcal{N}. \quad (6)$$

- Buying, placing, or replacing a controller generates monetary cost. In our model, once a controller is placed ($y_{tn} = 1$), it can not be removed or replaced, and it stays in the same spot until the end of all future transition steps ($y_{(t+1)n} = 1$).

$$-1 \leq y_{tn} - y_{(t+1)n} \leq 0, \forall n \in \mathcal{N}, \forall t \in \mathcal{T} \mid t < T. \quad (7)$$

Constraint (2) deals with the location aspect in our model. Considering the decision variables and constraints, the objective function of the CPP can be defined as a minimization of the number of controllers, as defined by the Equation 8.

2) CPP optimization problem::

$$\min \sum_{t=1}^{T} \sum_{n=1}^{N} y_{tn} \quad (8)$$

s.t.

$$(1), (2), (3), (4), (5), (6), \ and \ (7).$$

In Equation (8), the installation of a new controller is characterized by the first occurrence of the variable y_{tn}, *i.e.,* for a given node n, it is the smallest value that t can assume, where $y_{tn} = 1$. This is how we deal with the time part. Due to the propagation effect caused by constraint (7) that turns all $y_{(t+1)n} = 1$ until $t = T - 1$ from its first occurrence, the minimization can be generalized to the occurrence of all variables y_{tn} throughout all transition steps. Thus, our objective is the generalized minimization of controllers' occurrence for each SDN node in $G(\mathcal{N}, V)$. As the proposed model is a Binary Integer Linear Problem (BILP), it reduces to a combinatorial problem with non-polynomial complexity. Although combinatorial, our problem presents a linear objective and constraints, and it's the solution can be found with linear-programming solvers, such as C-PLEX[1] or OR-Tools[2].

To solve the optimization problem proposed, it must know all information available before the start. It means the number of transition steps is finite and known. It is also known which nodes will be upgraded to SDN at each stage. However, in unplanned, uncontrolled, or real scenario deployments, these statements cannot behold. For instance, in a University environment, different departments may choose the number and the legacy nodes that will be upgraded without prior notification to the Network Operation Center (NOC). Thus, NOC managers may not have all the required information to solve this optimization problem. Thereby, they will not be able to plan how optimally distribute and install new controllers at the University infrastructure. Consequently, the network managers will have to assume a sub-optimal strategy or some policy to deploy new controllers for the whole University. They will follow this approach in order to keep some control level of the academic network flows, despite the different department upgrade decisions.

V. Proposal

Solving the optimization problem proposed turns feasible to identify the minimum number, the placement, and at which transition step controllers must be installed for a given topology. However, this approach turns infeasible for uncontrolled scenarios, where the steps of the transitions occur without previous planning. In this case, we propose three heuristic solutions for choosing where to place a new controller, each one based on a different policy considering a graph invariant. These controller placement policies are:

- Most Connected Node (MCN);
- Highest Eccentricity Node (HEN); and

[1]C-PLEX-IBM Inc.-https://www.ibm.com/br-pt/analytics/cplex-optimizer
[2]OR-Tools - Google Inc. - https://developers.google.com/optimization

- Highest Load Node (HLN).

In the MCN policy, the node with the highest degree available is selected. This selection aims to decrease the average hop number between placed controllers and nodes, considering the intuition from [15]. Unlike the previous policy, HEN selects the node with the highest eccentricity, *i.e.,* this policy maximizes the distance from the chosen node to all others in the topology. Lastly, the HLN policy selects a node based on its load ($packet\_in/s$). Since the load is a direct function of the switch traffic experienced, placing a new controller on the highest load node will decrease signaling traffic and lower switch-controller latency.

Before implementing each policy, we propose Algorithm 1 that presents the Controller Location Mapping (CLM) algorithm. This algorithm is responsible for identifying if a new controller is required or not. This decision is based on the arrival of new SDN nodes, the network load and the switch-controller latency restrictions. CLM is also responsible for calling the Controller Node Chooser (CNC) algorithm (line 3) presented in Algorithm 2. The CNC will select an SDN node to install a new controller, based on one of the three available policies.

Algorithm 1: Controller Location Mapping (CLM)

```
input  : $\mathcal{N}$, $t$ and $policy$
output: A list $l$ of controllers location mapped for
        current $t$
1  map_switches($t$);
2  while any SDN switch is not mapped and limit_count
     $< \mathcal{N}$ do
3      $node$ = controller_node_chooser($\mathcal{N}$, $t$, $policy$);
4      insert_controller($node$, $t$);
5      map_switches($t$);
6  end
7  if any SDN node is not mapped then
8      stop;
9  end
10 return $l$;
```

The CLM algorithm receives as input the set of nodes $\mathcal{N}$, the current transition step t, and the $policy$ in use. The $policy$ value can be one of the defined selection heuristics as MCN, HEN, and HLN policies. Since we defined that a new step t means an upgrade of one or more switches, every execution of Algorithm 1 starts by calling the $map\_switches()$ function (line 1). This function re-configures the switch-controller assignment mapping for the current step t. Since the load on a controller depends only on the switches mapped to it, re-mapping switches process to the controllers located in different places does not affect the controller load, only the switch-controller latency. In case one or more SDN switches remain without being assigned, due to latency or load constraints, the algorithm enters into the while loop (line 2). Thus, CLM will select a node, based on the current running $policy$ (line 3), to place the controller (line 4), until all SDN switches are assigned. The mapping process is updated considering a new controller placed (line 5) or new SDN switches (line 1). At the end of execution, CLM returns all controllers' place and the switch-controller mapping for the current step (line 10). If there are SDN switches not assigned to any controller at the end of execution, the problem is infeasible.

Algorithm 2: Controller Node Chooser (CNC)

```
input  : $\mathcal{N}$, $t$ and $policy$
output: Chosen node
1  for $n \leftarrow 1$ to $\mathcal{N}$ do
2      if (n is SDN on t) and (n does not have a
         controller placed on it) then
3          if $metric(n, policy) > highest\_value$ then
4              $chosen\_node$ = $n$;
5              $highest\_value$ = $metric(n, policy)$;
6          end
7      end
8  end
9  return $chosen\_node$;
```

Algorithm 2 presents the CNC implementation. It receives as input parameter the set of nodes $\mathcal{N}$, the current transition steps t, and the used $policy$. Based on the $policy$ and the current transition step t (line 2), the CNC iterates over all SDN nodes (line 1), considering for it the SDN nodes without a controller installed on it (line 2). For each node, CNC will check if its node position maximizes the $policy$ metric value (line 3). At the end of the execution, the algorithm will return (line 9) the SDN node which maximizes the evaluated $policy$ metric value (lines 3 to 5).

All proposed algorithms were implemented using the Python programming language. This language allowed us to use a specific and optimized library for graph analysis, called NetworkX[3]. NetworkX was used to build the network graph topology and to compute graph operations, such as node degree and eccentricity. Also, the optimization problem was solved using OR-Tools from Google Inc. Next, we present the results gathered considering different data sets.

VI. Performance evaluation

To evaluate the optimization problem comparing to the selected heuristics, we used three data sets [14]: (*i*) NSFNet (14 nodes), (*ii*) GEANT2 (24 nodes), and (*iii*) SYNTH50 (50 nodes). The selected data sets topologies are illustrated in Figure 1. We conduct experiments considering the optimization and the three proposed heuristics with NSFNet and GEANT2. Afterwards, to test each heuristic in a more challenging environment, we selected a data set of $N = 50$ nodes, the SYNTH50 [14]. However, since SYNTH50 presents a high number of nodes and consequently an even higher number of variables to be considered, the optimization solver's execution time becomes impractical and their results are not shown.

The NSFNet data set was surveyed and detailed described in [16]. Since all the considered data sets follow the same

[3]NetworkX https://networkx.github.io

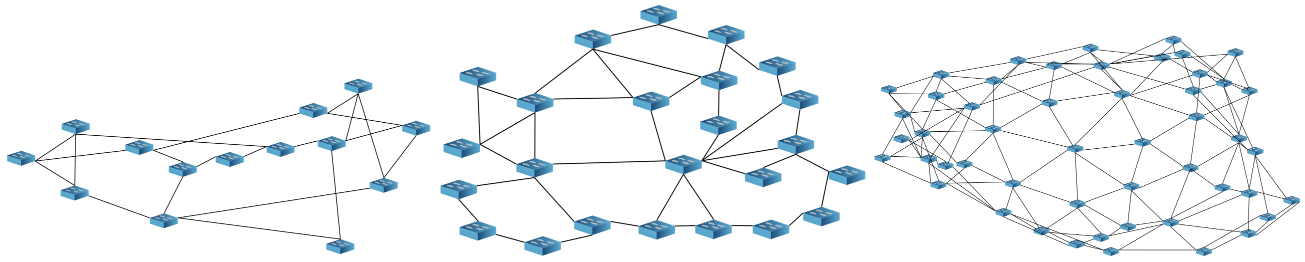

(a) NSFNet topology (b) GEANT2 topology (c) SYNTH-50 topology

Fig. 1: Data sets topologies

methodology, the mean latency experienced among nodes is averaged as 40ms, precisely in [16, Figure 7]. In this sense, we adopted the value for maximum switch-controller latency to the normalized value of 0.25, which corresponds to approximately 10ms in these scenarios. Also, we adopted a 20000 $packet\_in/s$ for the maximum controller capacity according to [17]–[20]. More details are presented in Table II.

TABLE II: Results scenario parameters.

Parameter	NSFNet	GEANT2	SYNTH50
$\mathcal{N}$	14	24	50
$\mathcal{T}$	{1, 5, 10}	{1, 5, 10, 15, 20}	{1, 5, 10, 15, 20}
L_{nm}	[0, 1.55684]	[0, 0.734737]	[0, 0.734347]
δ	0.25	0.25	0.25
K_n	[0, 14340]	[0, 9221]	[0, 16217]
σ	20000	20000	20000
X_{tn}	Random	Random	Random

As can be seen in Table II, each of the data sets presents a different topology and configuration. $\mathcal{N}$ represents the number of nodes. $\mathcal{T}$ is the set of transition steps. δ and σ are the maximum switch-controller latency and maximum controller workload constraints, respectively. L_{nm} and K_n are the range of the values for normalized latency and number of packets per second in the data set. X_{tn} is the randomly generated SDN switch transition for each step.

Throughout our experiments, we divided the network load in two categories: optimized load and heavy load. For the optimized load, we rely on the results of the overhead reduction strategy developed in [21] work, which characterizes the number of $packet\_in$ in terms of number of packets. We performed a linear interpolation on the graph in [21, Figure 8] resulting in a function we used to convert the number of packets in the data sets to $packet\_in$ messages presented in Figure 2, which we refer to as load. The heavy load is where one packet in the network corresponds to one $packet\_in$ in the controller. The load on a switch is the sum of the absolute number of packets transmitted from all adjacent switches to this switch.

For a generalization of the network changes, the upgrade plan is randomly created with one or more legacy switches becoming SDN every transition step until all become SDN-enabled. We ran 35 rounds of each experiment to achieve 95% confidence interval. To evaluate the impact of time, each experiment is evaluated for different transition step horizons spanning in the set $\mathcal{T}' = \{1, 5, 10, 15, 20\}$. Given that one step means at least one upgrade of a legacy node to SDN-enabled, we stop experimenting with the highest value that $t' \in \mathcal{T}'$ can assume that is smaller than the total number of nodes per data set. For example, NSFNet is a 14-nodes network data set, we executed the experiment until $t' = 10$.

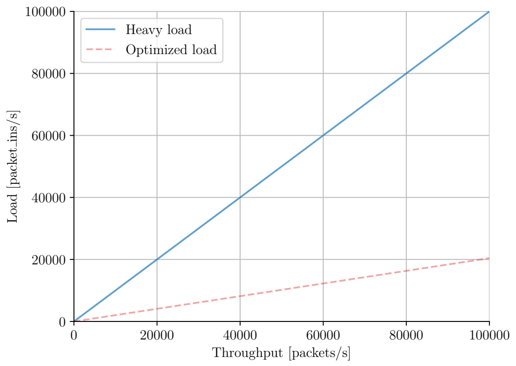


Fig. 2: Load [packet_in/s] as a function of the network throughput [packets/s]

We will first present the results of the experiments for the data sets. They are divided in optimized and heavy load scenario. Since the average controller load is complementary to the minimum number of controllers, we decided to omit them.

A. Optimized load results analysis

The main metric presented throughout our experiments is the average number of controllers at the end of the finite horizon of transition steps. In Figure 3, we have in the vertical axis, the average number of controllers being presented. In the horizontal axis, we have the cardinal value of different sets of transition steps T being presented. Each value of T contains a group of bars, one for each heuristic and the optimization solution. Since $T = 1$ means a transition from legacy to SDN in a single step, the X_{tn} variables are always 1 for every switch. In this case, the results for both optimized and heuristic approaches have no variations, for there is no insertion of uncertainty caused by the random steps present in other cases. Thus, there is no error bars on the first columns of every result being presented. Also, given that the heuristics are suboptimal, without readjustments, they can converge to suboptimal solutions that do not necessarily have the best result.

As expected, the proposed optimized model always requires the lowest amount of controllers and has a higher controller utilization reaching a concise average value of 4 controllers for the NSFNet displayed in Figure 3a. Compared to the optimization, the best heuristic for this use case is HLN, reaching the average value of 5.56 controllers, about two controllers more than the optimization. On the other hand, the worst heuristic to be considered is HEN, with 10 controllers required for the worst case and about 6.20 in the best case, about 3 controllers more than the optimization. The MCN policy presents a similar demeanor to the HEN heuristic.

Given that NSFNet is a small network of 14-nodes not fully connected, placing the controller at the most loaded node in HLN brings benefits regarding controllers processing workload locally. Whereas, using the highest eccentricity from HEN does not exploit the same benefit and increase the chance of latency restriction violation, such as presented in the range of L_{nm} in Table II. Another important point is that with the increasing of the number of transition steps, the three heuristics start to present similar results, enabling to use any of the policies considered for a network with the same topology and a large deployment time.

In Figure 3b, the GEANT2 minimum number of controllers is depicted by the optimization, presenting an average of 3.4 controllers in the worst case. The best policy for the GEANT2 network is the HEN heuristic, presenting seven controllers in the worst case and 5.4 in average at best. On the other hand, the worst policy is the HLN presenting 15 controllers at worst and achieving the same value of 9 controllers as the MCN for 20 transition steps.

As GEANT2 presents a highly connected topology, placing the controllers at the nodes with the highest eccentricity, such as in HEN, allows the assignment between controllers and SDN switches to be better executed reducing the number of hops between them. Nevertheless, a heuristic based on highest load, such as HLN, underperform in GEANT2 due to its highly connected topology reducing diameter among controllers, increasing the number of hops per assignment, and increasing the chance to violate latency restrictions. Also, it means that the load of an individual node is similar to the others in GEANT2, which is the opposite of what happens to the NSFNet, such as observed in Table II in K_n. To confirm such a statement, we execute experiments with the SYNTH50. Also, it is worth mentioning that the increase of transition steps reduces the difference from one heuristic to another, but differently from the NSFNet, the HEN policy for GEANT2 remains better despite the higher deployment time.

SYNTH50 is a highly connected topology with 50 nodes. As mentioned, the number of variables increases dramatically turning the optimization execution time infeasible. Thus, only the policies MCN, HEN and HLN have results considering this data set. As it can be seen, the HEN policy presents the best values among the three heuristics with 9 controllers required at worst. The MCN, in turn, presents the highest results requiring 33 controllers at worst, reducing accordingly to the increasing of the number of steps achieving the average of 19 controllers at best. HLN falls in between HEN and MCN, presenting an average of 14.2 controllers at best.

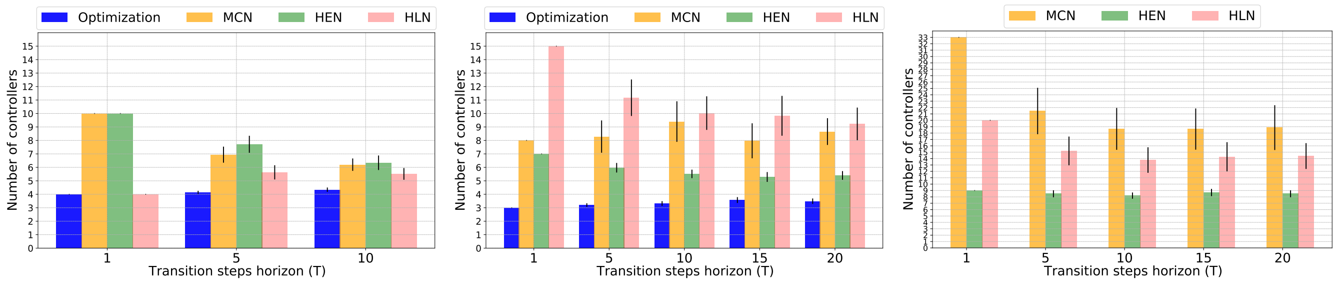
The results presented for SYNTH50 corroborate with the statement that a highly connected topology will require a smaller number of controllers when placing controllers in nodes with higher eccentricity. In this case, we can conclude that for optimized load scenarios, the HEN policy is recommended for highly connected topologies. Also, for lesser connected topologies, the HLN will present better results when planning to install new controllers in the network. Besides, a deployment time with several transition steps can balance the performance among the policies selected. However, highly connected topologies will still benefit with smaller number of controllers using the HEN policy.

B. Heavy load results analysis

Similar to the previous results, Figure 4 shows the results obtained regarding the average number of controllers for the heavy load scenario. In Figure 4a, regardless of the value of transition steps, the optimized model and HLN have the same number of controllers. MCN and HEN, in turn, have almost the same results. Since the NFSNet presents high traffic per node according to the range K_n in Table II, using the heavy load function that increases the load linearly to the traffic experienced, the average number of controllers are dominated by the load constraint. In this case, almost all switches of the network require to be installed with a controller, which is confirmed by the average value of 13 controllers at best by the optimization for a 14 nodes network, such as NFSNet.

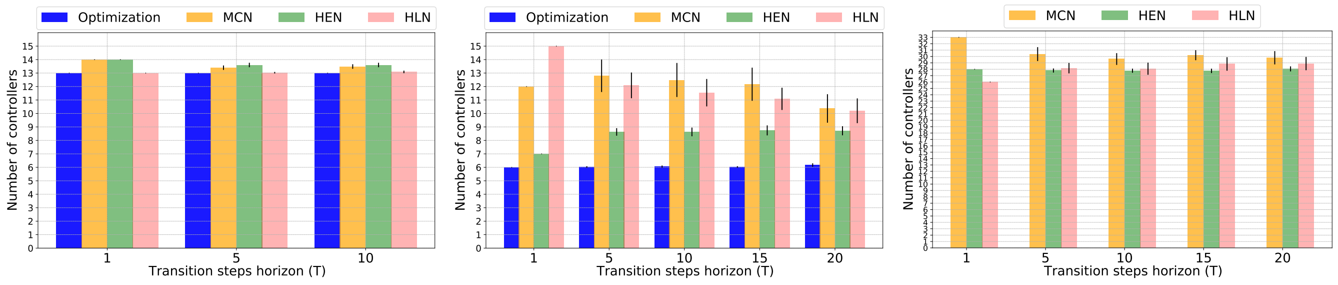
In Figure 4b, the average number of controllers of the GEANT2 topology is shown. Given that GEANT2 is highly connected with small traffic per node, its results are not dominated by the load constraint, such as occurred with the previous network. Even under heavy load, the best policy is the HEN with 8.5 controllers in the worst-case scenario. On the other hand, the policy that requires the highest number of controllers is HLN with 15 and 10 controllers for the worst (*i.e.,* , 1 transition step) and best case (*i.e.,* , 20 transition steps), respectively. MCN, in turn, performs a little better than HLN, but presents the same behavior through the transition steps. As previously analyzed for the optimized load scenario, the same occurs for the heavy load scenario, where the higher number of transitions serve to balance the heuristics results. However, HEN is still a better heuristic despite the number of transition steps, which is also confirmed by the variability analysis, where the confidence intervals are not overlapped.

The next analysis concerns SYNTH50 topology for the proposed heuristics, Figure 4c. The MCN policy resulted in the highest number of controllers, while HEN was the smallest with 28 controllers on average. The difference in the number of controllers between policies was more significant in the optimized load scenario from Figure 3 than in the heavy load displayed in Figure 4. Despite the SYNTH50 being a highly connected network, its high traffic per node also generated high load, making the average number of controllers



(a) NSFNet (b) GEANT2 (c) SYNTH50

Fig. 3: Number of controllers in scenarios under optimized load



(a) NSFNet (b) GEANT2 (c) SYNTH50

Fig. 4: Number of controllers in scenarios under heavy load

more sensible to the load constraint. This means that the results are almost the same despite the heuristic adopted or the number of transitions considered.

As a final observation, the heavy load scenarios depict a very poorly controlled network environment from which each data packet occurring creates a $packet\_in$ message to the controller. The signaling cost is the same as the data traffic which is rather very unlikely to occur in reality. At the same time, since we are not assessing nor optimizing the control feature of the SDN network, our results cover this kind of scenario for completeness.

VII. Conclusions and Future Work

In this paper, we proposed an optimization model to the Controller Placement Problem for the transitioning from a legacy network to hybrid until becomes fully deployed SDN. Given that the solution for the optimization problem requires full acknowledgment of each change of the transition of a network, we propose three alternative policies based on graph invariant with different heuristics to select the best candidate node to host a controller, namely (*i*) Most Connected Node; (*ii*) Highest Eccentricity Node; and (*iii*) Highest Load Node. The results showed that the optimal result has little impact throughout the transition from legacy to SDN. However, the policies results have a great influence of the total number of transitions enabling the number of controllers to be balanced at the end of a high number of transitions. Also, we detected that highly connected topologies can greatly reduce the number of controllers required when planned with the Highest Eccentricity Node policy. Finally, lesser connected topologies require lesser controllers when planned using the Highest Load Node policy for a small deployment time, *i.e.,* few transition steps.

Given this information, we recognize some limitations to the wide use of our model in real network scenarios. First, a controller cannot be migrated to another place once it is placed. We know that it might be cheaper to move a controller to another place instead of buying a new one. However, since we focus on a network operated by different and independent management teams, installing a controller in one department and then revoking it back might generate conflict between them. Second, we assume that the controller can be installed in the chosen place. Sometimes there is no infrastructure or budget to place and maintain this equipment in every switch location. Finally, network migration is random. Some companies can be able to trace a plan for upgrading their network to SDN. On the other hand, since we are considering networks managed by independent teams, the moment a department upgrades their switches depends on several variables, such as their needs, budget, infrastructure, and bureaucracy. So, we decided that a random switch migration would fit better in our study case.

As future work, we suggest exploring an optimization model on a centralized network, where the network migration plan is part of the output. Optimization with different objectives, such as resilience and load balancing, can also be explored. Also, the $map\_switches$ function from Algorithm 1 can be improved for optimized results in terms of load balancing.

References

[1] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implemen-

tation challenges for software-defined networks," *IEEE Communications Magazine*, pp. 36–43, 2013.

[2] J. N. Binlun, T. S. Chin, L. C. Kwang, Z. Yusoff, and R. Kaspin, "Challenges and Direction of Hybrid SDN Migration in ISP networks," in *2018 IEEE International Conference on Electronics and Communication Engineering, ICECE 2018*, 2019, pp. 60–64.

[3] M. A. Marotta, M. Kist, J. A. Wickboldt, L. Z. Granville, J. Rochol, and C. B. Both, "Design considerations for software-defined wireless networking in heterogeneous cloud radio access networks," *Springer Journal of Internet Services and Applications*, p. 18, 2018.

[4] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN Networks: A Survey of Existing Approaches," *IEEE Communications Surveys & Tutorials*, pp. 3259–3306, 2018.

[5] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, pp. 14–76, 2015.

[6] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "SDN Controller Placement at the Edge: Optimizing Delay and Overheads," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 684–692.

[7] T. Das and M. Gurusamy, "INCEPT: INcremental ControllEr PlacemenT in Software Defined Networks," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, vol. 2018-July, 2018, pp. 1–6.

[8] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2020.

[9] T. Das and M. Gurusamy, "Resilient Controller Placement in Hybrid SDN/Legacy Networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[10] H. Xu, H. Huang, S. Chen, G. Zhao, and L. Huang, "Achieving High Scalability Through Hybrid Switching in Software-Defined Networking," *IEEE/ACM Transactions on Networking*, pp. 618–632, 2018.

[11] A. H. Fakhteh, V. Sattari-Naeini, and H. R. Naji, "Increasing The Network Control Ability and Flexibility In Incremental Switch Deployment for Hybrid Software-Defined Networks," in *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, 2019, pp. 263–268.

[12] L. Csikor, M. Szalay, G. Retvari, G. Pongracz, D. P. Pezaros, and L. Toka, "Transition to SDN is HARMLESS: Hybrid Architecture for Migrating Legacy Ethernet Switches to SDN," *IEEE/ACM Transactions on Networking*, pp. 275–288, 2020.

[13] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An Incrementally-Deployable P4-Enabled Architecture for Network-Wide Heavy-Hitter Detection," *IEEE Transactions on Network and Service Management*, pp. 75–88, 2020.

[14] A. Cabellos. Knowledge-defined networking training datasets. [Online]. Available: https://knowledgedefinednetworking.org/

[15] S. ur Rahman, G. Kim, Y. Cho, and A. Khan, "Deployment of an sdn-based uav network: Controller placement and tradeoff between control overhead and delay," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, 2017, pp. 1290–1292.

[16] J.-J. Pedreno-Manresa, J.-L. Izquierdo-Zaragoza, and P. Pavon-Marino, "Joint fault tolerant and latency-aware design of multilayer optical networks," 2016, pp. 1–6.

[17] C. Tom, H. Yu, K. Li, and H. Qi, "An active controller selection scheme for minimizing packet-in processing latency in sdn," in *Security and Communication Networks*, 2019.

[18] L. Yao, P. Hong, and W. Zhou, "Evaluating the controller capacity in software defined networking," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, 2014, pp. 1–6.

[19] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "Sdn controller assignment and load balancing with minimum quota of processing capacity," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[20] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on sdn network throughput," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[21] A. A. Pranata, T. S. Jun, and D. S. Kim, "Overhead reduction scheme for sdn-based data center networks," *Computer Standards Interfaces*, pp. 1 – 15, 2019.