Revisiting the Controller Placement Problem

Md Tanvir Ishtaique ul Huque,[*,1,2] Guillaume Jourjon,[1] Vincent Gramoli[1,3]

Abstract: The controller placement problem (CPP) is one of the key challenges of software defined networks to increase performance. Given the locations of switches, CPP consists of choosing the controller locations that minimize the latency between switches and controllers. In its current form, however, CPP assumes a fixed traffic and no existing solutions adapt the placement to the load.

In this paper, we introduce the *dynamic controller placement problem* that consists of (i) determining the locations of controller modules to bound communication latencies, and of (ii) determining the number of controllers per module to support the load. We propose, LiDy, a solution that combines a controller module placement algorithm with a dynamic flow management algorithm. We evaluate the latency, the controller utilization, the power consumption and the maintenance cost of LiDy on sparse, normal and dense regions. Our results show that, in all settings, LiDy achieves a higher utilization at lower energy and maintenance costs than the most recent controller placement solution.

Keywords: SDN, distributed system, Network Management

[*]Corresponding author: `md.huque@nicta.com.au`
[1] Nicta, Sydney. Eveleigh, NSW, Australia, `first.last@nicta.com.au`
[2] University of New South Wales, Sydney, NSW, Australia
[3] University of Sydney, Sydney, NSW, Australia, `vincent.gramoli@sydney.edu.au`

ISSN: 1833-9646-8870

1 Introduction

Software Defined Network (SDN) decouples the network control (control plane) and forwarding functions (data plane). This decoupling enables network functions to become directly programmable as the underlying infrastructure becomes an abstraction for applications and network services [1]. Through these programmable functions, SDN helps reducing the network upgrading cost by up to 58.04% compared to the legacy technologies.[1] A major difference with traditional networks is that the SDN relies on centralized intelligence to control the network management. However, state-of-the-art SDN controllers, such as NOX-MT [4], can only process 1.6 Million flows per second (*Mfps*) with a median response time of 2 *msec* where in the case of a relatively large scale city such as Frankfurt, they would need to process 2.79 *Mfps* [5]. Hence, the scalability to a continent [6] naturally raises the problem of effectively exploiting multiple controllers.

Recently, Heller et al. defined the *controller placement problem (CPP)* that, given the locations of switches, consists of choosing controller locations so as to maximize performance while minimising the delay between switches and some controllers regardless of the network traffic variation [7]. As a consequence, all solutions to the CPP we are aware of [7, 9, 10, 11, 14, 15] are variants of solutions to the facility location problem, all assuming that the switches incur a fixed load. In real networks, however, the load applied to a controller depends on the number of the switches connected to it and the rate at which these switches encounter new flows [8]. It is thus necessary to consider two problems when setting up controllers, the controller placement problem placing controller "modules" to minimize communication latency, and the *controller scheduling problem* provisioning controllers at each of these controller modules to handle the varying load.

Our contribution is a combined solution, LiDy, for Location Independent controller module placement and DYnamic flow management. First, our solution takes a controller-switch latency constraint as an input and outputs the locations of controller modules so that a maximum number of switches can be operated from each selected location under the predefined latency constraint. Second, our solution adapts the number of controller at each controller module to maximize utilization as the traffic load[2] of switches varies: it deactivates controllers running at each module to minimize energy consumption and activates them to disentangle the control plane. We evaluate LiDy as a solution to a variant of the CPP [7] that takes into account the dynamic traffic load of switches, being the first initiative of this kind. To this end, we compare our solution to the most recent CPP solution namely YBLG [15]. YBLG also considers that the traffic load is fixed. We show that our solution utilizes a lower number of controllers to balances the same traffic load than YBLG. Overall, we have the following specific achievements:

1 **Controller utilization:** The controller utilization is measured by the number of switches (or flows) managed by a controller. Our results show that LiDy assigns more than twice as many switches per controller as YBLG. It also increases the number of flows per controller more than 100% compared to the YBLG.

2 **Power consumption:** Power consumption is reduced by more than 50% compared to the YBLG.

3 **Cost:** LiDy reduces the setup and maintenance cost to about 15% compared to the YBLG. This cost reduction rate increases as the number of switches in the SDN increases.

We also identify the potential drawbacks of state-of-the-art SDN controller placement methods, and present how the offered solution differs from the current approaches of the SDN controller placement problem.

The remainder of this paper is organized as follows: Section 2 presents the relevant research work showing the present research trend of the controller placement problem of the SDN. Section 3 clarifies the problem statement to give the benchmark of our motivation. Section 4 describes the working mechanisms of our proposed controller placement solution. Section 5 presents the system model, based on which the performances of proposed algorithms are evaluated. A comparative performance evaluation, between the proposed model and YBLG, is shown in Section 6. Section 7 focuses on the inequalities of the proposed

[1] *Dimension Data* a leading IT infrastructure solutions and services provider company, has recently disclosed a forecast that the SDN market is expected to extend over 35 billion USD within 2019 [3].

[2] In this paper, we refer to the traffic load of a switch as the number of new flows encountered by that switch

controller placement method compared to the available ones in the SDN. Finally, Section 8 ends with the conclusion and the direction for the future work.

2 Related Work

The controller placement problem of SDNs was first presented by Heller et al.[7] in 2012. This research was to estimate the minimum number of controllers and their placements in the SDN. Its result shows that a random placement of a controller yields five times worse latency than an optimal placement of that controller in the SDN. Followed by Heller et al.[7], during the last two years, a limited number of potential research works, [9, 10, 11, 12, 13, 14, 15], is found on this controller placement problem.

Hu et al.[9] present a result on the number of controllers and their placements in the SDN considering the reliability of the SDN control plane. This shows the trade off between the reliability and the latency. The authors agree that placing too many or too few controllers reduce the reliability. Both Heller et al.[7] and Hu et al.[9] consider the location of switches to get the optimal number of controllers and their corresponding locations. But they ignore the impact of inter-controllers effects. Hock et al.[10] first addresses this problem. They take into account the inter-controller latency and traffic load balancing for the optimal placement of controllers. Like Hu et al.[9], their primary concern is the reliability of the SDN control plane. In parallel, they also consider the network resiliency in terms of failure tolerance and traffic load balancing. But this technique is limited to the small and medium scale SDNs. To overcome this shortcoming, Hock et al.[10] extended their research work [11]. They propose a heuristic approach to find out the optimal number and locations of controllers for the large scale SDN.

All of the above mentioned research works, [7, 9, 10, 11], consider only the latency to get the number of controllers and their corresponding locations in the SDN. Besides latency, traffic load management among controllers is also considered by a number of research works, [12, 13, 14, 15], to get the number of controllers and their corresponding locations. Thus these research works, [12, 13, 14, 15], offer a better controller's placement compared to previous ones. Rath et al.[12] proposes a non-zero-sum game based dynamic controller placement technique. Here, a controller goes into sleep mode periodically based on the traffic demand. This technique only ensures the maximum utilization of controllers from a set of controllers in the SDN. But It does not define where to place the controllers in the SDN. By contrast, [13, 14, 15] defined the optimal placement of controllers in the SDN from a set of selected locations. Among them, Sallahi et al.[13] propose a mathematical model to define the optimal number and locations of controllers. They also consider heterogeneity of controllers and their interconnections. This model is however applicable only for the small scale ($1km \times 1km$) SDN. On the contrary, the controller placement models of Jimenez et al.[14] and Yao et al.[15] are not limited to the small scale SDN. Both of these research works consider the latency and the traffic load to estimate the number and the locations of controllers in the SDN. Their focus is to use a minimum number of controllers. The proposed model of Yao et al.[15] differs from the model of Jimenez et al.[14] based on the traffic load consideration manner. Jimenez et al.[14] considers that each switch has same traffic load, and the number of switches connected with a controller is considered as the load of that controller. whereas Yao et al.[15] considers unequal traffic load from switches to controller.

3 Problem Statement

From Section 2, we get different views to solve the controller placement problem of the SDN. The aforementioned solutions rely on a common simplification. Indeed, they estimate the location of a controller from a set of selected locations in SDN. These selected locations are the locations of switches in the SDN. Since this common approach selects the controller location from a set of fixed locations of a region, it may not result the most compatible locations of that region. We present in Figure 1 an illustrative example where four identical fully connected switches (A, B, C and D) are considered at four corners of a square region. Here the goal is to show the effect of the search of the controller locations, on the latency, the number of controllers and the network performance (traffic load of switches) of the SDN.

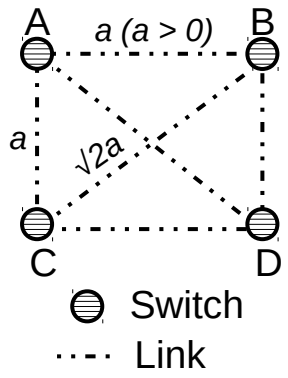


Figure 1: A software defined network with identical switches

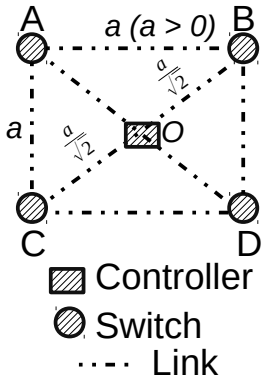


Figure 2: Effect of the controller placement on the latency of the SDN

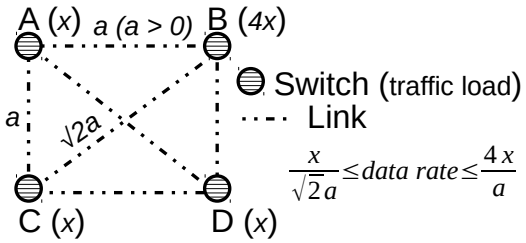


Figure 3: A software defined network consisting of switches having different traffic loads

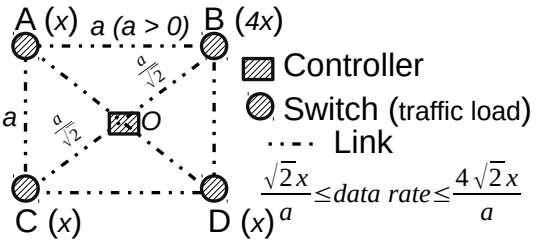


Figure 4: Effect of the controller placement on the traffic load of switches of the SDN

3.1 Effect of the controller placement on the latency

According to the common approach used by [7, 9, 10, 11, 12, 13, 14, 15], the optimal location of the controller is one of four locations of switches (A, B, C and D). For any of these locations the maximum latency, between the switches and the controller ($A \leftrightarrow D$ or $B \leftrightarrow C$), is $\sqrt{2}a$, shown in Fig. 1. But, if the controller is placed at location O, shown in Fig. 2, then the maximum latency becomes $\frac{a}{\sqrt{2}}$. This controller placement halves the latency compared to the placement in Figure 1. Thus instead of the fixed locations, if the entire region containing the switch's locations is searched, the location found further decreases the latency.

3.2 Effect of the controller placement on the number of controllers

Consider the same example from Figure 1 with the latency bound of $\frac{a}{\sqrt{2}}$. Now, according to the common controller placement approach of [7, 9, 10, 11, 12, 13, 14, 15], the controller placement is limited to the switch locations. Since the minimum latency between two consecutive switches is a, shown in Fig. 1, four controllers are required at four switch locations, to maintain the latency constraint. On the contrary, If the controller is placed at location O, shown in Figure 2, only one controller is required to maintain the latency constraint. It reduces the number of controller by four times. Thus instead of the fixed locations, if the entire region containing the switch's locations is searched, a limited number of controllers is used.

3.3 Effect of the controller placement on the traffic load

To illustrate the effect of controller placement on the traffic load of switches, the same example of Fig. 1 is considered. But, this time it is considered that the switches have different amount of traffic load (number of flows) to send, shown in Fig. 3. Here, switch A, B, C and D have x, $4x$, x and x number of flows, respectively. Now according to the common controller placement approach of [7, 9, 10, 11, 12, 13, 14, 15], if the controller is placed at one of four switch locations, then the maximum data rate (or flow rate) becomes $(\frac{flows}{latency} =)\frac{4x}{a}$. On the contrary, if the controller is placed at location O, shown in Figure 4,

then the maximum data rate becomes $\frac{4\sqrt{2}x}{a}$. This controller placement of Figure 4 doubles the data rate. Thus instead of the fixed locations, the entire region, containing the switch's locations, search, for placing the controller improves, the network performance (in term of data rate).

These observations motivated us to search the entire region, containing the switch's locations, to get the locations of controller, compared to the search of some selected locations of that region. To achieve this goal, we propose a controller placement solution, LiDy, in Section 4.

4 Proposed Controller Placement Solution

In this section, we propose a novel controller placement solution, LiDy, considering both the latency and the traffic load of switches, for the SDN. LiDy consists of two algorithms: the *Independent Controller Module Placement Algorithm* and the *Dynamic Flow Management Algorithm*.

4.1 Location Independent Controller Module Placement Algorithm

The proposed *Location Independent Controller Module Placement Algorithm* (LICMP) determines the locations of controller modules in the SDN. The controller module concept is presented in details in Section 4.2. LICMP works along with a searching algorithm, *Location Searching Algorithm* (LSA). It has two primary functions:

1 LICMP partitions the entire SDN into sub-regions based on a maximum latency bound (D_{req}).

2 For each sub-region, it finds out the location of the controller so that a maximum number of switches can be operated, and the maximum latency can be reduced.

LICMP starts with the border switch selection (step 2). Here, a border switch (A) is defined as a switch that has the highest coordinate among all available switches (S). LICMP considers a set (S_A) consisting of the border switch and all of its neighboring switches (step 3). From S_A, LICMP selects a switch that has the highest number of neighbors and its corresponding set, S_{select} (steps 4-8). Here, a switch, X, is a neighbor of another switch, Y, if the latency, $d(X, Y)$, between them is not greater than D_{req}. Then it calls LSA to estimate the location (K) of the sub-region (S_{select}) (step 9). Here, the sub-region is defined as a region that contains the switches of S_{select}. After getting K, LICMP searches again to add new switches in S_{select} (step 10). This sub-region formation and location selection process is continued until no switch is left.

Algorithm 1 Location Independent Controller Module placement Algorithm (LICMP)

Require: Set of all switches, S
Maximum latency, D_{req}
Ensure: Set of selected switches, S_{select}

1: **Initialize** $S_{select} \leftarrow \emptyset$, $t \leftarrow \emptyset$
2: Select border switch A: $A \in S$
3: $S_A \leftarrow$ Set of neighboring switches (x_i) of switch A
where for $\forall x_i : x_i \in S_A, S$, $d(x_i, A) \leq D_{req}$
4: **for** $i \leftarrow 1, |S_A|$ **do**
5: $S_(x_i) \leftarrow$ Set of neighboring switches (y_j) of switch x_i
where for $\forall y_j : y_j \in S_{x_i}$, $d(y_j, x_i) \leq D_{req}$
6: **end for**
7: **Update** $t \leftarrow t + 1$
8: $S_{select} \leftarrow S_{x_{max}} \in S_{x_i}$, $|S_{x_{max}}| = \max_{1 \leq i \leq |S_A|} |S_{x_i}|$
9: **Call LSA** Input: S_{select}, D_{req}, A
Output: K
10: **Update** S_{select}
where $\forall y_i : y_i \in S_{select}(t)$, $d(y_i, K) \leq D_{req}$
11: **Update** $S \leftarrow S - S_{select}(t)$
12: **if** $S_{select} \neq \emptyset$ **then**
13: $A \leftarrow B : B \in S, d(B, K) = \min_{N_i \in S} d(N_i, K)$
14: **goto** step - 3
15: **end if**

4.1.1 Location Searching Algorithm

LSA acts as a part of LICMP. It gets the location information of all switches of a sub-region (S_{select}) from LICMP. Then it yields the location (K) of the controller module of that sub-region. Its objective is to find out a location (K) within a sub-region to minimize the maximum latency between K and the switches. At the beginning, it forms the Delaunay triangles (S_T) considering each switch as a vertex of a triangle (steps 2-4). Then the centre of gravity (S_{CT}) of all of these triangles is estimated (step 5). Among all points of S_{CT}, the minimum one is defined as follows (step 6):

$$D_K(K,P) \leftarrow \min_{K_k \in S_{CT}} \max_{p_l \in S_p} d(K_k, p_l)$$

Here, D_K is the latency between the controller location, K, and the switch, P, of S_p. Now if D_K is not greater than the threshold latency ($D_{K_{pre}}$), then LSA continues the Delaunay triangle formation and location estimation process (steps 10-18). To stop this continuous process, a stopping criteria (D_{stop}) is used. D_{stop} is defined in (1).

$$D_{stop} = \frac{\min_{s,v \in S, s \neq v} d(v,s)}{n} \tag{1}$$

On the contrary, if D_K is greater than $D_{K_{pre}}$, then K will be replaced by K_{pre} (steps 7-9).

Algorithm 2 Location Searching Algorithm (LSA)

Require: Set of all switches, S_{select}
Maximum latency, D_{req}
Stopping criateria, D_{stop}
Border switch, A
Ensure: Location, K

1: **Initialize** $K \leftarrow \emptyset, D_K \leftarrow \emptyset, D_{K_{pre}} \leftarrow \emptyset,$
$K_{pre} \leftarrow k : k \in S_{select}, D_{req} \leftarrow d(k, A)$
2: $S_p \leftarrow$ set of switches, $p \in S_{select}$
3: $\forall p \in S_p$ form Delaunay triangles (T) using **Bowyer-Watson algorithm** [16]
4: $S_T \leftarrow$ set of Delaunay triangles
5: $S_{CT} \leftarrow$ set of gravity centres of Delaunay triangles
6: **Update** $D_K(K,P) \leftarrow \min_{K_k \in S_{CT}} \max_{p_l \in S_p} d(K_k, p_l)$
7: **if** $D_{K_{pre}} < D_K$ **then**
8: $K \leftarrow K_{pre}$
9: **else**
10: $P_{select} \leftarrow$set of vertices of Delaunay Triangles containing K
11: $K_{pre} \leftarrow K : K \in S_{CT}, D_K \leftarrow max_{p_l \in S_p} d(Kp_l)$
12: $S_p \leftarrow P_{select} \cup K$
13: **if** $D_K \geq D_{stop}$ **then**
14: $D_{K_{pre}} \leftarrow D_K$
15: **goto** step - 3
16: **end if**
17: **end if**

4.2 Dynamic Flow Management Algorithm

Instead of distributing a number of controllers in a SDN, we propose to use a limited number of controller modules. A controller module is a set of controllers. A many-core processor[17] can be used as a controller module. Each controller modules is placed in each sub-region in the SDN. Based on the new number of flows, generated by switches, a controller modules activates the required number of controllers. To manage this functionality, we have proposed *Dynamic Flow Management Algorithm* (DyFlow). The controller module runs DyFlow periodically to manage and update its functionality. DyFlow considers the heterogeneity of controllers of a controller module. Heterogeneity of controllers means different number of flow execution capability. To execute DyFlow, some preliminary system information of a controller module is required. The inputs of DyFlow are listed in Table 1; and its outcome is a set of active controllers ($C_{selected}$).

Table 1: List of inputs of DyFlow

Symbol	Definition
S	Set of all switches managed by a *Controller Module*
$w_{switch,i}$	Switch i's workability factor; shows that either it is active (1) or inactive (0)
$t_{switch,i}(t)$	Switch i's traffic carrying capacity factor; it is a function of time (t) and $0 \leq t_{switch}(t) \leq 1$
$T_{switch_i,max}$	Maximum number of new flow generated by the Switch i
C	Set of all controllers of a *Controller Module*
C_A	Set of active controllers of a *Controller Module*
$w_{controller,i}$	Controller i's workability factor; shows that either it is active (1) or inactive (0)
$t_{controller,i}(t)$	Controller i's traffic carrying capacity factor; it is a function of time (t) and $0 \leq t_{controller}(t) \leq 1$
$T_{controller_i,max}$	Maximum number of flow supported by the Controller i
$T_{controller_i}$	Number of new flows managed by the Controller i

Algorithm 3 Dynamic Flow Management Algorithm (DyFlow)

Require: $S, w_{switch}, t_{switch}(t), T_{switch,max}, C, C_A,$
$w_{controller}, t_{controller}(t), T_{controller,max}$
Ensure: $C_{selected}$: set of selected controllers

Initialize $C_{selected} \leftarrow \emptyset$
Compute $T_{switch,all}$ using (2)
$T_{controller,all}$ using (3)
$T_{controller,max}$ using (4)
if controllers are homogeneous **then**
  Compute N_c using (5)
  if $T_{switch,all} \geq T_{controller,max}$ **then**
    $C_{selected} \leftarrow C_A$
  else
    $C_{selected} \leftarrow N_C$, number of controllers in any order
  end if
else if controllers are heterogeneous **then**
  Initialize $C_k \leftarrow \emptyset$
  $j \leftarrow \emptyset$
  if $T_{switch,all} \geq T_{controller,max}$ **then**
    $T_{available} \leftarrow T_{switch,all}$
  else
    $T_{available} \leftarrow T_{controller,all}$
  end if
  while $T_{available} \neq 0$ **do**
    Update $j \leftarrow j + 1$
    Set $C_{selected}(j) \leftarrow C_k(T_{controller} : C_k \in C_A, C_{selected}(j) \in C_{selected})$
    $T_{controller} \leftarrow \min_{i \in C_A}(T_{available} - T_{controller,i})$
    Update $C_A \leftarrow C_A - C_{selected}(j)$
    $T_{available} \leftarrow T_{available} - T_{controller}$
  end while
end if

$$T_{switch,all} = \sum_{i \in S} (w_{switch,i} \times t_{switch,i}(t) \times T_{switch_i,max}) \tag{2}$$

$$T_{controller,all} = \sum_{i \in C} T_{controller,i} = \sum_{i \in C} (w_{controller,i} \times t_{controller,i}(t) \times T_{controller_i,max}) \tag{3}$$

$$T_{controller,max} = \sum_{i \in C} T_{controller_i,max}) \tag{4}$$

$$N_C = \begin{cases} \lceil |C| \times \frac{T_{switch,all}}{T_{controller,max}} \rceil & \text{if } T_{switch,all} \leq T_{controller,all} \\ \lceil |C| \times \frac{T_{controller,all}}{T_{controller,max}} \rceil & \text{if } T_{switch,all} > T_{controller,all} \end{cases} \tag{5}$$

$$P_{con}(t) = \sum_{C=1}^{N_{AC}} \{P_P(C,t) + P_S(C,t) + 400 \times 10^3 \times N(t)^2 \times N_{max,C} \times N_L \times P_B\} \tag{6}$$

Table 2: Specifications of Sparse, Medium and Dense networks

Network Type	Maximum number of Switches	Switches Distribution	Network Surface (km^2)
Sparse	5	uniform	25
Medium	10	uniform	25
Dense	20	uniform	25

5 System Model

5.1 Network Model

Three different types of networks (sparse, medium and dense) are considered for three different traffic scenarios (rural, suburban and urban regions). For simplicity, it is considered that all of the network elements of each network are fully connected. The general specification of these networks is listed in Table 2.

5.2 Flow Generation Model

We model the flows generation of switches with a traffic fluctuation model [18]. The number of flows, generated by a switch, is approximated by using (7). Equation (7) represents the normalized number of flows ($N(t)$) as a function of time (t). Equation (7) is also used to estimate the normalized number of active switches at any instant of time, t. Here, the maximum number of flows, generated by a switch, is considered 0.4 *Mfps* [15]. The maximum number of switches of each network is listed in Table 2.

$$N(t) = \begin{cases} -.133t + .97 & \text{if } 0 \leq t \leq 5, \\ .073t^{.89} & \text{if } 5 < t \leq 12, \\ .1417t^{.62} & \text{if } 12 < t \leq 23. \end{cases} \tag{7}$$

5.3 Controller Model

We consider the NOX-MT controller [4]. It can process 1.6 *Mfps* with an average response time of 2 *msec*. Its access bandwidth is 1 *Gbps*, and the flow length (N_L) is 8 *bytes*. A controller consists of a processor and a server. This processor is on a single server. The power consumption ($P_{con}(t)$) of the controllers (or the control plane) at time t is shown in (6). In (6), $P_P(C,t)$ is the power consumed by the processor of the controller C, at time t. $P_S(C,t)$ is the power consumed by the server of the controller C, at time t. P_B is the power consumption per bit, which is 18×10^{-9} $Watt$. $N_{max,C}$ is the maximum number of switches controlled by the controller C. N_{AC} is the number of all active controllers at time t.

The set up and maintenance cost of the control plane (or controllers) is also considered. Controller installation cost is referred to as the set up cost. It varies with the network type (or the number of controllers). The power consumption of the controller, due to its operation and cooling, is taken into

Table 3: Cost estimation of a Controller (Server and Processor)

Cost parameter		Value
Set up cost	Processor installation cost, C_P	\$215 [19]
	Server installation cost, C_S	\$1250 [20]
Maintenance cost	Processor power consumption, P_P	86 $Watt$ [19, 21]
	Server power consumption, P_S	367 $Watt$ [21]
Electricity cost, E_C		7.5 $\frac{cents}{Watt}$ [22]

account in the maintenance cost. It varies with the number of active controllers at any instant of time. These estimated costs are listed in Table 3. The set up and maintenance cost ($S_{cost}(t)$) of the control plane (or controllers) at time t is shown in (8).

$$S_{cost}(t) = E_C \times P_{con}(t) + \sum_{C=1}^{N_{AC}} \{C_P(C) + C_S(C)\} \tag{8}$$

In addition, some other considerations are taken into account to perform the simulation. These considerations are as follows,

1 It is considered that the latency is proportional to the distance, and the maximum latency bound (D_{req}) is 10 $msec$. In our considered system model, the network dimension ($5km \times 5km$) is much smaller. Thus for this maximum latency bound of 10 $msec$ we get one controller module. The more the network dimension is extended, the more controller module will be found.

2 The locations of all switches are known.

3 Although the proposed model considers the heterogeneity of both controllers and switches, to simplify the simulation it is assumed that the controllers are homogeneous in traffic carrying capacity.

4 It is considered that the set up and maintenance cost of a controller module is the summation of the costs of controllers of that controller module.

5 Number of controller of a controller module is set in such a manner that can handle the maximum traffic load of all switches connected with that controller module.

6 A switch with the highest x-coordinate is considered as the border switch (A).

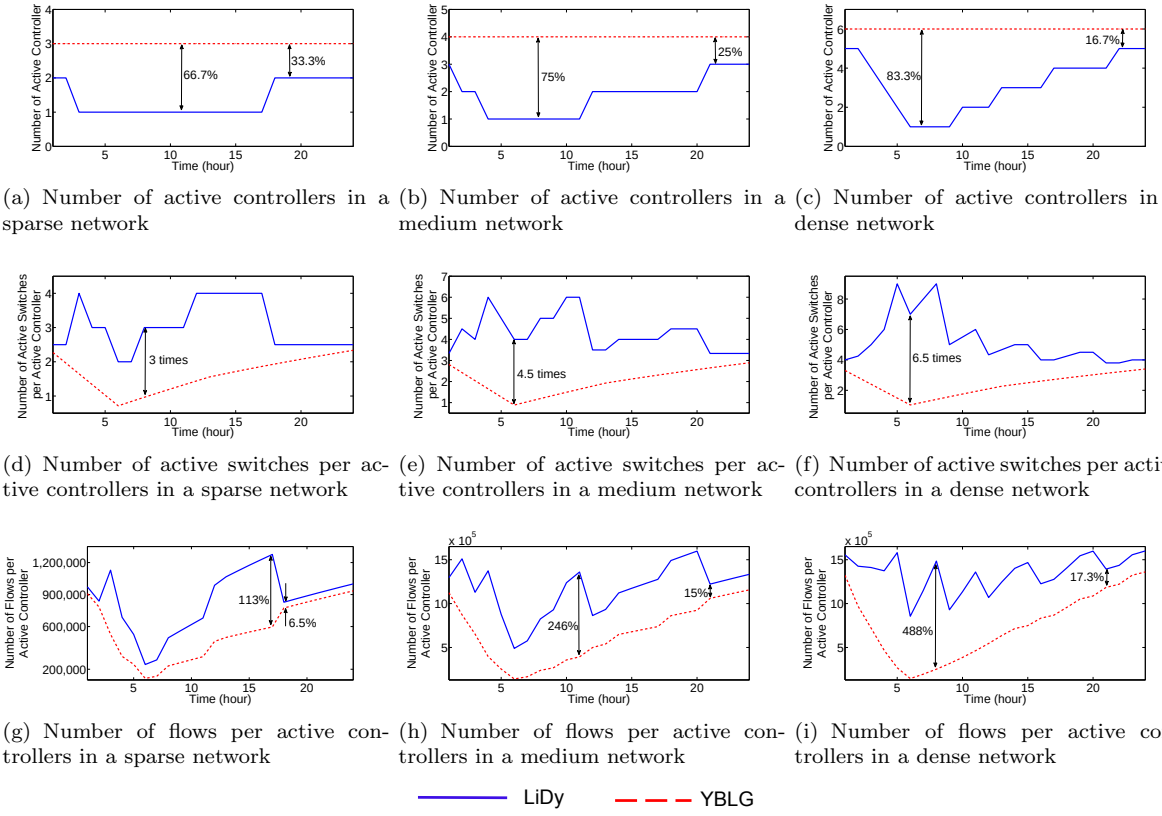
7 It is assumed that $n = 2$ in (1).

6 Result and Analysis

In this section, we present the simulation results of LiDy based on the system model presented in Section 5. These results are compared with YBLG, the most recent and relevant controller placement solution we are aware of, to show the comparative performance analysis.

6.1 Number of Active Controllers of the Control Plane

The effect of the number of active controllers obtained with using LiDy and YBLG is shown in Fig. 5(a), Fig. 5(b) and Fig. 5(c). This variation in the number of active controllers with respect to the time is shown during one day. In all cases, LiDy uses a lower number of active controller than YBLG. For the sparse network, it decreases the number of active controller from 33.3% to 66.7% as shown in Fig. 5(a). Similarly for the medium network and dense network, the decrease in the number of active controllers is 25% to 75% as shown in Fig. 5(b), and 16.7% to 83.3% as shown in Fig. 5(c), respectively. Here, we can observe that with the increasing number of switches (from the sparse network to the dense network), the difference in the number of maximum active controllers increases. Thus LiDy utilises a limited number of controllers to maintain the same functionalities offered by YBLG.



(a) Number of active controllers in a sparse network

(b) Number of active controllers in a medium network

(c) Number of active controllers in a dense network

(d) Number of active switches per active controllers in a sparse network

(e) Number of active switches per active controllers in a medium network

(f) Number of active switches per active controllers in a dense network

(g) Number of flows per active controllers in a sparse network

(h) Number of flows per active controllers in a medium network

(i) Number of flows per active controllers in a dense network

Figure 5: Performances of LiDy compared to YBLG

6.2 Controller Utilization of the Control Plane

In SDN, new flows, generated by numerous switches, are managed by controllers. Thus the number of active switches controlled by each active controller is considered as the measurement of utilization of the controller. Moreover, the average number of flows managed by each active controller is also taken into consideration to estimate its utilization. The more number of flows (or switches) an active controller manages, the more it is utilised. Fig. 5(d), Fig. 5(e) and Fig. 5(f) show the controller utilization for the sparse, medium and dense network, respectively. These figures show the variation of the number of active switches per active controller with respect to time of the day.

For the sparse, medium and dense network, LiDy increases the number of active switches per active controller around 3, 4.5 and 6.5 times, respectively, compared to YBLG. We get that, from the sparse network to the dense network as the number of switches increases, an increasing number of switches is controlled by each active controller.

Conversely, Fig. 5(g), Fig. 5(h) and Fig. 5(i) show that the average number of flows managed by each active controller for the sparse, medium and dense network, respectively. Compared to the YBLG, LiDy increases the number of flows per active controller around 113%, 246% and 488% for the sparse, medium and dense network, respectively. Like previous results, with the increasing number of switches (from the sparse network to dense network), the average number of flows per active controller is also increased by using LiDy. One interesting result is that by using LiDy, the average number of flows per active controller, for both medium and dense network, is found around 1.6 *Mfps*. Since the maximum capacity of the considered NOX-MT controller is 1.6 *Mfps*, at that time all of the active controllers are fully utilized.

6.3 Power Consumption of the Controllers

Fig. 6 shows the relative variation of the power consumption of the control plane (or controllers) with respect to time for the sparse, medium and dense networks. It presents the percentage of reduction in power consumption by using LiDy compared to the YBLG. The proposed model reduces the power

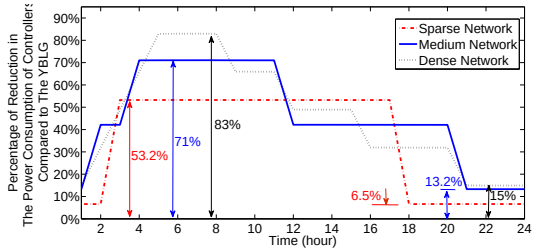


Figure 6: Percentage of reduction in the power consumption of controllers by using LiDy, compared to the YBLG, for the sparse, medium and dense networks

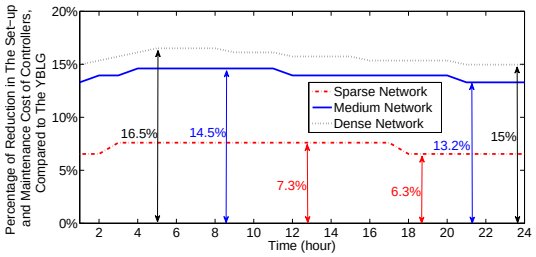


Figure 7: Percentage of reduction in the set up and maintenance cost of controllers because by using LiDy, compared to the YBLG, for the sparse, medium and dense networks

consumption from 6.5% to 53.2% for the sparse network, 13.2% to 71% for the medium network, and 15% to 83% for the dense network. It is found that by using LiDy with the increasing number of switches (from sparse network to dense network), the difference of the maximum power consumptions is also increased, compared to the YBLG.

6.4 The Set up And Maintenance Cost of the Controllers

According to the system model of Section VI, the set up cost is fixed for each type of network, but the maintenance cost is variable. The set up cost depends on the total number of controllers of the network, whereas the maintenance cost depends on the number of active controllers of the network. Fig. 7 shows the relative cost (set up and maintenance cost) variation with respect to time for the sparse, medium and dense networks. This figure shows the variation in the relative cost between the LiDy and the YBLG. The results shows that LiDy reduces these costs from 6.3% to 7.3% for the sparse network, 13.2% to 14.5% for the medium network and 15% to 16.5% for the dense network, compared to YBLG.

6.5 Effect of the Latency

One of our goal is to maintain the latency, between the switch and the controller, within the maximum latency bound of 10 $msec$. The simulation results show that LiDy has maximum latency of 0.015 $msec$, 0.015 $msec$ and 0.014 $msec$ for the sparse, medium and dense networks, respectively. Thus LiDy ensures the latency bound. The empirical cumulative distribution function (eCDF) of the latency, between the switch and the controller, is presented in Fig. 8. It shows the median value of latency of LiDy (9.63×10^{-6} sec) is smaller, compared to the YBLG (1.006×10^{-5} sec).

Moreover, to understand the effect of the maximum latency bound on the number of controllers, we simulated all three networks (sparse, medium and dense) for the different maximum latency bounds ($1msec$ to $10msec$). In all cases, we get the same number of controllers.

LiDy consists of three distinct algorithms: LICMP, LSA and DyFlow. Both, LICMP and DyFlow have the time complexity of $O(n)$. LSA uses the Bowyer-Watson Algorithm, to form the Delaunay triangles, which time complexity is $O(n^3)$. Thus the time complexity of LSA is $O(n^3)$. But there are several Delaunay triangle formation algorithms having the linear time complexity [23]. By using them, the time complexity of LSA can be reduced to $O(n)$. On the contrary, YBLG has the time complexity of $O(n^2)$. Here, the worst time complexity is considered as the time complexity.

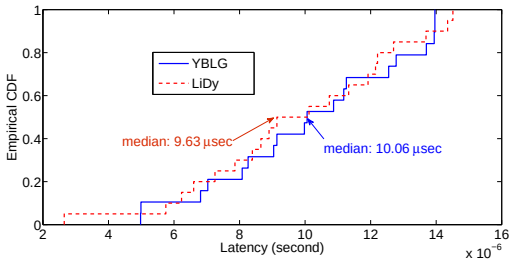


Figure 8: eCDF of latencies of switches with respect to their corresponding controllers for dense network

From the above analysis we can conclude that LiDy utilises a lower number of controllers and ensures their higher utilization, than the YBLG. This reduces the overall system cost and energy consumption extensively.

7 Discussion

In this section, we discuss how LiDy differs from the other research works on the controller placement problem of the SDN. They are as follows,

1 The key feature of the software defined network is its centralized controllability. To keep in conformity with this feature, we have introduced the controller module concept in LiDy. Because of using controller module, controllers are less distributed satisfying the required latency bound. Simulation results of Section 6 show that this approach enhance the controller utilizations.

2 The research work of the controller placement problem of the SDN can be broadly classified into two categories. This classification is based on the location searching manner for the controller of the SDN. In the first category, the research papers, e.g., [14, 15], propose to select the controller location from a set of selected locations (locations of switches). Whereas in the second category, the research paper's proposal is to selectively activate a controller from a set of controllers [12]. In this paper, we have proposed a new approach, known as LiDy, to place the controller in the SDN. Our proposal is to search the entire region containing the switches to get the location of the controller. The performance evaluation of LiDy, shown in Section 6, shows that it not only reduces the number of active controllers but also increases their utilizations.

3 From Section 2, we get that most of the recent research works [12, 13, 14, 15] on the controller placement problem consider both the latency and the load of switches to place the controllers in the software defined networks. In these research works, to place the controllers, a common approach is to consider the load of switches only once in conjunction with the latency. Although the latency is roughly fixed, the load of switches varies with time. Since the controller is placed based on both the latency and the load of switches. Thus with this continuous varying load, controllers need to be replaced continuously. But it is not practically feasible. Considering this dilemma, the latency and the load of switches are separately considered in LiDy. The outperforming simulation results of Section 6 support the consistency of this consideration.

8 Conclusion and Future Work

LiDy offers a combined solution of the controller placement problem and the controller scheduling problem of the SDN by taking as inputs, the locations of switches of the SDN and the required latency constraint. LiDy outputs the locations of the controller modules, and the number of active controllers of each controller module. Although the first output is static, the second one is dynamic. From the best of our knowledge, LiDy is the first controller placement solution in which dynamic traffic load of switches is taken into consideration. Our results show that LiDy achieves higher utilization at a lower cost and the similar latency than the most recent CPP solution.

As future work, we have intention to extend this work into two directions. First, we plan to design a new location searching algorithm, having lower computational complexity, for LICMP to get a better performance. Second, instead of a single latency constraint, a number of different latency constraints will be considered in the future extension of LiDy. In addition, we want to explore the effect of the controller module on larger scale SDNs.

References

[1] Open Networking Foundation. Software Defined Networking (SDN) definition. 2015. [Online]. Available: `https://www.opennetworking.org/sdn-resources/sdn-definition`

[2] B. Naudts, M. Kind, F. Westphal, S. Verbrugge, D. Colle, and M. Pickavet, "Technoeconomic analysis of software defined networking as architecture for the virtualization of a mobile network", *in Proc. of European Workshop on Software Defined Networking*, pp. 67- 72, October 2012.

[3] *Is the future of networking software defined*, White paper, Dimension Data, 2013. [Online]. Available: `http://www.dimensiondata.com/Global/Downloadable`.

[4] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks", *In Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, USENIX Association, Berkeley, CA, USA, 2012.

[5] DE-CIX, *Traffic Statistic*. 24 July 2014. [Online]. Available: `https://www.de-cix.net/about/statistics/`

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, and A. Vahdat, "B4: experience with a globally-deployed software defined wan", *In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 3-14, New York, USA, 2013.

[7] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem", *SIGCOMM Comput. Commun. Rev.*, Vol 42, No 4, pp.473-478, September 2012.

[8] T. Benson, A. Akella, and D. A. Maltz. "Network Traffic Characteristics of Data Centers in the Wild", *ACM SIGCOMM IMC*, pp. 267-280, 2010.

[9] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, "Reliability aware controller placement for Software-Defined Networks", *2013 IFIP/IEEE Int. Sym. on Integrated Network Management*, pp.672-675, 27-31 May 2013.

[10] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks",*25th Int. Teletraffic Congress* , pp.1-9, 10-12 September 2013.

[11] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, 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, March 2015.

[12] H. K. Rath, V. Revoori, S. M. Nadaf, A. Simha, "Optimal controller placement in Software Defined Networks (SDN) using a non-zero-sum game", *2014 IEEE 15th Int. Sym. on A World of Wireless, Mobile and Multimedia Networks*, pp.1-6, 19-19 June 2014.

[13] A. Sallahi, M. St-Hilaire, "Optimal Model for the Controller Placement Problem in Software Defined Networks", *IEEE Communications Letters*, Vol.19, No.1, pp.30-33, January 2015.

[14] Y. Jimenez, C. Cervello-Pastor, A. J. Garcia, "On the controller placement for designing a distributed SDN control layer", *2014 IFIP Networking Conf.*, pp.1-9, 2-4 June 2014.

[15] G. Yao, J. Bi, Y. Li, L. Guo, "On the Capacitated Controller Placement Problem in Software Defined Networks", *IEEE Communications Letters*, Vol.18, No.8, pp.1339-1342, August 2014.

[16] World eBook Library, *Delaunay Triangulation.* [Online]. Available: `http://ebook.worldlibrary.net/article/WHEBN0000008864/Delaunay%20triangulation#Algorithms`

[17] Intel, *Intel Xeon Phi Product Family.* [Online]. Available: `http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html`.

[18] S. Gebert, R. Pries, D. Schlosser, and K. Heck, "Internet access traffic measurement and analysis", *In Traffic Monitoring and Analysis*, Vol. 7189, pp. 29-42, 2012.

[19] Intel, *Intel Core i5-4670 Processor.* [Online]. 2013. Available: `http://ark.intel.com/products/75047/Intel-Core-i5-4670-Processor-6M-Cache-up-to-3_80-GHz`.

[20] Dell, "Dell PowerEdge T110 II Server". [Online]. 2014. Available: `http://www.dell.com/au/business/p/poweredge-t1102/pd?oc=u421102au&model_id=poweredge-t110-2`.

[21] Vertatique (Green ICT: Sustainable Computing, Madia, e-Devices), *Average power use per server & desktop.* [online]. 2011. Available: `www.vertatique.com/average-power-use-server`.

[22] Origin Energy, *Electricity Tariffs (QLD).* (Online). 2014. Available: `http://www.originenergy.com.au/2087/Electricity-tariffs-QLD`

[23] S. Fortune, "Computing in Euclidean Geometry", *Voronoi diagrams and Delaunay triangulations*, 2nd edition, World Scientific, 1995.