Large-Scale Dynamic Controller Placement

Md Tanvir Ishtaique ul Huque
University of New South Wales, Data61-CSIRO

Weisheng Si
Western Sydney University

Guillaume Jourjon
Data61-CSIRO

Vincent Gramoli
University of Sydney, Data61-CSIRO

Abstract

The controller placement problem (CPP) is one of the key challenges of software defined networks (SDN) to increase performance. Given the locations of n switches, CPP consists of choosing the controller locations that minimize the latency between switches and SDN 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 have addressed 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 dynamic load.

We propose an algorithm named *LiDy+* that runs in $O(n^2)$ and combines a controller module placement algorithm with a dynamic flow management algorithm. We evaluate the number of controllers, the controller utilization, the power consumption and the maintenance cost of *LiDy+* on both sparse and dense networks. Our comparison against a previous solution shows that *LiDy+* does not only achieve a smaller number of controllers and a higher controller utilization, but also incurs less energy and maintenance costs than the previous solution. Finally, we run *LiDy+* in a large-scale environment where the previous solution of time complexity $\Omega(n^2 \log n)$ is impractical.

1 Introduction

Software Defined Network (SDN) [1] relies on centralized intelligence to control the network management. However, state-of-the-art SDN controllers, such as Beacon [3, 27], can process ~10 Million flows per second (*Mfps*) with a median response time of 2 *msec* and cannot scale to a state of the size of New South Wales that could require the processing of ~70 *Mfps* [28]. To further scale a software defined network to the size of a continent, it is thus necessary to place multiple controllers at properly selected locations.

The *controller placement problem (CPP)* consists of choosing controller locations given the locations of switches to maximize performance while minimising the delay between switches and some controllers regardless of the network traffic variations [6]. Most solutions to this problem [6, 8–10, 13, 14] are variants of solutions to the facility location problem, all assuming that switches incur a fixed load. In real networks, however, the load applied to a controller depends on the number of switches connected to it and the rate at which these switches encounter new flows [7]. Hence, one has to consider two problems when setting up controllers, the controller placement problem placing controller "modules" to minimize communication latency, and the provisioning of controllers at each of these controller modules to handle the dynamic load.

Recently few tentative techniques tackled the problem of the dynamic load. We initially proposed the *LiDy* [15] controller placement solution. *LiDy* was shown to require a lower number of controllers to balance the same traffic load than *YBLG* [14], the latest controller placement solution we are aware of. The drawback of *LiDy* is, however, its complexity: although *LiDy* doubles the utilization of controllers and halves power consumption compared with *YBLG*, its time complexity is $\Omega(n^3)$.[1] Not only does this complexity slows the reaction of the algorithm to the variations of the dynamic load, but it makes any large-scale evaluation impossible.

To reduce the time complexity of *LiDy*, this paper presents *LiDy+* that achieves similar performance, in term of controllers utilization, power consumption, maintenance cost and latency profile, as *LiDy* in a time complexity of $O(n^2)$. Like *LiDy*, *LiDy+* is a large-scale controller placement technique for SDN-based wide area networks (WANs). The key novelty of *LiDy+* lies in an efficient circular decomposition of the plane rather than the triangular decomposition of the plane proposed by *LiDy*. In particular, *LiDy+* splits the network region into a constant number of subregions, and then run the smallest enclosing disk algorithm [24] on the set of switches located in each of these subregions. The simplicity of the

[1]Here $f(n) = \Omega(g(n))$ (resp. $f(n) = O(g(n))$) indicates that there exist positive constants c and n_0 such that $f(n) > cg(n)$ (resp. $f(n) < cg(n)$) for all $n > n_0$.

smallest enclosing disk algorithm running in $O(n)$ time enables *LiDy+* to achieve the complexity of $O(n^2)$.

Thanks to this optimization we evaluated *LiDy+* on a large-scale network of up to $100,000$ switches. We evaluated the performance of *LiDy+* in terms of (i) number of SDN controllers necessary to handle the dynamic load of switches, (ii) power consumed by the running SDN controllers and (iii) latency between every switch and its closest controller. We observe that *LiDy+* achieves similar performance to *LiDy* at small scale ($\sim 1K$ switches) and significantly better performance than *YBLG* at medium-scale ($\sim$10K switches). Unfortunately, we could neither evaluate the performance of *YBLG* nor *LiDy* at large-scale due to their time complexity, but we illustrate the scalability of *LiDy+* that places controllers at the scale of a continent ($\sim$100K switches) in about 5 hours.

We also evaluate our technique, *LiDy+* as a solution to a variant of the *CPP* [6] that takes into account the dynamic traffic load of switches. To the best of our knowledge, *LiDy+* is the first solution that addresses dynamic traffic load by using prediction technique (detailed in Section IV). Moreover, in *LiDy+*, we introduce the concept of "controller module", which is a set of controllers. LiDy+ adjusts to the dynamic traffic load by activating different number of controllers in a controller module. We compare *LiDy+* with *YBLG*, which also considers traffic load when deploying controllers although it does not support the constant changes of traffic load. We show that our solution utilizes a lower number of controllers to balance 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* and *LiDy+* assign more than twice as many switches per controller as *YBLG*. It also increases the number of flows per controller more than 100% compared to *YBLG*.

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

3 **Cost:** *LiDy* and *LiDy+* reduce the maintenance cost to about 6% compared to *YBLG*. This cost reduction rate increases as the number of switches in the network increases.

4 **Scalability:** While *LiDy* has complexity $\Omega(n^3)$ and *YBLG* has complexity of $\Omega(n^2 \log n)$ as we explained in Section 6, *LiDy+* has complexity of $O(n^2)$, which makes our solution scalable to hundreds of thousands of switches.

The remainder of this paper is organized as follows: Section 2 surveys the relevant research work, showing the present research trend of the controller placement problem of the SDN. Section 3 motivates this paper and gives the problem statement. Section 4 presents 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, among *LiDy+*, *LiDy* and *YBLG*, is shown in Section 6. Section 7 discusses the characteristics of *LiDy+*. Finally, Section 8 ends with the conclusion and directions for future work.

2 Related Work

In this section, we discuss the key research results of the controller placement problem (CPP). The distinguishing features of these research results, [6, 8–15], are presented in Table 1. Since, quality of services (QoS) is outside the scope of this paper, we do not discuss QoS.

The research into the SDN controller placement problem can be broadly classified into two categories depending on the system inputs. The research works of Heller et al. [6], Hu et al. [8], Hock et al. [9] and Lange et al. [10] fall into the first category. They consider only the latency, between switches and controllers, to estimate the number and locations of SDN controllers in the network. On the contrary, the research works of Rath et al. [11], Sallahi et al. [12], Jimenez et al. [13], *YBLG* [14] and *LiDy* [15] take into account both the latency and traffic load of switches to get the number and locations of SDN controllers. The research results of this latter category can be further classified based on the network dimension. The proposed solutions of Rath et al. [11] and Sallahi et al. [12] are applicable only for small scale networks, whereas the solutions proposed by Jimenez et al. [13], *YBLG* [14] and *LiDy* [15] are for large scale networks.

The SDN controller placement problem was first introduced by Heller et al. [6] in 2012. Later, several papers [8–15] investigated the controller placement problem in SDN. The SDN controller placement problem is to estimate the minimum number of controllers, and their placements in the network. Heller et al. showed that a random placement of a controller yields five times worse latency than an optimal placement of that controller in the network. Hu et al. [8] presented a research on the number of controllers and their placements in the network considering the reliability of the SDN control plane. Their research outlined a trade-off between the reliability and the latency. They also found that placing too many or too few controllers could reduce the reliability of the network. Heller et al. [6] as well as Hu et al. [8] considered locations of switches to get the number and locations of the controllers. They, however, ignored the impact of inter-controllers effects. Hock et al. [9] were the first to take the inter-controller latency into account to determine the optimal number and placement of controllers. Like Hu et al. [8], their primary concern was the reliability of the SDN control plane. In parallel, they also considered the resilience of the network as its tolerance to failures and the number of switches. Yet, this technique is limited to small and medium scale networks. To overcome this shortcoming, Hock et al. [9] later extended their research work [10]. They offered a heuristic approach to find out the optimal number and locations of controllers in a large scale network.

Table 1: The key research works into the SDN controller placement problem (CPP)

Technique	Large scale network	Open search	Performance metric Latency	Traffic load	QoS
LiDy [15]	✓	✓	✓	dynamic	×
Bari et al. [26]	✓	×	✓	dynamic	×
Jimenez et al. [13]	✓	×	✓	static	✓
YBLG [14]	✓	×	✓	static	×
Lange et al. [10]	✓	×	✓	×	✓
Rath et al. [11]	×	×	✓	static	✓
Sallahi et al. [12]	×	×	✓	static	×
Heller et al. [6]	×	×	✓	×	×
Hu et al. [8]	×	×	✓	×	✓
Hock et al. [9]	×	×	✓	×	✓

All of the above mentioned research works of Heller et al., Hu et al., Hock et al., Lange et al. considered only the latency to get the optimal number of controllers, and their corresponding locations in the network. Besides latency, traffic load of switches has also been considered in a number of research works, e.g., Rath et al. [11], Sallahi et al. [12], Jimenez et al. [13], Bari et al. [26], and Yao et al. [14], to get the number and locations of the SDN controllers. Rath et al. [11] proposed a non-zero-sum game based dynamic controller placement technique. Here, a controller is either activated or deactivated periodically based on the traffic demand of switches. This technique only ensures the maximum utilization of the SDN controllers in the network. But it did not define where to place the controllers in the network. By contrast, Sallahi et al. [12], Jimenez et al. [13], Bari et al. [26] and *YBLG* [14] defined the optimal placement of controllers in the SDN. Sallahi et al. proposed a mathematical model to define the optimal number and locations of the SDN controllers. They also considered the heterogeneity of SDN controllers, and their interconnections. This model is however applicable only for the small scale ($1\ km \times 1\ km$) network. By contrast, the controller placement models of Jimenez et al., Bari et al., *YBLG* and *LiDy+* are not limited to the small scale network. These models estimated both the number and the locations of SDN controllers. These large scale controller placement models can be further classified based on the traffic load type of switches. The considered traffic load of Jimenez et al., and *YBLG* was static, whereas Bari et al. and *LiDy+* considered dynamic traffic load of switches. Dynamic traffic load means the traffic loads of switches varies with time. Note that Jimenez et al. considered only the number of switches as the traffic load to the controller; and *YBLG* assumed that each switch sends a fixed traffic load to the controller. While Bari et al. [26] also considered the dynamic traffic load as *LiDy+*, *LiDy+* differs from Bari et al. [26] in that *LiDy+* employs a prediction technique to solve the problem.

3 Problem Statement

The aforementioned solutions rely on a common simplification. Indeed, they estimate the location of the SDN controller from a set of selected locations in the network. These selected locations are usually identical to the one of switches in the network. Since this *restricted search* selects the controller location from a set of fixed locations in a region (the geographical area containing the network), it may not result in the optimal location in that region. In the following parts of this section, we illustrate the importance of the *open search*, which allows the controller to be placed in any location in the geographical area, as opposed to the *restricted search* for placing the SDN controller in the WAN topology.

Figure 1 depicts an example in which four identical fully connected switches (A, B, C and D) are located at four corners of a square region. All of these switches generate the same number, say x, of flows. Here, the goal is to show the difference between the *open search*, and the *restricted search* from various aspects, e.g., the controller locations, the latency, the number of controllers and the performance (traffic load of switches) of the software defined network.

3.1 Effect of the Controller Placement on the Latency

According to the *restricted search*, used in [6, 8–14], the optimal location of the SDN controller is one of four locations of switches (A, B, C and D). For any of four locations, the maximum latency between switches and the controller ($A \leftrightarrow D$ or $B \leftrightarrow C$) is $\sqrt{2}a$, shown in Figure 1.[2] But, if the controller is placed at location O as shown in Figure 2, then the maximum latency becomes $\frac{a}{\sqrt{2}}$. This

[2]In all considered scenarios of Figure 1, and Figure 2, it is assumed that the latency between two switches is proportional to their distance.

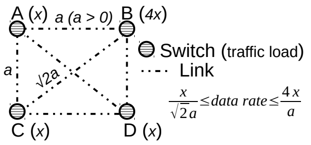


Figure 1: A software defined network consisting of switches having different traffic loads

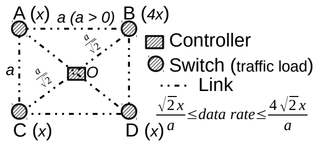


Figure 2: Effect of the controller placement on the traffic load of switches of the SDN

controller placement of Figure 2 halves the latency compared to the controller placement of Figure 1. Thus the *open search*, the entire region containing the switch's locations, decreases the latency compared to the *restricted search*.

3.2 Effect of the Controller Placement on the Number of Controllers

Consider the same example of Figure 1 with the maximum latency bound of $\frac{a}{\sqrt{2}}$. Now, according to the common controller placement approach, used in [6, 8–14], the controller placement is limited to the switch locations. Since the minimum latency between two consecutive switches is a, as 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. This controller placement reduces the number of controllers by four times. Thus, instead of using the *restricted search* if the *open search* is used, then a limited number of SDN controllers can cover the network.

3.3 Effect of the Controller Placement on the Traffic Load of Switches

To illustrate the effect of controller placement on the traffic load of switches, the same example of Fig. 1 has been considered. But, this time it is considered that switches have different amount of traffic load (number of flows) to send to controller, shown in Fig. 1. Switches A, B, C and D generate continuously x, $4x$, x and x number of flows, respectively. Now according to the common controller placement approach, used in [6, 8–14], if the controller is placed at one of four switch locations, then the maximum flow rate becomes $\frac{flows}{latency} = \frac{4x}{a}$. On the contrary, if the controller is placed at location O, shown in Figure 2, then the maximum flow rate becomes $\frac{4\sqrt{2}x}{a}$. The controller placement of Figure 2 doubles the flow rate compared to the controller placement of Figure 1. Thus the *open search* achieves a better utilization of the SDN controller, compared to the *restricted search*, to improve the network performance (in term of the flow rate).

The aforementioned observations motivated us to apply the *open search* in the entire intended region to get the optimal location of the SDN controller. To achieve this goal, we propose, in the next section, a controller placement solution called *LiDy+*.

4 Proposed Controller Placement Solution

In this section, we propose the SDN controller placement solution, *LiDy+*. To place the controller in the network, *LiDy+* considers both the latency and the traffic load of switches. *LiDy+* has the flexibility to adapt between the open search and the restricted search techniques based on the characteristics of geographic areas. If a geographic area allows open search (i.e., contains cities or towns almost everywhere, which enables the deployment of a controller module at nearly any location), the open search technique will be used; on the other hand, if a geographic area does not allow open search (i.e., contains mountains, sea or deserts where the deployment of a controller module is impossible), the restricted search technique will be used. *LiDy+* can take the geographic characteristics, i.e., the availability of possible locations of controller modules in the network, as input and chooses the suitable search technique. *LiDy+* can also predict the traffic load of switches and accordingly adjust the number of controllers in each controller module. It consists of mainly two algorithms: the *Location Independent Controller Module Placement Algorithm* (Algorithm 1) and the *Dynamic Flow Management Algorithm* (Algorithm 4).

4.1 Location Independent Controller Module Placement Algorithm

The *Location Independent Controller Module Placement Algorithm* (LICMP) determines locations of SDN controller modules in the network. A controller module consists of a set of SDN controllers. We present in more detail this concept in Section 4.4. LICMP (Algorithm 1) works along with the searching algorithm, *Location Searching Algorithm* (LSA). It has two primary functions:

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

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

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). Then LICMP considers a set, S_A, consisting of the border switch, A, and all of its neighboring switches (step 3). S_A gets all of those switches (x_i) that are within the latency bound of D_{req} from the border switch, A, in the network. 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}. For each switch (x_i) of S_A, LICMP estimates its neighboring switches (y_j) (steps 4-6). The set S_{x_i} consists of the switch x_i, and all its neighboring switches. After that, in S_A, LICMP selects a switch that has the highest number of neighbors, and its corresponding set, S_{select} (step 8). After obtaining S_{select}, LICMP invokes LSA (Algorithm 2) to estimate the optimal location (K), for placing the controller module, of the sub-region (S_{select}) (step 9). In this section, the sub-region is defined as a part of the network that contains the switches of S_{select}. Then LICMP checks the latency of all switches of S from the most appropriate location, K, of the S_{select}. It adds new switches in the S_{select} if these switches are within the latency bound of D_{req} (step 10). Finally, it updates the set S to continue selecting the optimal location (step 11), which terminates when there is no more switch (steps 12-15).

4.2 Location Searching Algorithm

LSA (Algorithm 2) acts as a part of LICMP (Algorithm 1). It takes the location information of a set of switches (S_p) in the network from LICMP; and yields the optimal location (K) to place the controller module. Here the objective function is to find out the optimal location (K) that minimizes the maximum latency between K and all switches of S_p.

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 $S_{select} \leftarrow \emptyset$, $z \leftarrow \emptyset$
 2: Select border switch $A$: $A \in S$
 3: $S_A \leftarrow$ Set of neighboring switches ($x_i$) of switch $A$
    where for $\forall x_i : x_i \in S_A, S_A \subset S, d(x_i, A) \le D_{req}$
 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 $\forall y_j : y_j \in S_{x_i}$, $d(y_j, x_i) \le D_{req}$
 6: end for
 7: Update $z \leftarrow z + 1$
 8: $S_{select} \leftarrow S_{x_{max}} \in S_{x_i}$, $|S_{x_{max}}| = \max_{1 \le i \le |S_A|} |S_{x_i}|$
 9: Call LSA Input: $S_{select}(z)$
    Output: $K$
10: Update $S_{select}(z)$
    where $\forall z_i : z_i \in S_{select}$, $d(z_i, K) \le D_{req}$
11: Update $S \leftarrow S - S_{select}(z)$
12: if $S \ne \emptyset$ 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
```

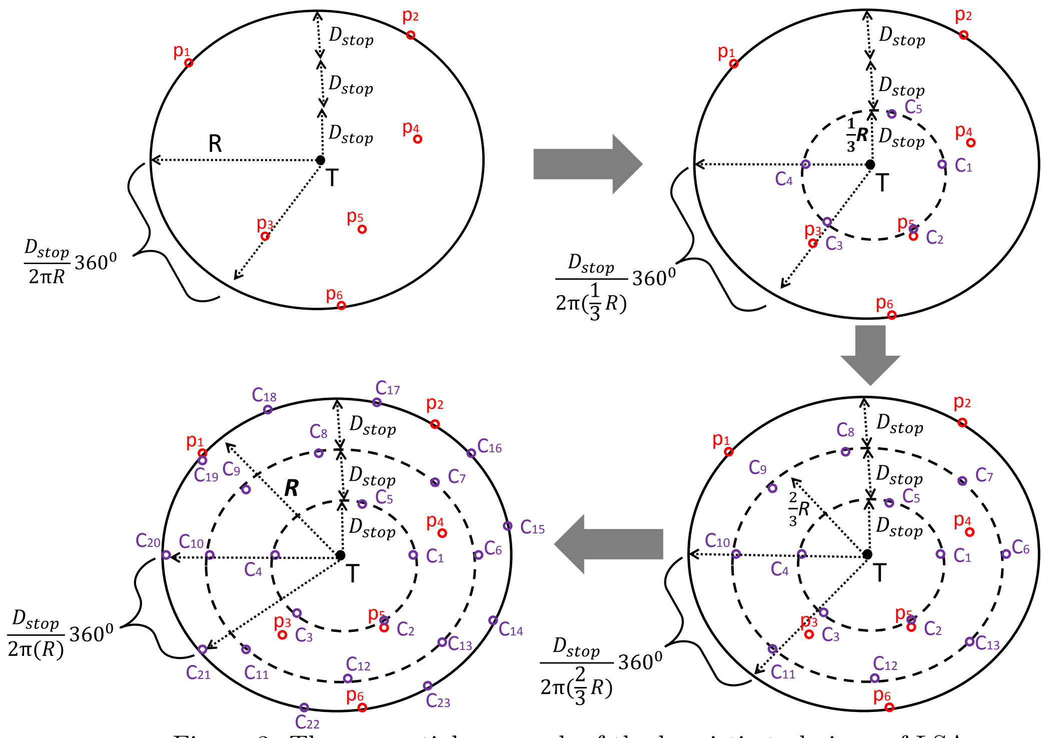
At first, LSA forms the smallest enclosing circle, by using the Welzl's algorithm [24]. This algorithm takes the locations of all switches of (S_p) as its input and it outputs a circle with a known center location (T) and a radius (R) (step 3). Then LSA uses a heuristic technique to get the set, S_{CT}, of all possible locations, C, of the controller module (steps 4-12). The proposed heuristic technique is depicted in Figure 3 where the smallest enclosing circle is formed based on the points p_1, p_2, p_3, p_4, p_5, and p_6. The center of the circle is T and its radius is R. In addition to the smallest enclosing circle, this heuristic technique considers a number of circles within the main circle, based on the center T of the main circle. The number of these circles is inversely proportional to D_{stop} that is the difference in radius between two consecutive circles. More specifically, D_{stop} is defined in Equation 1 where R is the radius of the main circle and n is the randomly selected number of circles.

$$D_{stop} = \frac{R}{n}. \tag{1}$$

The effect of D_{stop} for selecting different values of n is discussed in Section 6.3. The example of Figure 3 considers $n = 3$. So there are in total 3 circles having the center location of T. Here, the difference in radius between two successive circles is $D_{stop} = \frac{R}{3}$. Then LSA selects the points, C, on the circumference of each circle. These points, C, are considered as the possible locations of the controller module. Here, two successive points of the same circle of radius r is separated by an angle $\frac{D_{stop} \times 360°}{2 \times \pi \times r}$ based on the center point, T, of the circle. In Figure 3, all the five points C_1, C_2, C_3, C_4 and C_5 are taken from the same circle of radius $\frac{R}{3}$ and any of these two successive points are separated by an angle $\frac{D_{stop} \times 360°}{2 \times \pi \times \frac{R}{3}}$. Again, all the eight points C_6, C_7, …, C_{12}, C_{13} are taken from the same circle of radius $\frac{2 \times R}{3}$ and any of these two successive points are separated by the angle $\frac{D_{stop} \times 360°}{2 \times \pi \times \frac{2R}{3}}$. Similarly, the points C_4, C_5, …, C_{22}, C_{23} are determined from the circle of radius R. Therefore, for the example of Figure 3, all possible locations of the controller module becomes $S_{CT} = \{C_1, C_2, C_3, ..., C_{23}\}$. Finally, Among all possible locations, C, of S_{CT}, the optimal location K is selected to satisfy the following condition (step 13):

$$\min_{C_i \in S_{CT}} \max_{p_j \in S_p} d(C_i, p_j).$$

In this example depicted in Figure 3, the proposed heuristic technique leads to the optimal location C_{18}. Thus, for p_1, p_2, p_3, p_4, p_5, and p_6 of the set S_p (or S_{select}) the optimal location K is thus set at location C_{18}.



D_{stop}
R
T
$\frac{D_{stop}}{2\pi R}360^0$
$\frac{1}{3}R$
$\frac{D_{stop}}{2\pi(\frac{1}{3}R)}360^0$
$\frac{2}{3}R$
$\frac{D_{stop}}{2\pi(\frac{2}{3}R)}360^0$
$\frac{D_{stop}}{2\pi(R)}360^0$
p1 p2 p3 p4 p5 p6
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20 C21 C22 C23

Figure 3: The sequential approach of the heuristic technique of LSA

4.3 Restricted Location Searching Algorithm

We present in Algorithm 3 a modified version of Algorithm 2 where a limited number of controller locations are possible. In particular, LICMP uses R-LSA (Algorithm 3), instead of LSA (Algorithm 2), for those scenarios in which *open search* technique can not be applied because of limited resources or unusual attributes of the geographic region. In LSA, the *open search* technique is used to get the optimal location of the controller module, whereas the *restricted search* technique is used in R-LSA. Both of LSA and R-LSA have the similar working principle and the same objective function. The only difference is the heuristic technique (*steps 4-10* of Algorithm 2) used in LSA, is avoided in R-LSA. This technique offers a number of possible locations of the controller module within the considered region to get the best location to LSA, whereas R-LSA takes as an input a limited number of available locations (S_{CT}) to find out the optimal location.

At the beginning, R-LSA forms the smallest enclosing circle (C_p) based on locations of all switches (S_p) (step 3). Then it searches the available locations (S_{C_p}) of that circular region to place the controller module (step 4). Here, S_{C_p} takes only the user predefined input. It is a set of known and available locations in C_p where the controller module can be placed. In worst case scenario, $S_{C_p} = \emptyset$ leads to $S_{CT} = S_p$. In that case the controller module's location is selected among the switch locations (S_p). Finally, the optimal location (K) is selected from the available locations of S_{CT}. Here, the objective function, used to get K, is the same objective function of LSA's.

Algorithm 2 Location Searching Algorithm (LSA)

```
Require: Set of all switches, $S_{select}$
    Stopping criateria, $D_{stop}$
Ensure: Location, $K$

 1: Initialize $r \leftarrow D_{stop}, S_{CT} \leftarrow \emptyset$
 2: $S_p \leftarrow$ set of switches, where $\forall p \in S_p$, $S_p \leftarrow S_{select}$, and $p$ is the
    location of a switch
 3: Considering all switches' locations ($p$) of $S_p$ form the smallest
    enclosing circle using the Welzl's algorithm [24], where
    $T \leftarrow$ centre of the circle, and $R \leftarrow$ Radius of the circle
 4: while $r \leq R$ do
 5:    Update $r = r + D_{stop}$
 6:    Initialize: $\theta \leftarrow 0^\circ$
 7:    while $\theta \leq 360^\circ$ do
 8:       Update $S_C \leftarrow$ set of locations, where $\forall C \in S_C$, and
          $C(r \times cos(\theta), r \times sin(\theta))$ is a location
 9:       Update $\theta = \theta + \frac{D_{stop} \times 360^\circ}{2 \times \pi \times r}$
10:    end while
11:    $S_{CT} \leftarrow S_{CT} \cup S_C$, where $S_{CT}$ is the set of all possible loca-
       tions of the controller module including $T$
12: end while
13: Select $K$ where, $K \in S_{CT}$, and
    $\max_{p_j \in S_p} d(K, p_j) \leftarrow \min_{C_i \in S_{CT}} \max_{p_j \in S_p}, d(C_i, p_j)$
```

$$T_{switch,all} = -0.0008t^4 + 0.0078t^3 - 0.0131t^2 - 0.1954t + 1.0687 \quad for\ 1 \leq t \leq 24 \tag{2}$$

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

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

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

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

$$F_{controller,i}(t) = \frac{number\ of\ flows\ managed\ by\ the\ controller\ at\ time\ t}{maximum\ number\ of\ flows\ managed\ by\ the\ controller} \tag{7}$$

4.4 Dynamic Flow Management Algorithm

Instead of distributing a number of SDN controllers in the network, we propose to use a limited number of controller modules. In this case, a controller module is a set of SDN controllers that could be implemented on a many-core processor [17]. Each controller module runs the *Dynamic Flow Management Algorithm* (DyFlow) independently. DyFlow (Algorithm 4) is used periodically to predict the number of flows generated by switches to activate only the required number of controllers. Here, each controller of a controller module receives an equal number of flows generated by the switches as traffic loads. DyFlow considers the heterogeneity of controllers within a controller module in that each controller is capable of handling a different number of flows. To execute DyFlow, some preliminary system information of the controller module is required. The inputs of DyFlow are listed in Table 2 and its output is a set of active controllers ($C_{selected}$).

At the beginning, DyFlow predicts the number of new flows $T_{switch,all}$ generated by all switches (S) at time t (step 2). In Equation (2), the 6th order polynomial regression technique is used to predict the number of flows of switches at any instant of time [25, p.470]. In Equation (2), since the coefficients of x^6 and x^5 are 0, thus the highest polynomial order is 4. It is found that the 6th order polynomial regression technique has the estimation error at maximum of 6% using the traffic model of Gebert et al. [18]. Since this traffic model is used as the system model, presented in 5.2, the traffic load predictor, given in the Equation (2), is designed in such a way that it yields an overestimated value: 6% more than the actual predicted value. Then the controller module computes $T_{controller,all}$ to estimate the required capacity of the controller module to manage the flows, $T_{switch,all}$, of switches (S) in step 3. If controllers of the controller module are homogeneous, DyFlow uses steps 4-10 of the algorithm to set the required controller, $C_{selected}$. On the contrary, if the controllers are heterogeneous it executes steps 13-22. Here, the *traffic carrying capacity* factor, $F_{controller,i}$, is used to present the heterogeneity of the controllers. The *traffic carrying capacity* of a controller i at time t is defined in Equation (7). It shows the flow managing efficiency of an active controller. In the idle case, it is 1. In case of homogeneous controllers, if new flows $T_{switch,all}$ generated by switches exceed the maximum capacity $T_{controller,max}$ of the controller module, then DyFlow activates all available controllers of that controller module (steps 6-7). Otherwise, it uses Equation (5) to get the number of controllers N_C needed to be active (steps 8-9). If the controllers of the controller module are instead heterogeneous, the controller $C_{T_{controller}}$ is selected based on the following condition (steps 18-22):

$$\min_{i \in C_A} T_{available} - T_{controller,all}$$

This is a repetitive process. In each cycle, DyFlow

Algorithm 3 Restricted Location Searching Algorithm (R-LSA)

```
Require: Set of all switches, $S_{select}$
Ensure: Location, $K$

1: Initialize $S_{CT} \leftarrow \emptyset$
2: $S_p \leftarrow$ set of switches, where $\forall p \in S_p$, $S_p \leftarrow S_{select}$, and $p$ is the location of a switch
3: Considering all switches' locations ($p$) of $S_p$ form the smallest enclosing circle $C_p(R,T)$ using the Welzl's algorithm [24], where
   $T \leftarrow$ centre of the circle $C_p$, and $R \leftarrow$ radius of the circle $C_p$
4: $S_{C_p} \leftarrow$ set of available locations of the controller module in the circle $C_p$. Where $S_{C_p} \cap S_p \leftarrow \emptyset$
5: $S_{CT} \leftarrow S_p \cup S_{C_p}$, where $S_{CT}$ is the set of all possible locations of the controller module.
6: Select $K$ where, $K \in S_{CT}$, and
   $\max_{p_j \in S_p} d(K,p_j) \leftarrow \min_{C_i \in S_{CT}} \max_{p_j \in S_p}, d(C_i,p_j)$
```

Table 2: List of inputs of DyFlow

Symbol	Definition
S	Set of all switches managed by a *controller module*
C_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)
$F_{controller,i}(t)$	Controller i's traffic carrying capacity factor; it is a function of time (t), and $0 \leq F_{controller}(t) \leq 1$.
$T_{controller_i,max}$	Maximum number of flow supported by the Controller i
$T_{controller_i}$	Number of new flows managed by the controller i

selects a controller $C_{T_{controller}}$ of the available controllers of C_A. The selected controller $C_{T_{controller}}$ occupies the minimum flow capacity $T_{controller}$ among the available controllers of C_A. This outputs the minimum number of active controllers that ensures the maximum utilization of the controller module.

5 System Model

In this section, we present the experimental setup in detail.

Algorithm 4 Dynamic Flow Management Algorithm (DyFlow)

```
Require: $S, C_C, C_A, w_{controller}, F_{controller}(t), T_{controller}, T_{controller,max}$
Ensure: $C_{selected}$: set of selected controllers

 1: Initialize $C_{selected} \leftarrow \emptyset$
 2: Compute $T_{switch,all}$ using (2)
 3: Compute $T_{controller,all}$ using (3), and
            $T_{controller,max}$ using (4)
 4: if controllers are homogeneous then
 5:     if $T_{switch,all} \geq T_{controller,max}$ then
 6:         $C_{selected} \leftarrow C_A$
 7:     else
 8:         Compute $N_C$ using (5)
 9:         $C_{selected} \leftarrow C_{N_C}$, where $C_{N_C} \subset C_A$, and $N_C = |C_{N_C}|$
10:     end if
11: else if controllers are heterogeneous then
12:     Initialize $C_k \leftarrow \emptyset$, $j \leftarrow \emptyset$
13:     if $T_{switch,all} \geq T_{controller,max}$ then
14:         $T_{available} \leftarrow T_{switch,all}$
15:     else
16:         $T_{available} \leftarrow T_{controller,all}$
17:     end if
18:     while $T_{available} \neq 0$ do
19:         Update $j \leftarrow j + 1$
20:         Set $T_{controller} \leftarrow \min_{i \in C_A} (T_{available} - T_{controller,i})$, and
                $C_{selected}(j) \leftarrow C_{T_{controller}}$, where $C_{T_{controller}} \in C_A$,
            and $T_{controller}$ is the number of new flows managed by the
            controller $C_{T_{controller}}$.
21:         Update $C_A \leftarrow C_A - C_{selected}(j)$, and
            $T_{available} \leftarrow T_{available} - T_{controller}$
22:     end while
23: end if
```

Table 3: Specifications of Sparse, and Dense networks

Network type	Maximum number of switches	Switches distribution	Network surface (km^2)
Sparse	1k	uniform	7.7M
Dense	100k	uniform	7.7M

5.1 Network Model

Two different types of networks (sparse and dense) are considered for two different traffic scenarios (rural and urban regions). For simplicity, we consider the worst case scenario in that the network is fully reactive (i.e. each new flow in the network needs to get its own rule from the controller) and all network elements are logically fully connected. The general specification of these networks is listed in Table 3.

5.2 Flow Generation Model

We model the flow generation based on the data of the real-time traffic fluctuation of the wide area network topology presented by Gebert et al. [18]. The number of flows generated by a switch was approximated by using Equation (8) [25, p.515]. We used the *cubic spline interpolation* technique to drive Equation (8). This equation represents the normalized number of flows $N(t)$ as a function of time t. Equation (8) was also used to estimate the normalized number of active switches at any instant of time t expressed in hour. Here, the maximum number of flows, generated by a switch, was considered to be 0.4 *Mfps* [14]. The maximum number of switches of the dense network, and the sparse network is listed in Table 3.

$$N(t) = \begin{cases} -.133t + .97 & \text{if } 0 \leq t \leq 5, \\ .073t^{.89} & \text{if } 5 < t \leq 12, \\ .1417t^{.62} & \text{if } 12 < t \leq 23. \end{cases} \tag{8}$$

5.3 Controller Model

We consider the *NOX-MT* controller [2]. 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 all SDN controllers at time t is shown in Equation (6) where $P_P(C,t)$ 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 of a bit which is 18×10^{-9} $Watt$ [21]. $N_{max,C}$ is the maximum number of switches controlled by the controller C and N_{AC} is the number of all active controllers at time t.

Table 4: 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]

We now consider the setup and maintenance cost of controllers. The setup cost refers to the controller installation cost and varies with the network type (or the number of controllers). The power consumption of the controller, due to its operation and cooling, is considered a part of the maintenance cost. It varies with the number of active controllers at any time. These estimated costs are summarized in Table 4. The setup and maintenance cost, $S_{cost}(t)$, of all SDN controllers at time t is given in Equation (9).

$$S_{cost}(t) = E_C \times P_{con}(t) + \sum_{C=1}^{N_{AC}} \{C_P(C) + C_S(C)\} \quad (9)$$

In addition, we made the following assumptions for the simulation:

1 It was considered that the latency is proportional to the distance, and the maximum latency bound (D_{req}) is 4 *msec*.

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 was assumed that the controllers have the same traffic carrying capacity. It is $t_{controller,i} = 1$.

4 It was considered that the setup and maintenance cost of a controller module is the sum of the cost of all SDN controllers of that controller module.

5 A switch with the highest x-coordinate was considered as the border switch (A).

6 It was assumed that $n = 10$ in Equation (1).

7 Each controller module consists of a number of controllers that can manage the traffic flows of switches connected with it during the busiest hour of a day.

6 Result and Analysis

In this section, we present the simulation results of *LiDy+* compared to its previous version *LiDy*, and *YBLG*, considering system models described in Section 5 and given in *LiDy* [15]. Since the features of *YBLG*, shown in Table I, are closely relevant to *LiDy* compared to other controller placement techniques, *YBLG* and *LiDy* have been chosen as the candidates for comparison with *LiDy+*. To evaluate the performance of *LiDy+*, the *open search* technique described in 4.2 is adopted in all experiments. Since the main performance difference between the *open search* and the *restricted search* lies in latency, we present *restricted search* results in the upcoming Figure 7, Figure 8 and Figure 12. Therefore when not explicitly noted, we only present *open search* technique performance evaluation. This section is further divided into three subsections, each analysing the solutions in a different aspect.

* The *Large Scale Experimental Result Analysis* (Section 6.1) is based on the system model of Section 5. Here, the results of *LiDy+* are compared with *YBLG*.

* The *Small Scale Experimental Result Analysis* (Section 6.2) is based on the system model proposed in *LiDy* [15]. It compare the performance of *LiDy+*, *LiDy*, and *YBLG*.

* In the *Parameter Impact Analysis* (Section 6.3), the effect of the maximum latency bound (D_{req}) and the stopping criteria (D_{stop}) of *LiDy+* are further explained in terms of the number of active controllers and the latency of the network. This experiment exploits the dense network model of Section 5.

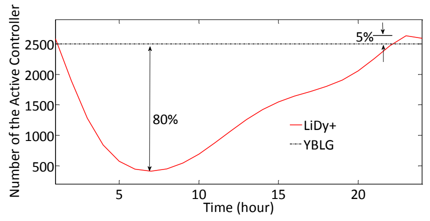
Finally, we present in Section 6.4 the time complexity analysis of *LiDy+*, *LiDy*, and *YBLG*.

For the simulation purpose we run *Matlab* on an *Intel(R) Core(TM) i7-5600U CPU @2.60GHz* with *8GB RAM*. Figure 13 reports the experienced run time of *LiDy+* and *YBLG* using the system model of Section 5. Here, the maximum time window is 650 *sec*. Since the minimum network dimension, i.e., $100K$ switches exceeds this time window we could not present the simulation run time of *LiDy* in Figure 13. Note that since *LiDy* experiences a significantly larger time complexity than *LiDy+* and *YBLG*, we could only present the full comparison of *LiDy*, *LiDy+* and *YBLG* at small scale.

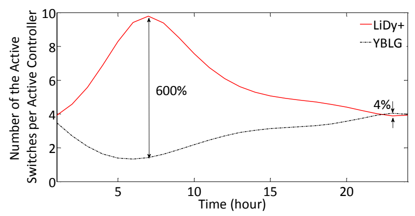
6.1 Large Scale Experimental Result Analysis

6.1.1 Number of Active Controllers of the Control Plane

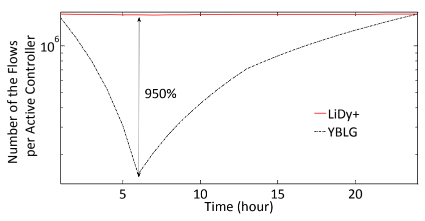
The number of active controllers obtained by using *LiDy+* and *YBLG* is shown in Figure 4(a) and Figure 4(d), respectively. This variation in the number of active controllers with respect to the time is given for a day. The controller number variations are shown in Figure 4(a) and Figure 4(d) for the dense network and the sparse network,



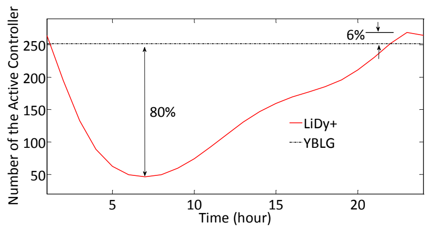
(a) Number of active controllers in the dense network



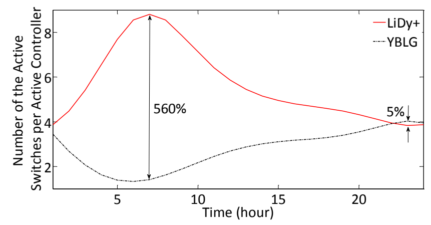
(b) Number of active switches per active controller in the dense network



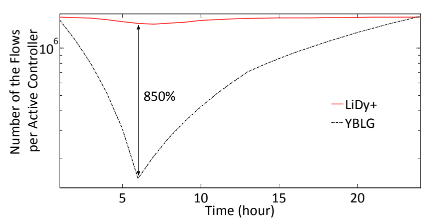
(c) Number of flows per active controller in the dense network



(d) Number of active controllers in the sparse network



(e) Number of active switches per active controller in the sparse network



(f) Number of flows per active controller in a sparse network

Figure 4: Performances comparison between *LiDy+* and *YBLG* in the large scale network

respectively. These results show the superior performance of *LiDy+* compared to *YBLG*. *LiDy+* activates a limited number of controllers to manage the same functionalities offered by *YBLG*. In both cases (dense and sparse networks), *LiDy+* reduces the number of active controllers by up to 80%. Unlike *YBLG*, it uses a traffic load prediction technique. Thus for some time periods, it activates more controllers than the required number, however, this fluctuation is limited to only 5% and only for a short period of time.

6.1.2 Controller Utilization of the Control Plane

In the network, the SDN controller controls a number of switches. The number of active switches controlled by the controller is used as a measurement of the utilization of controllers. Again, the controller manage the new flows generated by these switches. Thus, the average number of flows managed by a controller impacts its utilization: the higher the number of flows (or switches) an active controller manages, the more it is utilized. Fig. 4(b) and Fig. 4(e) show the controller utilization for the dense network and the sparse network, respectively. These figures show the variation of the number of active switches per active controller with respect to the time of the day. For the dense network, and the sparse network, *LiDy+* increases the number of active switches per active controller at maximum of 600% and 560% respectively, compared to *YBLG*. As already mentioned, the traffic load prediction technique used by *LiDy+* overestimates the traffic loads of switches. Because of this overestimated traffic loads, it activates more controllers than the required number at any time. Thus for some time periods it shows inefficiency compared to *YBLG* up to a certain limit, i.e., 5%, for a certain period of time.

Conversely, Fig. 4(c), and Fig. 4(f) show that the average number of flows managed by each active controller for the dense network, and the sparse network, respectively. These results show that *LiDy+* increases the number of flows per active controller up to 950% and 850% compared to *YBLG* for dense network and sparse network, respectively. Both results of Figure 4(c) and Figure 4(f) show that for different networks, i.e., dense and sparse networks, the average number of flows per active controller when using *LiDy+* remains almost unchanged. It is close to the maximum capacity of the considered *NOX-MT* controller, i.e., 1.6 *Mfps*. This shows that *LiDy+* maximizes the utilization of controllers.

6.1.3 Power Consumption of Controllers

Fig. 5, and Fig. 6 show the power consumption of the control plane (or the SDN controllers) with respect to the time for the dense and sparse networks, respectively. For both type of networks, *LiDy+* reduces the power consumption up to 80% compared to *YBLG*. Although for most of the time period, i.e., 92% of the day, *LiDy+* offers significantly lower power consumption than *YBLG*. But for a certain time time period of a day, i.e., 8%, its power consumption efficiency decreases to a maximum of 5% for the dense network and 6% for the sparse network. This happens due to its traffic load overestimates.

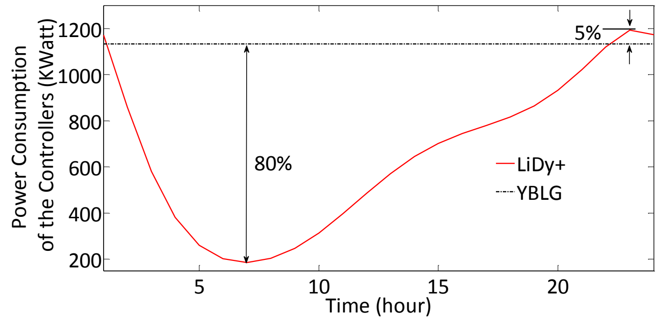


Figure 5: The power consumption of SDN controllers for the dense network

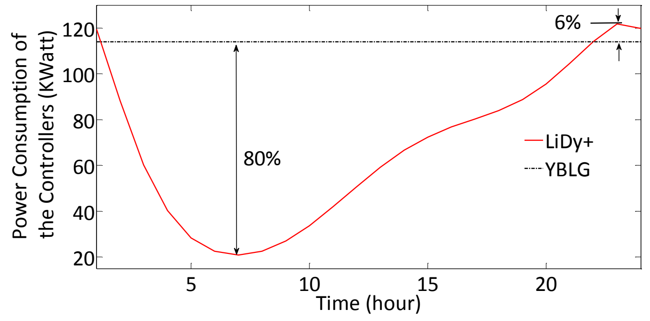


Figure 6: The power consumption of SDN controllers for the sparse network

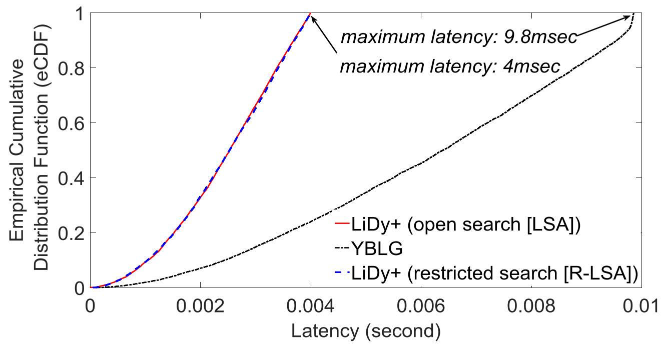


Figure 7: eCDF of latencies of switches with respect to their corresponding SDN controllers for the dense network

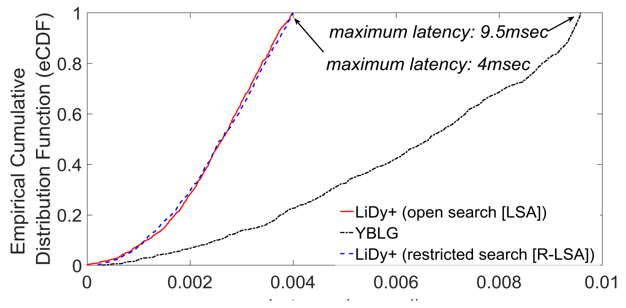


Figure 8: eCDF of latencies of switches with respect to their corresponding SDN controllers for the sparse network

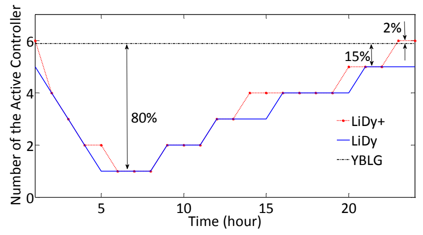
6.1.4 The Set-up And Maintenance Cost of Controllers

According to the system model of Section 5, the setup cost of controllers is fixed for each type of networks but the maintenance cost of controllers is variable. The setup cost depends on the total number of controllers of the network whereas the maintenance cost is proportional to the number of active controllers (or, the power consumed by active controllers) at any time. Since *LiDy+* activates less controllers than *YBLG*, as shown in Fig. 4(a) and Fig. 4(d), it offers a lower power consumption, as shown in Fig. 5 and Fig. 6. Thus *LiDy+* leads to a lower maintenance cost compared to *YBLG*. It reduces the maintenance cost by up to 80%. On the contrary, the setup cost (or, controllers' installation cost) of *LiDy+* is higher that *YBLG*. Because of its overestimate of the traffic load, *LiDy+* uses more controllers than the necessary. It increases its setup cost to 3.5% compared to *YBLG*.

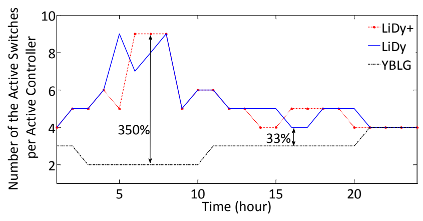
6.1.5 Effect of the Latency

The simulation results of Fig. 7 and Fig. 8 show the latency profile of *LiDy+* and *YBLG* for the dense network and the sparse network, respectively. For both types of networks, i.e., dense and sparse networks, *LiDy+* experiences a maximum latency of 4 *msec*. Whereas the maximum latency of *YBLG* is 9.8 *msec* and 9.5 *msec* for the dense network and the sparse network, respectively. In addition, the comparison in term of latency profile between the *open search* and the *restricted search* using *LiDy+* is also shown in Fig. 7 and Fig. 8. Here, in case of the *restricted search* technique, we have considered the worst case scenario where controllers can only be place at the same location of any given switch. Results of Fig. 7 and Fig. 8 show that for both type of networks (sparse and dense), the latency profile of the *open search* and the *restricted search* using *LiDy+* is almost similar.

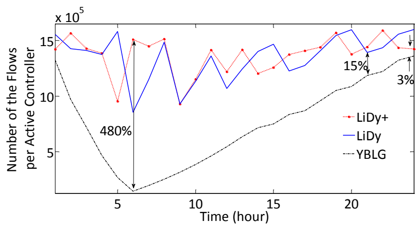
One of the goals of *LiDy+* is to maintain the latency between the switch and the controller within the maximum latency bound of 4 *msec*. Thus *LiDy+* ensures the latency bound in all environments even considering either the *open search* technique or the *restricted search* technique.



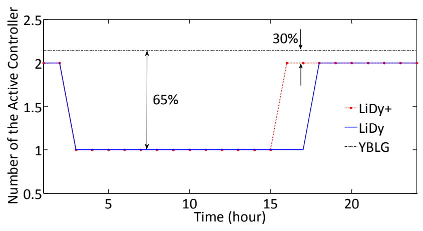
(a) Number of active controllers in the dense network



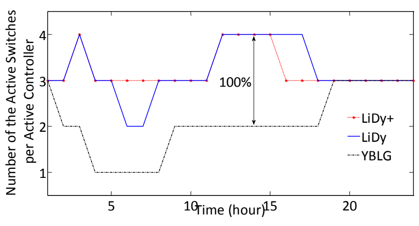
(b) Number of active switches per active controller in the dense network



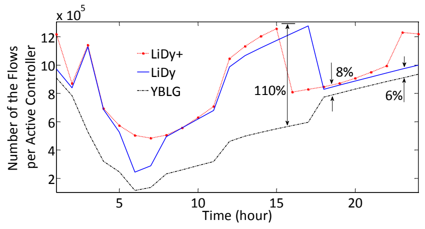
(c) Number of flows per active controller in the dense network



(d) Number of active controllers in the sparse network



(e) Number of active switches per active controller in the sparse network



(f) Number of flows per active controller in the sparse network

Figure 9: Performances comparison among *LiDy+*, *LiDy* and *YBLG* in the small scale network

6.2 Small Scale Experimental Result Analysis

6.2.1 Number of Active Controllers of the Control Plane

The number of the active controller variation of *LiDy+*, *LiDy*, and *YBLG* with respect to time is shown in Figure 9(a), and Figure 9(d). Figure 9(a), and Figure 9(d) show the variations for the dense network, and the sparse network respectively. For the sparse network, *LiDy+*, and *LiDy* decreases the number of active controllers from 30% to 65%, shown in Figure 9(d), compared to *YBLG*. For the dense network, *LiDy* decreases the number of active controllers from 15% to 80%, shown in Fig. 9(a). Although for the dense network, *LiDy+* decreases the number of the active controller at maximum of 80% compared to *YBLG*; but because of its overestimating traffic load prediction technique, the number of the active controller is increased to 2% compared to *YBLG* for a short period of time. Fig. 9(a), and Fig. 9(d) show that both *LiDy+*, and *LiDy* offer the same performance compared to *YBLG*. These figures also show that with the increasing number of switches (from the sparse network to the dense network), the difference in the number of maximum active controllers is also increased for both of these techniques (*LiDy+*, *LiDy*), compared to *YBLG*.

6.2.2 Controller Utilization of the Control Plane

The controller utilization is measured by (i) the number of active switches controlled by the controller and (ii) the average number of flows managed by the controller. The higher the number of flows (or switches) an active controller manages, the more it is utilized.

Fig. 9(b) and Fig. 9(e) show the variation of the number of active switches per active controller for the dense network and sparse network, respectively. These variations are shown for a day. For the dense network, *LiDy+*, and *LiDy* increases the number of active switches per active controller from 33% to 350% compared to *YBLG*. Again for the sparse network, *LiDy+* and *LiDy* increase the number of active switches per active controller up to a maximum of 100% compared to *YBLG*. In both cases, *LiDy+* and *LiDy* show very similar performance.

Figure 9(c) and Figure 9(f) show the variation of the average number of flows managed by each active controller for the dense network and the sparse network, respectively. Here, *LiDy+* increases the average number of flows managed by each active controller from 3% to 480% for the dense network, and from 8% to 110% for the sparse network, compared to *YBLG*. Again, *LiDy* increases the average number of flows managed by each active controller from 15% to 480% for the dense network, and from 6% to 110% for the sparse network, compared to *YBLG*. Since, *LiDy+* shows almost the same performance as *LiDy*, thus we can conclude that both *LiDy+*, and *LiDy* offers similar controller utilization compared to *YBLG*.

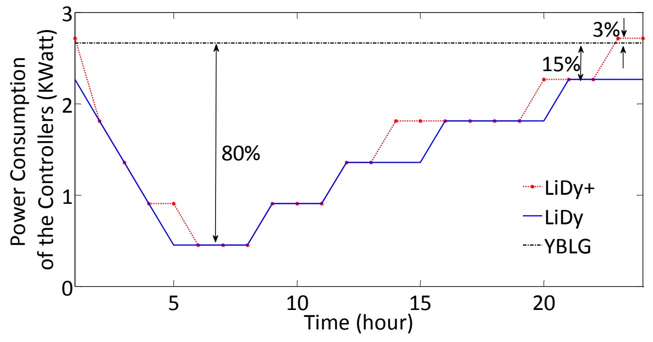


Figure 10: The power consumption of SDN controllers for the dense network

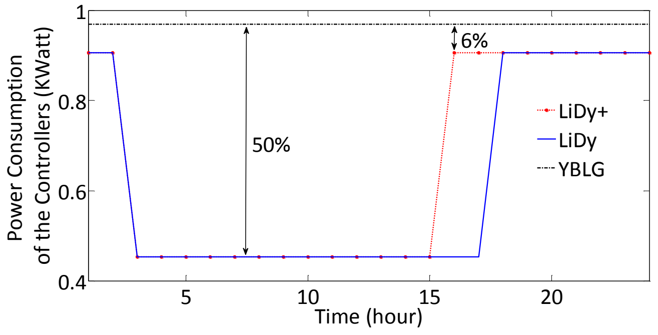


Figure 11: The power consumption of SDN controllers for the sparse network

6.2.3 Power Consumption of Controllers

Fig. 10, and Fig. 11 show the power consumption of the control plane (or, the SDN controllers) for the dense network, and the sparse network, respectively. The power consumption variations of *LiDy+*, *LiDy* and *YBLG* are shown with respect to the time of the day. For the sparse network, the power consumption is decreased from 30% to 65%, as shown in Figure 11, by using *LiDy+*, and *LiDy* compared to *YBLG*. For the dense network, both *LiDy+* and *LiDy* reduce the power consumption up to 80% when compared to *YBLG*. Although for the dense network both *LiDy+* and *LiDy* have the almost same power consumption, *LiDy+* shows an increased power consumption, 3% compared to *YBLG* for a certain period of time. It occurs because of activating more SDN controllers due to *LiDy+*'s overestimate of the traffic load.

6.2.4 The Setup And Maintenance Cost of Controllers

The setup and the maintenance cost of controllers is the sum of the setup cost, and the maintenance cost. Since the setup cost is almost 40 times higher than the maintenance cost, this combined cost mostly depends on the

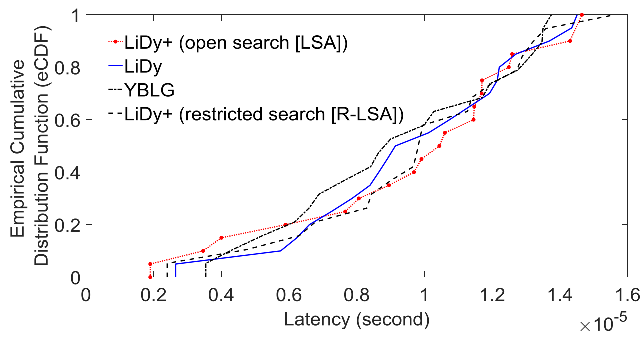


Figure 12: eCDF of latencies of switches with respect to their corresponding SDN controllers for the dense network

setup cost of controllers. Again, the setup cost is fixed, hence the setup and the maintenance costs of controllers are also almost static for different networks. The simulation results show that for the dense network, the setup and the maintenance costs of both *LiDy* and *YBLG* are similar, i.e., *AU*\$7350, while *LiDy+* reduces the cost around 15% compared to them. Again for the sparse network, both *LiDy+* and *LiDy* achieve 6% cost efficiency compared to *YBLG*.

6.2.5 Effect of the Latency

Figure 12 shows the latency profile of *LiDy+*, *LiDy* and *YBLG* for the dense network. This empirical cumulative distribution function (eCDF) of the latency between the switch and its controller illustrates that there is no significance latency variations among all of three controller placement techniques. Figure 12 also shows that the latency profile of *LiDy+* using either the *open search* or the *restricted search* is quasi-identical. Here, in case of the *restricted search* technique, we have once again considered switches locations as restricted locations for controllers.

6.3 Parameter Impact Analysis

6.3.1 Effect of the Maximum Latency Bound (D_{req})

The maximum latency bound (D_{req}) of *LiDy+* was varied in the range 1 *msec* to 10 *msec*. In all cases, the number of active controllers were almost unchanged. There was a negligible variation, i.e., 0.5%, in the number of active controllers. Again, from Figure 7 and Figure 8 we know that *LiDy+* exactly follows the *Maximum Latency Bound*. Thus, the *Maximum Latency Bound* has no effect on the number of active controllers.

The number of the controller modules depends on the maximum latency bound (D_{req}) of the network. The relation between these two parameters is inversely proportional. If the value of D_{req} is increased then the number of controller modules is decreased, and vice versa. But we found, in the experimental analysis, that both of these parameters have no impact on the total number of controllers of the network. We showed already that the variation in the number of controllers change the network performances (controller utilization, power consumption, and maintenance cost). Since the number of controller modules does not change the total number of controllers, controller modules have no effect on the network performances (controller utilization, power consumption, and maintenance cost).

6.3.2 Effect of the Stopping Criteria (D_{stop})

Surprisingly the simulation results of *LiDy+* for different stopping criteria (D_{stop}) were almost similar. We considered six different values of D_{stop} (from $n = 1$ to $n = 100$ in Equation (1)). The fluctuation in the number of active controllers was less than 1%. We also observed that the *Stopping Criteria* (D_{stop}) has no effect on the latency of the network. Again, the number of controllers and controller modules does not depend on D_{stop}.

All of these above mentioned analysis of D_{req}, and D_{stop} is based on the system model presented in Section 5.

6.4 Time Complexities

LiDy+ consists of three algorithms: LICMP, LSA (or, R-LSA) and DyFlow. LICMP and LSA (or, R-LSA) are for the controller placement, and DyFlow is for the load adaptation. LICMP uses a loop of $O(n)$ iterations, where n is the number of switches to control. In each iteration, LSA (or, R-LSA) is invoked. LSA (or, R-LSA) mainly calls the Welzl's algorithm [24] to compute the smallest enclosing disks, which has the complexity of $O(n)$. So the overall complexity of LICMP is $O(n^2)$. The time complexity of DyFlow algorithm is $O(m)$, where $m \leq n$ is the number of controllers of the controller module. Therefore, the time complexity of LiDy+ is $O(n^2)$.

Although *YBLG* was not analyzed in the paper, its time complexity can be obtained as follows. *YBLG* first calculates the pair-wise distances among all switches, and then sorts the n^2 distance values obtained in ascending order, which needs $\Theta(n^2 \log n)$. Then, it conducts a binary search on these n^2 distance values. In each comparison during the search, it needs to solve a linear program. While the complexity of this linear program is unknown, we denote it by $\Theta(x)$ here. So the complexity of the second step in *YBLG* is $\Theta(x \log n)$. Combining these two steps, the complexity of *YBLG* becomes $\Theta(n^2 \log n)$, which is asymptotically higher than *LiDy+*.

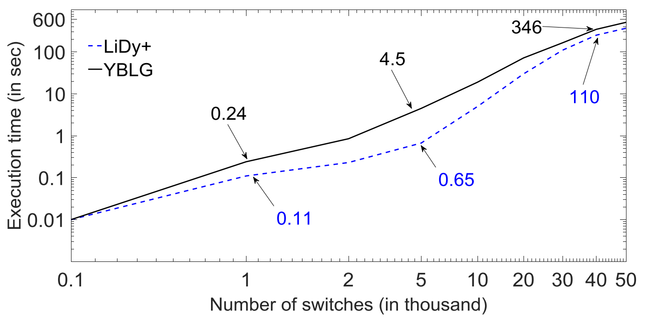


Figure 13: Simulation run time to place controllers using *LiDy+* and *YBLG*

From the above analysis we can conclude that *LiDy+* uses a lower number of controllers and achieves a better utilization than *YBLG*. *LiDy+* also offers a better latency profile, lower computational time complexity and reduces the system maintenance cost compared to *YBLG*. *LiDy+* has all features of *LiDy*. On top of that, it is more computationally efficient than *LiDy*, and uses a traffic load prediction technique.

7 Discussion

In this section, we discuss how *LiDy+* differs from other research works of the SDN controller placement problem. The distinguished features of *LiDy+* are as follows,

1 Inherited from *LiDy*, the controller module concept is introduced. Because of the controller module, controllers in the network are physically less distributed. Moreover, simulation results of Section 6 show that the use of the controller module increases the controller utilizations.

2 Our proposal is to search the entire region containing switches to get locations of SDN controllers. To do so, we have used the *open search* technique. Although this searching technique ensures the maximum utilization of controllers, it is only applicable in the best case scenario. Thus this technique can be used as a theoretical approach to solve the controller placement problem in SDN, and a reference to analyze the impact of placing controllers in the very large scale network before its real-world implementation. It helps to stretch to the maximum limit of the control plane utilization, in term of the power consumption, controllers' utilization, control plane maintenance cost. In addition, considering the real-world implementation, *LiDy+* can adopt the *restricted search* technique, instead of the *open*

search technique. The simulation results of Fig. 7, Fig. 8 and Figure 12 present that *LiDy+* offers similar latency profile using either the *open search* technique or the *restricted search* technique. Since the effect of using the searching (either the *open search* or the *restricted search*) technique is only limited to the latency profile of *LiDy+*, thus the other performance metrics, e.g., controllers' number, their utilizations, their power consumptions, their set-up and maintenance cost remain unchanged.

3 Recent research advances, listed in Section 2, of the SDN controller placement problem consider both the latency and the traffic load of switches to place controllers in the software defined network. To place the SDN controller, a common approach of these research results was 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, controllers would need to be replaced frequently, which is not practically feasible. Considering this dilemma, the latency and the traffic load of switches are separately considered in *LiDy+*. The related simulation results of Section 6 illustrate the benefit of this separate consideration.

4 *LiDy+* has a lower time complexity than *LiDy*. It is specially designed for the large-scale software defined network. Unlike *LiDy*, *LiDy+* uses a prediction technique to dynamically estimate the traffic load of switches in the network.

8 Conclusion and Future Work

LiDy+ is a modified version of *LiDy* that is applicable to software defined networks at large scale. As far as we know, *LiDy+* is the most efficient controller placement solution in which dynamic traffic load of switches was taken into consideration. It offers a combined solution of the controller placement problem and the controller provisioning problem by taking as inputs the locations of switches and the required latency constraint. It outputs the locations of controller modules and the number of active controllers of each controller module. Although the first output is static, the second one is dynamic. *LiDy+* achieves higher utilization at a lower cost than the previous CPP solution technique, called *YBLG*. Moreover, compared to *LiDy* and *YBLG*, *LiDy+* offers lower computational complexity and latency.

Although in this paper we have emphasized the use and the importance of the *open search* technique to maximize the utilization of controllers, we have also offered an alternative search technique for those scenarios in which the *open search* technique is not applicable. As future work, it would be also interesting to combine *LiDy+* with a new prediction algorithm that could get a better performance by reducing the overestimated traffic load of switches. In addition, we plan to relax some of our assumptions and work on a real-time scenario in which all switches are not fully connected with controllers.

References

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

[2] 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.

[3] D. Erickson, "The Beacon OpenFlow controller", *in Proceedings of the 2nd ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 1318, New York, USA, 2013.

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

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] M. T. Huque, G. Jourjon, V. Gramoli, "Revisiting the controller placement problem", *IEEE 40th Conference on Local Computer Networks*, pp.450-453, 26-29 October 2015.

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

[17] S. Mallon, V. Gramoli, and G. Jourjon, "Are Today's SDN Controllers Ready for Primetime?", *The 41st IEEE Conference on Local Computer Networks*, 7-10 November 2016.

[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.

[24] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)", New Results and New Trends in Computer Science (H. Maurer, Ed.),*Lecture Notes in Computer Science 555*, pp. 359370, 1991.

[25] S. C. Chapra, R. P. Canale, "Modelling, Computers and Error analysis", *Numerical methods for engineers*, 6th edition, McGraw-Hill, 1998.

[26] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani,R. Ahmed and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks", *IEEE CNSM 2013*, pp. 18-25, 14-18 October 2013.

[27] Y. Zhao, L. Iannone and M. Riguidel, "On the performance of SDN controllers: A reality check", *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 79-85, San Francisco, USA, 18-21 November 2015.

[28] B. Grubb, *These graphs show the impact Netflix is having on the Australian internet*, Digital Life News, The Sydney Morning Herald, 2 April 2015.