Towards Controller Placement for Robust Software-Defined Networks

Sheng Guo*†, Shu Yang*, Qi Li*, Yong Jiang*
*Graduate School at Shenzhen, Tsinghua University, Shenzhen, China, 518055
†Department of Computer Science and Technology, Tsinghua University, Beijing, China, 100084

***Abstract*—The core concept of software-defined network (SDN) is the separation between control plane and date plane. SDN provides a programmatic interface to network control, significantly simplifies network management and improves the efficiency of network utilization. To further improve network performance, scalability and reliability, it is recommended to deploy multiple controllers in SDN. However, network performance would degrade if operators randomly deploy the controllers, especially in the case of failure, e.g., router crashes, fiber cuts, etc.**

In this paper, we try to optimally place controllers while taking network failures into account. First, we formally define two problems, 1) Controller Placement under Comprehensive Network States *(CPCNS)* problem; and 2) Controller Placement under Single Link Failure *(CPSLF)* problem. Secondly, We propose a network states traversal based algorithm, which optimally solve the problem; and further propose another greedy-based algorithm, which can solve the problem in polynomial time. Finally, we evaluate the algorithms using real topologies and empirical data. The results indicate that the new controller placement strategies can significantly improve the performance when link failures happen.

***Index Terms*—software-defined network, controller placement, network optimization**

I. Introduction

Recently Software-Defined Networks (SDN) has attracted more and more attentions, due to its separated and centralized control plane logic, where a set of dedicated controller instances each manages one or more simplified packet-forwarding switches. A range of academic prototypes and industry products have emerged, such as the plane of 4D [1], [2] Ethane [3], OpenFlow-based controllers [4], [5] and enterprise wireless controller with CAPWAP access points [6].

SDN greatly simplifies control plane design and improves convergence. However, reliability and scalability should be more carefully considered in SDN networks. Generally, there are two interfaces in a controller: 1) the northbound interface which propagates network state and receives control information; 2) the southbound interface which communicates with the switches. Logically, the controller works in a centralized manner, and may become a choke point, especially in a large-scale network, such as a WAN. The capacity of controllers, including the CPU resources, memory, and input/output bandwidth are limited. Therefore, to avoid being a bottleneck, especially when link failures happen, multiple controllers are necessary.

To further improve network performance, including latency and robustness, operators should carefully place the controller [7]. Moreover, the controller placement problem needs to be more carefully designed when considering security issues, e.g., attack on the controllers, and link failures. In SDN networks, the propagation delay between switches and controllers affects flows setup time and network convergence time. Given an arbitrary number of controllers, the random placement may even degrade the network performance. Thus, an algorithm is needed for network operators to compute the controller placement, which leads to better performance, even when failure happens. This is because failures can change network topologies and degrade SDN performance. Fortunately, we are not the first one to address the problem. However, optimizing each metric is generally NP-hard, it is important to find an efficient placement algorithm.

Traditional optimization based on worst case latency ignores network failures. One of our simulation results shows that the worst case latency changes within a wide range in the case of failures and the minimum worst case latency hardly acquired when link failures happen from another point of view.Furthermore, a lot of works have been done to study the features of network failures. They find that links differ widely in their failure characteristics, which motivates us to explore a SDN failure model and optimize controller placement decision under this observation.

In this paper, we improve the network performance of SDN by employing optimal controllers placement strategies, especially under network failures. We minimize the propagation latency of control packets and formalize it as an optimal problem. To address the problem in different network failure scenarios, i.e., single failure and multiple failure scenarios, we develop a SDN failure model to describe network failures in SDN and formalize two optimization problems according to the model, i.e., Controller Placement under Comprehensive Network States *(CPCNS)* problem and Controller Placement under Single Link Failure *(CPSLF)* problem, which are NP-hard problems. Correspondingly, we put forward a network states traversal based algorithm, which optimally solve the problem. Furthermore, We propose a greedy algorithm to solve the problem and implement controller placement strategies with these algorithms. Finally, we evaluate the performance of the algorithms by simulations using real topologies. The simulation results demonstrate that the proposed controller placement strategies provide good solutions to the problem

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

we defined.

We make the following contributions in this paper:

1) we formalize a SDN failure analysis model to describe network failures where network state latency is defined as the optimization objective.

2) We develop two optimization problems to minimize network state latency and propose two algorithms to solve them and implement control placement strategies.

3) We evaluate our control placement strategies using real topologies and observe that they can properly solve the problem.

II. Background and Related Work

In SDN, the communication between the control plane and data plane could utilize standard transport layer security (TLS) or transmission control protocol (TCP). The whole control plane formed by hundreds of controllers must keep connecting with themselves in order to achieve a consistent global view of the network state. Generally, the Link Layer Discovery Protocol (LLDP) is applied to establish the communication path between switch and controller. Once link failure happens, switches at both ends of the failed link will try to report it to control plane and the controller recomputes the communication path, which means the switch would not lose control as long as there is one path working.

OpenFlow [8] is the most popular SDN prototype which derives from an academic research project and has a wide range of usage in industry, where a central OpenFlow controller defines rules for switches how to forward packets, thus enabling a centralized routing control. Applications that deal with multiple controllers must carefully design the distribution strategy and consistency model since some information may need to be accessible from all controllers and data need to be synchronized. OpenDaylight [9] is an open platform for network programmability with clustering as an integral feature.

The first work motivated the controller placement problem is [7]. The author points that in long propagation delay WANs, the placement scheme places fundamental limits on availability and convergence time and it has practical implications for network design, affecting whether they must push forwarding actions to forwarding elements in advance.

The logically single SDN control plane must maintain a unique Network view upon the whole network through each controller periodically synchronize their databases. Several design scheme of SDN [5], [10], [11] try to support the construction and interaction mechanism.

A lot of controllers have declared excellent performance e.g., [12]–[14], propagation delay is not the only factor that should be focused since the capacity of a controller is limited. In [15], the author defines a capacitated controller placement problem (*CCPP*), took the load of a controller into consideration and introduced an effective algorithm to solve the problem.

Normally, the mapping between a controller and a switch is configured statically, which will become a key limitation when considering the issues of scalability of the distributed controllers. In [16], the author proposes an elastic distributed controller architecture(*ElastiCon*), which dynamically assigns switches to controllers according to traffic conditions. However, this does not mean the strategies of controller placement is meaningless in this article since the locations of the controllers impact the performance of networks and will become more sensitive considering the issue of reliability.

In [17], the authors consider different aspects of the controller placement problem, such as the maximum latencies between nodes and controllers, failure tolerance issues as well as load balancing, finding that the optimal values for the metric quality and resilience are impossible to acquire simultaneously time and trade-offs should be considered.

The most similar work to us is [18], the authors introduce the Fault Tolerant Controller Placement Problem and present a heuristic algorithm which computes controller placement with at least five nines reliability. They find that each node is required to connect to 2 or 3 controllers, which typically provide ample reliability. The authors do not optimize the propagation delay, however, which is a key factor to network performance.

III. Problem Definition

A. Placement Constraints

For a network denoted by $G(V \cup F, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes(switches or routers) and F consists of the different facility locations where a controller can be deployed (We assume every node to be a candidate of the facility location, so we would not make the distinction between F and E in the following chapters). E is the set of physical links. $e_{ij} = (v_i, v_j) \in E.$, let n be the number of nodes, i.e., $n = |V|$. The set of controllers is denoted by $\Theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$. Each link $e_{ij} \in E$ has a cost(propagation delay) c_{ij}. If v_i and v_j are not neighbours, $c_{ij} = \infty$. The link failures between each node are statistically independent. The whole solution space of controller placement schemes is denoted by $P = \{p_1, p_2, \ldots, p_l\}$ And for $p \in P$, $\theta \in \Theta$, controller θ is placed at a facility's location denoted by $\psi_p(\theta)$, $\psi_p(\theta) \in F$. Then P can be formalization described by:

$$p = \{(\theta_1, \psi_p(\theta_1)), (\theta_2, \psi_p(\theta_2)), \ldots, (\theta_k, \psi_p(\theta_k))\} \quad (1)$$

$\phi_p(v)$ denotes the controller that node v be assigned to under p. $Q_{\phi_p}(\theta)$ denotes the set of switches controlled by controller θ. Then an assignment from controllers to switches is denoted by:

$$Q_{\phi_p} = \{(\theta_1, Q_{\phi_p}(\theta_1)), (\theta_2, Q_{\phi_p}(\theta_2)) \ldots, (\theta_k, Q_{\phi_k}(\theta_k))\} \quad (2)$$

When $\{\theta_1, \theta_2, \ldots, \theta_k\}$ are placed completely, then form a solution p_i, we assign each switch to the controller which has the minimum propagation latency between each other. So $Q_{\phi_p}$is dominated by p.

The worst case latency is an important metric which is defined as the maximum node to controller propagation delay:

$$C_{wc}(p) = \max_{v \in V} \min d(v, \phi_p(v)) \tag{3}$$

We also seek the optimal placements p from P, such that $C_{wc}(p)$ is minimum.

B. Network Failure Model

Since packet delivery is the basic function of the network layer, the resilience is of great importance to the network performance. Unfortunately, network failures frequently occur. Failures can happen at different protocol layers in the network for various reasons. A failure of optical equipment or a fiber cut may lead to loss of physical connectivity at the physical layer. The loss of connectivity happened between switches may also stem from software errors, switch processor overloads, protocol implementation or misconfiguration errors. Additionally, failures may occur due to scheduled network maintenance or just be unplanned. In the controllers placement problem, We consider the following two kinds of failures:

a) Node failure: A node failure could happen in the form of a switch failing to forwarding packets or losing control from its controller. Besides, It also can be a controller becoming unable to handle packets or managing forwarding rules. This kind of failures connotes that the switch or controller cannot be proper functioned anymore. Software errors, hardware failures, misconfigurations or network attacks can cause such kind of failures.

b) Link failure: When link failure happens, all packets passing through the failed link will be lost. Link failures can be caused by fiber cuts, equipment upgrades, maintenance operations, etc.

We name a network being the intact state when no failure happens. Either node failures or link failures lead to network state transforming.

For a given network $G(V \cup E)$, $n = |V|$ and $m = |E|$, the state space is 2^{n+m}. Let S be the set of whole states, then $S = \{s_1, s_2, \ldots, s_l\}$ i.e.,$l = 2^{n+m}$. Use indicator variables χ_s^v and χ_s^e denote states of node v and link e under network state s, then s can be formally denoted by:

$$s = \{\chi_s^{v_1}, \ldots, \chi_s^{v_n}, \chi_s^{e_1}, \ldots, \chi_s^{e_m}\} \tag{4}$$

C. Controller Placement under Comprehensive Network States

As mentioned above, we often use worst case latency $C_{wc}(p)$ to measure network performance under controllers placement p, $p \in P$. For network $G(V \cup E)$, define ρ_v and ρ_e as the probability of node $v \in V$ and $e \in E$ that function well. Use ρ_s^v and ρ_s^v to denote the state probability of node v and link e under network state $s \in S$, which means if $\chi_s^v = 1$,then $\rho_s^v = \rho_v$ otherwise $\rho_s^v = 1 - \rho_v$ when $\chi_s^v = 0$. ρ_s^e has the similar situation. Then the occurrence probability of network state $s \in S$ can be denoted by:

$$\rho_s = \prod_{v \in V} \rho_s^v \prod_{e \in E} \rho_s^e \tag{5}$$

We use $C_{wc}^s(p)$ to denote the worse case latency of the whole network G under state s and controllers placement p. For each placement p, the assignment scheme is decided by p under intact network state, which means the assignment between switches and controllers remain unchanged even if state transitions happen. In this way, avoiding the complexity of control status migration between different controllers could achieved. Then the network state latency under solution p can be defined as:

$$C_{wc}^S(p) = \sum_{s \in S} C_{avg}^s(p) \rho_s \tag{6}$$

Definition 1. *Controller Placement under Comprehensive Network States (CPCNS):*

$$\min C_{wc}^S(p) \tag{7}$$

s.t.

$$p \in P, s \in S \tag{8}$$

The *CPCNS* is an NP-Hard problem since the optimization goal can be viewed as a polynomial combination of the one of traditional K-centre problem which is another NP-Hard problem already been proved. The computational complexity of solving the *CPCNS* is too large according to various network states constructed by a large amount of nodes and links. Otherwise, we rather try to solve a pure mathematical problem than taking it into network scenarios. Furthermore if a number of nodes or links are out of working, there is no effective placement strategy can ensure a better network performance.That is why we should filter out some extreme situations. The work in [19] shows that 70% of the whole network failures affect only single link at a time and links differ widely in their failure characteristics. These observations have motivated us to explore a more simple network failure model and focus on single link failure.

D. Controller Placement under Single Link Failure

Suppose the network states is divided into two categories, namely, the intact state and single link failure states. The intact state has just one scenario that all the links and nodes work well, and single link failure states consist of m scenarios which means each link $e \in \{e_1, e_2, \ldots, e_m\}$ may fail at a time. Then we get $m + 1$ network states, denoted by $S' = \{s_0, s_1, \ldots, s_m\}$. Given ρ_e, the probability of link e operating well, $e \in \{e_1, e_2, \ldots, e_m\}$, then the occurrence probability of intact state is:

$$\rho_{s_0} = \prod_{e \in E} \rho_e \tag{9}$$

Therefore the probability of single link failure states is $1 - \rho_{s_0}$, which is shared altogether by m scenarios. The occurrence probability of each scenario reflects the probability of each link failure. In the effort of normalization, we define $F = \sum_{e \in E} (1 - \rho_e)$ to be the sum of probability for each link failure. Then the relative probability of each link $e_i \in E$ is :

$$\hat{\rho_{e_i}} = \frac{1 - \rho_{e_i}}{F} = \frac{1 - \rho_{e_i}}{m - \sum_{e \in E} \rho_e} \tag{10}$$

For each single link failure scenario $s_i \in \{s_1, s_2, \ldots, s_m\}$, we suppose e_i to be failed. Then the occurrence probability of scenario $s_i, i \in 1, 2, \ldots, m$, is:

$$\rho_{s_i} = (1 - \rho_{s_0})\hat{\rho_{e_i}} = \frac{(1 - \prod_{e \in E} \rho_e)(1 - \rho_{e_i})}{m - \sum_{e \in E} \rho_e} \tag{11}$$

We also use $C^s_{wc}(p)$ to denote the worse case latency of network G under state S as well as controller placement scheme p, then the S scenarios is replaced by S' since only single link failure states are taken into consideration.

Definition 2. *Controller Placement under Single Link Failure (CPSLF):*

$$\min C^{S'}_{wc}(p) \tag{12}$$

s.t.

$$p \in P, s \in S' \tag{13}$$

$C^{S'}_{wc}(p)$ is an irreducible polynomial over $C_{ws}(p)$, at the same time, computing $C_{ws}(p)$ is NP-Hard. Therefore, Calculating the *CPSLF* is also an NP-Hard problem.

IV. PLACEMENT ALGORITHM

To solve the *CPSLF* problem, the optimal placement algorithm is proposed to be an evaluation baseline for estimating other algorithms. Since it will exhaust computer resource and run too much time if a Software-Defined network contains hundreds or even thousands of switches, we also propose a greedy placement algorithm based on optimal placement algorithm, which has a lower time complexity without much performance degradation.

A. Optimal Placement Algorithm for CPSLF

The Controller Placement in Single Link Failure State *(CPSLF)* problem is NP-hard, which means if we want to work out the optimal placement scheme, the space of whole placement schemes need to be traversed. We develop the Optimal Placement Algorithm for *CPSLF* based on state-first search. The main idea of state-first search, obviously, is to traverse all the network states for each placement scheme. For each state, we calculate the worst case latency and multiply it by state occurrence probability and then we add up all the latencies to form the final network state latency of that scheme. We can optimize the algorithm during the states traversal procedure by comparing the current network state latency under current state with the global network state latency calculated in the previous steps. If the current network state latency is larger than the global network state latency calculated in previous state traversal, it means in the previous steps, we already found some placement scheme which has a better performance than current scheme traversal procedure. By this way we can avoid some unnecessary traversal steps. Since most of the procedures are similar to the following Greedy Placement Algorithm for *CPSLF*, details is omitted.

The first step of the algorithm is to calculate and store the shortest distance between any two nodes which cost $O\left(|V|^3\right)$ times. We have $\binom{|V|}{k}$ placement schemes to traverse. In each scheme traversal, calculating the worse case latency under $|E| + 1$ network states costs $O\left((|E| + 1)k|V|^2\right)$ times. So the complexity of Optimal Placement Algorithm for *CPSLF* is bounded by $O\left(k|E||V|^{2+k}\right)$.

B. Greedy placement Algorithm for CPSLF

The complexity of Optimal Placement Algorithm for *CPSLF* is exponential which is still too high for computing large scale Software-Defined Network. However, if k is constant, the complexity will become polynomial. To reduce computing complexity, we develop the Greedy Placement Algorithm for *CPSLF*. The main idea of the algorithm is to place k controllers iteratively. The input network data of current iterative process is the output result of the previous iterative process. Only iterate k times would we get the final placement scheme. Although most of the time, it would not be optimal, still acceptable. We will discuss the simulation results afterwards. The idea of Greedy Placement Algorithm For *CPSLF* is detailed in algorithm 1. The input datum include a network $G = (V, E)$, V is the set of nodes and E is the set of links. C is the set of propagation delays and $\mathcal{P}$ is the set of operation probabilities of all links. We also need to give the number of controllers, which is symbolized by k. The output result include a subset of V,denoted by Θ which means the set of places that controllers should be deployed. Q_{θ_p} is the collection of sets, and each set represents an administrative area, meaning the nodes are assigned to a same controller.

The first steps are some variable assignments and since we need distance operation in many next steps, it is better to pre-compute the shortest distances between any two nodes and store them in the matrix. We assume every node is expressed numerically and then the rows and columns in the matrix can be viewed as the corresponding nodes. Mention that it takes $O(n^3)$ operations to get the distance matrix in a general graph, but there are more efficient algorithms [20] at least theoretically for sparse graphs. The $length\_matrix$ and $path\_matrix$ computed from function $CalShortestLength()$ store the shortest path distance and the involved links separately. We need to iterate k times to get the final placement scheme.During each iteration, we choose a new potential controller namely $candidate$ from can_list which denotes the set of nodes that haven't been placed controller and add it into cur_Θ, forming a new placement scheme. And then we compute assignments and the worst case latency applied by $length\_matrix$. Since they are computed in intact network state, so we multiply the worst case latency by intact state probability and add it to tmp_prob_dist, denoting the current network state latency. Afterwards, we consider the link failures by assign ∞ to each c_e separately, meaning the corresponding link is unavailable. Recompute the worse case latency and multiply it by current link failure probability, then add it to the tmp_prob_dist. At last compare the tmp_prob_dist to the network state latency $prob\_dist$ calculated from previous traversal steps. If tmp_prob_dist is smaller, it means we acquire a better placement scheme. Then assign it to network state latency $prob\_dist$ for next

Algorithm 1: Greedy Placement Algorithm for *CPSLF*

Data: $G = (V, E), C, \mathcal{P}, k$
Result: Θ with $\Theta \subseteq V$,Q_{θ_p}
begin
  $\Theta \longleftarrow \emptyset$; $Q_{\theta_p} \longleftarrow \emptyset$; $prob\_dist \longleftarrow \infty$
  $compli\_prob \longleftarrow \prod_{e \in E} \rho_e$
  $length\_matrix, path\_matrix \leftarrow$
  $CalShortestLength(G, C)$; $can\_list \leftarrow V$
  for $i \longleftarrow 1$ **to** k **do**
    $cur\_\Theta \longleftarrow \Theta$
    for $candidate \in can\_list$ **do**
      $placement\_matrix \longleftarrow \infty$; $plt \longleftarrow \emptyset$
      $tmp\_Q_{\theta_p} \longleftarrow \emptyset$
      $cur\_\Theta \longleftarrow cur\_\Theta + \{candidate\}$
      for $node \in cur\_\Theta$ **do**
        Calculate $\phi_p(node)$,
        $tmp\_Q_{\theta_p}$,max_min_dist
      $tmp\_prob\_dist \leftarrow compli\_prob * min\_dist$
      for $e \in E$ **do**
        $ori\_length \longleftarrow c_e$; $c_e \longleftarrow \infty$
        $fail\_dist \longleftarrow min\_dist$
        for $node \in V$ **do**
          if $e \in path\_matrix[node][\phi_p(node)]$ **then**
            $tmp\_dist \longleftarrow$
            $Shortestlength(node, \phi_p(node))$
            if $tmp\_dist > fail\_dist$ **then**
              $fail\_dist \leftarrow tmp\_dist$
        $c_e \longleftarrow ori\_length$;
        $tmp\_prob\_dist \leftarrow tmp\_prob\_dist +$
        $fail\_dist * \frac{(1-compli\_prob)(1-\rho_{e_i})}{|E|-\sum_{e \in E} \rho_e}$
        if $tmp\_prob\_dist > prob\_dist$ **then**
          break
      if $prob\_dist > tmp\_prob\_dist$ **then**
        $prob\_dist \leftarrow tmp\_prob\_dist$
        $cur\_can \leftarrow candidate$; $\Theta \leftarrow cur\_\Theta$;
        $Q_{\theta_p} \leftarrow tmp\_Q_{\theta_p}$
      $cur\_\Theta \longleftarrow cur\_\Theta - \{candidate\}$
    $can\_list \leftarrow can\_list - \{cur\_can\}$

traversal step. In each iteration process, we traverse all the $candidate$ in can_list and find the formed placement scheme with the minimum network state latency within the current iteration. Concurrently, store corresponding placement scheme and assignment scheme in Θ and Q_{θ_p}.

The first step of the algorithm is also to calculate and store the shortest distance between any two nodes which cost $O\left(|V|^3\right)$ times. The placement scheme space is reduced to $k\binom{|V|}{1}$. In each scheme traversal, calculating the worse case latency under $|E|+1$ network states costs $O\left((|E|+1)k|V|^2\right)$ times.The complexity of Greedy Placement Algorithm for *CPSLF* is bounded by $O\left(k^2|E||V|^3\right)$.

V. Performance Evaluation

A. Simulation Setup

We evaluate the performance of the Optimal Placement Algorithm *(OPA)* and Greedy Placement Algorithm *(GPA)* for *CPSLF* using the topologies from the Internet Topology Zoo [21] and Internet2 [22] as well as Cernet2. A case study on Internet2 and Cernet2 would be in the next section.

1) Topology: The Internet Topology Zoo is public available repository, from where we exclude all disconnected ones and final choose 124 topologies at the Point-of-Presence level. Hence, each PoP corresponds to a node in the network. The Internet2 in the continental US consists of 34 nodes and 42 links, in different cities, connecting via high-speed links. The Cernet2 plays as a counterpart of Internet2 in China mainland, which consists of 21 nodes and 23 links. We consider each PoP as a candidate for hosting a controller.

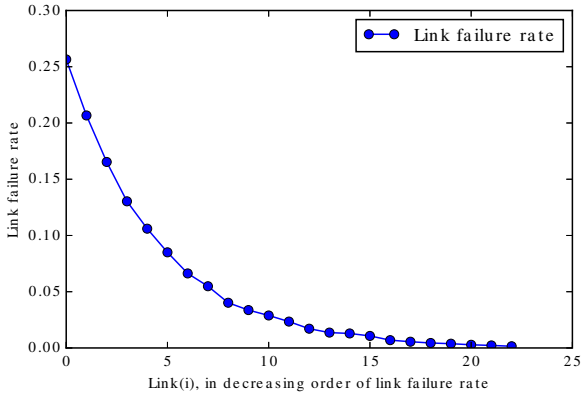

Fig. 1: Per-link failure rate, in descending order.

2) Operational Probability: The operational probabilities of links coming from Topology Zoo and Internet2 are assumed to follow different i.i.d. Weibull distribution. The probability density for the Weibull distribution is $p(x) = \frac{\alpha}{\lambda}(\frac{x}{\lambda})^{\alpha-1}e^{-(x/\lambda)^{\alpha}}$. We set $\alpha = 0.9$ and $\lambda = 15526$ and such configuration is intended to reproduce the "long tails" in the downtime distribution of Wan links reported in the literature. As for Cernet2, we apply authentic operational probability data collected from Oct. 10, 2008 to Nov. 2, 2008. During this period, we have observed 240 failures. The failure rate of each link is shown in Fig.1. We can clearly see that the failure shows a "long tails" feature. Actually, we can see that almost 60% of failures are caused by a small set of links, e.g., only 17% (4 out of 23 links). This character is also observed from the ASes of Sprint.

3) Singularity: It is worth mentioning that a well-designed network should have sufficient redundancy and be carefully engineered in the case of link failures. For *CPSLF* problem, there may have nodes disconnected with others in the process of traversal, which we call singular nodes. These singular nodes will bring calculation deviation since the latency will become Infinity. In our simulation, if single link failure results

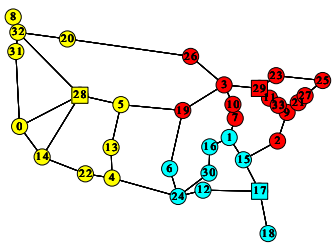

Fig. 2: Placement of Internet2 computed by OPA at k=3

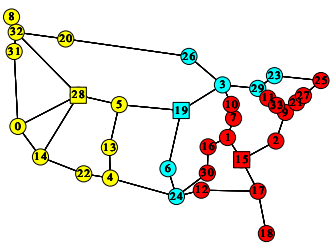

Fig. 3: Placement of Internet2 computed by GPA at k=3

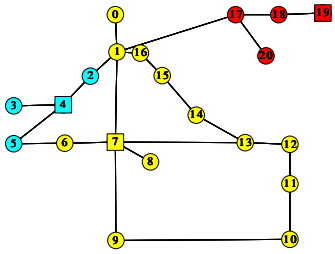

Fig. 4: Placement of Cernet2 computed by OPA at k=3

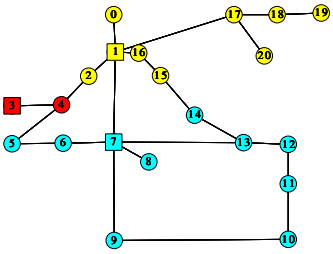

Fig. 5: Placement of Cernet2 computed by GPA at k=3

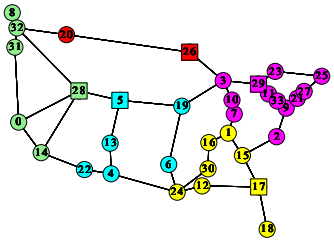

Fig. 6: Placement of Internet2 computed by OPA at k=5

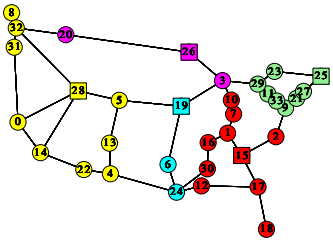

Fig. 7: Placement of Internet2 computed by GPA at k=5

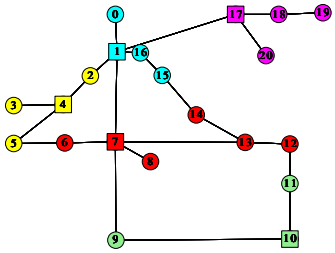

Fig. 8: Placement of Cernet2 computed by OPA at k=5

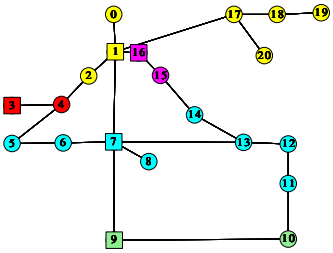

Fig. 9: Placement of Cernet2 computed by GPA at k=5

in singular nodes, we add a punitive delay. More accurate penalty function is left for further research.

B. Simulation Result

1) Location of controllers: We place 1 to 5 controllers in the topologies Internet2 and Cernet2. Fig.2-Fig.9 show the placement scheme computed by two algorithms at $k = 3$ and $k = 5$. The circle nodes denote switches and square node denote controllers. Each switch is assigned to the controller with the same color. As we can see from the figures, the placement schemes vary but the placement of controllers is still in common. For Internet2, the result of *OPA* shares one location with the one of *GPA* which is node 28 when $k = 3$. The number grows to two, which are node 26 and node 28 when $k = 5$. This may result from the similar network states traversal steps. The *GPA* is based on the idea of placing the controllers step by step, so the chosen controllers computed by small k value will be included in the result computed by larger value of K. The result of *OPA* is unsuitable this pattern. As for Cernet2, a similar situation could be obtained.

2) Network state latency: Minimizing the network state latency is our optimization objective. Fig.10 and Fig.13 show the network latency of Internet2 and Cernet2 calculated by the two algorithms and k is assigned from 1 to 5. For Internet2, the network state latencies are equal both in two algorithms when k=1, and for Cernet2, the network state latencies calculated by *GPA* and *OPA* are equal when $k = 1$ and $k = 2$. As the number of controllers increasing, the network state latencies all decline for both algorithms and topologies. Clearly, network state latencies from *OPA* are always better than the ones of *GPA*. For Internet2, the peak latency deviation between the results of *OPA* and *GPA* acquires at $k = 2$ where *OPA*'s is 17.6% better than the one of *GPA*, and the deviation declines under 5% when $k > 2$. For Cernet2, the gap between the results of *OPA* and *GPA* is not obvious and the peak latency deviation be acquired at k=5 where *OPA*'s is 4.4% better than the one of *GPA*. We can indicate that there is no observable difference in performance between the results of *OPA* and *GPA* if we place more controllers into the network.

3) Worse case latency distribution: Fig.16-Fig.18 show the cumulative distribution of worse case latency computed by all two algorithms at $k = 1, 3, 5$ for Internet2 and Fig.19-Fig.21 show the one of Cernet2. We could observe that the worst case latency changes within a wide range, which shows that the minimum worst case latency hardly acquired when link failures happen from another point of view. When $k = 1$ the placement schemes calculated by two algorithms are same in both topologies. For Internet2 Over 75% network states have the worse case latency less than 10 *ms* and nearly 50% network states have a latency of $9.5ms$. We can see a significant decrease in worse case latency calculate by *OPA* and *GPA* in Fig.16 when $k = 3$. About 38% network states of the result of *OPA* and over 33% network states of of the one of *GPA* have a latency decreased to $5.86ms$. It is interesting that the maximal worse case latency of *OPA* is $20.1ms$, larger than the $16.2ms$ of *GPA*'s. It seems some link failure has a greater impact on the worst case latency of the placement scheme calculated by *OPA*. The worse case latency distribution deviation gets smaller between the results of *OPA* and *GPA* as k grows. We can get a similar conclusion from the result computed in Cernet2. Finally, we could find that the distribution of worse case latency has a close relationship with topology and the number of controllers.

4) Runtime: Fig.11 and Fig.14 show the runtime of all two algorithms for Internet2 and Cernet2 respectively. We can observe an obvious runtime disparity between two algorithms among which *OPA* needs much longer time than *GPA* for

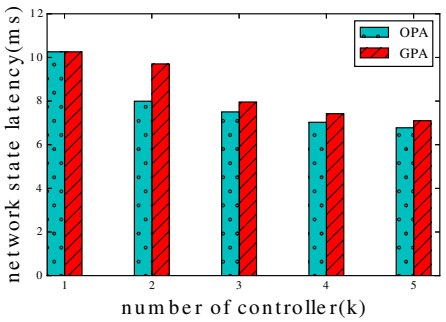

Fig. 10: Network state latency of Internet2

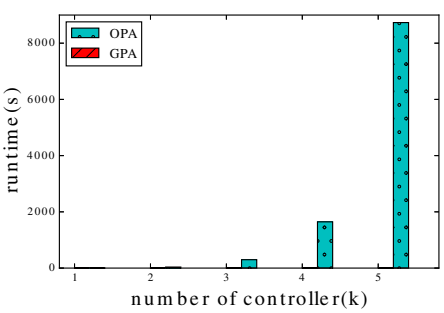

Fig. 11: Runtime of Internet2

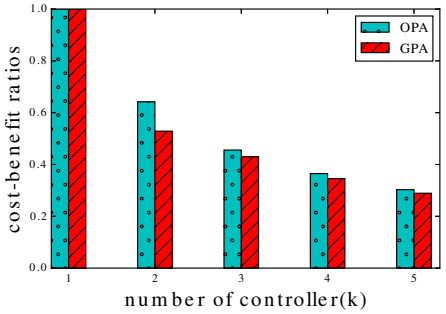

Fig. 12: Cost benefit ratios of Internet2

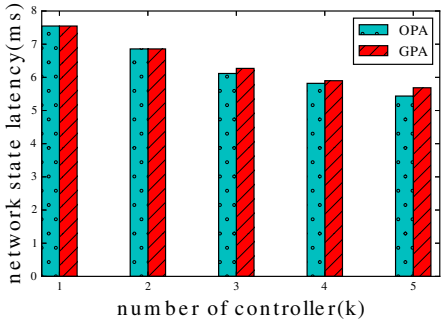

Fig. 13: Network state latency of Cernet2

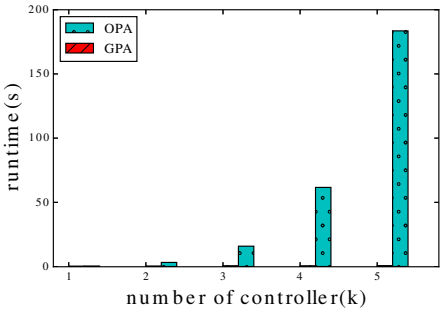

Fig. 14: Runtime of Cernet2

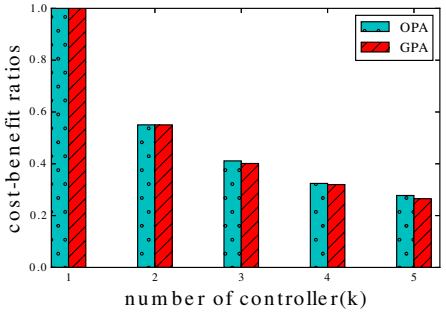

Fig. 15: Cost benefit ratios of Cernet2

both topologies. Actually, for Internet2 the runtime of *OPA* is $9398.97s$ at $k = 5$, 2564 times the runtime of *GPA*. The minimum deviation is at $k = 1$, where *OPA* runs $2.51ms$, 1.9 times the runtime of *GPA*. We could observe a similar situation in Cernet2. It is easy to understand that the time complexity of *OPA* is exponential and *GPA*'s polynomial, so with the growth of the number k, the runtime of *OPA* increasing rapidly. The time complexity expression of *GPA* has a coefficient related to k, so k increases the computation time as it grows in size. On the other hand, k doesn't play a decisive role in the time complexity expression of *GPA*. Thus, it can be seen *GPA* runs the minimum time with worse performance and *OPA* runs the maximum time with limited performance improvement.

5) Cost-benefit ratios: In Fig.12 and Fig.15, we study the cost-benefit ratios of all two algorithms for both topologies. We define the cost-benefit ration as $(latency_1/latency_k)/k$. A value of 1.0 implies a proportional reduction; where k controllers reduce latency to $1/k$ of the one-controller latency. Higher is better. In Internet2, reducing the network state latency to half that at $k = 1$ requires 3, 2 controllers for *OPA* and *GPA* respectively. As for Cernet2 it is requires 3 controllers for both algorithms.

VI. Conclusion and Future Work

In this paper, we proposed that network states transformation due to network failures has a huge impact on the worst case latency between data plane and control plane, and will further influence network performance. We considered the transformation between network states and defined network state latency as a new evaluation criterion of the controller placement scheme. Accordingly, we defined the Controller Placement under Comprehensive Network States *(CPCNS)* problem and Controller Placement under Single Link Failure *(CPSLF)* problem. Accordingly, we proposed two algorithms, namely, Optimal Placement Algorithm *(OPA)* for *CPSLF* and Greedy Placement Algorithm *(GPA)* for *CPSLF*.

We evaluated our algorithms comprehensively using Internet2 and Cernet2 as well as other topologies. Later, we presented a case study on Internet2 and Cernet2. We study various aspects of two algorithms. We illustrated that the controller placement scheme computed by *OPA* has the minimum network state latency but spending the more time. The runtime of *GPA* has a huge degree of reduction over *OPA* without much performance degradation. With the number of controllers increasing, the deviation of network state latencies becomes smaller between the *OPA* and *GPA*.

In the future, we try to work out a good evaluation procedure for *CPCNS* problem. Our next work includes taking the node operational probability and controller operational probability into consideration and propose a complete evaluation model.

VII. Acknowledgment

This work was supported in part by the R&D Program of Shenzhen under grant No. ZDSYS20140509172959989, No. JSGG20150512162853495, and No.Shenfagai[2015]986.

References

[1] H. Yan, D. A. Maltz, T. E. Ng, H. Gogineni, H. Zhang, and Z. Cai, "Tesseract: A 4d network control plane." in *NSDI*, vol. 7, 2007, pp. 27–27.

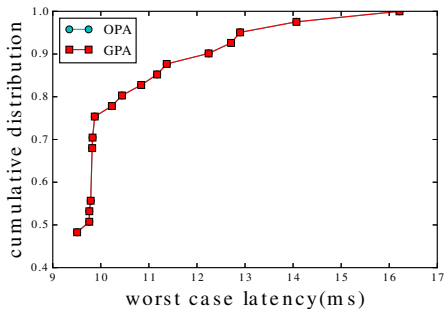

Fig. 16: Worse case latency distribution of Internet2 at k=1

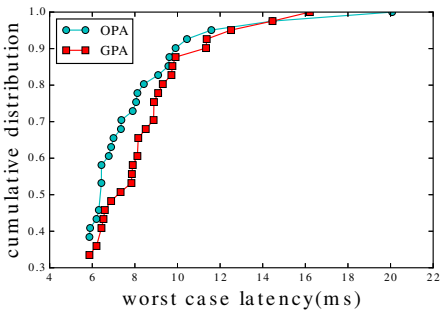

Fig. 17: Worse case latency distribution of Internet2 at k=3

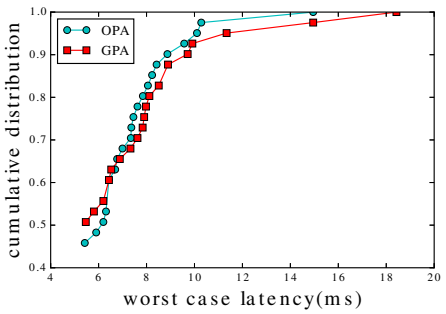

Fig. 18: Worse case latency distribution of Internet2 at k=5

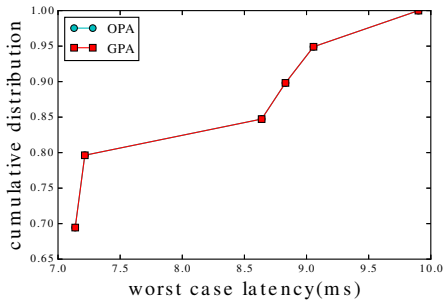

Fig. 19: Worse case latency distribution of Cernet2 at k=1

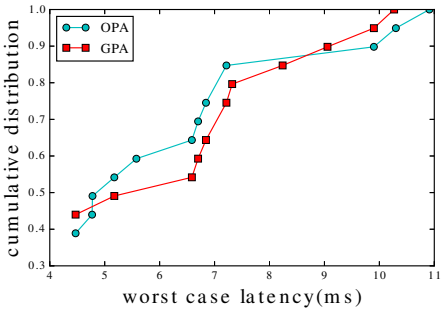

Fig. 20: Worse case latency distribution of Cernet2 at k=3

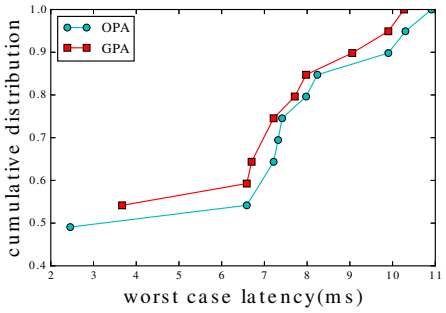

Fig. 21: Worse case latency distribution of Cernet2 at k=5

[2] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 1–12, 2007.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[6] L. Yang, P. Zerfos, and E. Sadot, "Architecture taxonomy for control and provisioning of wireless access points (capwap)", rfc 4118," 2005.

[7] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[9] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[10] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[11] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[12] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, vol. 54, 2012.

[13] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[14] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H. J. Chao, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Computer Networks*, vol. 68, pp. 95–109, 2014.

[15] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," 2014.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[17] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in sdn-based core networks," in *Teletraffic Congress (ITC), 2013 25th International*. IEEE, 2013, pp. 1–9.

[18] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 31–36.

[19] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Transactions on Networking (TON)*, vol. 16, no. 4, pp. 749–762, 2008.

[20] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, Jan. 1977. [Online]. Available: http://doi.acm.org/10.1145/321992.321993

[21] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.

[22] "Internet2 open science, scholarship and services exchange." http://www.internet2.edu/products-services/advanced-networking/layer-2-services.