Validation of Distributed SDN Control Plane under Uncertain Failures

Junjie Xie, Deke Guo, Chen Qian, Bangbang Ren, Honghui Chen

Abstract—The design of distributed control plane is an essential part of SDN. While there is an urgent need for verifying the control plane, little, however, is known about how to validate that the control plane offers assurable performance, especially across various failures. Such a validation is hard due to two fundamental challenges. First, the number of potential failure scenarios could be exponential or even non-enumerable. Second, it is still an open problem to model the performance change when the control plane employs different failure recovery strategies. In this paper, we first characterize the validation of the distributed control plane as a robust optimization problem and further propose a robust validation framework to verify whether a control plane provides assurable performance across various failure scenarios and multiple failure recovery strategies. Then, we prove that identifying an optimal recovery strategy is NP-hard after developing an optimization model of failure recovery. Accordingly, we design two efficient failure recovery strategies, which can well approximate the optimal strategy and further exhibit the good performance against potential failures. Furthermore, we design the capacity augmentation scheme when the control plane fails to accommodate the worst failure scenario even with the optimal failure recovery strategy. We have conducted extensive evaluations based on an SDN testbed and large-scale simulatons over real network topologies. The evaluation results show the efficiency and effectiveness of the proposed validation framework.

Index Terms—Robust validation, Distributed controllers, Uncertain Failures, Software-defined networking.

1 Introduction

Software-defined networking (SDN) is currently attracting significant attention from both academia and industry. It allows network operators to dynamically manage resources and behaviors of the underlying network [1], and enables the flexible development of novel network applications [2]. This is done by decoupling the control plane that makes the routing decisions for flow requests from the underlying data plane that accordingly forwards flows. SDN adopts a logically centralized control plane to maintain the global network view. The control plane usually consists of multiple distributed controllers, to guarantee the scalability and availability.

In SDN, a designed mapping from the control plane to the data plane is realized by assigning each switch a list of available controllers, including a master controller and multiple slave (standby) controllers [3]. It is the master controller that receives flow requests from the covered switches, computes a routing decision for each flow, and installs flow rules on all involved switches on the route. In this way, switches can forward flow requests according to the routing decisions derived from the control plane. Meanwhile, to enable the above design, the control plane and the data plane have to communicate with each other via secure links. The secure links can be dedicated control channels (out-of-band control) [4] or share the same network links as the data plane (in-band control) [5]. In this paper, we consider the former one, which is supported by many controllers [3][6][7].

It is critical to offer assurable performance at acceptable cost when designing a control plane. The routing requests experience the long-tail response latency when the number of requests to be processed exceeds the capacity of any one controller in SDN [8]. Furthermore, the long-tail latency would cause the network applications to suffer from severe performance degradation. While there is an urgent need, little, however, is known about how to validate whether the control plane and the assignment of controllers to switches can cope with various failures and achieve the design goal. This problem is referred as the robust validation of the control plane, and is very hard due to two-fold challenges.

First, the control plane suffers from the failures of not only controllers but also those secure links. Any such failure will isolate a switch from its master controller and make the switch become uncovered and no longer able to deal with new flow requests [6]. Meanwhile, the number of potential failure scenarios is exponential to the total number of controllers. Therefore, directly validating all failure scenarios is prohibitive. Second, it is still an open problem to model the performance change when different failure recovery strategies are employed to adapt to failures. That is, the control plane re-elects an appropriate master controller for each uncovered switch from its slave controllers under a failure scenario. Then, some controllers exhibit higher utilization due to serving additional uncovered switches. It is clear that the failure scenario and the related failure recovery strategy jointly affect the performance of the control plane.

In this paper, we take a first step towards the robust validation of the control plane of SDN by developing an optimization framework over uncertain failure scenarios

- *Junjie Xie, Deke Guo, Bangbang Ren and Honghui Chen are with the Science and Technology Laboratory on Information Systems Engineering, National University of Defense Technology, Changsha Hunan, 410073, China. E-mail: {xiejunjie06, guodeke}@gmail.com.*
- *Chen Qian is with the Department of Computer Engineering, University of California Santa Cruz, CA 95064, USA. E-mail: cqian12@ucsc.edu*

and multiple failure recovery strategies. Note that the utilization of each controller can efficiently reflect the controller performance of responding to flow requests as discussed in Section 2.2.3. Furthermore, for the distributed SDN control plane, we adopt the maximum controller utilization (MCU) among all controllers as the performance metric of the control plane. Given a failure scenario and a failure recovery strategy, we can evaluate the MCU in that case. Then, the robust validation of the control plane aims to verify if the highest MCU is lower than the demand performance under all combinations of the failure scenarios and the failure recovery strategies. Although the validation problem is so hard to solve, it becomes tractable by solving the following two challenging problems step by step.

The first one is to design a proper failure recovery strategy. Different failure recovery strategies would incur different MCUs, even though they meet the same setting of the control plane and the same failure scenario. Although our validation framework can cope with the existing failure recovery strategies, it still lacks a failure recovery strategy to minimize the MCU of the whole control plane. Therefore, we first develop an optimization model of failure recovery, and further prove that designing an optimal recovery strategy is NP-hard. Accordingly, we design two efficient failure recovery strategies, which can well approximate the optimal strategy and offer good performance against potential failures. The second key is to find the worst failure scenario, which results in the highest MCU among all failure scenarios. If the control plane can accommodate the worst failure scenario, the control plane can accommodate all uncertain failures. To solve this problem, we design a recursive solution to find the worst-case performance of the control plane, and further propose a branch-and-bound method to accelerate the solution.

Furthermore, even with the optimal failure recovery strategy, if the result of validation indicates that the value of the highest MCU is larger than 1 under the worst failure scenario, we further conduct the capacity augmentation on demand to tackle this scenario. That is, the capacity of a set of selected controllers can be extended incrementally, such that all failure scenarios can be handled after the capacity augmentation. Consequently, our validation framework can derive the most effective way to augment the capacities of a few involved controllers while incurring the minimal augmentation cost.

The major contributions of this paper are summarized as follows.

1) We characterize the validation of the distributed control plane as a robust optimization problem and further propose a robust validation framework to verify whether a control plane provides assurable performance across various failure scenarios.
2) We develop an optimization model for the failure recovery strategy, and further prove that designing an optimal recovery strategy is NP-hard. Two approximation solutions are designed to minimize the MCU across all failure scenarios.
3) We design the capacity augmentation scheme, which can effectively guide the capacity augmentation of the control plane when the value of MCU is still larger than 1 even with the optimal failure recovery strategy.
4) We evaluate the validation framework via experiments on a SDN testbed, with the real topology and traffic matrices. Furthermore, we measure our validation framework through large-scale simulations under two typical topologies. The extensive evaluations demonstrate the promise and effectiveness of our design.

The rest of this paper is organized as follows. Section 2 presents the motivation and background of the robust validation of distributed control plane. Section 3 formalizes the robust validation problem and the failure recovery strategy. In Section 4, we solve the validation problem by three steps, including approximating the optimal recovery strategy, finding the worst failure scenario and augmenting the capacity of the control plane. Section 5 evaluates the performance of our validation framework. We introduce the related work and conclude this paper in Section 6 and Section 7, respectively.

2 Motivation and Preliminaries

2.1 Motivation of robust validation

In designing the control plane of SDN, operators must determine how many controllers to employ and how much capacity to provision for each controller. The control plane then derives an efficient allocation of controllers to each switch such that each switch can connect to multiple controllers, including one master controller and several salve (standby) controllers [3]. Note that the master controller of some switches may be a slave controller of other switches simultaneously. It is clear that a controller can not tackle too many flow requests per unit time due to the capacity limitation. Furthermore, the routing requests would suffer from the long-tail response latency when the number of requests exceeds the controller's capacity [8]. The validation of the control plane means to verify if the current capacity setting of controllers and the mapping between controllers and switches can meet the demand performance.

Meanwhile, any failure scenario may make one or several involved switches become no longer covered by their master controllers. In this scenario, while the set of controllers and their capacities are difficult to change, a failure recovery strategy could improve the availability of the control plane by re-electing a master controller for each uncovered switch. For example, when a controller instance of ONOS [3] fails, the remaining controller instances would re-elect a new master controller for each uncovered switch from the slave controllers. Therefore, the validation across failures aims to verify if the control plane can meet the demand of design under a given failure recovery strategy across various failure scenarios, especially the worst failure scenario.

While there is an urgent need for verifying the control plane, little, however, is known about how to validate that the control plane offers assurable performance, especially across various failures. The insight of our validation framework is the theory of robust optimization [9], which minimizes the given objective across an uncertain parameter set. In the optimization process, a rich set of adaptation strategies could be utilized according to the application contexts. For the distributed SDN control plane, the uncertain set records all uncertain failure scenarios, consisting of the failures of controllers and the failures of secure links, which make controllers lose connection to switches.

2.2 Preliminaries

2.2.1 *The uncertain failures of the control plane*

We seek to validate that a control plane performs well across various failure scenarios. The failure scenarios consist of the failures of controllers and the failures of secure links. The secure links are the infrastructure of the control channels, which are used to connect switches with controllers. The failure of a secure link will cause a switch to lose the connection to its master controller. Note that failure probability of secure links depends on the hop count of its physical path. However, our validation framework is irrelevant to the failure probability. The main concern of the validation is if the control plane can still meet the demand performance when the failures occur. Not only the failures of secure links but also the failures of controllers lead to the disconnection between switches and their master controllers. More precisely, for the failures of controllers, a typical set of failure scenarios to validate is the simultaneous failures of f or fewer controllers.

2.2.2 *Modeling the adaptation of the control plane*

The control plane can naturally tackle the aforementioned failure scenario by using special adaptation methods, such that the high availability and the low utilization of controller can be reserved. This can also be achieved by determining an optimal new master controller for each uncovered switch that fails to reach its initial master controller under a given failure scenario. However, the state-of-the-art re-election strategy is far from the optimal solution. In the current design of ONOS [3], all available controllers of each switch form a list in a preference order. When a switch fails to communicate with its master controller, one of its slave controllers is orderly selected as the new master controller. It is obvious that such a failure recovery strategy could not always make a reasonable decision. Furthermore, the sequent selection method would degrade the performance of the control plane.

For example, switch s_x has the master controller c_i and two slave controllers c_{ii} and c_{iii}, which are set in a preference order. When switch s_x loses the connnection to controller c_i, the control plane orderly sets controller c_{ii} as its new master controller. However, at that time, the utilization of controller c_{ii} may be very high, and controller c_{iii} may have more residual capacity. In this case, a better choice is to set controller c_{iii} as the new master controller of switch s_x. Based on the above analysis, the validation problem of control plane across failures is dominated by the failure recovery (master re-election) strategy. Note that, an efficient failure recovery strategy should fully exploit the residual capacity of the control plane.

2.2.3 *The performance metrics of the control plane*

In SDN, the controllers calculate the routing paths for each new arrival flow and insert the forwarding rules into switches [3][5][6]. For a received flow request at a controller, The response latency of that controller reflects the performance of the control plane. Meanwhile, this latency consist of the latency of calculating path in the controller and the transfer latency of inserting the flow rules into switches. Note that not all packets of a flow are sent to the controller. Only a flow request is sent to the controller for calculating a routing path when a switch receives the first packet of a new flow. If a controller is far away from a switch, i.e., the transfer latency in the secure link fails to satisfy the demand performance, the controller will not be set as a master or slave controller of the switch. Therefore, to evaluate the performance of the control plane, the main latency is caused by the process of calculating the routing path at each controller. Furthermore, if a controller receives too many flow requests, they will wait in a queue for calculating the routing pathes [8].

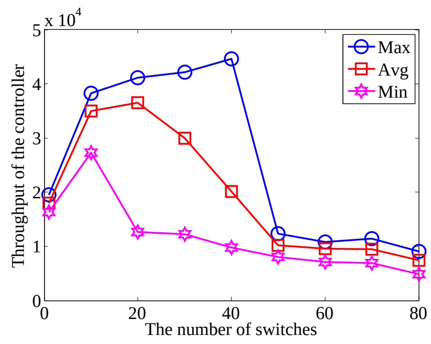


Fig. 1. The changing trend of throughput of a controller as increasing number of switches.

Given the capacity provision of the control plane, the utilization of each controller can efficiently reflect the perfromance of the control plane. Furthermore, to validate the distributed SDN control plane, we adopt the maximum controller utilization (MCU) among all controllers to capture the performance of distributed control plane against failures. When $MCU{>}1$, the control plane is no longer satisfying the demand of interest. Assume that a controller can process ρ flow requests per second. The controller utilization is defined as $U{=}r/\rho$, while receiving r flow requests in one second.

For any controller, the value of U directly affects its throughput and dominates the response delay of a flow request. As an evidence, we measure the influence of the controller utilization on the controller throughput using the *cbench* performance test suite [4], which can test the maximum, average and minimum throughputs in each setting. We deploy the cbench and the ONOS controller [3] (Falcon 1.5.0) on a machine with 8GB memory and an Intel Core 2.4GHz CPU. After 10 rounds of experiments under each different number of switches, Fig. 1 plots the changing trend of the controller's throughput across different number of switches. It is clear that the controller's capacity will be saturated when the amount of switches comes up to a threshold since each emulated switch sends requests at the highest possible rate. Then, the throughput of the controller starts to go down when its utilization exceeds 1. We can see from Fig. 1 that the maximum throughput decreases sharply when the number of switches exceeds 40.

2.2.4 *Predicted requests*

The number of flow requests at a switch can be predicted from the history records, a common method used in practice. As mentioned in [10], predictors usually estimate the traffic matrix for a given interval, based on a convex combination of previously seen matrices. Thus, the validation framework can be optimized using a set of historical data $\{r^j\}_{j\in H}$, where r^j records the number of flow requests in the j_{th} time slot and $H{=}\{1,2,...,h\}$. It is desirable to verify the design for the convex hull of $\{r^1,r^2,...,r^h\}$, which ensures that all controllers can respond to all flow requests from switches while keeping a reasonable utilization. Specifically, this can be modeled by replacing r_i as constraints $r_i{=}\sum_{j\in H} y_j r_i^j, y_j{\geq}0$ and $\sum_{j\in H} y_j{=}1$, where r_i indicates the number of flow requests produced by switch s_i per unit time.

3 Formalizing robust validation of distributed SDN control plane

3.1 Framework of robust validation

Let V denote the set of uncertain failures (possibly non-enumerable), over which the design of distributed control plane must be validated. Those uncertain failures fall into two categories, including the controller failure and the failure of the secure link. Let w denote a failure recovery strategy determined by the control plane to tackle a failure scenario v. That is, a failure recovery strategy w would generate the new master controllers for those uncovered switches. Formally, the validation problem of the distributed control plane may be written as:

$$F^* = \max_{v\in V} \min_{w\in W} F(v,w) \tag{1}$$

The inner minimization captures that for any failure scenario $v{\in}V$, the control plane picks w from a set of permissible strategies $W(v)$ to minimize an objective function $F(v,w)$. The selected w ensures that each uncovered switch, caused by the failure scenario v, can get one and only one new master controller. There may exist multiple permissible strategies to tackle a failure scenario, since each uncovered switch may have multiple slave controllers, each of which has opportunity to be elected as the master controller. The outer maximization robustly captures the worst-case performance across the set of failure scenarios V, assuming the control plane adapts to each failure scenario in the best possible fashion. Therefore, Formula (1) indicates that the robust validation problem is to verify if the performance of the control plane can meet the demand of interest while encountering the worst failure scenario and adopting the best failure recovery strategy.

In this paper, we utilize the MCU metric in Section 2 as the objective function $F(v,w)$, which reflects the performance of the distributed control plane. Formula (1) is referred as the validation problem, since it can be used to verify whether a given design of distributed control plane meets a desired optimization goal. For instance, $F^*{>}1$ indicates that the distributed control plane is not sufficiently provisioned to handle all failure scenarios.

TABLE 1
A list of terms used throughout the paper.

Term	Description
l	The number of failed secure links.
f	The number of failed controllers.
N	The number of switches in a network.
M	The number of controllers in a network.
v_j^f	The state of controller c_j.
v_{ij}^l	The state of the secure link between switch s_i and controller c_j.
m	The mastership between switches and controllers.
m_{ij}	The mastership between switch s_i and controller c_j.
$w(v)$	A failure recovery strategy for a failure scenario v.
$W(v)$	A set of failure recovery strategies for a failure scenario v.

3.2 Concrete validation problems

We use a notation $v{=}(v^l,v^f)$ to refer a general failure scenario faced by the distributed control plane. Here, v^l denotes a failure scenario of a secure link and v^f denotes a controller failure. Let v_{ij}^l be a binary variable which is 1 if the secure link between switch s_i and controller c_j has failed, and otherwise, $v_{ij}^l{=}0$. Let v_j^f be a binary variable which is 1 if controller c_j has failed, and otherwise, $v_j^f{=}0$. Let m_{ij} denote the mastership between switch s_i and controller c_j. If controller c_j is the master controller of switch s_i, $m_{ij}{=}1$; otherwise, $m_{ij}{=}0$. Meanwhile, when $v_{ij}^l{=}1$ or $v_j^f{=}1$, let $m_{ij}{=}0$, which indicates that the switch s_i loses the connection with its initial master controller c_j. In this case, the failure recovery strategy needs to re-elect a new master controller for switch s_i. After that, the new master controller will take over the switch s_i. Accordingly, the utilization of the new master controller would increase due to additionally taking over switch s_i.

Furthermore, we define a notation $w{=}(m,U)$, where $m{=}[m_{ij}]_{N\times M}$ denotes the mastership between N switches and M controllers, and U denotes the desired utilization metric. It indicates that a failure recovery strategy w is related to the mastership m and the utilization U. Consider that our focus is on minimizing the MCU among all controllers. The inner problem in Formula (1) could be expressed as $\min_{w\in W(v)} U$, which means that the utilization of each controller is at most U. Here, $W(v)$ corresponds to the constraints of the failure recovery strategy, which will be detailed in Section 3.3.

Table 1 gives a list of terms used in this paper. Formula (1) indicates a two-stage formulation problem [11]. The optimal variable (w) in the second stage depends on the variable (v) in the first stage. Therefore, this problem can be re-expressed as a single-stage optimization problem, where all variables can be determined simultaneously. Then, we formulate the validation problem as an integer programming (IP). For any failure scenario v, we use $\hat{w}(v)$ to denote the resultant failure recovery strategy in Section 4.1. Furthermore, the key of the validation problem is to find the worst failure scenario, which maximizes the MCU of the control plane. Then, the robust validation can be formalized as Formulation (2).

$$\max \hat{w}(v)$$
$$s.t. \begin{cases} v \in V \\ \{v_j^f, v_{ij}^l\} \in \{0,1\} \quad \forall i,j \end{cases} \tag{2}$$

Here, V denotes the set of failure scenarios involving the failure of f or fewer controllers and the failure of l or fewer secure links simultaneously. The failure model is used commonly in practice [12]. We aim to tackle the validation problems, including the design of the control plane and the availability of the control plane against uncertain failures. Furthermore, incorporating this model results in replacing the constraint $v{\in}V$ in Formulation (2) with the constraints $\sum_{j=1}^{M} v_j^f{\leq}f,\ v_j^f{\in}\{0,1\}$ and $\sum_{i=1}^{N}\sum_{j=1}^{M} v_{ij}^l{\leq}l,\ v_{ij}^l{\in}\{0,1\}$. Then, we can simplify Formulation (2) as:

$$\max \hat{w}(v)$$
$$s.t. \begin{cases} \sum_{j=1}^{M} v_j^f = f \\ \sum_{i=1}^{N}\sum_{j=1}^{M} v_{ij}^l = l, \\ \{v_j^f, v_{ij}^l\} \in \{0,1\} \quad \forall i,j \end{cases} \tag{3}$$

In our validation problem of the control plane across failures, the inner problem $min_{w\in W(v)}F(v,w)$ is an IP in variable $w{=}(m,U)$, where m indicates the mastership between switches and controllers. Furthermore, the mastership m is determined by a failure recovery strategy for a fixed failure scenario v.

3.3 Formalizing the failure recovery strategy

Let a binary variable b_{ij} denote the mapping between switches and controllers. We have $b_{ij}{=}1$, if and only if controller c_j is a master or slave controller of switch s_i, and 0 otherwise. The mapping b_{ij} is set based on the capacity of controller c_j and the transfer latency between controller c_j and switch s_i, when operators deploy controllers in a SDN. Note that, the mapping b_{ij} can affect the performance of the control plane and determine the ability of the control plane against failures. Furthermore, the failure recovery strategy should keep the utilization of each controller low. An optimal failure recovery strategy, which is defined as the Utilization Minimization Re-election (UMR) problem, aims to minimize the MCU among all controllers.

Let ρ_j denote the capacity of controller c_j, which indicates the maximum number of flow requests the controller can respond to per unit time. Let r_i denote the number of flow requests that switch s_i produces in a unit time. For the validation, r_i can be calculated in advance. Further, r_i can also be predicted based on the discussion about the *predicted requests* in Section 2. Given a failure scenario, some switches would become uncovered switches. Accordingly, we use S^f to denote the set of uncovered switches, which can be achieved by the control plane [3]. Then, $|S^f|$ denotes the number of uncovered switches. Note that S denotes the set of all switches. Thus, $S{-}S^f$ denotes the set of switches, which are not affected by a given failure scenario. Recall that we use $m{=}[m_{ij}]_{N\times M}$ denote the mastership between N switches and M controllers, and is determined at the design stage of the control plane. Let $m^*{=}[m_{xj}^*]_{|S-S^f|\times M}$ denote the mastership, which is not affected by the failure scenario. Then, the optimization problem is to determine $[m_{ij}]_{|S^f|\times M}$ for each uncovered switch $s_i{\in}S^f$ such that the MCU of the entire control plane is minimized. Furthermore, we formalize the UMR problem as follows:

$$\min U$$
$$s.t. \begin{cases} U\rho_j(1-v_j^f) \geq \sum_{x=1}^{|S-S^f|} r_x m_{xj}^* + \sum_{i=1}^{|S^f|} r_i m_{ij} \quad \forall j \\ b_{ij}(1-v_{ij}^l) \geq m_{ij} \quad \forall i,j \\ \sum_{j=1}^{M} m_{ij} = 1 \quad \forall i \\ m_{ij} \in \{0,1\} \quad \forall i,j \end{cases} \tag{4}$$

Here, $\sum_{i=1}^{|S-S^f|} r_x m_{xj}^*$ indicates the number of flow requests received by controller c_j from those covered switches, which still communicate with their master controller c_j under a given failure scenario. In Formulation (4), the first constraint ensures that (i) the utilization of controller c_j is at most U for all non-failed controllers; and (ii) no flow request is sent to a failed controller since the value in the left of the formulation is 0 when $v_j^f{=}1$. The utilization U reflects the maximum utilization among all controllers in the control plane since the first constraint is applied to each controller. The second constraint captures that (i) a controller manages a switch based on the available secure link with $v_{ij}^l{=}0$; and (ii) the control plane re-elects the master controller for each uncovered switch s_i only from its slave controllers with $b_{ij}{=}1$, which indicates that controller c_j is a available controler of switch s_i. Note that $v_{ij}^l{=}1$ denotes the secure link between controller c_j and switch s_i has failed. In this case, m_{ij} can not be set as 1. The third constraint ensures that the uncovered switch s_i is only assigned one master controller again.

An instance of UMR is specified by variables ($|S^f|$,M,k), where $|S^f|$ denotes the number of uncovered switches, M denotes the number of controllers, and k denotes the number of available controllers, which a switch can utilize among all controllers. The available controllers include the master and slave controllers for a switch. Theorem 1 proves that the UMR($|S^f|$,M,k) problem is NP-hard when controllers are identical. In addition, the UMR problem will be more complicated when controllers differ in their capacities.

Theorem 1. UMR($|S^f|$,M,k) is NP-hard, when all controllers in the control plane are identical.

Proof: Scheduling jobs on identical parallel machines (SIPM) is NP-hard [13], it is considered as follows. There are $|S^f|$ jobs to be assigned to M identical machines, running in parallel. Each job $j{=}1,...,|S^f|$, must be processed on one of such machines for r_j time units without interruption, and each job is available for processing at time 0. Each machine can process at most one job at a time. The aim is to complete all jobs as soon as possible, that is to minimize the makespan.

Here, we describe a polynomial reduction from the SIPM problem to our UMR($|S^f|$,M,M) problem. Assume that the SIPM instance has $|S^f|$ jobs with each has the size $(r_1, r_2, ...r_{|S^f|})$ and M identical machines with each has the processing speed ρ. We divide each job into several tasks $\{r_1{=}(t_{11}, t_{12}, ...), r_2{=}(t_{21}, t_{22}, ...)...\}$, and each task represents one flow request in UMR($|S^f|$,M,M). Just as that the tasks from one job must be processed on the same machine, the flow requests from one switch only can be sent to the same controller, i.e., its master controller. M identical machines represent M controllers, each of which offers capacity ρ. In UMR($|S^f|$,M,M), each switch has M available controllers, and any one of them can be the master controller of the switch. That also means each job can be scheduled to any one machine. In this setting, minimizing the makespan in SIPM equals to minimizing the MCU in UMR($|S^f|$,M,M). Note that we can construct all these procedures in polynomial time, we have shown that SIPM$\leq_p$UMR($|S^f|$,M,M), and then UMR($|S^f|$,M,M) is also NP-hard.

Additionally, if there exists a polynomial algorithm for UMR($|S^f|$,M,k), where each switch has k available controllers, then we can set $k{=}M$. This means that there is also a polynomial algorithm to solve UMR($|S^f|$,M,M). However, UMR($|S^f|$,M,M) is NP-hard; hence, there exists no polynomial algorithm for UMR($|S^f|$,M,k). Thus, we can conclude that UMR($|S^f|$,M,k) is NP-hard. □

3.4 Making validation tractable

In this paper, we solve the validation problem of distributed control plane via three steps. First, we solve the inner problem $min_{w\in W(v)}F(v, w)$ in Formula (1) for any failure scenario v. The goal is to find the optimal failure recovery strategy, which minimizes the MCU of the control plane. However, calculating the optimal failure recovery strategy is an NP-hard problem. Thus, we propose two approximation strategies to re-elect a new master controller for each uncovered switch, which is caused by a failure scenario, in Section 4.1. Second, the outer problem $\max \hat{w}(v)$ in Formula (1) needs to find the worst failure scenario, which leads to the highest MCU among all failure scenarios. To find the worst failure scenario, we design a recursive solution, and further propose a branch-and-bound method to reduce the running time of the recursive solution in Section 4.2. Third, for the validation problem, if $MCU{>}1$ under the worst failure scenario even using an efficient failure recovery strategy, the capacity of the control plane would be augmented at the minimal cost of the augmentation in Section 4.3. After that, the control plane always holds that $MCU{\leq}1$.

4 Validation of Distributed Control Plane

In this section, we indicate how to use our validation framework to verify the design of the control plane, to check the performance of the control plane across various failures, and to guide the capacity augmentation to the control plane. We start with two efficient failure recovery strategies, which are solutions to the inner problem $min_{w\in W(v)}F(v, w)$ in Formula (1) for any failure scenario v.

4.1 Approximating the optimal recovery strategy

As aforementioned, it is NP-hard to finding the optimal failure recovery strategy across various failure scenarios. For this reason, we first design two efficient solutions to approximate the optimal strategy of failure recovery, which aims to minimize the MCU of the control plane.

4.1.1 A Rounding-based solution for UMR

This section develops a rounding-based algorithm, called RB-election, to solve the UMR problem. The RB-election algorithm consists of two major steps. The first step relaxes the NP-hard UMR problem by relaxing the integer programming (Formulation (4)) as a linear problem, i.e., $m_{ij} \geq 0$. We can solve the linear programming in polynomial time, and obtain the fractional solution, denoted by $\tilde{m}$.

In the second step, the fractional solution would be rounded to the 0-1 solution for the UMR. The set S^f records those uncovered switches that are caused by a failure scenario. We first choose an uncovered switch, denoted by s_i, from set S^f. Then, the algorithm chooses a controller $c_{\hat{j}}$, whose $\tilde{m}_{i\hat{j}}$ is the maximum one among all controllers covering switch s_i, and set $\hat{m}_{i\hat{j}}{=}1$ and other $\hat{m}_{ij}{=}0$ for switch s_i. That is, controller $c_{\hat{j}}$ would become the new master controller of switch s_i. Moreover, we update $S^f{=}S^f{-}s_i$. The algorithm would be terminated until all switches are covered by appropriate master controllers. Additionally, Theorem 2 proves that the RB-election algorithm can achieve the $(k{-}1)$-approximation for the UMR problem.

Theorem 2. The RB-election algorithm can achieve the $(k{-}1)$-approximation for the UMR problem.

Proof: In second step of the RB-election algorithm, we first choose an uncovered switch s_i. Let k denote the number of all available controllers that can cover the switch s_i. That is, there is one master controller and $(k{-}1)$ slave controllers. When the master controller fails, the control plane needs to re-elect a new master controller from $(k{-}1)$ slave controllers. Since M is the total number of controllers in the control plane, $k{\leq}M$. Assume that $\tilde{m}_{i\hat{j}}$ is selected after some iterations. Since $\sum_{j=1}^{M} \tilde{m}_{ij}{=}1$ for switch s_i, it follows $\tilde{m}_{i\hat{j}} \geq \frac{1}{k-1}$.

After solving the linear programming in the first step of the RB-election algorithm, we derive a fractional solution $\tilde{m}$ and an optimal result $\tilde{U}$ for the relaxed UMR problem. Let U^* denote the optimal result of the UMR problem. According to the algorithm description, the final utilization of controller c_j is:

$$\begin{aligned}
\forall j \quad U_j &= \frac{\sum_{x=1}^{|S-S^f|} m^*_{xj} + \sum_{i=1}^{|S^f|} \hat{m}_{ij}}{\rho_j} \\
&\leq \frac{\sum_{x=1}^{|S-S^f|} m^*_{xj} + \sum_{i=1}^{|S^f|} (k-1)\cdot \tilde{m}_{ij}}{\rho_j} \\
&\leq (k-1)\cdot \tilde{U} \leq (k-1)\cdot U^*
\end{aligned} \tag{5}$$

Thus, Theorem 2 is proved. □

Algorithm 1 MM-election strategy of failure recovery

```
Require: The set of uncovered switches $S^f$ and the mapping $b{=}[b_{ij}]_{|S^f|\times M}$ between controllers and uncovered switches.
Ensure: A mastership $[m_{ij}]_{|S^f|\times M}$.
1: while $S^f \neq \emptyset$ do
2:     Calculate the set of available controllers $C_{S^f}$ based on the set $S^f$ and the setting $b{=}[b_{ij}]_{|S^f|\times M}$;
3:     Select controller $c_j$ with the maximum residual capacity from the set $C_{S^f}$;
4:     Calculate the set of uncovered switches $S_j$, which can be covered by controller $c_j$;
5:     Select switch $s_i$ with the largest amount of flow requests from the set $S_j$;
6:     Set controller $c_j$ as the new master controller of switch $s_i$, i.e., $m_{ij}{=}1$;
7:     $S^f{=}S^f{-}s_i$;
8: end while
```

4.1.2 A lower-complexity solution using Double-Max

When a re-election event is triggered by a failure scenario, we expect that the control plane can immediately derive an appropriate failure recovery strategy. Note that the RB-election algorithm needs to solve the linear programming, and the number of its variables mainly depends on the amount of switches and controllers in a SDN. Thus, the linear programming may contain a large number of variables for a large-scale network; hence, it is rather costly in practice to solve such a linear programming problem. Therefore, we further design an efficient algorithm with lower complexity for the UMR problem.

We design the failure recovery strategy based on the Double-Max principle, abbreviated as MM-election strategy. To minimize the MCU, the switch with the maximum amount of flow requests should be covered by the controller with the maximum residual capacity. Inspired by this idea, we first find the controller with maximum residual capacity from those available controllers of all uncovered switches. Then, let the selected controller c_j takes over the uncovered switch, which has the maximum number of flow requests and can be covered by controller c_j. In addition, it is very hard to derive the global optimal solution, since the optimal controller may not be available to the corresponding switch.

The details of the MM-election algorithm are shown in Algorithm 1. S^f is the set of uncovered switches that fail to connect to their predefined master controllers. Let C_{S^f} denotes the set of available controllers, which can connect to at least one uncovered switch in the set S^f based on the mapping $b{=}[b_{ij}]_{|S^f|\times M}$ where $b_{ij}{=}1$ denotes controller c_j is an available controller to switch s_i. Algorithm 1 first calculates the set C_{S^f} based on the set S^f and the mapping $b{=}[b_{ij}]_{|S^f|\times M}$. Second, Algorithm 1 chooses a controller c_j with the maximum residual capacity from the set C_{S^f}. Third, Algorithm 1 calculates the set of uncovered switches S_j, which can be covered by controller $c_j{\in}C_{S^f}$. Fourth, Algorithm 1 selects a switch s_i with the largest amount of flow requests from the set S_j. Fifth, Algorithm 1 sets controller c_j as the new master controller of switch s_i, i.e., $m_{ij}{=}1$. After that, switch s_i would send flow requests to its new master controller c_j. Furthermore, we delete switch s_i from the set S^f, i.e., $S^f{=}S^f{-}s_i$. Algorithm 1 repeats the above process until all switches are covered. Theorem 3 shows that the time complexity of our MM-election strategy is $O(k{\times}|S^f|^2)$.

Algorithm 2 FWorst: find the worst failure scenario

```
1: Define and initialize two global variables, $\hat{V}$ and $\hat{U}$;
2: Initialize $V$, $U$, $level$←0, $start$← 1;
3: GETMCU($start$, $V$, $U$, $level$);
4: return $\hat{V}$, $\hat{U}$;
5: function GETMCU($start$, $V$, $U$, $level$)
6:     $level$ ++;
7:     if $level{>}f$ then
8:         return;
9:     end if
10:    for $j{=}start$ to $M$ do
11:        if $v_j^f$ == 1 then
12:            continue;
13:        else
14:            $v_j^f \leftarrow 1$;
15:            Calculate the maximal utilization of controller $U$;
16:            if $U{>}\hat{U}$ then
17:                $\hat{U} \leftarrow U$;
18:                $\hat{V} \leftarrow V$;
19:            end if
20:            GETMCU($j+1$, $V$, $U$, $level$);
21:            $v_j^f \leftarrow 0$;
22:        end if
23:    end for
24: end function
```

Theorem 3. The time complexity of our MM-election strategy is $O(k{\times}|S^f|^2)$. Here, $|S^f|$ is the number of uncovered switches under a failure scenario, and k denotes the maximum number of available controllers connected by each switch in a SDN.

Proof: Algorithm 1 consists of $O(|S^f|)$ iterations. In each iteration, it first takes $O(k{\times}|S^f|)$ time to achieve the set C_{S^f}. It then consumes $O(|C_{S^f}|)$ time to choose a controller c_j with the lowest utilization from C_{S^f}. The time complexity of Step 4 is at most $O(|S^f|)$. Meanwhile, it selects switch s_x with the most number of flow requests from S_j at the cost of consuming $O(|S^f|)$ time. It is clear that the time complexity of Step 6 is $O(1)$. Finally, it takes $O(|S^f|)$ time to delete s_x from S^f. In summary, the total time complexity of Algorithm 1 is $O(k{\times}|S^f|^2)$. □

4.2 Finding the worst failure scenario

The key to the robust validation is to find the worst failure scenario. If the performance of the control plane can meet the demand of interest under the worst failure scenario, the current design of the control plane can accommodate all failure scenarios. In general, it is hard to find the worst failure scenario for the original validation problem (2) that results in the highest MCU, since the failure scenarios may be exponentially many and potentially non-enumerable. However, through analysis, we find that Formulation (3) is tractable, and we use a recursive algorithm to find the worst failure scenario. The details of the recursive algorithm is shown in Algorithm 2. Furthermore, we design a branch-and-bound method to efficiently reduce the running time of Algorithm 2.

A recursive solution. Algorithm 2 first defines two global variables, $\hat{V}$ and $\hat{U}$, which record the worst failure scenario and the highest MCU, respectively. Then, it initializes the two set variables $\hat{V}$ and V, and set the variables in the two sets as 0. At the beginning, the variables $\hat{U}, U, level$ are all 0, and the variable $start$=1. Algorithm 2 invokes the function $getMCU()$ to calculate the values of $\hat{V}$ and $\hat{U}$. The variable $level$ records the depth of the recursive function $getMCU()$. Here, the maximal value of $level$ is f, where f is the maximum number of simultaneously failed controllers. The function $getMCU()$ would calculate the maximal utilization of controller U in each failure scenario. If a failure scenario produces a larger U than previous validated failure scenarios, it is recorded. The recursive algorithm would be terminated and return the calculated results when $level{>}f$. In addition, for validating the failure of secure links, the value of $level$ can be up to l where l is the maximum number of simultaneously failed secure links. Meanwhile, in Step 10, the length of the for loop should be equal to the number of secure links. To validate the hybrid failures of controllers and secure links, the maximal value of $level$ is $f{+}l$. The variable $start$ can efficiently reduce the running time of Algorithm 2 while ensuring the optimal value.

Furthermore, we analyze the feasibility of Algorithm 2. Its time complexity is $O(\binom{M}{f}{\times}k{\times}|S^f|^2)$, where $O(k{\times}|S^f|^2)$ is the time complexity of MM-election strategy for calculating $w(v)$. Algorithm 2 is practical for the validation problem of the distributed control plane based on the following two reasons. First, the validation problem usually need not to immediately response the result; hence, there is a considerable long time to solve and optimize the validation problem. Second, $\binom{M}{f}{\leq}M^f$, where the values of variables M and f are usually not very large. The number of deployed controllers M for achieving a distributed control plane is not too many. Furthermore, $f{\leq}k$, where k is the number of available controllers for each switch. Normally, the value of k is lower than 10 since it is not necessary that each switch has more than 10 slave controllers. Meanwhile, if any controller can manage all switches [5], we have $k{=}M$. In this case, it will be easier to solve the validation problem because the MCU will be equal to the ratio of the number of all flow requests from all switches to the capacity of the whole control plane.

A branch-and-bound method. Although Algorithm 2 is feasible, it still takes long time to derive the final solution. To reduce the running time of Algorithm 2, we further design a branch-and-bound method to cut a large number of failure scenarios, which will not result in a higher MCU. For the worst failure scenario, it would make an uncovered switch just keep a few of available controllers, even only one available controller. Accordingly, the switch is taken over by the controller even though the controller has already exhibited high utilization. That is, for an uncovered switch, the worst failure scenario means that its master and multiple salve controllers fail at the same time. Note that each uncovered switch has k available controllers. Furthermore, for each uncovered switch s_i, we can relax Formulation (3) as:

$$\max \hat{w}(v)$$

$$s.t. \begin{cases} \sum_{j=1}^{k} v_j^f = f \\ \sum_{j=1}^{k} v_{ij}^l = l, \\ 0 \leq v_j^f \leq 1, 0 \leq v_{ij}^l \leq 1 \quad \forall j \end{cases} \tag{6}$$

Formulation (6) is a relaxation LP, which can determine the worst failure scenario, causing the highest MCU. Those branches with $MCU{<}1$ will be pruned. Those unexplored switches is visited continually. The process is repeated until a failure scenario is found such that $MCU{>}1$. This indicates that the control plane with the current setting fails to accommodate the failure scenario. The search procedure solves at most as many LPs (Formulation (6)) as the number of uncovered switches in the network to find the worst failure scenario. That is, the number of the explored branches is at most the same as the number of uncovered switches. If all MCUs solved by those LPs are lower than 1, it means that the design of the control plane can accomodate to all given failure scenarios.

4.3 Augmenting capacities to existing controllers

In this section, we discuss how our validation framework can guide the capacity augmentation to the control plane. The capacity of a set of given controllers can be extended incrementally, such that all failure scenarios can be handled after the capacity augmentation. The augmentation process needs to consider two goals, enabling $MCU{\leq}1$ and minimizing the additional cost of augmentation. The augmentation problem is the development of Formulation (1); hence, we formalize the problem of capacity augmentation as follows:

$$\min_{\delta\geq 0} \max_{v\in V} \min \left\{ \sum_{j=1}^{M} \varphi_j \delta_j \;\middle|\; \begin{array}{c} (\rho_j{+}\delta_j)(1{-}v_j^f) \geq \\ \sum_{x=1}^{|S-S^f|} r_x m_{xj}^* + \sum_{i=1}^{|S^f|} r_i m_{ij} \\ b_{ij}(1{-}v_{ij}^l) \geq m_{ij} \\ \sum_{j=1}^{M} m_{ij} = 1 \end{array} \right\} \tag{7}$$

In Formulation (7), the set V contains all failure scenarios. We increase the capacity of controller c_j by δ_j. Let φ_j denote the cost of an unit capacity. The first constraint captures that the number of flow requests received by controller c_j should not exceed its capacity $\rho_j{+}\delta_j$ after augmentation. The outermost $\min$ can minimize the cost of capacity augmentation.

Inspired by the solution of Formulation (1), we use an iterative approach to solve Formulation (7), and then guide the capacity augmentation. In each iteration, we first identify the worst failure scenario, according to the method in Section 4.2. Then, we resolve the problem of augmenting capacities under the worst failure scenario and add the calculated capacities to those involved controllers. In the next iteration, we would continue to solve the validation problem in Formulation (1) to identify the worst failure scenario after updating the capacity of the control plane. The iterative process continues until the vailidation result shows

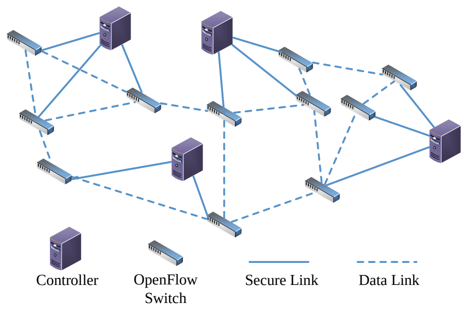


Fig. 2. A SDN testbed consists of Pica8 switches and ONOS controllers under the Abilene backbone topology.

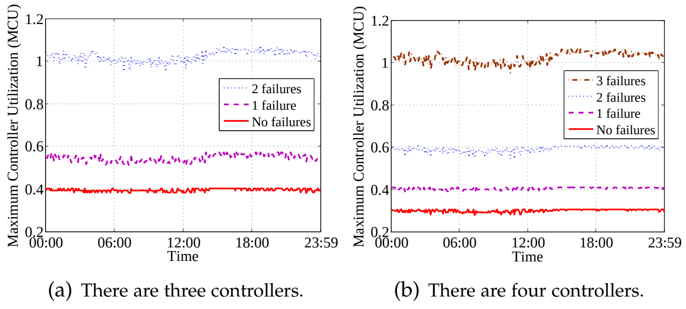


Fig. 3. The impact of the failure scenarios on the MCU on the SDN testbed.

$MCU \leq 1$. The iterative solution works well in practice for the augmentation of capacity.

5 Performance Evaluation

In this section, we demonstrate the ability of our validation framework through experiments on a SDN testbed and extensive simulations on two practical and typical topologies.

5.1 Experiments on a SDN testbed

We build a small-scale SDN testbed, employing the open-source ONOS controller platform, Falcon 1.5.0 [14], and the SDN switches, Pica8 P-3297 [15]. The SDN switch realizes the industry-leading OpenFlow 1.4 protocol through the integration of Open vSwitch (OvS) 2.0. Those OpenFlow switches are interconnected according to the Abilene backbone topology [12]. Furthermore, we adopt the real traffic traces of the Abilene backbone network on April 14, 2004 [16], and inject them into our testbed. The traffic data is recorded every five minutes. Therefore, we produce flow requests every five miniutes based on the flow record from a source to a destination in the traffic matrix. We implemented the validation framework, which is written in Java. The validation framework collects the number of flow requests from active controllers. Note that our validation framework can be applied to any traffic data. In SDN, the control plane calculates the routing path for each new arrival flow. Therefore, if there is a flow record, the related source switch sends a flow request to its master controller for configuring the routing path. The capacity of each controller is set as 500 since the traffic traces indicate that the number of received flow requests per second is not too many.

We first verify the performance of the control plane with three controllers against failures. We can see from Fig. 3(a) that the MCU fluctuates around 40% when no failures occur. Meanwhile, Fig. 3(a) indicates that the design of the control plane with 3 controllers is resilient to all scenarios of one failed controller. In this case, the MCU fluctuates around 55%. However, the control plane fails to accommodate all failure scenarios of two controllers since the MCU exceeds 1 at certain moments in that day. Furthermore, we consider another setting of our testbed with four controllers where the network topology is shown in Fig. 2. As shown in Fig. 3(b), the MCU always fluctuates around 60%; hence, the control plane can accommodate all failure scenarios of two controllers. However, the MCU exceeds 1 at some moments, when three controllers fail simultaneously. This indicates that the control plane is not resilient to failures of three controllers at the worst case.

In addition, Fig. 3 shows that the MCU always varies over the time slots in one day. In each time slot, we can get a MCU among all controllers. Furthermore, a robust control plane needs to ensure $MCU \leq 1$ across all time slots in the whole day. The fluctuation is caused by the varied number of flow requests. More precisely, for scenarios of three simultaneous failures, the MCU is always lower than 1 from 6 am to 12 am but is higher than 1 from 12 am to 6 pm, as shown in Fig. 3(b). Note that each switch generates flow requests according to the real trace in our testbed validation. Furthermore, for the case of other real applications, we predict the flow requests using many existing methods, such as the exponential moving average method [12].

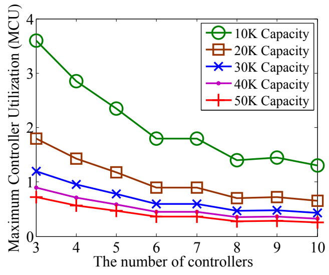
5.2 Evaluations based on large-scale simulations

In this section, we evaluate our validation framework and the failure recovery strategies by large-scale simulations.

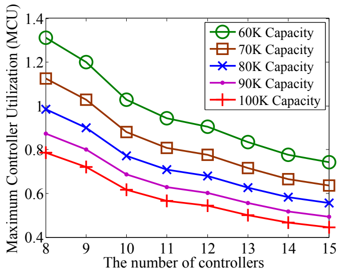
5.2.1 The settings of simulations

We further conduct large-scale simulations using two practical and typical topologies, a campus network [17], and a datacenter topology of a 16-ary fat-tree [18]. The former topology contains 100 switches, 397 links and 200 servers. The latter topology utilizes 320 switches, 3072 links and 1024 servers. All LPs and IPs in our proposals are resolved by CPLEX. Each server generates 500 flow requests per second on average.

When there is no failed controller, the utilization of the whole control plane is about 50%. The number of flow requests and the capacity of a controller can be adjusted in practice. Different settings of such two factors determine the number of controllers employed by the control plane, however, do not influence the effectiveness of our validation framework. To indicate that our validation framework can accommodate a rich set of failure recovery strategies, we compare four failure recovery strategies. They are the MM-election strategy, the RB-election strategy, the optimal solution (using CPLEX to solve the Integer Programming), and the prior election strategy. The prior election strategy selects a slave controller in turn to replace the initial master controller, when the master controller of a switch fails.

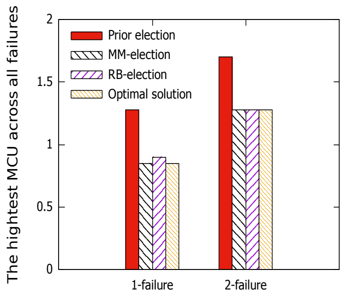


(a) The campus network.

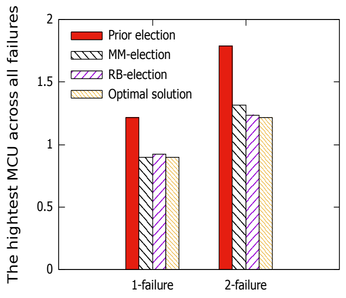


(b) The datacenter topology.

Fig. 4. The impact of the number of controllers on the MCU under varied settings of the controller capacity.



(a) Validation for the campus network.



(b) Validation for the datacenter network.

Fig. 5. The highest MCU across all failures under different failure recovery strategies.

5.2.2 *Validating the design of a control plane*

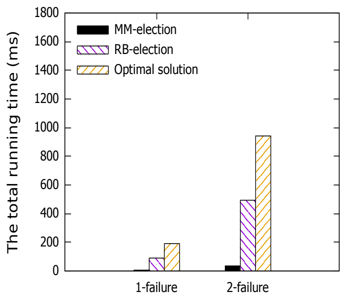
We verify if the design of a control plane meets the demand of the MCU, under different settings of the amount and capacity of controllers.

The campus network. The number of controllers deployed in this campus network increases from 3 to 10, while each controller manages the same number of switches as possible. The capacity of each controller changes from 10k to 50k. Fig. 4(a) plots the validation results of the control plane. If the design demand is that the MCU should be lower than 70% when no failures occur, the control plane needs at least 4 controllers, and each controller has the capacity of 50K. Accordingly, 10 controllers, each of which has the capacity of 20K, are also enough to meet the design demand. In addition, if the control plane requires that the MCU does not exceed 40%, 5 controllers are enough to meet the demand where each controller has the capacity of 50K.

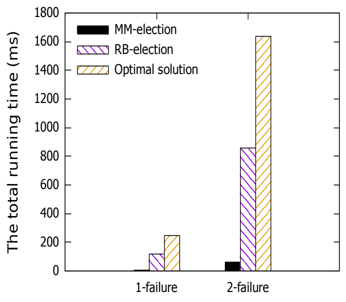
The datacenter topology. To reflect the difference in the amount of flow requests sent out by each switch, the number of flow requests resulting from each server is randomly set from 100 to 1000. The number of controllers varies from 8 to 15, and the capacity of each controller increases from 60k to 100k, Fig. 4(b) shows the changing trend of the MCU with the increase of the number of controllers. We can see from Fig. 4(b) that the MCU decreases as the number of controllers increases, and a lower MCU is achieved when each controller has higher capacity. Furthermore, the validation results can guide the design of the control plane. For example, if the demand of interest is that the MCU is under 60%, Fig. 4(b) indicates that the control plane requires at least 11 controllers, and the capacity of each controller should be 100K. However, the setting of 15 controllers is also sufficient to meet the demand of interest if each controller has the capacity of 80K.

5.2.3 *Validation across failure scenarios*

In this section, we validate the performance of the control plane across various failure scenarios. The key issue is to find the worst failure scenario. Note that the control plane can not cope with the simultaneous failure of three controllers in the worst case when each switch is only assigned one master controller and two slave controllers. Here, the three controllers of a switch may fail at the same time. In this case, the switch would not be covered by any one controller. Therefore, we verify if the design of a control plane can meet the demand of utilization under the 1-failure and 2-failures scenarios. In addition, the control plane that maintains more slave controllers for each switch can resist the simultaneous failure of more controllers.



(a) Validation for the campus network.

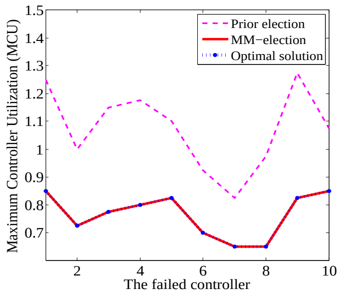


(b) Validation for the datacenter network.

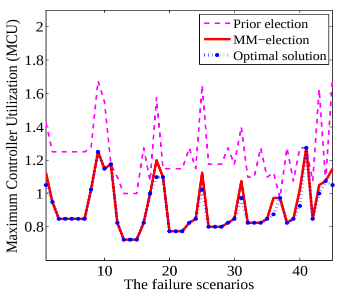
Fig. 6. The total running time of validation across all failures under different failure recovery strategies.

Figure 5 indicates the highest MCU across all failures under different failure recovery strategies. Fig. 5(a) and Fig. 5(b) plot the validation results across all 1-failure and 2-failures scenarios in the campus network and the datacenter network, respectively. The highest MCUs, achieved by our MM-election and RB-election strategies, are obviously lower than prior election strategy, and are very close to the optimal solution.

Fig. 6 shows the total running time of validation across all failures under different failure recovery strategies. Figs. 6(a) and 6(b) reflect the total validation time in the campus network as well as the datacenter network, respectively. We can see that our MM-election strategy significantly saves the running time than our RB-election (solving LPs) and the optimal solution (solving IPs) strategies. Moreover, as the network scale increases, the advantage of the running time of our MM-election strategy would be more obvious, especially when a failure scenario causes more uncovered switches. That is, our MM-election strategy takes less time to re-elect the new master controller for each uncovered switch. This makes the control plane can quickly (online) adapt to various failure scenarios. Although the validation can be offline, the failure recovery is required to make a decision online, which is latency-critical. Next, we further report the details of the validation under our MM-election strategy since it almost achieves the similar performance

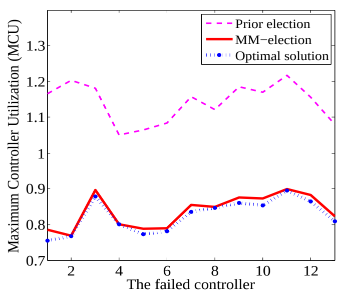


(a) Scenarios of one failure.

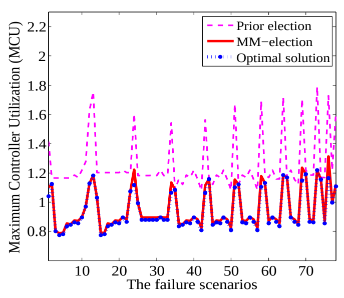


(b) Scenarios of two simultaneous failures.

Fig. 7. Finding the worst failure scenarios, under different failure recovery strategies in the campus network.



(a) Scenarios of one failure.



(b) Scenarios of two simultaneous failures.

Fig. 8. Finding the worst failure scenarios, under different failure recovery strategies in the datacenter network.

with the optimal solution, while exhibiting considerably lower time complexity than the optimal solution.

Validation in the campus network. In this section, the campus network employs 10 controllers, each of which is set to process at most 20k flow requests per second. Fig. 7(a) plots that our MM-election strategy achieves the same MCUs as that of the optimal strategy under the scenarios of one failure. The similar results can be observed in Fig. 7(b) when the MCUs are lower than 1 under the scenarios of two simultaneous failures. However, the MM-election strategy achieves higher MCUs than the optimal strategy when the MCUs are higher than 1 in some failure scenarios. Meanwhile, in our experiments, we note that those peak values appear in Fig. 7(b) when the failure scenarios make the master and multiple salve controllers of some switches fail at the same time. In this case, those switches ony can be taken over by the controllers that have already exhibited high utilization. Furthermore, Fig. 7 reflects also that our MM-election strategy achieves significantly lower MCUs than the prior election strategy under various failure scenarios.

Validation in the datacenter network. For processing a large amount of flow requests in a datacenter, the control plane employs 13 controllers, each of which can deal with 80k flow requests per second. The evaluation results are plotted by Fig. 8(a). It is clear that all MCU do not exceed 1, when adopting our MM-election strategy. This means that the current design of the control plane can cope with all 1-failure scenarios under our MM-election strategy. Fig. 8(b), however, indicates that some scenarios of two failures cause that the MCUs exceed 1. Thus, the control plane can not withstand the worst scenario of two failures even adopting the optimal strategy. Moreover, we find that the highest MCUs are not the same under different failure recovery strategies in Fig. 8(b). The highest MCU is 1.315 and is achieved in the 76th failure scenario with our MM-election failure recovery strategy. However, when the control plane employs the prior failure recovery strategy, our validation framework identifies the 73th failure scenario as the worst case with the MCU of 1.79. Therefore, our validation framework closely interdepends to the employed failure recovery strategy , and it is critical to determine the failure recovery strategy before conducting the validation.

TABLE 2
Iterative capacity augmentation for the campus network.

Iteration Solution	Validation Step			Augmentation Step
Iteration step	MCU	The worst scenario	Identified Controller	New (old) capacities
1	1.275	c_7 and c_9	c_8	25.5K (20K)
2	1.25	c_1 and c_{10}	c_9	25K (20K)
3	1.2	c_3 and c_4	c_2	24K (20K)
4	1.175	c_2 and c_4	c_3	23.5K (20K)
5	1.15	c_2 and c_3	c_1	23K (20K)
6	1.1	c_3 and c_5	c_4	22K (20K)
7	1.05	c_1 and c_2	c_{10}	21K (20K)
8	1.05	c_8 and c_9	c_7	21K (20K)
9	1.05	c_7 and c_8	c_6	20.5K (20K)
10	1.0	–	–	–

5.2.4 *Guiding the augmentations of capacities*

In Section 5.2.3, we have found that the design of the control plane can not cope with all failure scenarios of two simultaneous controllers in the campus network even adopting our MM-election failure recovery strategy. To meet the demand of interest, Table 2 illustrates the iterative solution to incrementally augment the capacities of the controllers in the campus network.

The iteration procedure consists of two steps, including a validation step and an augmentation step. The first step verifies if $MCU \leq 1$. When $MCU > 1$, the validation step will identify the controller with the maximum utilization and the related worst failure scenario. For example, in the first row of Table 2, the controller c_8 has the maximum utilization and the worst failure scenario means that controllers c_7 and c_9 fail simultaneously. The second step is to minimize the augmentation cost of the identified controller, while coping with the worst failure scenario. Recall that the capacity of controller c_8 is 20K at the beginning. To cope with the simultaneous c_7 and c_9 failures, the capacity of controller c_8 should be added at least 5.5K. Then, in the next validation step, another MCU would appear. In this case, the new worst failure scenario would be identified and the augmentation step would be conducted again. The iteration terminates when $MCU \leq 1$ in the validation step. In addition, some practical constraints can be easily incorporated to the augmentation step.

6 Related work

Chang et al. proposed the robust validation of network designs [12], which aims to validate that the network designs provide assurable performance in a wide-area network. However, it is obvious that the proposed method is not applicable to our problem because it focuses on bounding worst-case link utilizations in the data plane, which is orthogonal to our work. In this paper, we focus on validating the worst-case performance of the control plane in SDN. Furthermore, we design two efficient failure recovery stratigies for the control plane.

Distributed SDN control plane: At current, there have been many researches on the design of distributed control plane in SDN. The Onix [6] distributed controller partitions application and network state across multiple controllers using distributed storage. Hyperflow [7] is an SDN controller where network events are replicated to the controller replicas using a publish-subscribe messaging paradigm among the controllers. ONOS [3] is an experimental controller platform that provides a distributed, but logically centralized, global network view and fault tolerance by using a consistent store for replicating application state. Panda et al. present the design of a simple coordination layer (SCL) that can seamlessly turn a set of single-image SDN controllers into a distributed SDN system, based on the eventual correctness [5]. Current controllers can manage those switches by using either a separate control network (out-of-band control) [4] or using the same networks as the one being controlled (in-band control). In this paper, we focus on the distributed control plane and just consider the out-of-band control mechanism between the control plane and the data plane. We leave the validation on the hierarchical control plane [19][20] and the control plane adopting the in-band control mechanism [5] as our future work.

Controller fault-tolerance: There have been several researches on the controller fault-tolerance, whose main concern is how to ensure consistency under controller failures. Statesman [21] takes the approach of allowing incorrect switch state when a master fails. Ravana [4] is a fault-tolerant SDN controller platform that processes the control messages transactionally and exactly once (at both the controllers and the switches). In addition, some researchers study the problem of dynamic controller provision [22], [23], [24], [25], which can dynamically re-assigned controllers to switches based on the variations of network states. However, our validation framework is able to verify if the control plane offers assurable performance across uncertain failures. Furthermore, our UMR is to re-select a new master controller for the switch from its slave controllers after assigning the master and slave controllers to each switch. Therefore, the dynamic controller provision is orthogonal to our work.

7 Conclusion

Little is known about how to validate whether the control plane offers assurable performance, especially across various failures. In this paper, we develop a general framework to derive the worst-case performance of the control plane under any failure recovery strategy, across various failure scenarios. We prove that designing an optimal recovery strategy is NP-hard, and further design two approximation solutions to minimize the MCU under failures. We then develop efficient methods to find the worst failure scenario, which leads to the highest MCU among all failure scenarios. Moreover, we show how our validation framework can effectively augment the capacity of the control plane. The extensive evaluations via a testbed and large-scale simulations demonstrate the promise and effectiveness of our design.

Acknowledgments

This work is partially supported by the National Natural Science Foundation for Outstanding Excellent young scholars of China under Grant No.61422214, National Natural Science Foundation of China under Grant Nos.61772544 and U1536202, National Basic Research Program (973 program) under Grant No.2014CB347800, the Hunan Provincial Natural Science Fund for Distinguished Young Scholars under Grant No.2016JJ1002, and the Guangxi Cooperative Innovation Center of cloud computing and Big Data under Grant Nos.YD16507 and YD17X11.

References

[1] S. Jia, X. Jin, G. Ghasemiesfeh, J. Ding, and J. Gao, "Competitive analysis for online scheduling in software-defined optical wan," in *Proceedings of IEEE INFOCOM*, May 2017.

[2] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *Proceedings of IEEE INFOCOM*, May 2017.

[3] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, and W. Snow, "Onos: towards an open, distributed sdn os," in *Proceedings of ACM HotSDN*, Chicago, Illinois, August 2014.

[4] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: controller fault-tolerance in software-defined networking," in *Proceedings of ACM SOSR*, June 2015.

[5] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "Scl: Simplifying distributed sdn control planes," in *Proceedings of 14th USENIX NSDI*, March 2017.

[6] 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 *Proceedings of USENIX OSDI*, Vancouver, BC, Canada, October 2010.

[7] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of USENIX INM/WREN*, SAN JOSE,CA, April 2010.

[8] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE Journal on Selected Areas in Communications*, vol. PP, no. 99, pp. 1–1, 2018.

[9] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Review*, vol. 53, no. 3, pp. 464–501, 2011.

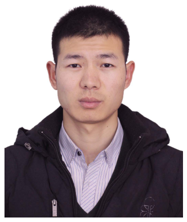
[10] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: Traffic engineering in dynamic networks," in *Proceedings of ACM SIGCOMM*, September 2006.

[11] D. Bertsimas and V. Goyal, "On the power and limitations of affine policies in two-stage adaptive optimization," *Mathematical programming*, vol. 134, no. 2, pp. 491–531, 2012.

[12] Y. Chang, S. Rao, and M. Tawarmalani, "Robust validation of network designs under uncertain demands and failures," in *Proceedings of 14th USENIX NSDI*, March 2017.

[13] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[14] "This wiki documents the current development version of onos (master)," [Online]. Available: https://wiki.onosproject.org/display/ONOS/, accessed April 2018.
[15] "Pica8 p-3297 white pages." [Online]. Available: https://www.pica8.com/wp-content/uploads/pica8-datasheet-48x1gbe-p3297.pdf, accessed April 2018.
[16] "Abilene traffic matrices." [Online]. Available: http://www.maths.adelaide.edu.au/matthew.roughan/project/traffic_matrix/, accessed April 2018.
[17] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu, "Minimizing flow statistics collection cost of sdn using wildcard requests," in *Proceedings of IEEE INFOCOM*, May 2017.
[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM*, August 2008.
[19] C. Chen, C. Liu, P. Liu, B. T. Loo, and L. Ding, "A scalable multi-datacenter layer-2 network architecture," in *Proceedings of ACM SOSR*, 2015.
[20] L. Fang, F. Chiussi, D. Bansal, V. Gill, T. Lin, J. Cox, and G. Ratterree, "Hierarchical sdn for the hyper-scale, hyper-elastic data center and cloud," in *Proceedings of ACM SOSR*, 2015.
[21] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin, "A network-state management service," *Acm Sigcomm Computer Communication Review*, vol. 44, no. 4, pp. 563–574, 2014.
[22] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *Proceedings of ACM HotSDN*, 2013.
[23] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An efficient elastic distributed sdn control plane," in *Proceedings of ACM HotSDN*, 2014.
[24] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of IEEE CNSM*, 2013.
[25] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *Proceedings of IEEE INFOCOM*, April 2016.



Junjie Xie Junjie Xie received the B.S. degree in computer science and technology from the Beijing Institute of Technology, Beijing, China, in 2013. He received the M.S. degree in management science and engineering from the National University of Defense Technology (NUDT), Changsha, China, in 2015. He is currently a Ph.D. student in NUDT, from 2016. He is also a joint Ph.D. student in the University of California, Santa Cruz (UCSC), USA, from October 2017. His study in the UCSC is supported by the China Scholarship Council (CSC). His research interests include distributed systems, software-defined networking and edge computing.

Deke Guo Deke Guo received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of Information System and Management, National University of Defense Technology, and a Professor with the School of Computer Science and Technology, Tianjin University. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.

Chen Qian Chen Qian is an Assistant Professor at the Department of Computer Engineering, University of California Santa Cruz. He received the B.Sc. degree from Nanjing University in 2006, the M.Phil. degree from the Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from the University of Texas at Austin in 2013, all in Computer Science. His research interests include computer networking, network security, and Internet of Things. He has published more than 60 research papers in highly competitive conferences and journals. He is a member of IEEE and ACM.

Bangbang Ren Bangbang Ren received the B.S.degree and M.S.degree in management science and engineering from National University of Defense Technology,Changsha,China,in 2015 and 2017. He is currently a Ph.D. student in NUDT. His research interests include software-defined network,data center network and network function virtualization.

Honghui Chen Honghui Chen received the MS degree in operational research and the PhD degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 1994 and 2007, respectively. Currently, he is a professor of Information System and Management, National University of Defense Technology, Changsha, China. His research interests include information system, cloud computing and Information Retrieval.