PROACTIVE CONTROLLER ASSIGNMENT SCHEMES IN SDN FOR FAST RECOVERY

by

Selcan Güner

B.S., Computer Engineering, Boğaziçi University, 2011

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2019

PROACTIVE CONTROLLER ASSIGNMENT SCHEMES IN SDN FOR FAST RECOVERY

APPROVED BY:

Prof. Fatih Alagöz (Thesis Supervisor)	.....
Prof. Tuna Tuğcu	.....
Assoc. Prof. Berk Canberk	.....

DATE OF APPROVAL: 30.07.2019

ACKNOWLEDGEMENTS

I would like to thank first and foremost to my family for their support and motivation that encouraged me during my academic career.

I would like to offer my deep gratitude to my instructor and thesis supervisor Prof. Fatih Alagöz, who has given me the chance to continue my research work despite a very long thesis period and sometimes lacking the appropriate level of effort on my side. I have been able to finish my M.S. thesis thanks to his support and supervision.

I am also deeply thankful to my friends and fellow graduate students at SATLAB, who have supported and inspired through thesis process.

Last but not the least, I deeply thank Gürkan Gür for all his support and guidance on this very long way to finish line.

ABSTRACT

PROACTIVE CONTROLLER ASSIGNMENT SCHEMES IN SDN FOR FAST RECOVERY

Software-defined networking (SDN) is an emerging networking paradigm which entails separate control and data planes. In SDN, a controller is a centralized network control entity responsible for management of the data plane. Typically, a sizeable software defined network with a single controller responsible for all forwarding elements is potentially failure-prone and inadequate for dynamic network loads. To this end, having multiple controllers improves resilience and distributes network control load. However, when there is a disruption in the control plane, a rapid and performant controller-switch assignment is critical, which is a challenging technical question. Therefore, controller-to-switch assignment and robust operation are challenging research questions, since control plane hazards such as disconnection between control and forwarding planes may lead to packet loss and failures in the system. In this thesis, we propose a proactive switch assignment scheme PREF-CP in case of controller failures using genetic algorithm considering controller load distribution, reassignment cost and probability of failure. Moreover, we compare the performance of our scheme with other algorithms developed during our thesis work, namely random and greedy algorithms. Experiment results show that our proposed PREF-CP framework has better assignment performance in terms of probability of failure and controller load distribution.

ÖZET

YAZILIM TANIMLI AĞLARDA HIZLI KURTARMA İÇİN PROAKTİF DENETLEYİCİ ATAMA YÖNTEMLERİ

Yazılım tanımlı ağ (YTA), veri ve kontrol katmanlarını birbirinden ayırmak üzere ortaya çıkan ağ paradigmasıdır. YTA'da bir denetleyici, veri düzleminin yönetiminden sorumlu olan merkezi ağ kontrol birimidir. Tipik olarak, tüm anahtar elemanlarından sorumlu olan tek bir denetleyiciye sahip büyük bir yazılım tanımlı ağ, mevcut veri yükü ihtiyaçlarının artması durumunda sistem yükünü taşımak için yeterli olmayacaktır. Bu amaçla, birden fazla denetleyiciye sahip olmak, sistemin esnekliğini arttırmakla kalmayıp, ağ denetim yükünün kontrolcüler arasında dağıtılmasını da sağlar. Birden fazla kontrolcünün olduğu YTA'larda, anahtar-denetleyici ataması çözmesi zor bir teknik problemdir. Kontrol düzleminde bir kesinti olduğunda, kontrol ve veri katmanları arasındaki kopukluk sistemde paket kaybına ve arızalara neden olabilir. Bu yüzden, sistemdeki bir kesinti durumunda hızlı ve performanslı bir kontrolcü-anahtar ataması önemli bir rol taşır. Bu tezde; denetleyici yük dağılımını, yeniden atama maliyetini ve arıza olasılığını göz önünde bulundurarak genetik algoritma kullanan; denetleyici arızalarında proaktif olan bir anahtar-denetleyici atama şeması PREF-CP'yi öneriyoruz. Tez çalışmamız sırasında, şemamızın başarımını geliştirilen benzer görevli diğer algoritmalar ile (rastgele ve açgözlü algoritmalar) karşılaştırdık. Deney sonuçları, önerilen PREF-CP şemasının, arıza olasılığı ve denetleyici yükü dağılımı açısından daha iyi başarıma sahip olduğunu göstermektedir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xi
LIST OF ACRONYMS/ABBREVIATIONS . . . xii
1. INTRODUCTION . . . 1
1.1. Main Contributions . . . 2
1.2. Thesis Outline . . . 4
2. RELATED WORK . . . 6
2.1. Multiple Controllers . . . 6
2.2. Failure Recovery and Reassignment Metrics . . . 7
2.2.1. Reliability . . . 8
2.2.2. Controller Load . . . 9
2.3. Backup Strategies . . . 10
3. PROACTIVE CONTROLLER-SWITCH ASSIGNMENT: PRELIMINARIES 12
3.1. OpenFlow . . . 12
3.2. Failure Scenarios . . . 13
3.2.1. Controller Failure . . . 13
3.2.2. Switch Failure . . . 13
3.2.3. Link Failure . . . 14
3.3. Topology Management Metrics . . . 14
3.3.1. Probability of Connectivity Failure (PoF) . . . 14
3.3.2. Controller Load Distribution . . . 15
3.3.3. Reassignment Cost . . . 16
4. PREF-CP: PROACTIVE RECOVERY FOR SDN CONTROL PLANE . . . 17
4.1. Optimization Model . . . 17
4.2. Objective Formulation . . . 19

4.2.1. Probability of Connectivity Failure $\tilde{P}$ . . . 19
4.2.2. Controller Load Distribution σ . . . 19
4.3. Constraints . . . 20
4.4. PREF-CP Scheme . . . 20
4.4.1. Fitness Function . . . 21
4.5. Other Assignment Algorithms . . . 22
4.5.1. Random Assignment . . . 22
4.5.2. Inter-Controller Greedy Algorithm (ICA) . . . 22
5. EXPERIMENTS AND RESULTS . . . 24
5.1. Experimental Setup . . . 24
5.1.1. Emulation and Simulation Tools . . . 24
5.1.1.1. ONOS . . . 24
5.1.1.2. Mininet . . . 25
5.1.1.3. Docker . . . 26
5.2. PREF-CP Implementation . . . 26
5.3. Simulation Setup . . . 28
5.3.1. Experiments . . . 28
5.3.1.1. Traffic Model . . . 31
5.4. Experiment Results . . . 31
5.4.1. Reassignment of Switches After Failure . . . 31
5.4.2. Impact on Load Distribution . . . 33
5.4.3. Load Distribution vs Probability of Connectivity Failure (PoF) . 35
5.4.4. PoF Characteristics . . . 37
5.4.5. Algorithmic Computation Times . . . 38
5.4.6. Controller Load Behavior over Time . . . 38
5.4.7. Cascaded Failure Characteristics . . . 40
5.4.8. Reassignment Cost . . . 41
5.4.9. ONOS Standard Assignment Algorithm vs PREF-CP . . . 42
6. CONCLUSION . . . 44
REFERENCES . . . 45

LIST OF FIGURES

Figure 1.1. Network Evolution from Conventional to Programmable Networks. 3

Figure 4.1. General Flow of Genetic Algorithm. 21

Figure 4.2. Genetic Algorithm 22

Figure 4.3. ICA 23

Figure 5.1. ONOS Cluster Synchronization. 25

Figure 5.2. PREF-CP Architecture. 27

Figure 5.3. Simulation Setup. 29

Figure 5.4. Internet2 OS3E Topology. 29

Figure 5.5. Controller-Switch Assignment Before Failure. 32

Figure 5.6. Controller-Switch Assignment After Controller Failure. 32

Figure 5.7. PACKET_IN Message Distributions for Different Algorithms. . . . 33

Figure 5.8. PREF-CP Load Distribution σ (pim) for Different Topologies. . . 35

Figure 5.9. Individual Controller Loads in pim After Controller-3 Fails. 36

Figure 5.10. PREF-CP Performance - Load Distribution σ (pim) and Probability of Failure. 36

Figure 5.11. PoF Calculated by Different Algorithms. 37

Figure 5.12. PoF Values for Different Topologies. 38

Figure 5.13. Run-times for PREF-CP and ICA algorithms. 39

Figure 5.14. Controller Loads Over Time - PREF-CP $\alpha = 0.7$ $\gamma = 0.8$. 39

Figure 5.15. Load Distribution Under Controller Failures for Different Topologies. 40

Figure 5.16. Load Distribution Under Controller Failures for Different Algorithms. 41

Figure 5.17. Individual Controllers' Loads After C3 and Then C1 Fail. 42

Figure 5.18. ONOS Reassignment Algorithm (baseline) vs PREF-CP. 43

LIST OF TABLES

Table 4.1. System Parameters. 18

Table 5.1. Test Network Topologies. 30

Table 5.2. Simulation Parameters. 30

Table 5.3. Load Distribution For Single Controller Failure. 34

Table 5.4. PREF-CP Load Distribution σ (pim) vs Number of Switches S. . . 34

Table 5.5. Load Distribution For Two Controller Failure Scenario. 41

Table 5.6. Reassignment Cost. 42

LIST OF SYMBOLS

E	Set of links
L_c	Load of controller
L_cn	Load of switch n assigned to Controller c
$\tilde{P}$	Probability of failure
P_e	Probability of link failure
P_v	Probability of failure of switch n
R_{ab}	Shortest path from switch a to b
$\tilde{R}$	Reassignment cost
S	Set of switches
S_f	Number of switches of failed controller
V	Set of nodes
x_{cn}	Switch n assigned to Controller c
z_{cn}	Switch n's new assignment to Controller c
α	Weight parameter in objective function
γ	Reassignment cost constraint multiplier
σ	Controller load distribution
Θ	Genetic algorithm maximum population size

LIST OF ACRONYMS/ABBREVIATIONS

D-ITG	Distributed Internet Traffic Generator
ICA	Inter-Controller Greedy Algorithm
IoT	Internet of Things
JVM	Java Virtual Machine
MM	Monitoring Module
OF	OpenFlow
ONOS	Open Network Operating System
pim	PACKET_IN Messages
PoF	Probability of Failure
PREF-CP	Proactive Recovery Framework for SDN Control Plane
RM	Reassignment Module
SDN	Software-Defined Networking
SPoF	Single Point of Failure

1. INTRODUCTION

Conventional computer networks consist of network elements such as switches and routers with many complicated protocols, policies and interfaces implemented on them. Their structure is restrictive for network operators to change the network with the needs of changing traffic demands and service requirements, which are becoming more taxing with the increasing number of mobile devices and impact of big data flows [1, 2]. The proliferation of advanced wireless systems such as Beyond 5G networks and massive connectivity of IoT impose stringent performance requirements on wired networks such as backhaul and core networks [3]. The idea of Software Defined Networking (SDN) is proposed to facilitate network evolution while addressing these challenges to realize the Future Internet concept [4]. With SDN, control decisions and network intelligence is moved out of individual network nodes, which transforms the network to simpler forwarding hardware augmented with decision making network controllers.

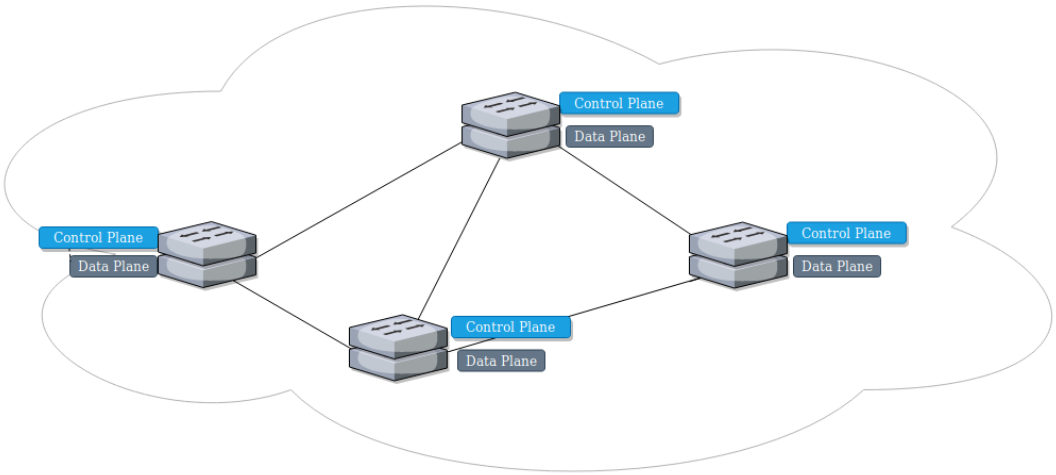
For most systems, a single controller might be enough since a single controller has a limited level of capacity [5]. Furthermore, resilience is an important aspect when designing a network architecture. As the system is performing, failures can occur in both control and forwarding planes. A switch, a controller or links between them may fail. For instance, when a switch is disconnected from its controllers, it can not forward new flows and becomes unresponsive except residual flows in time. A failure may eventually cause loss of data which reduces the reliability of the system. Therefore, improving resilience in a software-defined networks has great importance [2]. In a network with a single controller, when that controller fails, the network will be left without a control framework, i.e. become "headless". To overcome this single point of failure (SPoF) problem, control plane is distributed over multiple controllers to increase resilience and provide adaptation to dynamic network loads [6]. The distribution of switches to controllers influences multiple aspects of network system such as controller-switch latency, load balancing and network reliability [7]. Therefore, with use of multiple controllers, the dynamic and responsive controller-switch assignment becomes crucial.

SDN has great potential to change the way networks operate. The practical benefits of SDN range from centralized control, simplified algorithms, commoditizing network hardware and enabling the design and the deployments of third-party apps. Traditional routers and switches are mostly closed systems, and every routes and switch provides a limited and vendor specific control interfaces. Therefore, once they are deployed and in production, it is difficult to update and evolve network infrastructure according to changing needs. Basically, deploying new versions of existing protocols or deploying new protocols and services are obstacles in evolution of current networks. As seen in Figure-1.1(a), since data and controller planes are tightly coupled, deployment of new network applications or functionalities are not easy. Various network devices provide different control interfaces, which makes it a harder task and causes and high effort to configure network or policy enforcement due to lack of a single control interface for all devices [1].

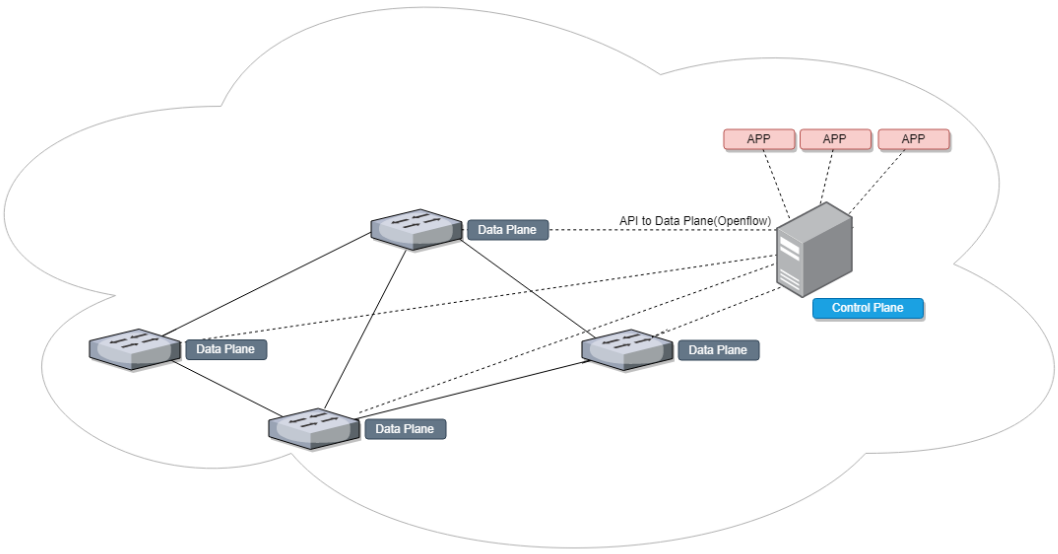
SDN is developed to speed up innovation and provide a simpler programmatic control interface for network. As visualized in Figure-1.1(b), the separation of data and controller planes make it easier to maintain network management, monitoring and deploying new applications and protocols. Instead of enforcing policies and running protocols on closed system devices, control decisions and network intelligence are moved out of network devices into decision making network controllers.Thus, network devices in data layer are reduced to simple forwarding hardware [1]. However, as noted above, how controllers are deployed, configured and assigned is critical for this promising paradigm, which we elaborate in this work form the aspect of controller assignment, recovery and resilience.

1.1. Main Contributions

In case of controller failures, the load of the failed controller will be distributed between other controllers which may result in the overload of one or more controller. If there is physical or widely-impacting problem that causes a controller to fail such as natural disasters or flash crowds, there is a higher possibility that another controller to be failed might be the closer one to failed one. In this work, in order to prevent and



(a) Conventional Data Network Architecture.



(b) SDN Architecture.

Figure 1.1. Network Evolution from Conventional to Programmable Networks.

to reduce chance of multiple controller failures, we consider calculating a backup map for possibility of consecutive controller failure. Thus, so called "cascaded controller failure" is an important aspect to consider for efficient recovery of a data network.

In this thesis, we investigate an efficient controller-to-switch assignment scheme in SDN to overcome negative effects of control plane failures. We also consider controller load for preventing any future cascaded failures. We propose a proactive switch assignment scheme in case of controller failures via genetic algorithm considering controller load distribution, reassignment cost and probability of failure. The main premise is to have a pre-calculated mapping for the network calculated at run-time and applied rapidly when failure incidents happen.

1.2. Thesis Outline

The thesis outline is as follows. For presenting the current state of research related to our research topic and providing a background, we examine the current studies in the literature in Chapter 2.

In Chapter 3 the preliminaries for proactive controller-switch assignment are explained. Network components failures that might occur are described. That chapter is concluded by explaining topology management metrics on calculating proactive assignment maps.

In Chapter 4, we propose our controller-switch assignment framework PREF-CP and its components such as objective function, constraints and assignment algorithms that are implemented as part of PREF-CP.

In Chapter 5, we explain our simulation environment setup and the technical tools that are used for experiments. First, we give detailed information about the emulation and simulation tools. Then, we explain PREF-CP algorithm implementation as part of our simulation environment. Finally, we conclude that chapter with the presentation of results of our experiments.

In the last chapter, Chapter 6, we summarize and discuss the results of the proposed framework. Finally, we conclude the thesis with future research directions.

2. RELATED WORK

In this chapter, we present and discuss related work in the literature for our thesis topic, specifically multiple controller frameworks, failure recovery, reliability, controller load issues and backup mapping (controller-switch assignment) strategies.

2.1. Multiple Controllers

Physical centralization of the network intelligence in one controller increases the risk of single point of failure (SPoF) compromising network reliability and availability. In a network managed by a single controller, when that controller fails, the network will be left without a control framework [8]. For instance, during disaster scenarios if network has only one controller, its failure is catastrophic. Therefore, to overcome the SPoF problem and increase resilience, control plane is distributed over multiple controllers [6]. That way, when one controller is wrong, the other controllers can take over the whole data plane. A multi-controller SDN should maintain a logically centralized view even though control plane itself is physically distributed among multiple controllers while considering resilience and load requirements [2].

A multi-controller setup is also beneficial to simultaneously provide adaptation to dynamic network loads [5]. When network traffic surges due to user or application dynamics, a single controller's limited capacity may be insufficient to respond to flow requests received from switches. A load-balancing approach may provide higher resilience against overloading in such scenarios.

Multi-Controller Challenges Having multiple controllers can solve SPoF encountered by single controller networks. Nevertheless, new technical challenges/questions are introduced in software-defined networks. With the use of multiple controllers, one of the critical issues is how to assign switches to controllers for management. Furthermore, once there is more than one controller in the network, it brings the question of how to address issues related with reliability and load balancing. Besides that, con-

trollers may suffer from the unexpected failures due to bugs and external malicious attacks, which influence the reliability of the control plane. Thus, how to balance controllers and reliability of network are important topics for multi-controller SDN research [8].

2.2. Failure Recovery and Reassignment Metrics

As a software-defined network is operating, failures can occur in control or forwarding planes due to catastrophic events, security incidents and configuration errors. A switch, a controller or links between them may fail. As considered in [5], a static assignment configuration may not be sufficient forever since the network conditions change over the time.

In case of a controller failure, the switches assigned to the failed controller must be migrated to an alternative controller. One can randomly define a backup controller in advance or define mechanism to calculate new switch-to-controller assignment map considering metrics such as load, reliability and connectivity. As studied in [9], during controller failures, a recovery mechanism can pick a controller with the highest priority or alternative controllers can be chosen according to controller load and CPU utilization. However, a proper solution needs to handle failure recoveries in a feasible and optimal way during the calculation of the switch assignment so that the chance of controller overload after failures are reduced [10].

In papers [11,12], they have examined that load is not distributed evenly through all controllers while the network is operating. The proposed system, *ElastiCon*, is a load balancing mechanism which monitors loads periodically, and if there occurs an imbalance, the load is distributed from highly loaded controller to lightly loaded controllers. Load balancing is done for only migrating selected switches to their new master controllers. However, if we consider load balancing during a failure, more switches might need to be distributed in addition to the failed controller's switches. Therefore, a global reassignment is needed for having a better distribution of load.

When calculating controller-switch assignment in SDN, multiple parameters such as controller load, probability of failure between network components or reassignment cost are considered [5, 13]. Controller failure recovery mechanisms in the literature are considering controller reliability of the network, switch-controller delay, or controller load. In regards to controller load, [10] proposed a reassignment algorithm which chooses a controller with highest controller capacity. For considering only switch-controller delay, [6] proposed a failure recovery mechanism that, for switches under control by a failed controller, the nearest controller takes over these switches.

2.2.1. Reliability

To assure the reliability of the control plane, the most important aspect is to keep switches connected to controllers which are up and running. For making this condition realized, failures of the controllers must be prevented so that they can work properly [14]. In evaluating a network design, reliability and resilience of a software-defined network are important while reliability of SDN control plane has great importance in failure handling in SDN since a failure may cause widespread loss of data and system unavailability [2].

For failure recovery or for protection from failures, there are various metrics developed or proposed in the literature. In [7], probability of failure is defined as likelihood of communications failure from node a to node b. The authors try to minimize the average disconnection chance in the split-architecture network by minimizing the probability of failure. The resilience of connection between controller and switches is studied in order to minimize the data loss in case of node and link failures. Similarly, a minimized number of hops between switch and controller can reduce probability of failure because fewer nodes in a path are prone to fewer failures [15]. Connectivity, which is defined as the highest number of disjoint paths between switch and controller, is also an important factor in calculation of reliability of network [10].

Another reliability metric that is defined to analyze reliability of the SDN control layer is the expected percentage of control path loss, where the control path loss is the

number of broken control paths due to network failures. The optimization target for a reliability-enhancing scheme is then to minimize the expected percentage of control path loss [16].

A switch may lose connectivity to its controller due to failures on the intermediate links or nodes along the path. Thus in [7], Beheshti and Zhang proposed a protection metric for having a resilient connectivity from a controller to its switches. The algorithm ensures connectivity by taking distance and resiliency into account and building a routing tree. To increase connectivity, the routing tree is calculated considering shortest path between the switches and the controller which results is higher resiliency.In [10], Muller proposed enhancing connectivity by increasing path diversity with using disjoint paths.

2.2.2. Controller Load

The increasing load of controllers can be the one of the main reasons of cascaded failures. Beside the case of controller failure, the load of the controllers which carry the load of the failed controller can exceed the capacity of them, and then cascading failures of controllers might happen. The counteraction for a failed controller may result in ripple effects, which again may cripple the system operation [17].

As studied in [10,18], controller capacity and controller overload are critical issues on switch assignment planning. To assign switches of failed controllers, the controller instance with the highest capacity can be selected.

Multi Controller Load Balancing Even though there is no failure, a balanced controller load is important for preventing future controller failure. Network traffic may vary and if due to the network traffic variation and if mapping between switches and controllers is static, this may cause some controllers to be overloaded even though there are enough capacity in other controller in the network. On other words, if load is not distributed evenly, imbalanced load distribution among controllers will reduce the network performance. Therefore, for a given multi-controller SDN network, it is

essential to ensure the nice load balancing performance of multi-controller [8].

2.3. Backup Strategies

Physical separation of the control and forwarding planes reduces the reliability of the communication between these two structures [19]. If there is any failure that disconnects the forwarding plane from the controller, the forwarding plane could be disabled. Accordingly, fast recovery from network failures is an important requirement for satisfactory service quality in high-speed networks. To minimize disruptions in case of controller failure, quick reactions are needed with failover mechanism(s) ensuring that the system rapidly and efficiently recovers connectivity with remaining devices and links [6].

In addition to making the decision on according to which parameters the reassignment algorithms should run, what else important is when to calculate the reassignment map for counteraction. There are two types of recovery schemes – proactive and reactive recovery [19]. If the actions of controllers are calculated prior to failures, remedial actions can be taken quickly which is denoted as proactive recovery. On the other hand, reassignment can be computed after a failure has occurred, that is a reactive recovery. [6] and [18] also generate backup map before a failure occurs. There are also other works which calculate reassignment on the fly when a failure occurs [10].

Depending on the back-up calculation time, both of the approaches have pros and cons. Proactive approaches may be faster to recover from a failure. On the other hand, calculation on the fly may use live network data while proactive calculations will use the last state of the system when the calculation algorithm is run. However, to prevent long restoration and backup calculation time, as used in [15], a proactive approach is instrumental. Also in [7], the solution suggests pre-configuring some backup outgoing links for switches and reconnecting switches to controllers if a failure is detected. The strategy in [6] computes results of backup algorithm proactively while controllers share their state information about the network topology.

As discussed by Fang et al. [18], during failover incidents, to calculate controller-switch assignment schemes, local optimization or global optimization can be aimed. Recovery mechanisms that aims local optimization reassign switches one by one until all switches are assigned to some controllers. However, if global optimization is the goal, recovery algorithms compute the best switch-controller reassignment for the switches of the failed controller, and enforce switch reassignment accordingly. For instance, in [18], the recovery plan is derived using a genetic algorithm. The proposed scheme is better than alternative approaches because it aims for global optimizations and computes switch-controller reassignment with switch reassignments being done once for all of the switches.

3. PROACTIVE CONTROLLER-SWITCH ASSIGNMENT: PRELIMINARIES

In case of controller failure(s), there are different assignment algorithms that can be used to calculate backup assignment mapping between controllers and switches. As a simple approach, the switches of failed controllers can be randomly assigned to other controllers. However, the random placement of switches naturally do not provide efficient performance [13]. Thus, one needs a satisfactory assignment algorithm that deals with failures at run time in a rapid and efficient manner.

In this thesis, we propose an assignment calculation using genetic algorithm to increase the resilience of a running software-defined system as described in Chapter 4. The proposed scheme *PREF-CP (Proactive Recovery for SDN Control Plane)* calculates a backup switch assignment map considering each controller might fail and taking load balancing into account to recover the network from possible failures and adapt new conditions if a failure occurs despite all the efforts. Load aware switch assignment is essential to avoid overload, specially during fail over. Thus, the overall approach including other assignment schemes determines a backup controller for each forwarding node according to load of controllers, switch reassignment cost and probability of failure from a switch to its controller. In the case of controller failure, a switch will be assigned to its proactively calculated backup controller.

3.1. OpenFlow

Driven by the SDN principle of decoupling control and data planes, OpenFlow standardizes information exchange between these two planes. In the OpenFlow architecture, the forwarding device or OpenFlow switch contains one or more flow tables. Flow tables consist of flow entries each of which determines how packets belonging to a flow will be processed and forwarded. Upon a packet arrival at the OpenFlow switch, firstly flow tables are checked to find a matching flow rule. If a matching entry is found,

the switch applies the appropriate set of instructions associated with the matched flow entry. If the flow table look-up procedure does not result on a match the action taken by the switch is to forward the packet to controllers over the OpenFlow channel. Using OpenFlow protocol, a remote controller and associated network applications can, add update or delete flow entries from the switch's flow tables [1]. Regarding the relationship between an OpenFlow switch and controller, the switch has no local control plane logic and relies entirely on the controller to populate its forwarding table [13].

3.2. Failure Scenarios

3.2.1. Controller Failure

Software or hardware malfunction(s) on the machine hosting a controller can cause failures on the control plane. Since controllers are the most essential devices in the system, the probability of their failure is usually lower than the others due to extra precautions and hardening applied to them and their host platforms [20]. Nevertheless, a broken controller can no longer get flow requests from its switches and east-west interface messages from other controllers. When a controller fails, its switches must be assigned to other controllers.

When there is a malfunctioning controller, the network reconfiguration may cause one of the remaining controllers to overload or if there is a physical hazard such as a flooding or fire that causes controllers to fail, multiple controllers in proximity may be affected that can lead to cascaded failures [17].

3.2.2. Switch Failure

Failure in a switch results in that packets cannot be transferred from or to that switch. If a switch is connected to only another switch, and when the parent switch fails, child switches may also be affected from failure [21]. If a switch's parent fails, it will be parentless. Although it is an important problem to consider, parentless switches are out of scope of this work.

3.2.3. Link Failure

A failure in the link means that traffic over the link cannot be transferred any more [21]. When a link fails between a switch and its controller, if there are any alternative paths between them, one of those will be used to keep them connected. The next path between the node and controller is chosen by the shortest path criterion according to the number of hops. If there is no such path, even though the switch is working, it will be disconnected and traffic through that switch will not be transmitted anymore. In our proposed scheme, when a link failure occurs and if the affected switch has any other links, the assignment is calculated using this alternative path.

3.3. Topology Management Metrics

For an assignment to increase resilience and improve recovery for network components, it should utilize existing network connectivity among the switches. To maximize connectivity, network components can be connected to form a mesh topology. However, mesh connection will significantly increase the deployment cost and wiring complexity. Additionally, it is not scalable as the network size grows to a large number of switches spreading across multiple geographical locations [21].

3.3.1. Probability of Connectivity Failure (PoF)

If the connection between controller and the forwarding plane is broken because of network failures, some switches will be left without any controller and will be disabled. Network availability should be ensured to increase reliability of SDN. Since improving reliability is important to prevent future network failures, for defining a switch assignment algorithm, reliability must be considered as an assignment metric [22]. To formalize reliability for control plane availability better, it can be defined as the probability of connectivity failure of a path from a switch to its controller as described below in Equation 3.1:

$$1 - \prod_{e_i, v_j \in R_{ab}} (1 - P_{e_i})(1 - P_{v_j}) \tag{3.1}$$

where P_{e_i} and P_{v_j} are probability of link failure and switch failures and R_{ab} is the shortest path from switch a to b.

3.3.2. Controller Load Distribution

Network load balancing directly affects network performance, thus improving load balancing is important for a network [23]. In case of controller failure, when the switches are assigned to other controllers, this may cause some controllers to overload. Thus reassignment should be done taking controller loads into account. Controller load is calculated as total number of PACKET_IN messages(pim) from a controller's switches. Load variance is reflecting how this message count is distributed among controllers in the network. In our approach we consider controller load L_c and load variance σ^2 as described below:

$$ControllerLoad(L_c) = \sum_{i \in n} x_{C,n} R_n \tag{3.2}$$

$$LoadVariance(\sigma^2) = \frac{1}{c} \sum_{i \in c} (\overline{L} - L_c) \tag{3.3}$$

3.3.3. Reassignment Cost

When a controller fails, its controllers will be distributed among other controllers. Reassignment cost $\tilde{R}$ is considered when calculating reassignment map to measure the cost of applying that reassignment. It represents the trade-off between flexibility for assignments and the overhead of implementing them in a practical software-defined network. It can be calculated as in Equation 3.4 as the total number of switches whose controllers are changed in case of failure when reassignment is performed. Sepcifically, x_{ij} is the previous assignment map and z_{ij} is the new assignment of switches after failure. We take XOR of two values, i.e. if a switch is assigned to a new controller $z_{ij} = 1$, otherwise $z_{ij} = 0$.

$$\tilde{R} = \sum_{i \in S, j \in C} x_{ij} z_{ij} \tag{3.4}$$

The number of reassigned switches in the system will be at least as much as the number of switches of the failed controller. Besides, when network size increases, it is possible that reassignment cost will also increase. Thus, when setting reassignment cost as a constraint, we expect it to be smaller than the total number switches S multiplied with a scale factor γ.

4. PREF-CP: PROACTIVE RECOVERY FOR SDN CONTROL PLANE

In this chapter, we describe our proposed approach PREF-CP which relies on an optimization model solved using a genetic algorithm based approach. Moreover, we describe other assignment schemes which are used for performance evaluation.

4.1. Optimization Model

The SDN physical network presented as a graph which is indicated as $G(V, E)$, where V is the set of nodes, E is the set of links. Set of controllers is denoted as $V_c \in V$. The model parameters are listed in Table 4.1.

Objective Function: The goal of the proposed strategy is to keep in balance the controller load distribution uniformity and probability of failure minimization goals.

Failure Recovery The goal of the developed strategies is to recover when failure occurs so that system should continue processing with minimum harm. For each switch, it aims to find a backup controller with respect to some criteria and decision parameters. Specifically, the goal of Proactive Recovery Framework for SDN Control Plane (PREF-CP) is to calculate proactively to minimize controllers' load distribution and probability of failure while satisfying demand and reassignment cost constraints. Then the network can recover from controller failures in a robust setting. This reassignment approach also allows to ensure that network can be prevented from prospective cascaded failures; and even if failure occurs, network can recover from failures quickly. For that purpose, PREF-CP periodically checks the load of the controllers L_c according to the demand of each device n represented by R_n.

If a controller failure occurred, reassignment of the switches is performed according to proactively calculated reassignment map.

Table 4.1. System Parameters.

Symbol	Definition
V	set of nodes
S	set of switches
C	set of controllers
E	set of links
P_{fc}	Probability of failure of controllers
P_v	Probability of failure of switches
P_e	Probability of link failure
$\tilde{P}$	Probability of failure
γ	Reassignment cost multiplier
σ	Standard deviation for controller load distribution
U_c	Maximum load that controller C can handle
$(1-\alpha_c)U_c$	A controller's safe capacity
R_n	Number of requests of network device n

4.2. Objective Formulation

4.2.1. Probability of Connectivity Failure $\tilde{P}$

Failure probability P_{e_i} for link e_i and failure probability for P_{v_i} for node v_i (switch i) in the network, the chance that communications from node a to b fails is given in Equation 3.1. The product term in the Equation-3.1 is probability that all nodes and links on the path from node a to node b work finely [21]. The objective of a resilience-oriented reassignment scheme should include this parameter.

4.2.2. Controller Load Distribution σ

Equation 3.3 considers how existing control load in the system is distributed among controllers. If failed controller's switches are assigned randomly, one of the remaining controllers may overload which can cause cascaded failures [17].

On next assignments, when a failure occurs, the reassignment of switches should be calculated in a matter that probability of failure and load distribution are minimized.

Overall, the objective function for our reassignment optimization model has the form:

$$Min(\sigma + \tilde{P}) \tag{4.1}$$

4.3. Constraints

A switch will be controlled by exactly one controller:

$$\sum_{c \in C} x_{n,c} = 1, \quad \forall_n \in N \tag{4.2}$$

Controller capacity cannot be exceeded:

$$\sum_{n \in N} x_{c,n} R_n \leq (1 - \alpha_c) U_c \tag{4.3}$$

α values fall in range (0,1):

$$0 \leq \alpha \leq 1 \tag{4.4}$$

The reassignment cost $\tilde{R}$ can not exceed the value calculated by the number of switches S multiplied by γ:

$$\sum_{i \in S, j \in C} x_{ij} z_{ij} \leq \gamma \times S \tag{4.5}$$

4.4. PREF-CP Scheme

A genetic algorithm is a meta-heuristic that reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. The general workflow of a genetic algorithm is shown in Figure 4.1.

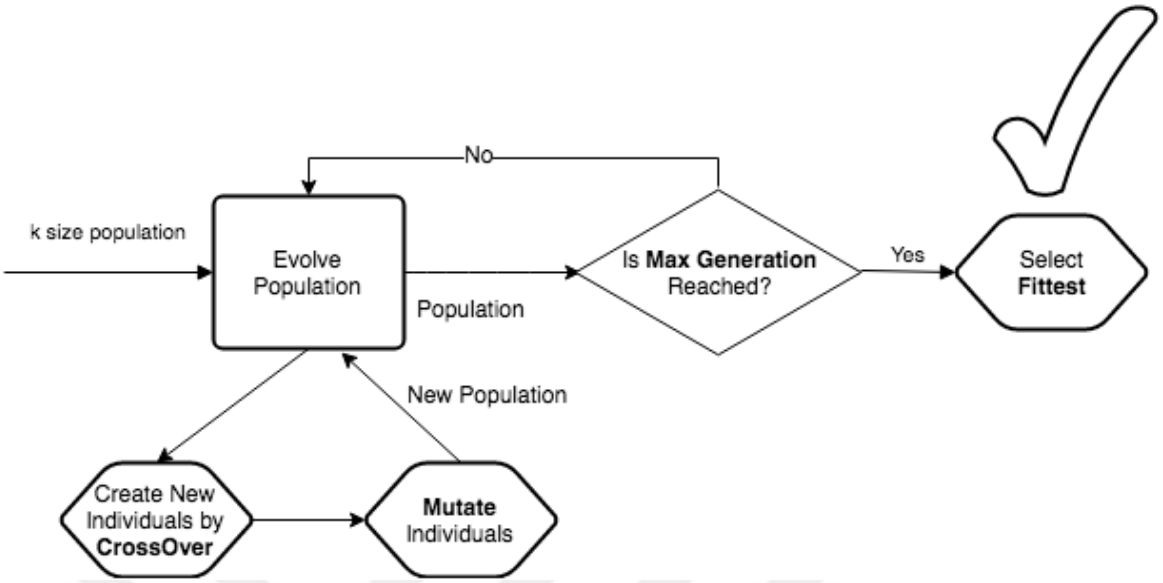


Figure 4.1. General Flow of Genetic Algorithm.

4.4.1. Fitness Function

The performance of a genetic algorithm relies on how well a fitness function is derived. Fitness function in our approach considers minimizing maximum probability of failure of the shortest path of a switch of the controller (P) and optimizing load distribution among controllers (σ).

Equation 4.6 is used during crossover for choosing which controller a switch will be assigned to for chromosome evaluation. That is, when the crossover operation has to choose a better gene from two parent genes during the crossover, Equation 4.6 is used to evaluate these two parent genes.

$$Eval = \alpha * \sigma + (1 - \alpha) * P \tag{4.6}$$

Require Input: Topology G, Switch Load S, Number of Controllers N, Probability of Failures, CrossoverFunction, $Population_size$
Ensure Generate solution x^c initialize R_{max} and T S_best
$Population \leftarrow InitializePopulation(Population_size)$
Evaluate population
$S_{best} \leftarrow$ BestSolution(population)
while population has not converged **do**
  Selection : Parents $\leftarrow$ SelectParent(Population, $Population_{size}$)
  Children $\leftarrow 0$
  Crossover
  if Compute fitness **then**
    pick parent
  end if
end while

Figure 4.2. Genetic Algorithm.

4.5. Other Assignment Algorithms

4.5.1. Random Assignment

Although this is a trivial algorithm, it is typically used as a baseline case for performance evaluation. Moreover, it can be practical for situations where very low-complexity requirements are in force. In random assignment algorithm, each candidate controller have a uniform probability of hosting a switch. In that case, reassignment algorithm randomly chooses a controller among all potential ones (i.e. ones that are still running).

4.5.2. Inter-Controller Greedy Algorithm (ICA)

This algorithm generates a list of the potential assignments, which is ranked increasingly based on failure probabilities of the shortest path from switch to controller

and load of switches. Then, it chooses one switch at a time for assigning to controllers. The algorithm iterates until all switches are assigned [24].

Require Input: Topology G, Switch Load S, Number of Controllers N, Probability of Failures P
for i in Devices Size **do**
  weight[i] = CalculateDeviceWeight (L, P)
end for
Sort switches s in ascending order of their weights considering failure properties P_f and loads L_n as set S'
while Devices left to add to Assignment Map **do**
  for j in Controllers Size **do**
    Among all devices select the device lowest weight from S'
    Add switch s_i to Controller c_j's backup map
  end for
end while
return AssignmentMap

Figure 4.3. ICA.

5. EXPERIMENTS AND RESULTS

In this chapter, we describe the experimental setup including utilized tools, implementation of PREF-CP and simulation results.

5.1. Experimental Setup

5.1.1. Emulation and Simulation Tools

5.1.1.1. ONOS. ONOS (Open Network Operating System) is an open-source SDN controller written in Java. It is used for control plane management for software-defined networks. Using ONOS, network components such as switches and links, software programs or modules to provide communication services from end hosts or neighboring networks can be managed.

Since ONOS is an operating system, it is an useful environment for developing software programs and applications for specific use cases. Those applications can be developed for monitoring services or for setting up network paths by visualizing network topologies in a web browser. With the help of the provided application subsystem, it is easier to deliver and manage software among multiple ONOS instances. All those instances that are members of same cluster share an inventory of applications. The application subsystem replicates this shared inventory by using a consistent map and inter-node communication. This approach helps new software components to be attached or existing components to be detached from ONOS without rebuilding and deploying ONOS itself.

An ONOS cluster consists of one or multiple ONOS instances. Each instance is identified with an unique identifier named as `NodeId`. Each of those instances in the network has information about a subsection of the network state. This local state information is distributed among all cluster nodes by events. The shared inventory stores those generated events and then shares it with all nodes by services that are built

into distributed stores. As seen in Figure 5.1, the same subsystems of two different nodes synchronize directly with one another through the Store [25].

OVerall, ONOS provides:

- A platform and set of applications that act as an extensible, modular, distributed SDN controller.
- A configurable software that helps hardware and services to be managed and deployed easily.
- A scale-out architecture to provide the resiliency and scalability required to meet the continuous changes in production environments.

ONOS can run as a distributed system across multiple servers, allowing it to use the CPU and memory resources of multiple servers. The ONOS kernel and core services, as well as ONOS applications, are written in Java as bundles that are loaded into the Karaf OSGi container. With use of OSGi, which is a component system for Java, modules can be installed and run on a single JVM. Running ONOS on the JVM, enables a range of OS platforms to run ONOS on.

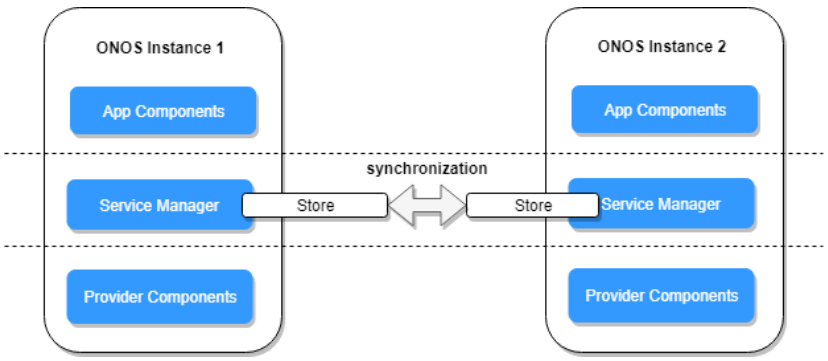


Figure 5.1. ONOS Cluster Synchronization.

5.1.1.2. Mininet. Mininet is a network emulator which creates a network of virtual hosts, switches, controllers and links. It simulates hosts and switches in separate network namespaces and connects them with virtual Ethernet interfaces. Mininet hosts run standard Linux network software, and its switches support OpenFlow for

highly flexible custom routing and Software-Defined Networking. Mininet allows an entire OpenFlow network to be emulated on a single machine, simplifying the initial development and deployment process [26].

5.1.1.3. Docker. Docker is a containerisation platform in which several applications can be created, deployed and run by using containers. Docker containers contain everything for an application to run by wrapping up pieces of software into a complete file system.

A docker container is a unit of software that packages an application's code and dependencies. Using a container, any application can be deployed and run rapidly in various environments. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at run time and in the case of Docker containers - images become containers when they run on Docker Engine. Software running on containers will always run same regardless of the infrastructure. Environment can differ such as development and staging, however applications that run on containers are isolated from their environments which secures applications to run uniformly despite all the differences [27].

5.2. PREF-CP Implementation

PREF-CP works proactively to enable system continue working in case of a controller failure in an SDN architecture. PREF-CP evaluates the instantaneous switch-to-controller assignment and calculates a backup map considering each controller might fail. For k number of controllers, k backup maps are calculated. If a reassignment is performed, PREF-CP recalculates the switch-to-controller assignment considering possible future failures. PREF-CP contains two modules as depicted in Figure 5.2 and explained below:

- *Monitoring Module - MM* tracks controllers' health and pulls relevant measurements and statistics. These data contain x_{cn} current assignment, L_{cn} controller loads, Probability of Failures P_n. MM also detects failures that occur in the control plane, and in case of controller malfunction informs the other component, reassignment module.
- *Reassignment Module - RM* periodically checks the collected statistics from the monitoring module and calculates reassignment map accordingly. In case of controller failure, reassignment module performs new assignment based on proactively calculated backup map.

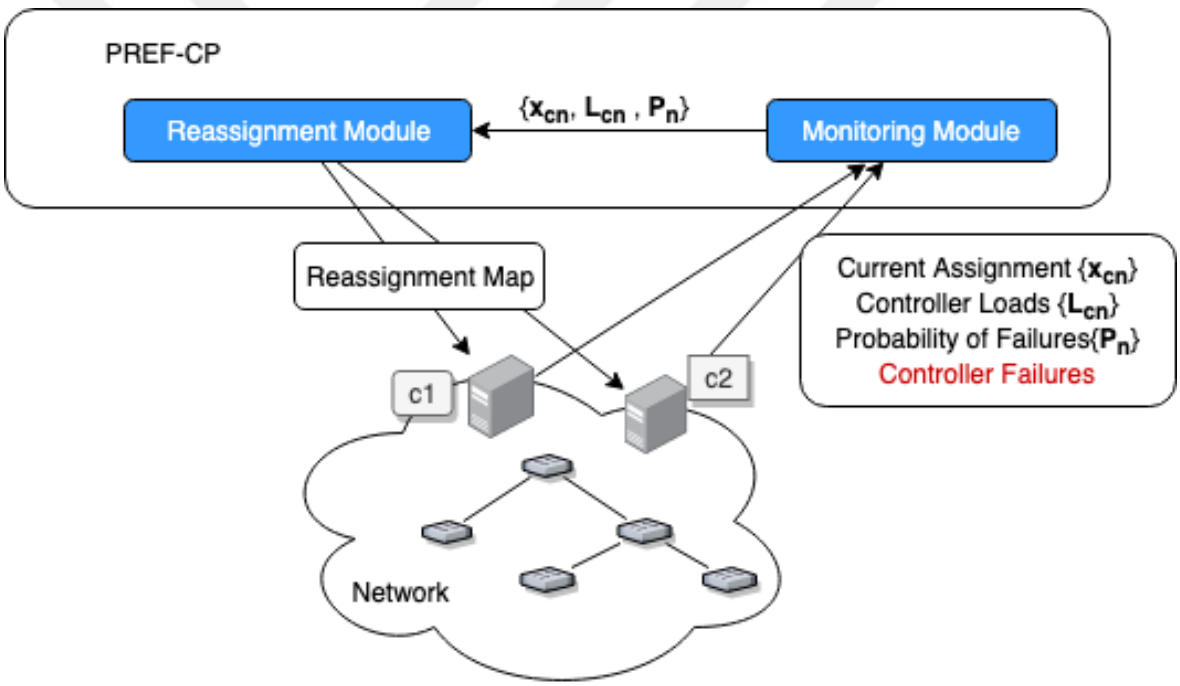


Figure 5.2. PREF-CP Architecture.

We have designed and implemented our proposed approach PREF-CP to work as an ONOS application. ONOS uses a replication application inventory that is shared by all ONOS instances in a cluster which provides all instances to know network states by communicating with each other [25].

5.3. Simulation Setup

We use ONOS Nightingale 1.13.1 as SDN controller to implement the controller plane and Mininet 2.2.1 for creating network topologies to implement data plane. All simulations are run on a physical machine with OS Ubuntu 14.1, 8GB RAM and Intel Core i7 6700HQ CPU. In our system, controllers are running inside Docker containers. We run Mininet on that physical machine and ran the controllers in Docker 17.09 containers. For traffic generation, we use Distributed Internet Traffic Generator (D-ITG) 2.8.1 to create traffic flows from hosts as shown in Figure 5.3. The switches run in Open vSwitch mode to deliver OpenFlow functionality, meaning that they implement the same interfaces as defined by Open vSwitch [28].

For network traffic, we generated the traffic flows of VoIP, video and two different types of online games with D-ITG traffic generator. D-ITG generates game traffic with Counter Strike characteristics related to the active phase of the game or an idle player. We ran our experiments on different network sizes and different network traffic flows. As seen in Table 5.1, our investigated topology sizes (T1-T5) contain 34, 40, 63, 85 and 121 switches. Topology T2 and T4 are used for testing algorithm runtime. Simulations are run on topologies T1, T3 and T5. Internet2 OS3E topology used as T5 is shown in Figure 5.4. In our system, five ONOS instances are actively running as the distributed controller plane during the simulations. For genetic algorithm implementation, the maximum number of generations per iteration was set to 50.

5.3.1. Experiments

In this section, we describe experiments that are carried out for performance investigation. We run our experiments in different topologies under random controller failures. Additionally, we run experiments to simulate *cascaded failures* where two controllers fail one after another. Our statistics collector is run at the time when ONOS instances start running while traffic generator starts and at t = 10 sec. At time t-reassignment, reassignment algorithm is performed using one of the reassign-

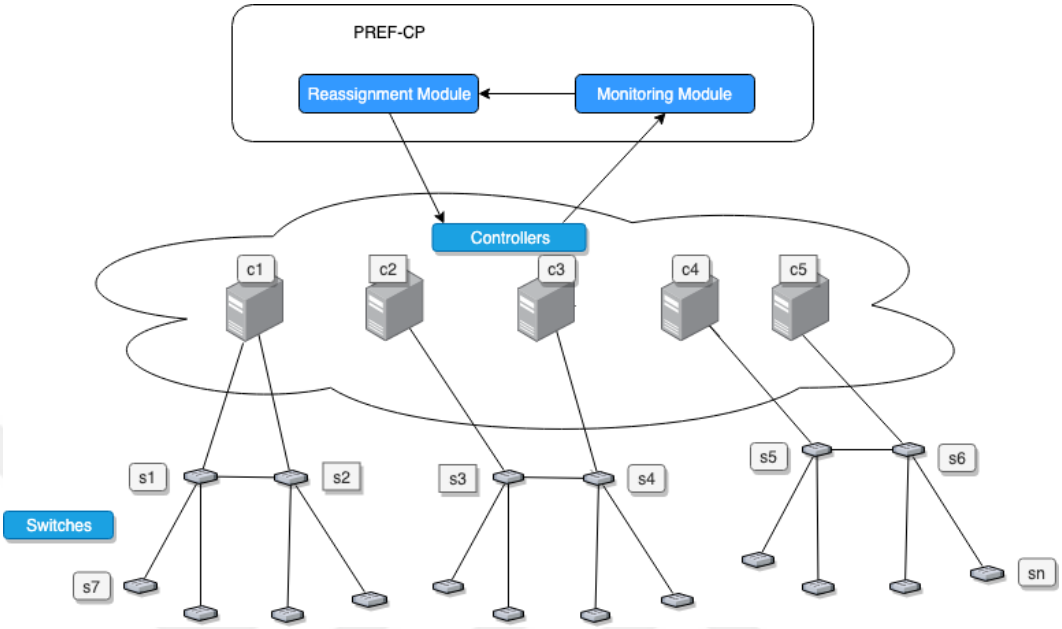


Figure 5.3. Simulation Setup.

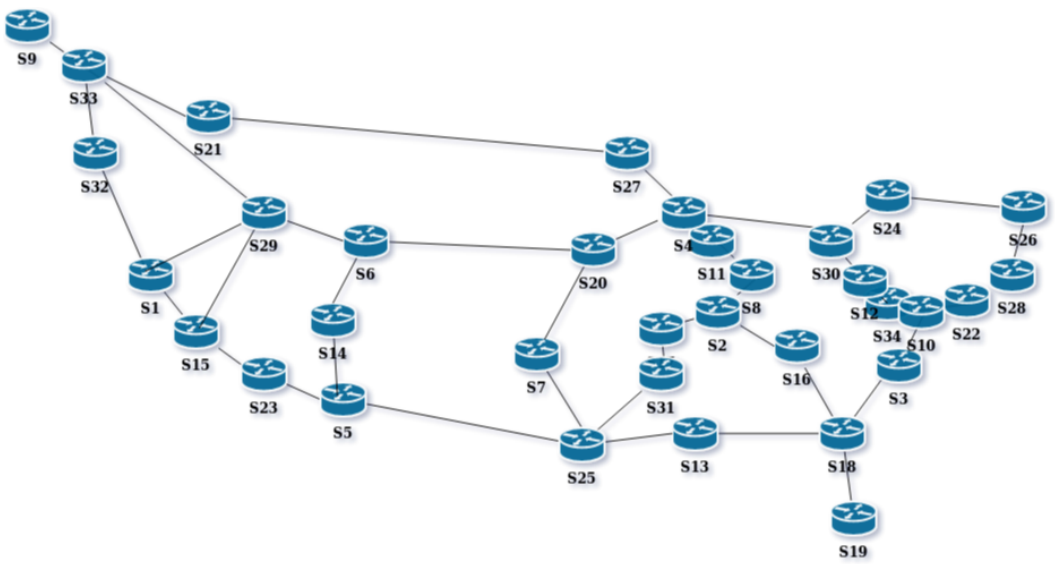


Figure 5.4. Internet2 OS3E Topology.

Table 5.1. Test Network Topologies.

Topology	**Type**	**Properties**	**Number Of Switches**	**Number of Hosts**
T1	Tree	Depth:4 Fanout:3	40	81
T2	Tree	Depth:4 Fanout:4	85	256
T3	Tree	Depth:6 Fanout:2	63	64
T4	Tree	Depth:5 Fanout:3	121	243
T5	-	Internet2 OS3E	34	105

Table 5.2. Simulation Parameters.

Parameter	**Symbol**	**Value**
Number of switches	S	{34, 40 , 63, 85, 121}
Weight parameter	α	{0.0, 0.5, 0.7, 1.0}
Maximum Population Size	θ	50
Number of controllers	V	5
Reassignment Cost Multiplier	γ	0.8

ment methods (random, ICA or PREF-CP) according to collected statistic values, PACKET_IN message load of controllers and probability of failures. At time t-failure a controller is artificially made to fail. When Monitoring Module detects that controller failure, switches are assigned to new controllers by the Reassignment Module.

We run random, ICA and PREF-CP algorithms ten times each for different α values shown in Table 5.2, randomly failing one controller on each run and then measuring load distribution and probability of connectivity failure.

For cascaded controller failure scenario, we run the same algorithms ten times for $\alpha = 0.7$ and $\gamma = 0.8$. In that case, the first controller is randomly failed and for second

controller failure we pick the controller with the highest load to see the after-effect impact of distributing load among other controllers after the initial failure occurred.

5.3.1.1. Traffic Model. For network traffic, we generated the traffic flows of VoIP, video and two different types of online games with D-ITG traffic generator. For packet generation, we generated 500 packets per second (pkt/s) with uniform distribution between minimum rate 500 - maximum rate 1000 pkt/s. Online game traffic generated by D-ITG emulates *Counterstrike* traffic for active or idle player [29]. At time point t=10 seconds, new flows are generated between different and randomly selected source and destination hosts. The number of generated flows in the network is equal to the number of hosts.

5.4. Experiment Results

In this section, we present and discuss experiment results of PREF-CP and other baseline algorithms in terms of various metrics considering PoF, load failures, reassignment cost and computational time of algorithms.

5.4.1. Reassignment of Switches After Failure

In Figure 5.5, a state is shown before there is any failure in the network. Initial switch assignment is done randomly, one by one all switches are assigned to randomly selected controllers. At time t-reassignment, reassignment algorithm is performed according to collected statistic values and when there is a failure, switches are assigned to new controllers calculated by reassignment algorithms. When Controller-2 fails, its switches are assigned to other controllers and for a more balanced load distributions some other switches' controller is also changed. After recovery state, the new configuration can be seen in Figure 5.6 with failed controller marked in orange.

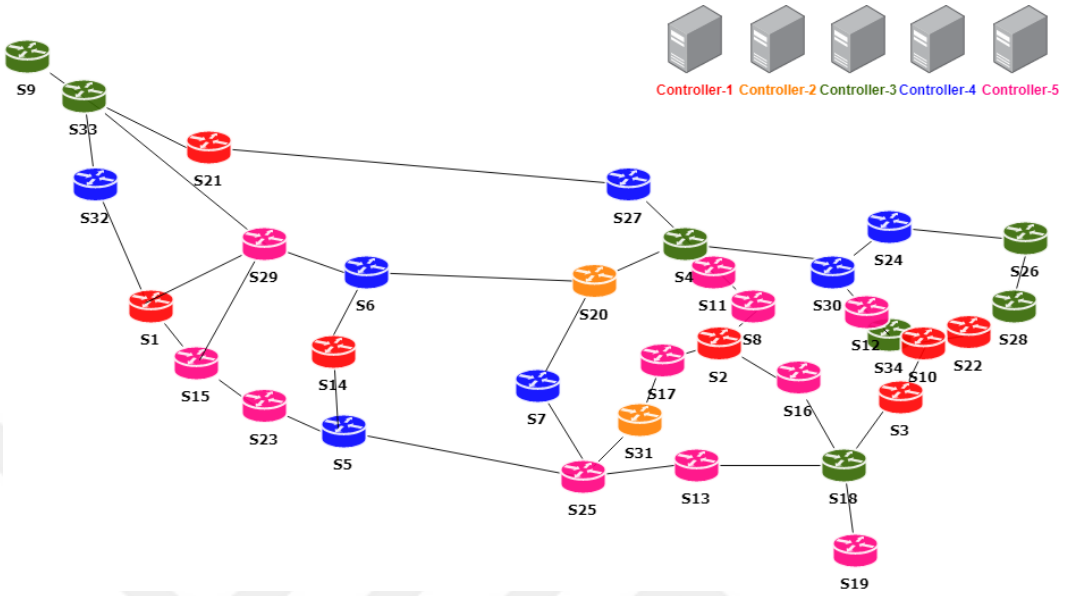


Figure 5.5. Controller-Switch Assignment Before Failure.

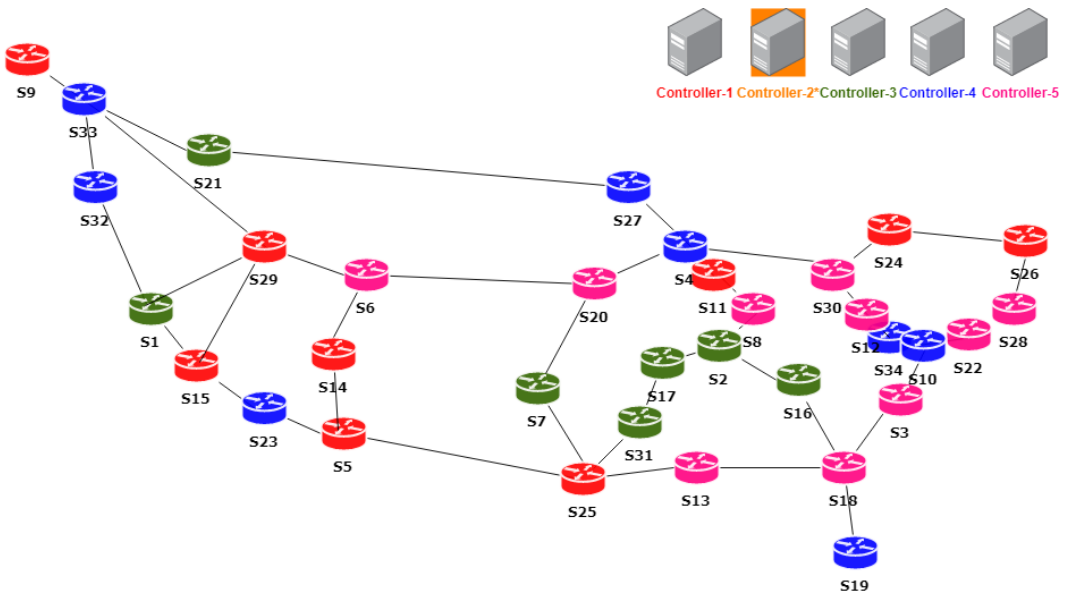


Figure 5.6. Controller-Switch Assignment After Controller Failure.

5.4.2. Impact on Load Distribution

Load distribution characteristics can be affected by several parameters of the network; specifically reassignment algorithms, α values and number of switches in the network. To see the effects of these parameters, we compared load variance of number of PACKET_IN messages that are distributed among controllers. Increasing α increases the priority of the load distribution as defined in Equation 4.1, thus load distribution σ decreases as α increase. The test results are showing that in Figure 5.7, PREF-CP distributes controller load by 8.38%, 11.5%, and 16.6% better from ICA for α values 0, 0.5, and 1, respectively. Random assignment is plainly for reference and it is not affected by the α since it assigns switches to random controllers without considering load distribution. For random algorithm, load distribution value equals to 9284.9 pim, which is higher than both PREF-CP: 3933.9 pim and ICA: 4438.9 pim. Although ICA algorithm outperforms random assignment, it is not as good as PREF-CP algorithm. The load distribution σ per algorithm can be seen in Table 5.3.

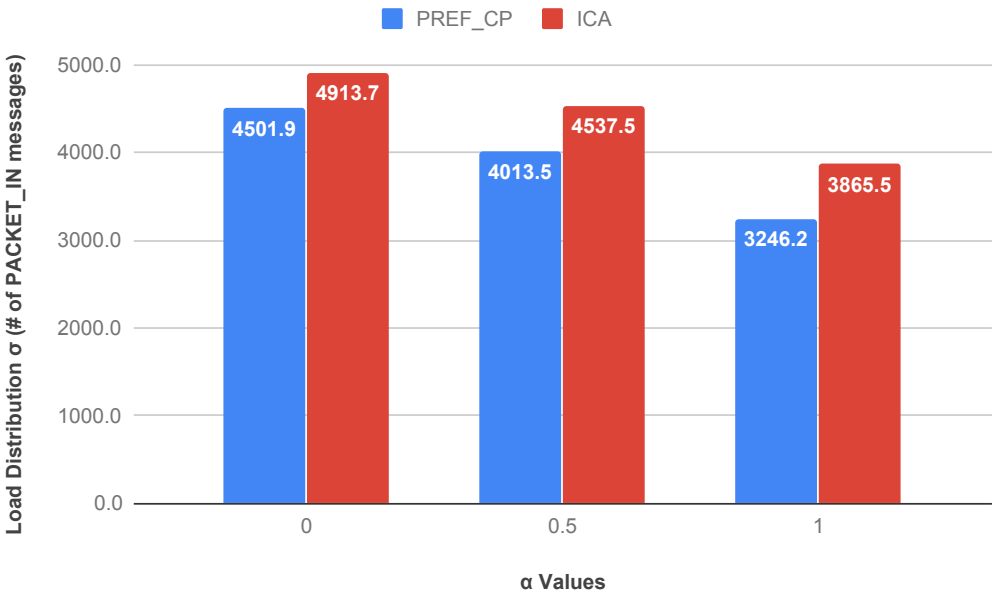


Figure 5.7. PACKET_IN Message Distributions for Different Algorithms.

Table 5.3. Load Distribution For Single Controller Failure.

Algorithm	**Load Distribution σ (pim)**
PREF-CP	3933.9
ICA	4438.9
Random	9284.9

Figure 5.8 shows the results of normalized values for load distribution of PREF-CP on topologies of different size : T1-40 switches, T3-63 switches and T5-34 switches. What we expected was that the load distribution performance should not be affected by the number of switches. However, load distribution efficiency is also depending on the weight factor α in objective function Equation 4.1. When $\alpha = 0$, topology sizes affect load distribution performance more than for $\alpha \geq 0.5$. As seen in Table 5.4, results are slightly close and increasing number of switches does not negatively impact the performance of PREF-CP algorithm. To have better performance at distributing load in a more balanced way, α should be picked up greater than 0.5 in our system. For greater α values, the network size is less effective on load distribution performance of PREF-CP algorithm. In other words, lower α values make performance more vulnerable to size because PoF's weight is at highest and its performance depends on the network size more than load distribution.

Table 5.4. PREF-CP Load Distribution σ (pim) vs Number of Switches S.

Number of Switches (S)	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
34	0.373	0.360	0.356
40	0.369	0.363	0.356
63	0.364	0.359	0.350

To see in more detail how control load is distributed on each controller after one controller's failure, we examined load distribution of PACKET_IN messages on controllers in T5-34 switch topology for $\alpha = 0.5$. Figure 5.9 shows individual controllers' load. Since failing controllers are picked randomly, in this graph we show only one

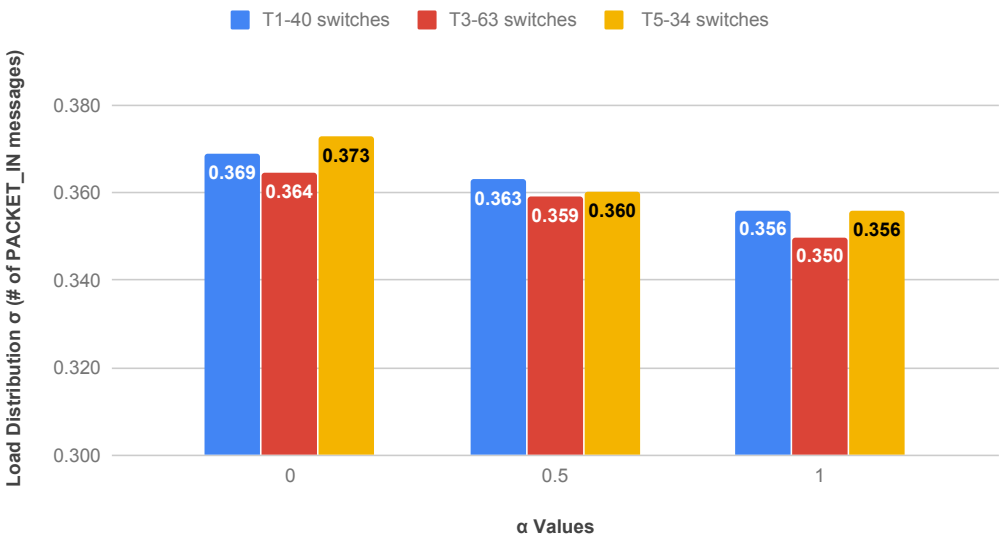


Figure 5.8. PREF-CP Load Distribution σ (pim) for Different Topologies.

test run result. In this case, Controller-3* is failing and subsequent loads of individual controllers are displayed comparing PREF-CP, ICA and random algorithms. As seen in Figure 5.9, control load distribution for PREF-CP is lowest at 1533.3 pim, for ICA it is 4439.9 pim and for random 9284:9 pim for this test case. Controller traffic is distributed more balanced among controllers when the new assignment is calculated with PREF-CP algorithm.

5.4.3. Load Distribution vs Probability of Connectivity Failure (PoF)

Figure 5.10 shows the average load distribution for three simulation topologies and probability of failure with varying α values. As we change α to prioritize load distribution or probability of failure in our optimization objective function, we expect that as α increases, PREF-CP algorithm will calculate an assignment map considering load over probability. Figure 5.10 shows the effect of different α values on prioritization of assignment metrics. When α is zero, PREF-CP algorithm tries to find an assignment algorithm with lower PoF. As stated in the constraint in Equation 4.4, α values range in [0,1]. These components behave as expected in Figure 5.10.

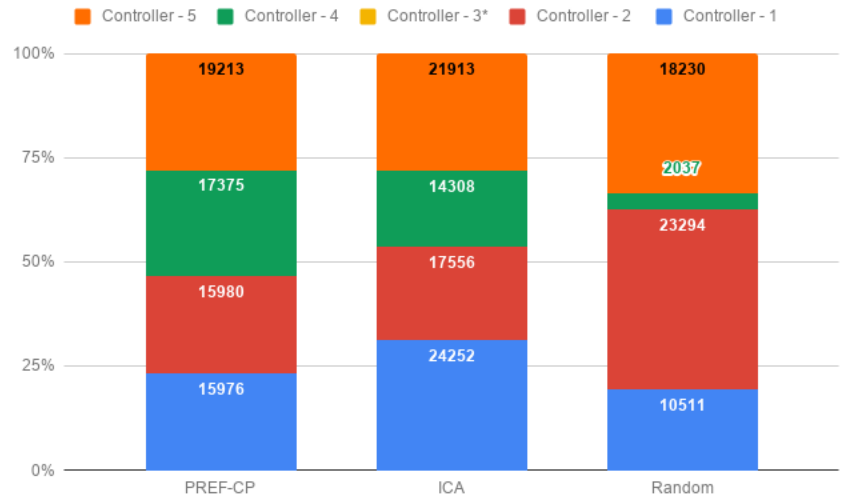


Figure 5.9. Individual Controller Loads in pim After Controller-3 Fails.

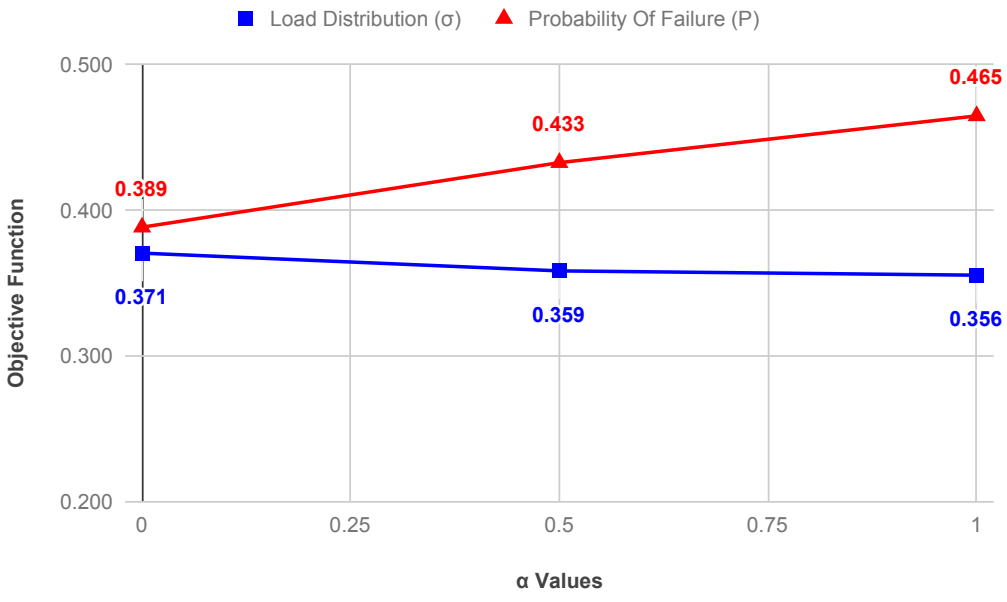


Figure 5.10. PREF-CP Performance - Load Distribution σ (pim) and Probability of Failure.

5.4.4. PoF Characteristics

We compared the reliability parameter, probability of connectivity failure, for different algorithms to examine their performance. There are basically two aspects in our network that affects PoF. The first one is α: lower α values result in better PoF. As seen in Figure 5.11, PREF-CP outperforms ICA and random assignment algorithms. ICA algorithm is slightly higher than random counterpart. However, the random algorithm is for reference since it assigns switches randomly regardless of load or PoF.

The second one is, as seen in Figure 5.12 the number of hops in the test topology: when that value increases, probability of failure is also increasing. This outcome is expected because PoF directly depends on the number of the switches and links between from node a to b. If we compare two tree topologies T1 and T3, T1 has depth of 4 while T3 has depth 6 and average PoF values are 0.373 and 0.509, respectively for average of α values.

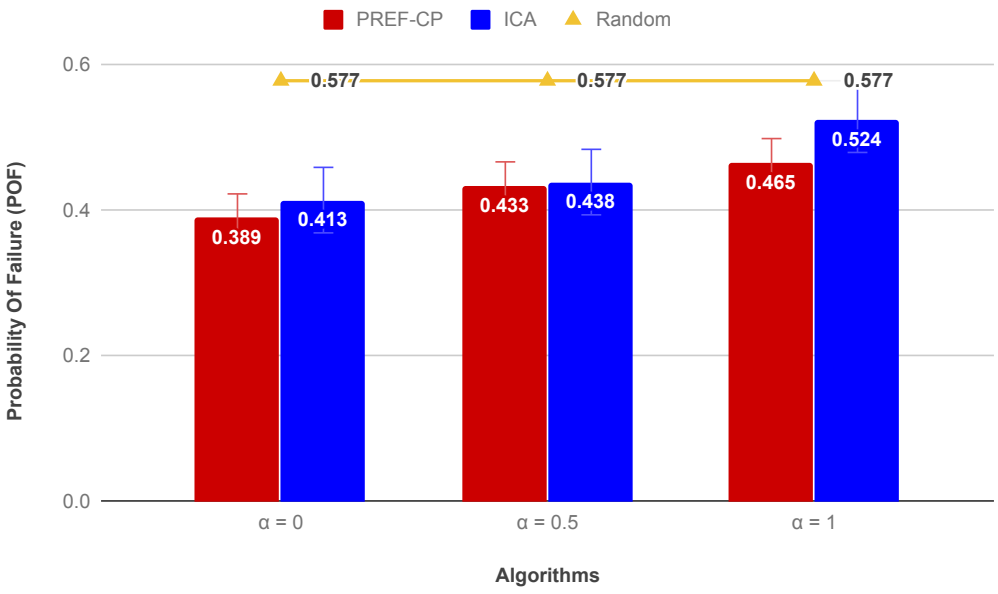


Figure 5.11. PoF Calculated by Different Algorithms.



Figure 5.12. PoF Values for Different Topologies.

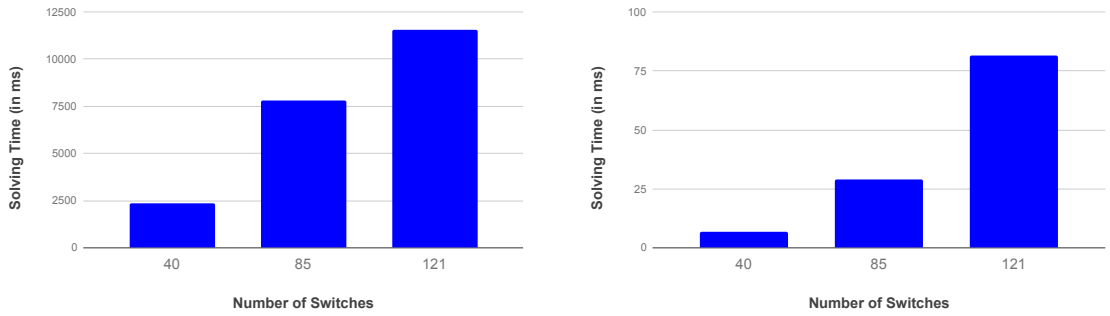
5.4.5. Algorithmic Computation Times

Solving time of PREF-CP algorithm was compared for different number of switches in order to investigate if suggested switch assignment can provide reasonable calculation times. For comparison, ICA algorithm's solving time was also examined to see the affect of increasing number of switches on the calculation time.

As seen in both Figure 5.13(a) and Figure 5.13(b), when the number of switches increases in the network, calculation time for backup reassignment map also increases. ICA algorithm's solving time is around 10ms while PREF-CP runs around 2.5secs. Considering the load distribution, PREF-CP results in a better load distribution. Since we are calculating assignment map before a failure happens, there is sufficient time to calculate assignment map to have better reliability.

5.4.6. Controller Load Behavior over Time

To see the effect of controller failure and the consecutive reassignment, we observe the loads of individual controllers over the experiment time period. This graph is an



(a) PREF-CP Algorithm. (b) ICA Algorithm.

Figure 5.13. Run-times for PREF-CP and ICA algorithms.

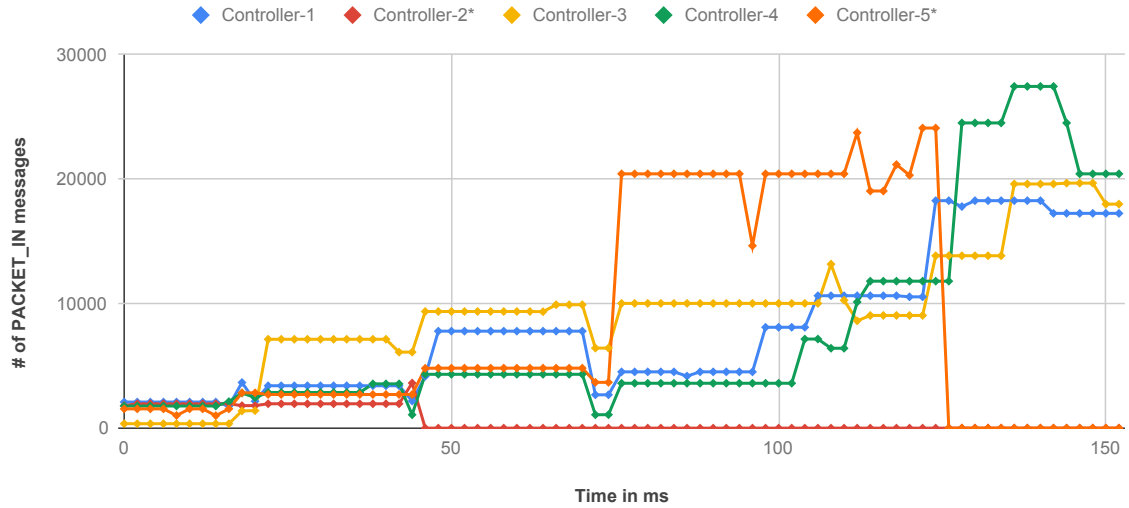


Figure 5.14. Controller Loads Over Time - PREF-CP $\alpha = 0.7$ $\gamma = 0.8$.

output of PREF-CP run with parameters $\alpha = 0.7$ and $\gamma = 0.8$, and on Topology T5-34 switches. Figure-5.14 shows controller control load changes by time. Controller-2 fails at time t=40, and then its switches are distributed to other controllers. As seen i nthe figure, controller loads of remaining controllers are increased after reassignment as expected. At t=125, second controller failure occurs. The controller with the highest load is the failed Controller-5.

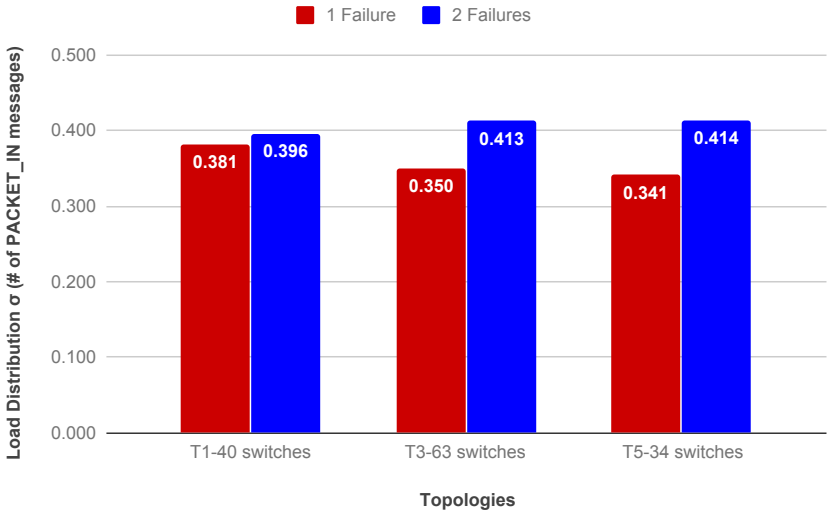


Figure 5.15. Load Distribution Under Controller Failures for Different Topologies.

5.4.7. Cascaded Failure Characteristics

When two controllers fail in tandem, the residual load is distributed among the remaining $V - 2$ (three in our case) controllers. In Figure 5.15, load distributions are shown with respect to different topologies. Tree topologies are effected by multiple controller failures more compared to Internet2 topology.

Figure 5.16 shows load distribution for all algorithms in case of one controller failure and two controller failures. PREF-CP algorithm outperforms both random and ICA algorithms. The presented results shown in Figure 5.15 are the average of load distribution for three different topologies.

To examine in more details how control load is distributed on each individual controller ün Internet2 topology with 34 switches and $\alpha : 0.5$; Figure 5.17 shows individual controller load after the failure of double controllers. In this case, Controller-3* and then Controller-1* fails one after another and we compare PREF-CP performance in handling this double controller situation. Load distributions on average for *before-failure*, *after-one-failure* and *after-two-failures* are shown in Table 5.5.

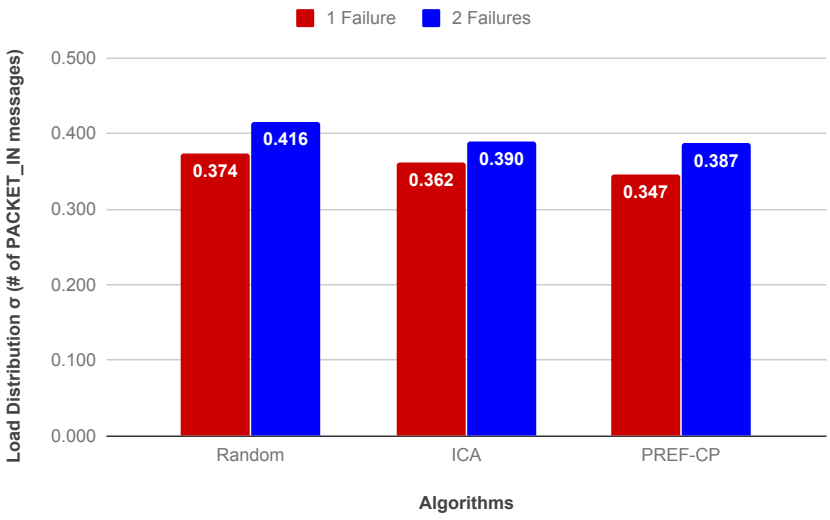


Figure 5.16. Load Distribution Under Controller Failures for Different Algorithms.

Table 5.5. Load Distribution For Two Controller Failure Scenario.

Time	**Load Distribution σ (pim)**
Before Failure	4260.1
After C3 Fails	3944.9
After C1 Fails	5182.2

5.4.8. Reassignment Cost

When there is a failed controller in the network, the minimum number of reassigned switches can be as much as the failed controller's switch count. As the limit case, the maximum number can be as high as the total number of switches. However, switch re-assignment is a costly operation which may include synchronization and context sharing between controllers and signaling between the control and data planes. Furthermore, there is also an inherent trade-off: when a load-aware reassignment is to be performed, setting reassignment cost excessively high (to avoid reassignments in the system) affects the performance of objective function and the load is not distributed homogeneously due to reduced flexibility. To reduce reassignment cost and keeping a

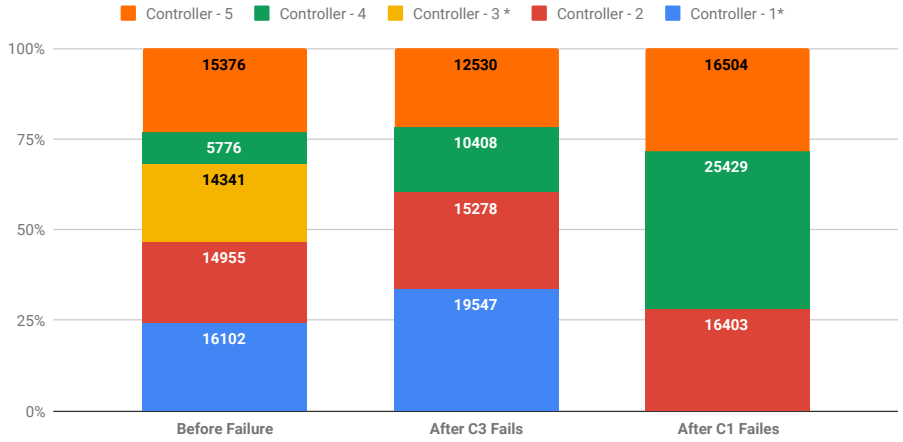


Figure 5.17. Individual Controllers' Loads After C3 and Then C1 Fail.

balanced load distribution, after running for different reassignment cost percentages such as 60%, 70 % and 80% empirically, we picked the specific value for cost constraint to be 80% of the number of switches in the network as shown in Table 5.6.

Table 5.6. Reassignment Cost.

Number of Switches	**Number of Failed Controller's Switches**	**Reassigned Switch Count**	**Reassignment Cost Percentage**
34	6.2	26.6	76.2 %
40	9.8	31.5	77.5 %
63	12.4	49.6	78.12 %

5.4.9. ONOS Standard Assignment Algorithm vs PREF-CP

ONOS also has a native recovery mechanism where the switches of that controller is distributed to remaining controllers when a controller is failed or shut-down. We compared our PREF-CP algorithm with ONOS's existing solution. As results show in Figure 5.18, PREF-CP provides clearly better load distribution for handling controller failure scenarios.

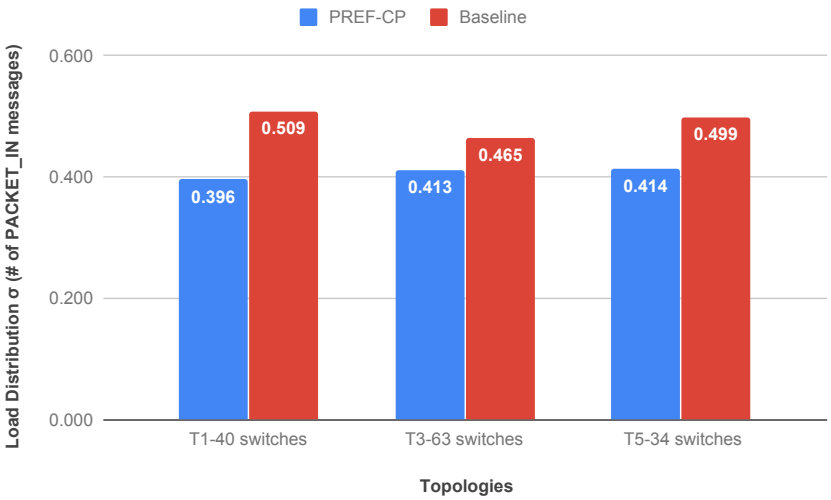


Figure 5.18. ONOS Reassignment Algorithm (baseline) vs PREF-CP.

6. CONCLUSION

In this thesis, we consider control plane failures and how to recover from them with an efficient proactive reassignment approach. To address this challenge, we propose a proactive switch assignment approach PREF-CP which acts considering controller load distribution, reassignment cost and probability of failure when controller failure(s) occur. Moreover, we implemented and simulated random assignment and greedy algorithm ICA to comparatively analyze the performance of PREF-CP. Our test results show that when controllers' load distribution and probability of connectivity failure are considered, PREF-CP performs better than ICA algorithm. Secondly, when reassignment map calculation time is considered, ICA is a faster algorithm as expected. However, since we are making the algorithm calculation proactively before a failure happens, the impact of that calculation time may not have that much of importance. This is especially valid for networks where the calculation time is small compared to inter-arrival time of significant load variations and failure incidents.

For future work, the performance experiments can be run on more topologies and larger network sizes for better diversity. For a more thorough comparison, latency and throughput results can also be considered. In addition, when picking the assignment with PREF-CP, we always pick the best fitness value. This may cause our generation (pool of solutions) to be biased. For better evaluation with PREF-CP algorithm, for some cases the second best can be selected to increase diversity. Finally, one more heuristic algorithm can be implemented to have a comparison in a wider scope.

REFERENCES

1. Nunes, B., M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", *Communications Surveys Tutorials, IEEE*, Vol. 16, No. 3, pp. 1617–1634, Third 2014.

2. Sezer, S., S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks", *Communications Magazine, IEEE*, Vol. 51, No. 7, pp. 36–43, July 2013.

3. Özçelik, M., N. Chalabianloo and G. Gür, "Software-Defined Edge Defense Against IoT-Based DDoS", *2017 IEEE International Conference on Computer and Information Technology (CIT)*, pp. 308–313, Aug 2017.

4. Campbell, A. T., H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente and D. Villela, "A Survey of Programmable Networks", *SIGCOMM Comput. Commun. Rev.*, Vol. 29, No. 2, pp. 7–23, Apr. 1999.

5. Bari, M., A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks", *Network and Service Management (CNSM), 2013 9th International Conference on*, pp. 18–25, Oct 2013.

6. Obadia, M., M. Bouet, J. Leguay, K. Phemius and L. Iannone, "Failover mechanisms for distributed SDN controllers", *2014 International Conference and Workshop on the Network of the Future (NOF)*, Vol. Workshop, pp. 1–6, Dec 2014.

7. Beheshti, N. and Y. Zhang, "Fast failover for control traffic in Software-defined Networks", *Global Communications Conference (GLOBECOM), 2012 IEEE*, pp. 2665–2670, Dec 2012.

8. Hu, T., Z. Guo, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey", *IEEE Access*, Vol. PP, pp. 1–1, 03 2018.

9. Yi-Chen Chan, Kuochen Wang and Yi-Huai Hsu, "Fast Controller Failover for Multi-domain Software-Defined Networks", *2015 European Conference on Networks and Communications (EuCNC)*, pp. 370–374, June 2015.

10. Müller, L. F., R. R. Oliveira, M. C. Luizelli, L. P. Gaspary and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving SDN survivability", *2014 IEEE Global Communications Conference*, pp. 1909–1915, Dec 2014.

11. Dixit, A., F. Hao, S. Mukherjee, T. Lakshman and R. Kompella, "Towards an Elastic Distributed SDN Controller", *SIGCOMM Comput. Commun. Rev.*, Vol. 43, No. 4, pp. 7–12, Aug. 2013, http://doi.acm.org/10.1145/2534169.2491193.

12. Dixit, A. A., F. Hao, S. Mukherjee, T. Lakshman and R. Kompella, "ElastiCon: An Elastic Distributed SDN Controller", *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '14, pp. 17–28, ACM, New York, NY, USA, 2014, http://doi.acm.org/10.1145/2658260.2658261.

13. Heller, B., R. Sherwood and N. McKeown, "The Controller Placement Problem", *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pp. 7–12, ACM, New York, NY, USA, 2012.

14. Nguyen, K., Q. T. Minh and S. Yamada, "A Software-Defined Networking Approach for Disaster-Resilient WANs", *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–5, July 2013.

15. Vizarreta, P., C. M. Machuca and W. Kellerer, "Controller placement strategies for a resilient SDN control plane", *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pp. 253–259, Sep. 2016.

16. Hu, Y., W. Wendong, X. Gong, X. Que and C. Shiduan, "Reliability-aware controller placement for Software-Defined Networks", *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 672–675, May 2013.

17. Yao, G., J. Bi and L. Guo, "On the cascading failures of multi-controllers in Software Defined Networks", *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pp. 1–2, Oct 2013.

18. Fang, K., K. Wang and J. Wang, "A fast and load-aware controller failover mechanism for software-defined networks", *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pp. 1–6, July 2016.

19. Chen, J., J. Chen, F. Xu, M. Yin and W. Zhang, "When Software Defined Networks Meet Fault Tolerance: A Survey", pp. 351–368, 11 2015.

20. Vizarreta, P., P. Heegaard, B. Helvik, W. Kellerer and C. M. Machuca, "Characterization of failure dynamics in SDN controllers", *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pp. 1–7, Sep. 2017.

21. Zhang, Y., N. Beheshti and M. Tatipamula, "On Resilience of Split-Architecture Networks", *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6, Dec 2011.

22. nan HU, Y., W. dong WANG, X. yang GONG, X. rong QUE and S. duan CHENG, "On the placement of controllers in software-defined networks", *The Journal of China Universities of Posts and Telecommunications*, Vol. 19, Supplement 2, No. 0, pp. 92 – 171, 2012.

23. Zhou, Y., Y. Wang, J. Yu, J. Ba and S. Zhang, "Load balancing for multiple controllers in SDN based on switches group", *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 227–230, Sept 2017.

24. Guner, S., H. Selvi, G. Gur and F. Alagoz, "Controller placement in software-defined mobile networks", *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, pp. 2619–2622, May 2015.

25. Roth, M., *ONOS Wiki Home*, 2019, `https://wiki.onosproject.org/display/ONOS/Wiki+Home`, accessed in May 2019.

26. *Mininet Overview*, 2018, `http://mininet.org/overview/0`, accessed in May 2019.

27. *Get Started with Docker*, 2019, `https://www.docker.com/get-started`, accessed in May 2019.

28. *Open vSwitch*, 2016, `https://www.openvswitch.org/`, accessed in May 2019.

29. Dainotti, A., A. Botta, A. Pescapé and G. Ventre, "Searching for Invariants in Network Games Traffic", *Proceedings of the 2006 ACM CoNEXT Conference*, CoNEXT '06, pp. 43:1–43:2, ACM, New York, NY, USA, 2006, `http://doi.acm.org/10.1145/1368436.1368488`.