Reactive Controller Assignment for Failure Resilience in Software Defined Networks

Faruk Açan
Dept. of Computer Engineering
Bogazici University
İstanbul 34342, Turkey
faruk.acan@boun.edu.tr

Gürkan Gür
Inst. of Applied Information Technology
Zurich University of Applied Sciences (ZHAW)
Winterthur 8401, Switzerland
gueu@zhaw.ch

Fatih Alagöz
Dept. of Computer Engineering
Bogazici University
İstanbul 34342, Turkey
fatih.alagoz@boun.edu.tr

***Abstract*—Resilience in SDN control plane is a challenging goal when a single controller is employed. Thus, distributed controllers are deployed to realize a resilient and reliable software defined network. However, such a strategy can not succeed without an efficacious controller-switch assignment scheme. In addition to zero-day assignment, online re-assignment is crucial since due to network failures, the connections between controllers and switches may break off intermittently and impair the network operation. In this paper, we propose a reactive assignment model against network failures using integer linear programming based on load distribution of controllers. We augment our proposal with simulated annealing and random assignment approaches for switch, link and controller failures. The experimental results show that our model gives resilience against network failures and load-awareness is a effective strategy for controller assignment.**

I. Introduction

In recent years, the idea of programmable networks has re-gained significant momentum due to the emergence of Software-Defined Networking (SDN) paradigm [1]. SDN is an important enabler for Future Internet solutions with two main characteristics: the decoupling of the control plane from the data plane and the programmability for network applications. Both of these characteristics are not new, but combining them brings potential benefits of enhanced configuration, improved performance and boosted innovation in network design and operations [2].

The emergence of software-defined networks is accompanied with the the fact that data communications and networking infrastructure is becoming the indispensable part of human activities. From daily mundane tasks to critical infrastructure operation, networks have become the underlying substrate of human civilization. Therefore, it is crucial to achieve robust and flexible communication networks against interruptions and failures. Accordingly, various research works have focused on the performance and resilience of SDN as a key networking technology in the literature [3]–[6]. In that regard, architectures with distributed controllers have been proposed to mitigate performance bottlenecks and improve resilience [7]. For instance, Tootoochan et al. proposed HyperFlow [8] which is an application running on each controller and provides synchronization among them via a cloud system. Another tool FlowVisor [9] developed by Sherwood et al. enables multiple controllers in an OpenFlow network by slicing network resources and delegating the control of each slice to a single controller. However, their approach increases latency.

This work was partially supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under Grant 117E165.

When multiple controllers are employed, a key technical challenge becomes how to assign them to switches for resilience objectives. In [5], Killi et al. proposed an optimization model for deploying controllers but they achieved resilience against only a pre-determined number of controller failures. Hu et al. argues that balancing the controller loads provides scalable and reliable control plane in [6]. They improved the controller throughput with low migration costs when the network scale changes. In another work, Gillani et al. implemented Resilient Control Network architecture (ReCON) that minimizes the sharing of critical resources among data and control traffic to improve resilience against DDoS attacks in SDN [10]. Moreover, Savaş et al. proposed an algorithm for recovery-aware switch-controller assignment to enable fast data-path recovery after a set of failures, and they achieved shorter data-path restoration times after any failure with a minor increase in resource consumption of control paths [11].

In this paper, we focus on the failure resilience problem for more reliable and resilient software defined networks while considering performance and overhead. Although the studied problem stems from the SDN's very own paradigm (i.e. centralized control plane), to achieve these goals, SDN provides extra features via its programmability. We propose a reactive controller-to-switch assignment framework for SDN (RAFRES) which monitors the network and re-assigns forwarding elements according to control plane loads when a calamity occurs. RAFRES entails three different methods to perform controller-switch assignment, namely random, simulated annealing (SA) and Integer Linear Programming (ILP).

II. Reactive Assignment Framework for Resilient SDN (RAFRES)

For failure cases in SDN, connectivity, capacity, and recovery must be considered to achieve resilience and robustness. To serve this objective, RAFRES works re-actively against network failures in a software defined network architecture. These failures can cause from device failures, security attacks or any user error. When a change due to failure occurs in the

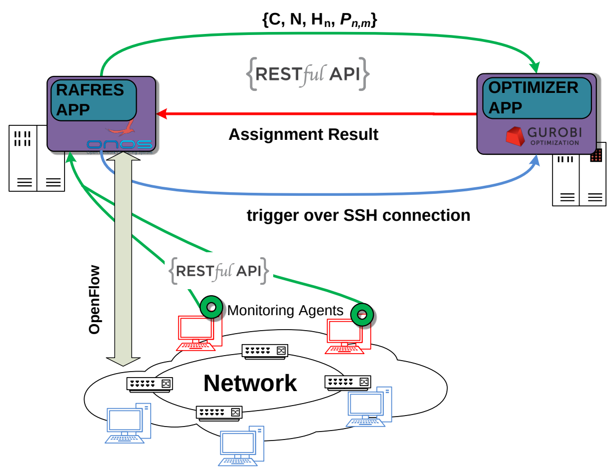


Fig. 1: RAFRES architecture.

topology, the application detects and re-assigns the network nodes onto controllers. Both switch-to-switch and switch-to-controller connectivity must be assured and the load of network nodes should be considered not to exceed the capacity of controllers in controller assignment. After re-assignment, all network devices can be managed by a controller node despite the incumbent failure. Overall mechanism is shown in Figure 1. The tuple $(C, N, H_n, P_{n,m})$ denotes controllers, network nodes, hosts of node n and number of paths between node n and m, respectively.

Architecturally, RAFRES consists of three components: (1) a controller application, (2) monitoring software agents in the network edge and (3) remote optimization engine. The controller application has the whole topology view by the help of centralized control and is aware of changes in the network. Edge-resident monitoring agents are used as beacons to monitor the network from the point of end users. The specific use-case is the monitoring of link quality and recovery. They send heartbeats, measure the round trip time (RTT) information for each controller instance, send them to the controller application via REST API. Hereafter, when a change in the network occurs, the application sends the aggregated network information to the optimization engine via REST API and triggers it using a remote connection. During this process, the engine uses RTT values to decide the forbidden controller $\widetilde{C}$ by calculating average RTT $\widehat{D}$ for each switch with monitoring agent hosts. The decision rule is simply "mark the controller with the highest $\widehat{D}$ as $\widetilde{C}$". After the calculation of reassignment, the engine sends the results back to the application. Finally, the application applies the assignment via the controller and returns back to the monitoring mode for a prospective change in the network.

A. RAFRES Algorithms

Three different algorithms are used to decide on controller-switch assignments in RAFRES, namely random assignment, simulated annealing and integer linear programming (ILP). Although ILP aims the optimal result, the other algorithms exploit the complexity vs. objective-performance tradeoff as well as provide a benchmark for evaluating ILP method.

For devising our algorithms, we represent the system as a network graph $G(N, L, F, H)$, where N is the set of network nodes, L is the set of links, F is the set of monitoring agents and H is the set of hosts. Additionally, let C denote the set of controllers and $\mathcal{P}_{n,m}$ denote the number of paths between node pairs $n \in N$ and $m \in N$.

1) Random Assignment: Using random assignment, the algorithm randomly assigns each network node $n \in N$ to a controller $c \in C$. When assigning a network node to a controller, RAFRES ensures that each node n will be controlled by exactly one controller c. The main advantage of this naive approach is simplicity and reduced response time.

2) Simulated Annealing (SA): SA is a generic probabilistic meta-heuristic which is based on the analogy between the simulation of the annealing of solids and the problem of solving large combinatorial optimization problems [12]. Reducing search space and shortening the computing time to find near optimal solution are major benefits of simulated annealing. It is an iterative procedure that continuously updates one candidate solution until a termination condition is reached as listed on Algorithm 1. In this model, the algorithm tries to create node clusters up to the number of controllers according to *Gain* function. In every iteration, a candidate model is created with a small change, and accepted or rejected according to our conditions. Along the run, the model develops the result, and at the end, it is completed with a more accurate outcome. The gain function of the clustering solution is calculated by use of *ratio cut formula* [13] shown in (1):

$$Gain = \frac{\sum\limits_{A \subset C} \sum\limits_{n \in A} \sum\limits_{m \in A'} \mathcal{P}_{n,m}}{\prod\limits_{A \subset C} |A|} \tag{1}$$

where $|A|$ is the cardinality of cluster A, and A' is the complementary set of cluster A. The ratio cut formula is the ratio of total number of inter-cluster paths to the product of cardinality of cluster sets. We aim to minimize the gain formula to increase resilience on our system.

First, SA randomly distributes the nodes into clusters and calculates the gain. The algorithm starts with starting temperature T_0. Then, in each iteration, M *move_states* occurs. For each *move_state*, algorithm randomly selects a node n to move from one cluster to another and calculates a new gain. The moves can be accepted according to $AcceptGainChange(\Delta Gain, T)$ function. If the moves are accepted, the current gain is updated and the algorithm continues. If the moves rejected, then node n returns the original cluster, and the algorithm continues. After each iteration, system temperature T cools down by the rate of cooling factor α. The algorithm stops if there have been no changes to the solution after t_s iteration. While this scheme is based on the model that was proposed by Manikas et al. in [14], we improved their model by adapting the gain function for more

Algorithm 1: Simulated Annealing Algorithm

```
T = T_0;
t_stop = t_s;
CurrentGain = CalculateGain();
while t_stop > 0 do
    AcceptMove = FALSE;
    for i = 1 to M do
        randomly select node n to move from one cluster to another;
        NewGain=CalculateGain();
        ΔGain = NewGain − CurrentGain;
        if AcceptGainChange(ΔGain, T) then
            CurrentGain = NewGain;
            AcceptMove = TRUE;
        else
            return n to original cluster;
        end
    end
    if AcceptMove then
        t_stop = t_s;
    else
        t_stop = t_stop − 1;
    end
    T = T * α
end
```

than two sets, and making $AcceptGainChange(\Delta Gain, T)$ function load-aware.

3) Optimal Model: For ILP, a mathematical model is constructed as descibed below. Our model is originated from Survivor [15] and extended by using the load distribution of controller instances as a new objective function. It tries to partition network nodes to achieve nearly equal requests for each controller. Moreover, placement-related constraints are omitted since we do not pursue placement problem and additional capacity-related constraints are added in for a more consistent model. The mathematical model used for ILP solution is presented below:

Input: Tuple $I = \{G(N, L, F, H); C; f_{n,c}; U_c; \alpha_c; \beta; \gamma; \omega; H_n; F_n; B_n\}$ is the input of our mathematical model. Graph G denotes the physical topology of the network. The set of controller instances is shown as C. The capacity of each controller represented by $U_c : c \in C$ and α_c indicates the percentage of backup capacity set to each controller. $|\cdot|$ denotes cardinality. β, γ and ω are weight constants for hosts, monitoring agents and bytes that are received by forwarding nodes in order. H_n and F_n shows hosts and monitoring agents those are connected to node $n \in N$. Lastly, B_n denotes number of bytes those are received by node n.

Output: Tuple $V = \{x_{n,c}; w_c; R_n\}$ represents the variables of the output. Device assignments are given by $x_{n,c} \in$ {0,1}; they indicate whether node n is assigned to controller c. $w_c \in \mathbb{R}^+$ denotes total load of controller c. Lastly, $R_n \in \mathbb{Q}^+$ is number of requests of each device n.

Objective Function: The aim of this mathematical model is to minimize the difference between controllers' loads. This goal is represented as:

$$\min \frac{\sum_{c \in C} (w_c - \mu_c)^2}{|C|} \quad \text{(Objective)}$$

TABLE I: ILP Model Parameters

Symbol	Definition
$f_{n,c} \in \{0, 1\}$	Whether controller *c* is forbidden for node *n*
U_c	Maximum number of requests that controller *c* can handle
α_c	Percentage of capacity reserved as backup in controller *c*
β	Weight of requests of hosts
γ	Weight of requests of monitoring agents
ω	Weight of bytes received by nodes
H_n	Set of hosts that are connected to node *n*
F_n	Set of monitoring agents connected to node *n*
B_n	Number of bytes that are received by node *n*
$x_{n,c} \in \{0, 1\}$	Whether device *n* is mapped to controller *c*
$w_c \in \mathbb{R}^+$	Number of total requests of each controller *c*
$R_n \in \mathbb{Q}^+$	Number of requests of each device *n*

which minimizes the variance of total number of requests for controller c. μ_c denotes the average number of requests for each controller.

Constraints: There are two types of constraints for this model: *assignment-related* and *capacity-related*.

The first three constraints (1-3) are assignment-related. They provide the correctness of controller-switch assignments.

$$\sum_{c \in C} x_{n,c} = 1, \forall n \in N. \quad \text{(C1)}$$

Constraint 1 ensures that each node n will be controlled by exactly one controller c.

$$\sum_{n \in N} x_{n,c} \geq 1, \forall c \in C. \quad \text{(C2)}$$

Constraint 2 guarantees that each controller c will control at least one node n. This additional constraint guarantees that no idle controllers will remain.

$$x_{n,c} \geq 1 - f_{n,c}, \forall c \in C, \forall n \in N. \quad \text{(C3)}$$

Constraint 3 provides that each node n will not be assigned to its forbidden controller c. Forbidden controllers are decided according to round-trip times to controllers from monitoring agents.

The other three constraints (4-6) are capacity-related. They guarantee that the controller assignments will not exceed the controllers' capacity.

$$R_n = \beta \cdot |H_n| + \gamma \cdot |F_n| + \omega \cdot \frac{B_n}{\sum_{n \in N} B_n}, \forall n \in N. \quad \text{(C4)}$$

Constraint 4 calculates the number of requests for each node n using H_n, F_n and B_n. This is the key element of our proposed mathematical model. It calculates the instant load of network nodes to use for load-distribution.

$$\sum_{n \in N} x_{n,c} R_n \leq (1 - \alpha_c) \cdot U_c, \forall c \in C. \quad \text{(C5)}$$

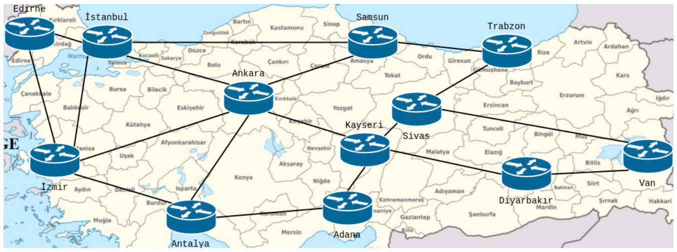


Fig. 2: WAN of major cities in Turkey.

Constraint 5 provides that the controller capacity will not be exceeded taking into account the backup capacity.

$$w_c = \sum_{n \in N} x_{n,c} R_n, \forall c \in C. \tag{C6}$$

Constraint 6 defines the number of total requests for each controller c.

III. Performance Evaluation

In this section, we investigate the performance of RAFRES for different cases. We first describe the experimental environment, followed by the evaluation results.

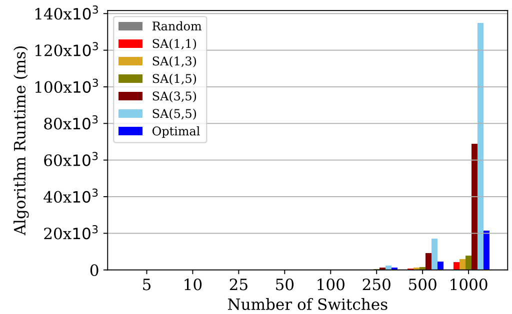
A. Experimental Environment

To test our framework, we use a machine with Intel® Core™ i7-4790 (3.60GHz × 8), 16GB RAM and 1TB HD to host four virtual machines (VMs). Each VM has 4GB memory and 100GB HD space and is running a controller instance. Three of them are used for distributed controller mechanism. We use ONOS Nightingale 1.13.1 as SDN controller and controller application is developed using Java. Mininet 2.2.1 [16] is used for creating network topologies on Ubuntu 14.04 LTS with Open vSwitch 2.0.2. Gurobi Optimizer 8.1.0 [17] as the ILP solver and Distributed Internet Traffic Generator (D-ITG) 2.8.1 [18] to generate traffic from hosts and monitoring agents are used in our experiments.

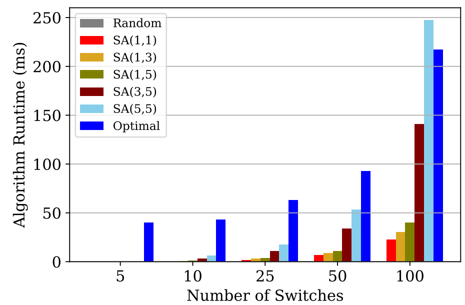
For our experiments, a Wide Area Network (WAN) topology is used. It is based on ULAKNET academic network in Turkey as shown in Figure 2 This network includes the important cities for Turkey such as border cities, the biggest cities in every region and cities with large universities. The topology is created according to real connections between these cities and is changed by creating redundant links for more complicated test cases in the Mininet simulation environment. Monitoring agents provide a heartbeat mechanism from some edge users which are connected to critical switches for the network. For network traffic, we generated the traffic flows of VoIP, video and five different types of online games with D-ITG traffic generator. The parameters of these traffic types are obtained from the literature which rely on actual traffic flows [19], [20].

B. Algorithmic Run-time Analysis

To evaluate the framework, run-times of three controller assignment algorithms on topologies of various sizes are tested. The computational complexity of random assignment is $O(N)$ where N stands for number of switches in the topology. SA and ILP run with $O(M * N^2)$ and $O(2^N)$, respectively, and *M* represents the number of *move_state*s. These theoretical time complexities render run-time behavior expected from RAFRES algorithms and guide to establish a trade-off between run-time and network performance. During these run-time experiments, we used eight different sizes of topology, i.e. 5 to 1000 switches. These topologies were generated artificially in different sizes after examining a sample topology in terms of traffic, number of hosts and link structure. For SA, different stopping values (t_{stop}) and number of *move_state*s per iteration (M) values are tested. All algorithms were run 100 times and the mean values were reported in the graphs.



(a) Runtimes of RAFRES algorithms.



(b) Runtimes for up to 100 switches (zoomed from (a)).

Fig. 3: Runtimes of RAFRES algorithms. For topology with five switches, SA has no results because it needs at least two switches for each controller instance to operate.

As shown in Figures 3a and 3b, random assignment is the quickest algorithm to find a solution as expected. The results show that the algorithms give the expected results according to their run-time complexities. SA is a greedy heuristic and ILP finds the optimal solution which is a time-consuming process. However, when the number of switches exceeded a certain level, since the number of test solutions increased exponentially due to increasing number of iterations based on t_{stop} and M, SA starts to work more slowly than ILP. However, SA with the parameters M=1,t_s=5 runs quicker than ILP for large-sized networks, specifically in less than half of ILP's run-time. Therefore, ILP can be used for small and medium sized networks, but SA with appropriate parameters must be chosen for large topologies.

C. Experimental Results

To assess the RAFRES performance, we first analyzed the distribution of number of `PACKET_IN` messages on

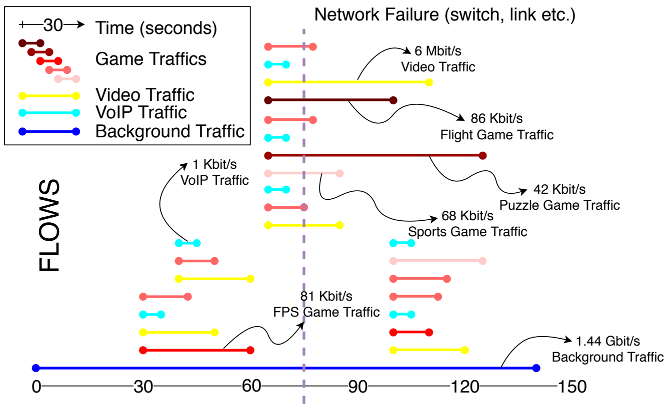


Fig. 4: Timeline for Experiment Scenarios.

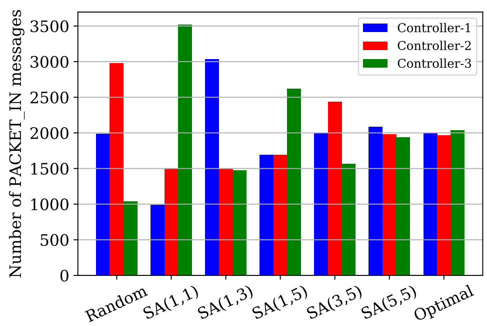


Fig. 5: Number of `PACKET_IN` Messages

controllers. Latency and throughput of data traffic were also examined under various failure scenarios with the dynamic RAFRES mechanisms. All experiments in this section were run ten times and mean results are reported.

Experiments are started with a base traffic to establish the background load on the network throughout the entire scenario. Then, at various intervals, different types of traffic are generated over certain periods of time among the randomly selected users for a dynamic traffic behavior. A network failure (e.g. switch or link) happens at some point in the timeline. Failure type is decided before the scenario starts, and failure occurs at a randomly selected element in the topology. The overall test duration is 150 seconds and follows the timeline shown in Figure 4.

We measured the number of PACKET_IN messages to examine the impact of RAFRES on controller load. To measure the statistics, we utilized Control Plane Management Application (CPMAN) [21] which is a built-in ONOS application. After performing the scenario in Figure 4, we calculated the average number of the packets received by the controller instances. The results can be seen in Figure 5. ILP gives the best results and random assignment is the worst among the algorithms. SA(5,5) gives nearly same results with the optimal one, but run-times differ at that point with a SA having a very low run-time. However, as seen in Fig. 3, for a large network (e.g. 1000 switches), the advantage of SA disappears and, to achieve equivalent load distribution performance, it takes more than 140 seconds with the SA(5,5) while the optimal model calculates the assignment in 20 seconds.

TABLE II: Experimental results for link and controller failure cases.

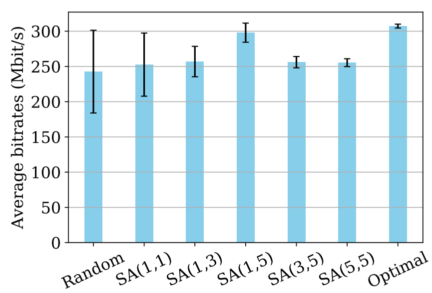
Link Failure			
Assignment Algorithms	Average Bitrate (Kbit/s)		Maximum Delay (sec)
	Mean	Std.Dev.	
Random	183.47	27.77	0.14
SA(1,1)	236.78	24.14	0.11
SA(1,3)	246.08	13.11	0.10
SA(1,5)	246.78	6.13	0.08
SA(3,5)	239.21	6.09	0.09
SA(5,5)	242.56	5.33	0.09
Optimal	247.35	5.03	0.05
Controller Failure			
Assignment Algorithms	Average Bitrate (Kbit/s)		Maximum Delay (sec)
	Mean	Std.Dev.	
Random	153.32	12.58	0.15
SA(1,1)	166.58	12.53	0.11
SA(1,3)	167.02	10.41	0.11
SA(1,5)	169.17	8.10	0.10
SA(3,5)	209.13	4.11	0.09
SA(5,5)	227.07	3.29	0.08
Optimal	235.78	1.18	0.07

To examine the framework against network failures, we apply switch, link and controller failures in the experiments. While the traffic flows according to the dedicated scenario, a failure is triggered at a certain time point. For each type of failure scenario, average delay, maximum delay and average bitrate values were measured as performance metrics after RAFRES performed reassignment as a response to failure.

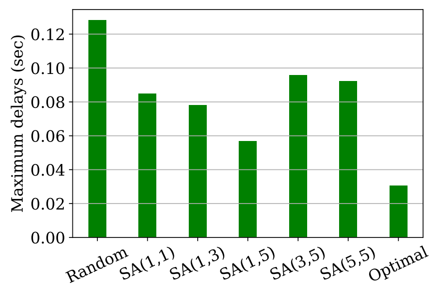
1) Switch Failure: Results of this experiment can be seen in Figure 6. Optimal model achieves average bitrate of 307 Mbps with a small variance. SA is able to provide a maximum bitrate of 297 Mbps for different values of M and t_s. In terms of maximum delays, random assignment gives the worst value as 128 msec, while the optimal model has the smallest value with 30 msec. SA managed to achieve 56 msec at best with the parameter values of M=1 and t_s=5.

2) Link Failure: Results can be seen in Table II for the link failure scenario. Optimal model has the best value of average bitrate with the value of 247 Mbps. The second most successful assignment method is SA(1,5) which achieves 246 Mbps. SA(3,5) and SA(5,5) have worse results for all three metrics than expected, but these these results show a glimpse of the trade-off between performance and the number of iterations. During the link failure incident, optimal model has a maximum delay of 54 msecs and there is no better result among the other algorithms.

3) Controller Failure: After one of the controllers fails, the switches that are assigned to the failed controller remain idle. After the failure incident, ONOS makes an election to choose a master for controller instances and RAFRES reassigns the switches to surviving controllers based on the current number of requests. Results can be seen in Table II. Election of the master among the controllers decreases the average bit-rate and increases the delays. On the other hand, it balances the reassignment calculation time of SA(3,5) and SA(5,5) with others, therefore they can end up with more usual results. ILP achieves 235 Mbps and SA(5,5) follows the optimal with 227



(a) Average bitrate.



(b) Maximum delay.

Fig. 6: Experimental results for switch failure case.

Mbps. Random assignment has the lowest bit-rate with 153 Mbps. In terms of maximum delay, ILP has the best result with 74 msecs and random assignment causes nearly 150 msecs of delay. Maximum delay values are higher than the other scenarios due to the election of master controller.

Overall, experimental results show that random assignment has the worst results, and the proposed ILP based model outperforms the others. Because SA is a meta-heuristic, it has a variety of results in every setting. However, on average, it scores between random and ILP algorithms. It also has worse results for some parameter values. These results can be explained by the slowness of the assignment calculation which causes higher latency and lower throughput in the network. This degradation is caused from also the number of switches in the network. Therefore, SA with suitable parameters is a better choice for large networks. For instance, SA with the parameters M=1 and t_s=5, may be preferred for large networks because the calculation times for optimal model may be too long according to the experimental results. If the effects of switch, link and controller failures on the network are considered, controller failure is the most impactful according to all three metrics as expected. It causes smallest average bit-rate and higher delays. When switch and link failures are compared, link failure seems to affect system performance more than the switch failure considering the maximum delays.

IV. Conclusions

The adoption of distributed controllers is not solely adequate to achieve resilience and reliability goals in SDN. Such an architecture must employ a dynamic and high-performance controller-switch assignment strategy. In this paper, we propose a reactive assignment framework RAFRES based for failure resilience in SDN. We also use edge-resident software agents as network beacons via the flexibility of SDN. As future work, using both reactive and proactive mechanisms and taking into account new performance metrics are planned.

References

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

[2] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "JESS: Joint entropy-based DDoS defense scheme in SDN," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2358–2372, Oct 2018.

[3] K. Phemius and M. Bouet, "OpenFlow: Why Latency Does Matter," *2013 IFIP/IEEE International Symposium on Integrated Network Management*, 2013.

[4] A. Tootoonchain, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud and Enterprise Networks and Services*, 2012.

[5] B. P. R. Killi and S. V. Rao, "Towards Improving Resilience of Controller Placement with Minimum Backup Capacity in Software Defined Networks," *Computer Networks*, vol. 149, pp. 102–114, 2019.

[6] T. Hu, J. Lan, J. Zhang, and W. Zhao, "EASM: Effiency-aware Switch Migration for Balancing Controller Loads in Software-Defined Networking," *Peer-to-Peer Networking&Applications*, vol. 12, pp. 452–464, 2019.

[7] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 333–353, 2018.

[8] A. Tootoochan and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," *Proceedings of 2010 Internet Network Management Conference on Research on Enterprise Networking*, 2010.

[9] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and P. G., "FlowVisor: A Network Virtualization Layer," *OpenFlow Consortium*, Oct. 2009.

[10] F. Gillani, E. Al-Shaer, and Q. Duan, "In-design Resilient SDN Control Plane and Elastic Forwarding Against Aggressive DDoS Attacks," *Proceedings of the 5th ACM Workshop on MTD*, pp. 80–89, Oct. 2018.

[11] S. S. Savas, M. Tornatore, F. Dikbiyik, A. Yayimli, C. U. Martel, and B. Mukherjee, "RASCAR: Recovery-Aware Switch-Controller Assignment and Routing in SDN," *IEEE Transactions on Network and Service Management*, vol. 5, pp. 1222–1234, Nov. 2018.

[12] P. van Laarhoven and E. Aarts, "Simulated annealing," in *Simulated Annealing: Theory and Applications. Mathematics and Its Applications*. Dordrecht: Springer, 1987, ch. 2, pp. 7–8.

[13] Y. Wei and C. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, vol. 10, no. 7, pp. 911–921, Jul. 1991.

[14] T. W. Manikas and J. T. Cain, "Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem," *Computer Science & Eng. Research*, no. 1, 5 1996.

[15] L. F. Müller, 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 (GLOBECOM)*, Dec. 2014.

[16] B. Lantz, B. Heller, R., and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," *Proceeedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[17] Gurobi Optimization Inc., "Gurobi Optimizer." [Online]. Available: http://www.gurobi.com

[18] Universita' degli Studi di Napoli "Federico II", "D-ITG (Distributed Internet Traffic Generator)." [Online]. Available: http://traffic.comics.unina.it/software/ITG/

[19] Q. A. He, "Analysing the Characteristics of VoIP Traffic," Master's thesis, University of Saskatchewan, Saskatoon, 7 2007.

[20] M. Manzano, M. Urueña, M. Sužnjević, E. Calle, J. A. Hernández, , and M. Matijasevic, "Dissecting the Protocol and Network Traffic of the OnLive Cloud Gaming Platform," *Multimed. Syst*, vol. 20, no. 5, pp. 451–470, Oct. 2014.

[21] J. Li, J.-H. Yoo, and J. W.-K. Hong, "CPMan: Adaptive Control Plane Management for Software-Defined Networks," *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 121–127, Nov. 2015.