Reactive Controller Assignment for Failure Resilience in Software Defined Networks

Faruk Acan
Dept. of Computer Engineering
Bogazici University
Istanbul 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 Alagoz
Dept. of Computer Engineering
Bogazici University
Istanbul 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 cannot 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.

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, Savas 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
Although ILP aims the optimal result, the other algorithms simulated annealing and integer linear programming (ILP). Switch assignments in RAFRES, namely random assignment, prospective change in the network. The controller and returns back to the monitoring mode for a prospective change in the network. Finally, the application applies the assignment via application.  

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 \( P_{n,m} \) 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):

\[
Gain = \sum_{A \in C} \sum_{n \in A} \sum_{m \in A'} P_{n,m} \prod_{A' \subseteq C} |A| 
\]

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 are 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 \( \alpha \). 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

\[ T = T_0; \]
\[ t_{stop} = t_s; \]
\[ CurrentGain = \text{CalculateGain}(); \]
while \( t_{stop} > 0 \) do
    \[ \text{AcceptMove} = \text{FALSE}; \]
    for \( i = 1 \) to \( M \) do
        randomly select node \( n \) to move from one cluster to another;
        \[ \text{NewGain} = \text{CalculateGain}(); \]
        \[ \Delta \text{Gain} = \text{NewGain} - \text{CurrentGain}; \]
        if \( \text{AcceptGainChange}(\Delta \text{Gain}, T) \) then
            \[ \text{CurrentGain} = \text{NewGain}; \]
            \[ \text{AcceptMove} = \text{TRUE}; \]
        else
            \[ \text{return } n \text{ to original cluster}; \]
        end
    end
    if \( \text{AcceptMove} \) then
        \[ t_{stop} = t_s; \]
    else
        \[ t_{stop} = t_{stop} - 1; \]
    end
end
\[ T = T \ast \alpha \]

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

3) Optimal Model: For ILP, a mathematical model is constructed as described 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 \( U_c; \forall c \in C \) and \( \alpha_c \) indicates the percentage of backup capacity set to each controller. \( | \cdot | \) denotes cardinality. \( \beta, \gamma \) and \( \omega \) 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:

\[
\min_{c \in C} \frac{\sum_{c \in C} (w_c - \mu_c)^2}{|C|} \quad \text{(Objective)}
\]

which minimizes the variance of total number of requests for controller \( c \). \( \mu_c \) 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.

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_{n,c} )</td>
<td>Whether controller ( c ) is forbidden for node ( n )</td>
</tr>
<tr>
<td>( U_c )</td>
<td>Maximum number of requests that controller ( c ) can handle</td>
</tr>
<tr>
<td>( \alpha_c )</td>
<td>Percentage of capacity reserved as backup in controller ( c )</td>
</tr>
<tr>
<td>( \beta )</td>
<td>Weight of requests of hosts</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>Weight of requests of monitoring agents</td>
</tr>
<tr>
<td>( \omega )</td>
<td>Weight of bytes received by nodes</td>
</tr>
<tr>
<td>( H_n )</td>
<td>Set of hosts that are connected to node ( n )</td>
</tr>
<tr>
<td>( F_n )</td>
<td>Set of monitoring agents connected to node ( n )</td>
</tr>
<tr>
<td>( B_n )</td>
<td>Number of bytes that are received by node ( n )</td>
</tr>
<tr>
<td>( x_{n,c} )</td>
<td>Whether device ( n ) is mapped to controller ( c )</td>
</tr>
<tr>
<td>( w_c )</td>
<td>Number of total requests of each controller ( c )</td>
</tr>
<tr>
<td>( R_n )</td>
<td>Number of requests of each device ( n )</td>
</tr>
</tbody>
</table>
Constraint 5 provides that the controller capacity will not be exceeded taking into account the backup capacity.

\[ w_c = \sum_{n \in N} x_{n,c} R_n, \forall c \in C. \] (C6)

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

### III. Performance Evaluation

In this section, we investigate the performance of RA斐RES 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 x 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 (DITG) 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 DITG 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 \( O(M \times N^2) \) and \( O(2^N) \), respectively, and \( M \) represents the number of move states. These theoretical time complexities render run-time behavior expected from RA斐RES 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 states per iteration \( M \) values are tested. All algorithms were run 100 times and the mean values were reported in the graphs.

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 \( M=1, t_{stop}=5 \) 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 RA斐RES performance, we first analyzed the distribution of number of PACKET_IN messages on
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 (CPMAN) which is a built-in ONOS application.

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 latency, maximum latency 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 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

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Average Bitrate (Kbit/s)</th>
<th>Maximum Delay (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>183.47</td>
<td>0.14</td>
</tr>
<tr>
<td>SA(1,1)</td>
<td>236.78</td>
<td>0.11</td>
</tr>
<tr>
<td>SA(1,3)</td>
<td>246.08</td>
<td>0.10</td>
</tr>
<tr>
<td>SA(1,5)</td>
<td>246.78</td>
<td>0.08</td>
</tr>
<tr>
<td>SA(3,5)</td>
<td>239.21</td>
<td>0.09</td>
</tr>
<tr>
<td>SA(5,5)</td>
<td>242.56</td>
<td>0.09</td>
</tr>
<tr>
<td>Optimal</td>
<td>247.35</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Average Bitrate (Kbit/s)</th>
<th>Maximum Delay (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>153.32</td>
<td>0.15</td>
</tr>
<tr>
<td>SA(1,1)</td>
<td>166.58</td>
<td>0.11</td>
</tr>
<tr>
<td>SA(1,3)</td>
<td>167.02</td>
<td>0.11</td>
</tr>
<tr>
<td>SA(1,5)</td>
<td>169.17</td>
<td>0.10</td>
</tr>
<tr>
<td>SA(3,5)</td>
<td>209.13</td>
<td>0.09</td>
</tr>
<tr>
<td>SA(5,5)</td>
<td>227.07</td>
<td>0.08</td>
</tr>
<tr>
<td>Optimal</td>
<td>235.78</td>
<td>0.07</td>
</tr>
</tbody>
</table>
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.

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


Fig. 6: Experimental results for switch failure case.