ROBUST CONTROLLER ASSIGNMENT FOR SDN SURVIVABILITY

by

Faruk Açan

B.S., Computer Engineering, Boğaziçi University, 2014

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

ACKNOWLEDGEMENTS

I would like to thank first and foremost to my family, especially my brothers, for their support and motivation that encouraged me during my academic career. I am very lucky and delighted to be a part of my family.

During my undergraduate period, İsmail Arı has been a great inspiration for me to start my academic journey. I want to offer my gratitude to him. I also would like to thank fellow academician Dr. Gürkan Gür for supporting me during my M.S. studies. He has spent a large amount of time to mentor me and convey to how to conduct scientific research. I, therefore, have been able to complete my this research, written my first academic paper and started to think as a researcher thanks to him.

I would like to offer my deep gratitude to my instructor and thesis supervisor Prof. Fatih Alagöz, who has given me a second chance to continue my academic career. I have been able to finish my M.S. thesis thanks to this opportunity and his supervision.

I am also deeply thankful to my friends at SATLAB, who have inspired and encouraged me by their talks. I would like to thank to my colleagues at the Computer Center for their savvy manners.

I especially would like to offer my sincere gratitude to my close friend and colleague Hüseyin Temiz who has provided valuable help and insight about research on Software Defined Networks.

I have been lucky enough to have wonderful friends. Last but not the least, I deeply thank my beloved friend Yasemin Gök for encouraging talks and support. She has been always near me and given me her support generously. I would like to offer my deepest gratitude to the friends that I have shared many enjoyable moments. These moments gave me the power to go forward and finish my thesis work.

ABSTRACT

ROBUST CONTROLLER ASSIGNMENT FOR SDN SURVIVABILITY

Robustness in software defined networks (SDN) control plane is a challenging goal when a single controller is employed. Thus, distributed controllers are deployed to realize a robust, 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 thesis, we propose a reactive assignment model against network failures using integer linear programming based on the load distribution of controllers. We also use a complementary strategy to monitor the network by using monitoring agents as network beacons at the edge via the flexibility of SDN. Real-time information about network connectivity and latency is provided by these agents and used in the assignment decisions. We augment our proposal with simulated annealing and random assignment approaches for switch, link and controller failures. Our proposed framework is evaluated using different types of traffic such as VoIP, video, and network gaming. Our model is also investigated using two different test scenarios to assess the contribution of monitoring agents. The experimental results show that our model gives resilience against network failures while load-awareness is an effective strategy for controller assignment.

ÖZET

YAZILIM TANIMLI AĞLARIN BEKASI İÇİN GÜRBÜZ KONTROLÖR ATAMASI

Yazılım tanımlı ağların (YTA) kontrol düzleminde sağlamlık, tek bir kontrolör kullanıldığında zorlayıcı bir amaçtır. Bu nedenle, dağıtık kontrolörler sağlam, esnek ve güvenilir bir yazılım tanımlı ağ oluşturmak için görevlendirilir. Buna rağmen, böyle bir strateji, etkili bir kontrolör-ağ anahtarı atama şeması olmadan başarılı olamaz. Sıfırıncı gün atamasına ek olarak, çevrimiçi yeniden atama çok önemlidir, çünkü ağdaki arızalar nedeniyle kontrolörler ve anahtarlar arasındaki bağlantılar aralıklı olarak kopabilir ve ağın çalışmasını bozabilir. Bu çalışmamızda, kontrolörlerin yük dağılımına dayanan tamsayılı doğrusal programlama kullanarak ağdaki arızalara karşı reaktif bir atama modeli önerdik. Ayrıca, YTA'nın esnekliği sayesinde, izleme ajanlarını kenarda ağ fenerleri olarak kullanarak da ağı izlemek için tamamlayıcı bir stratejiden yararlandık. Ağ bağlantısı ve ağdaki gecikmeler hakkında gerçek zamanlı bilgiler, reaktif denetleyici-anahtar atamasında kullanılması için izleme ajanları tarafından sağlandı. Anahtar, link ve kontrolör arızaları için benzetilmiş tavlama ve rasgele atama yaklaşımlarıyla önerilerimizi çoğalttık. Önerilen çerçevemiz VoIP, video ve oyun gibi farklı trafik türleri kullanılarak değerlendirildi. Modelimiz ayrıca, izleme ajanlarının yardımını anlamak için iki farklı test senaryosu kullanılarak test edildi. Deneysel sonuçlar, modelimizin ağdaki arızalara karşı dayanıklılık kazandırdığını ve yük farkındalığının kontrolör-anahtar ataması için etkili bir strateji olduğunu gösterdi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A	Subset cluster of network nodes
A'	Complementary set of cluster A
B_n	Number of bytes those are received by node $n \in N$
C	Set of controllers
$\widetilde{C}$	Forbidden controller
$\widehat{D}$	Average round trip time
f	Prohibition matrix
$f_{n,c}$	Indicator for whether controller $c \in C$ is forbidden for node $n \in N$
F	Set of monitoring agents
F_n	Set of monitoring agents that are connected to node $n \in N$
H	Set of hosts
H_n	Set of hosts that are connected to node $n \in N$
L	Set of links between network nodes
M	Number of *move_states* per iteration
N	Set of network nodes (e.g. switches)
$\mathcal{P}_{n,m}$	Number of paths between node pairs $n \in N$ and $m \in N$
R_n	Number of requests of node $n \in N$
t_s	Number of iterations
t_{stop}	Stopping value
T	System temperature
T_0	Starting temperature
U_c	Capacity of the controller $c \in C$
w_c	Aggregate load of controller $c \in C$
$x_{n,c}$	Indicator for whether node $n \in N$ is assigned to controller $c \in C$
α	Cooling factor
α_c	Percentage of backup capacity for the controller $c \in C$

β	Weight constant for hosts
γ	Weight constant for monitoring agents
μ_c	Average number of requests for each controller $c \in C$
ω	Weight constant for switch loads

LIST OF ACRONYMS/ABBREVIATIONS

D-ITG	Distributed Internet Traffic Generator
DISCO	Distributed Multi-domain SDN Controllers
Gbit	Gigabit
GB	Gigabyte
HDD	Hard Disk Drive
IDT	Inter-departure Time
IoT	Internet of Things
Mbps	Megabit per Second
OF	OpenFlow
ONOS	Open Network Operating System
RAFRES	Reactive Assignment Framework for SDN
RAM	Random Access Memory
RTT	Round Trip Time
SA	Simulated Annealing
SDN	Software Defined Networks
SSD	Solid-state Drive
VoIP	Voice over Internet Protocol
VM	Virtual Machine
WAN	Wide Area Network

1. INTRODUCTION

The development and utilization of computer networks continue to expand on a daily basis and quality of service expectations of users from the communications services are becoming extremely demanding. These requirements range from very high bandwidths to anytime-anywhere uninterrupted connectivity. However, as there is no method to completely eliminate network failures, the recommended methods are only to reduce and quickly resolve the damages. Moreover, as computer networks evolve and the number of users increases with more advanced services, incumbent methods become inadequate and novel methods are necessary.

In recent years, the idea of programmable networks has regained significant momentum with the emergence of the Software-Defined Networking (SDN) paradigm to address the networking challenges [1]. SDN is an important enabler for Future Internet solutions with two main characteristics. The first one is the decoupling of the control plane from the data plane while the other is 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]. Open Network Foundation and its SDN proposal, OpenFlow [3], has gathered strong support from industry, research and academia and OpenFlow's SDN architecture has become the de-facto SDN standard [1]. The SDN architecture consists of three layers: application layer, control layer and infrastructure layer. These layers operate together to provide scalable and flexible network services.

The emergence of software-defined networks is accompanied with the fact that data communications and networking infrastructure is becoming an indispensable part of human activities. From daily mundane tasks to critical infrastructure operations, networks have become the underlying substrate of human civilization. Therefore, it is crucial to achieving robust and flexible communication networks against interruptions and failures. Accordingly, various research works focused on the performance and resilience of the SDN as a key networking technology in the literature [4–10]. The

findings indicate that, due to the flow processing mechanism of OpenFlow, the controller can become a bottleneck and thus affects the performance of the entire network. Therefore, architectures with distributed controllers have been proposed to mitigate bottlenecks and improve performance [11]. Studies in this area generally support the proposition that one controller is enough for performance issues but not for reliability issues. In that regard, a single controller can meet the response-time goals, but not the fault tolerance in medium and large size networks.

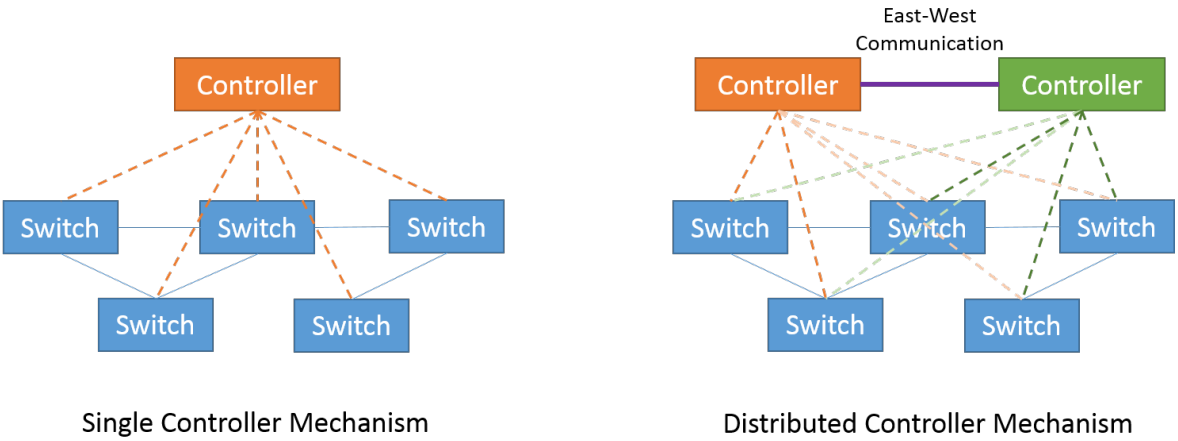

Figure 1.1. Single vs. distributed controller mechanism

When multiple controllers are employed, a key technical challenge becomes how to assign them to switches for reliability and robustness objectives. There are many techniques to assign controllers such as random selection, greedy methods, meta-heuristics and brute-force techniques. Studies also show that there is a need for a trade-off between performance and reliability for the assignment of controllers. Besides, Killi at al. searched for results on resiliency on SDN in their work [9]. They proposed a model to achieve full resilience against a pre-specified number of controller failures by mapping each switch to a pre-specified number of controllers. Also, they minimized the back-up capacity of controllers while fixing the number of controllers in the system.

Besides SDN, Internet of Things (IoT) is also an emerging technology. IoT is a paradigm where everyday objects can be equipped with identifying, sensing, networking and processing capabilities that will allow them to communicate with one another

and with other devices and services over the Internet to accomplish some objective [12]. Future Internet will entail billions of IoT devices which will become an interconnected network. They will serve various purposes from daily computation and communication services to more critical functions such as infrastructure protection and ICT security. Real-time network monitoring related services are good candidates for IoT applications since they require quick response and availability to meet their goals. Moreover, devices on the network edge are inherently numerous which provides a distributed infrastructure to monitor network events and metrics such as latency or connectivity loss.

1.1. Key Contributions

In this thesis, we focus on failure resilience for more reliable and robust software defined networks while considering performance and overhead. To this end, we aim to assign network nodes onto controllers to achieve a favorable distribution of load (i.e., requests) for each controller. As contributions, we develop a mathematical model to solve using Integer Linear Programming (ILP) and use monitoring agents to monitor real-time traffic metrics to achieve a more robust controller-switch assignment.

The contributions of this thesis can be listed as follows:

- *Load-aware ILP Model:* We propose a mathematical model to assign switches onto controllers. In this model, real-time loads of network nodes are used for the assignment to achieve equally distributed controller requests. A load-aware controller-switch assignment helps to prevent catastrophic failures caused by the exceeded capacity of controllers.
- *Monitoring Agents:* We introduce the utilization of monitoring agents as network beacons at the network edge. These agents periodically provide real-time information about network delays and host connections. This information is used for the controller-switch assignment by the ILP model to achieve robust controller-switch connectivity.

- *Online Controller-Switch Assignment:* In addition to zero-day assignment, we propose an online controller-switch reassignment mechanism considering changes in the network such as switch, link and controller failures. Online reassignment provides robustness against future failures and facilitates a more efficient use of the new topology after changes.

Overall, we propose a *Reactive controller-to-switch Assignment FRamEwork for SDN* (RAFRES) which monitors the network and reassigns forwarding elements according to control plane loads when a calamity occurs. RAFRES entails three different methods to perform the controller-switch assignment: We augment our ILP-based proposal with simulated annealing and random assignment approaches for switch, link and controller failures.

1.2. Thesis Outline

First, Chapter 2 presents the related works in the literature to provide a summary of current studies in our research field. The selection of distributed controller mechanism, controller-switch assignment algorithms and performance metrics are based on prior proposals in the literature.

Then, we explain our key contributions listed above and their performance in the network in Chapter 3 and Chapter 4, respectively. Specifically, in Chapter 3, we propose our controller assignment framework with all components such as how controller-switch assignments are made, how monitoring agents are used and integrated into the framework. The assignment models that we used and which improvements are made are listed. The mathematical definition of our proposed model for ILP and explanation of simulated annealing (SA) algorithm is also described.

In Chapter 4, in addition to the simulation environment and experimental scenarios, performance analysis of RAFRES is presented. First, the used distributed controller mechanism and the controller application development process is described. Then, how the network topology is setup and traffic flows are generated between hosts

are detailed. Performance metrics are listed and explained one by one so that our criteria to analyze the performance of the framework are explicitly present.

Lastly, in Chapter 5 we summarize our contributions and discuss the results of the proposed model in the thesis. Finally, we conclude by explaining the potential future research directions.

2. RELATED WORK

In this chapter, we summarize the contributions that are made in literature in the field that we are focused on. First, distributed controller mechanisms that are proposed by other researchers are examined. How they handle the east-west communication in the control plane is described. Then, the controller-switch assignment methods are discussed by explaining their techniques and objectives. Lastly, we look at the studies on performance analysis on SDN controllers and how they deal with the failures in the data plane of SDN.

2.1. Distributed Controller Mechanisms

After the distributed controller paradigm is proposed, east-west communication became an important issue for the SDN architecture [11]. There are many different proposals as distributed controllers like running on the same layer [13–18] or running hierarchically [19, 20].

Tootoochan *et al.* propose HyperFlow [13] which is an application running on each controller and provides synchronization among them via a cloud system. However, their implementation requires minor changes to the core controller code, mainly, to provide appropriate hooks to intercepts commands and serialize events. Hyperflow is a C++ application and works on NOX [21] controller.

Another tool FlowVisor [14] develop 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. FlowVisor sits between each OpenFlow controller and the switches, to make sure that a guest controller can only observe and control the switches it is supposed to. However, this approach has the drawback of increased latency.

Phemius *et al.* propose Distributed Multi-domain SDN Controllers (DISCO) [18] where each controller communicates with neighbor domain controllers to exchange total network-wide information. To handle east-west communication, they use a messenger module which explores neighboring controllers and remain a distributed publish/subscribe communication channel, and different agents that use this channel to exchange network-wide information with other controllers. This east-west communication is based on a lightweight and highly manageable control channel. DISCO is developed on the basis of Floodlight [22] SDN controller using Java language. Similarly, Berde *et al.* enable deployment of multiple controllers using an event notification system namely Hazelcast [23] to synchronize these controllers. Therefore, they develop Open Network Operating System (ONOS) [17] based on Floodlight SDN controller. It also contains a distributed registry, Zookeeper [24], to manage switch-to-controller mastership. Their goals are to make remote operations as fast as possible, to reduce the number of remote operations that ONOS has to perform.

Some of the researchers propose hierarchical architectures for distributed controller mechanisms to deal with east-west communication of SDN controllers. As a hierarchical system, Yeganeh *et al.* propose a two-layered architecture for controllers. While the controllers in the bottom layer have no interconnection between each other and no knowledge of the network-wide state, and the top layer is a logically centralized controller that maintains the network-wide state. The bottom layer is used as a shield for the top layer and handles most of the frequent events, but this architecture can cause scalability issues due to the possibility of a new bottleneck in controller architecture [19].

On the other hand, Google's B4 [20], a private software defined wide area network (WAN) connecting their data centers, proposes a two-level hierarchical control framework for improving scalability. At the bottom layer, each data-center site is handled by an Onix-based [15] SDN controller that executes local site-level control applications. These site controllers are managed by a gateway that collects network information from all sites and sends them to a logically centralized server which also manages at the top layer of the control hierarchy. The top layer performs high-level policies that are aimed

at optimizing bandwidth allocation between competing applications across the different data-center sites. thereby allowing controllers to run protocols at a coarse granularity based on a global controller view and, more importantly preventing architecture from becoming a serious performance bottleneck.

2.2. Controller Assignment Strategy

Distributed controllers unveil new problems to deal with. Using distributed controllers are not enough to achieve a robust SDN controller mechanism since an efficacious strategy must be applied for the controller-switch assignment. Upper bound of the number of controllers are also important for an SDN architecture, because the more number of controllers are used, the more control traffic and delay happens. Apparently, the number of controllers depends on the number of nodes in the network topology and its geographical size. About this problem, Hu *et al.* found that the best controller number is in between $[0.035n, 0.117n]$ where n is the number of nodes [25]. However, in this thesis, we adopt a different number of controllers in order not to be restricted by it for achieving more robustness and resilience in the network.

Controller placement and controller-switch assignment are very related problems with each other and a proposed solution for one of them can usually be applied for the other with proper modifications. Therefore, in this section, both controller placement and controller-switch assignment methods that are proposed in the literature are summarized.

The first proposed procedure to realize the controller-switch assignment is proposed by Heller *et al.*, and they tried all possible combinations of controllers by considering some metrics such as average-case latency, worst-case latency and nodes within a latency bound [8]. They state that using an algorithm to balance switches dynamically among available controllers decreases latency for the control plane.

Four different strategies are experimented by Hu *et al.* in [25]. They made their assignment using random assignment, l-w-greedy that is a proposed method for

assignment by considering failure probabilities of switches, simulated annealing and brute-force technique. When assigning switches onto controllers, their objective is to minimize the expected percentage of control path loss and achieve a more reliable SDN controller.

Bo *et al.* [26] apply the multi-objective genetic algorithm in order to obtain the connection matrix of controllers and switches, so the network can be partitioned into several domains and then, minimum delay algorithm is adopted to find an optimal place which has a minimizing sum of the distance to the other points in each domain. Then each domain is assigned to a controller to complete the assignment. For performance analysis, they apply a fitness test to indicate the quality of their assignments and calculate propagation delay and load diversity of controllers as performance metrics.

In another study, Wang *et al.* formulate the dynamic controller assignment problem as online optimization to minimize the total cost caused by response time and maintenance on the cluster of controllers [27]. They also propose a hierarchical two-phase algorithm that combines key concepts from both matching theory and coalitional games to solve it efficiently. Response times and control traffic overhead are analyzed as performance metrics to indicate that their proposed strategy works efficiently.

Müller *et al.* propose a controller placement strategy called Survivor [28] that proposes an ILP model to assign controllers and a greedy heuristic to decide back-up controllers for each switch in case of a network failure. The ILP model aims to maximize the number of disjoint paths between controller and switches, which is an important parameter for redundancy. To evaluate the performance of the proposed strategy, they use the number of overloaded controllers when a calamity occurs and the load distribution for each of the controller instances, as metrics.

Manikas *et al.* use the genetic algorithm and simulated annealing for circuit partitioning problem [29]. This problem also has similar objectives and constraints with the controller-switch assignment problem and their solution strategies can be applied to the controller-switch problem smoothly by applying relevant modifications.

In their work, they use ratio cut formula [30] as a clustering method to reduce inter-cluster traffic in the network.

2.3. SDN Controllers under Network Failures

Network failures can be divided into two groups in terms of the layer in which they occur. Some of them happen in the control plane of SDN such as controller and control path failures and the others occur in the data plane such as switch and link failures. There are also cases where these failures are correlated, e.g. a physical disconnection in the network which affects the control layer and data plane communications at the same time.

Some researchers specifically studied on control plane resilience in SDN against failures. Killi *et al.* propose an optimization model for deploying controllers but they achieved resilience against only a pre-determined number of controller failures [9]. To achieve this, their model decides on a pre-specified alternative controller for each switch in the network in zero-day assignment. Therefore, each switch knows its replacement controller when its master controller fails. They also introduce a model to minimize the switch-to-controller latency with resilience against controller failures.

Hu *et al.* propose Efficiency-Aware Switch Migration to balance the controller loads and improve migration efficiency [10]. Their work introduces a load difference matrix and triggers factor to measure load balancing on controllers. They analyze their proposal using controller response time, controller throughput, migration cost and load balancing rate as performance metrics. 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 that minimizes the sharing of critical resources among data and control traffic to improve resilience against DDoS on the SDN control plane [31]. It measures the control network resilience by creating a resilience metric based on the link isolation and the link criticality. Then, this metric is used to allocate SDN control plane resources.

There are also works aiming to handle the issues when a network failure occurs in the SDN data plane. Chu *et al.* proposed a single-link failure recovery mechanism in a hybrid SDN [32]. By redirecting traffic on the failed link to SDN switches through pre-installed IP tunnels, their approach can react to the failures quickly. They also discover multiple backup paths for the failure recovery to avoid congestion in the post-recovery network by choosing proper backup paths. They analyze the maximal link utilization in the post-recovery network, to understand their proposed mechanism whether works efficiently. 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 [33]. They use the number of restored data paths after failures as a metric to analyze their model's performance.

In another work, Lin *et al.* present a fast failover mechanism and a fast switchover mechanism to deal with link and switch failures and congestion problems. By monitoring periodically the status of each port of each OpenFlow switch, OF switches can failover the affected flows to another path when the link fails [34].

3. PROPOSED MODEL: REACTIVE ASSIGNMENT FRAMEWORK FOR SDN (RAFRES)

In this chapter, we describe our proposed model Reactive Assignment Framework for SDN (RAFRES), rendering various aspects such as how controller-switch assignments are made and how monitoring agents are used and integrated into the framework. The assignment algorithms that we adopt and on which improvements are made are listed. The communication and data exchange between the controller application and the optimization engine in RAFRES is also described briefly.

3.1. System Model

For failure cases in SDN, connectivity, capacity, and recovery must be considered to achieve resilience and robustness. To serve this objective, RAFRES works reactively against network failures in a software defined network architecture. These failures can be caused by device failures, security attacks or any user error. When a change due to network failure occurs in the topology, the application detects and reassigns the network nodes onto controllers to achieve robustness in newly formed condition. 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 the controller assignment. After reassignment, all network devices can be managed by a controller node despite the incumbent failure. The overall mechanism in RAFRES is shown in Figure 3.1. The tuple $(C, N, H_n, \mathcal{P}_{n,m})$ denotes controllers, network nodes, hosts of node n and the number of paths between node n and m, respectively.

Architecturally, RAFRES consists of three main components:

(i) A controller application that orchestrates and enforces the reassignment
(ii) Monitoring software agents in the network edge
(iii) A remote optimization engine

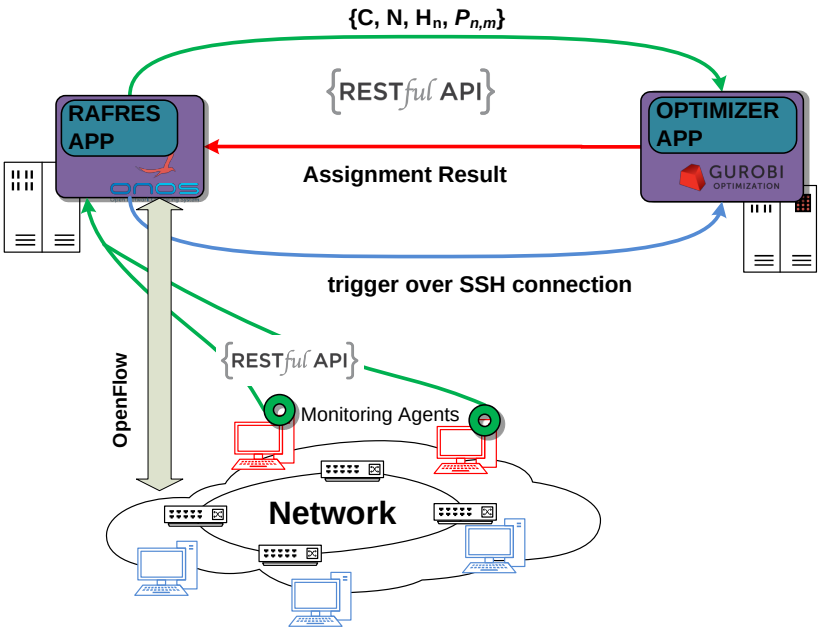

Figure 3.1. RAFRES architecture.

This architecture provides a modular system where new optimization techniques or monitoring data (from beacons or other network entities) can easily be integrated for developing our framework further.

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. Monitoring agents are implemented as software entities on hosts at the edge. The specific use-case of these agents is the monitoring of link quality and recovery. They send heartbeats in every 15 seconds, measure the round trip time (RTT) information for each controller instance, and 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 which are sent by monitoring agents to decide a "forbidden" controller $\widetilde{C}$ by separately calculating average RTT $\widehat{D}$ for each switch with monitoring agent hosts. If a monitoring agent has not sent a heartbeat within the last 30 seconds, the RTT information which is provided by this agent is ignored and the agent is considered dead by the controller. The decision rule for the forbidden controller

is simply "mark the controller with the highest $\widehat{D}$ as $\widetilde{C}$" for each switch. After the calculation of reassignment, the engine sends the controller-switch assignment results back to the controller application. Finally, the application applies the assignment via the controller and returns back to the monitoring mode for a prospective change in the network.

3.2. Controller Assignment 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 the 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 between the network nodes, F is the set of monitoring agents and H is the set of hosts. Essentially, F is a subset of H and the only difference between them is that they are also running as network beacons. Additionally, C denotes the set of controllers and $\mathcal{P}_{n,m}$ denotes the number of paths between node pairs $n \in N$ and $m \in N$.

3.2.1. Random Assignment

Using random assignment, RAFRES randomly assigns each network node $n \in N$ to a controller $c \in C$. When assigning a network node to a controller, it ensures that each node n will be controlled by exactly one controller c and no controller will remain idle. The main advantage of this naive approach is simplicity and reduced response time.

3.2.2. Simulated Annealing

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 [35]. Reducing search space and shortening the computation time to find near-optimal solutions 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 in Figure 3.2.

In SA based assignment, the algorithm tries to create node clusters up to the number of controllers according to *Gain* function. *Gain* function tries to form switch clusters up to the number of controllers to minimize the inter-cluster data paths. The smaller the gain function results, the more successful results the algorithm will have. 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 the use of *ratio cut formula* [30] shown in (3.1):

$$Gain = \frac{\sum_{A \subset N} \sum_{n \in A} \sum_{m \in A'} \mathcal{P}_{n,m}}{\prod_{A \subset N} |A|} \tag{3.1}$$

where A is a cluster of the network nodes and a subset of N, $|A|$ is the cardinality of cluster A, and A' is the complementary set of cluster A. The *ratio cut formula* is the ratio of the total number of inter-cluster paths to the product of cardinality of cluster sets. We aim to minimize the gain formula to increase resilience and to have a more favorable assignment in our system.

First, SA randomly distributes the network nodes into clusters and calculates the gain using Equation 3.1. The algorithm starts with starting temperature T_0. Then,

Require the knowledge of starting temperature (T_0), number of iterations (t_s), number of *move_states*(M);

Randomly distribute the network nodes into clusters

$T \Leftarrow T_0$;

$t_{stop} \Leftarrow t_s$;

Calculate *CurrentGain* using Eq. 3.1;

while $t_{stop} > 0$ **do**

  $AcceptMove \Leftarrow FALSE$;

  for $i = 1$ to M **do**

    Randomly select node n to move from one cluster to another;

    Calculate *NewGain* according to newly formed clusters using Eq. 3.1;

    $\Delta Gain \Leftarrow NewGain - CurrentGain$;

    if $AcceptGainChange(\Delta Gain, T)$ **then**

      $CurrentGain \Leftarrow NewGain$;

      $AcceptMove \Leftarrow TRUE$;

    else

      Return node n to original cluster;

    end if

  end for

  if $AcceptMove$ **then**

    $t_{stop} \Leftarrow t_s$;

  else

    $t_{stop} \Leftarrow t_{stop} - 1$;

  end if

  $T \Leftarrow T \times \alpha$;

end while

Figure 3.2. Simulated Annealing Algorithm.

Require the knowledge of the gain difference ($\Delta Gain$), the system temperature (T);
if controller overflows or first cluster has less then two nodes **then**
  return $FALSE$;
else if $\Delta Gain < 0$ **then**
  return $TRUE$;
else
  $R \Leftarrow$ random number$(0 < R < 1)$;
  $Y \Leftarrow e^{\frac{-\Delta Gain}{T}}$;
  if $R < Y$ **then**
    return $TRUE$;
  else
    return $FALSE$;
  end if
end if

Figure 3.3. *AcceptGainChange* function.

in each iteration, M *move_state*s occur. For each *move_state*, the 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* function shown in Figure 3.3. Moves are rejected immediately when the capacity of the controllers is exceeded or a cluster with only two switches is formed. 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 α. The algorithm stops if there have been no changes to the solution after t_s iteration.

While this structure is based on the model that was proposed by Manikas *et al.* in [29], we improved their proposal by adapting the gain function for more than two clusters, and making *AcceptGainChange* function load-aware. Hence, we are able to use this method for the controller-switch assignment to achieve a robust software defined network.

3.2.3. ILP Based Scheme

For ILP, a mathematical model is constructed as described below. Our model is originated from Survivor [28] 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 by using real-time loads of switches in the network. Moreover, placement-related constraints are omitted since we do not pursue placement problem and additional assignment-related constraints and capacity-related definitions are added in for a more consistent model. The mathematical model used for the ILP solution is presented below:

The input variables of our proposed mathematical model are represented as an 11-tuple:

$$I = \{G(N, L, F, H); C; f_{n,c}; U_c; \alpha_c; \beta; \gamma; \omega; H_n; F_n; B_n\} \tag{3.2}$$

where

- G is the physical topology of the network,
- C is the set of controller instances,
- $f_{n,c} \in \{0, 1\}$ is the indicator for whether controller $c \in C$ is forbidden for node $n \in N$,
- U_c is the capacity of the controller $c \in C$,
- α_c is the percentage of backup capacity for the controller $c \in C$,
- β is the weight coefficient for hosts,
- γ is the weight coefficient for monitoring agents,
- ω is the weight coefficient for switch loads,
- H_n is the set of hosts that are connected to node $n \in N$,
- F_n is the set of monitoring agents that are connected to node $n \in N$,
- B_n is the number of bytes that are received by node $n \in N$.

The output variables are represented as a 3-tuple:

$$V = \{x_{n,c}; R_n; w_c\} \tag{3.3}$$

where

- $x_{n,c} \in \{0, 1\}$ indicates whether node n is assigned to controller c,
- $R_n \in \mathbb{Q}^+$ is the traffic load of each device n,
- $w_c \in \mathbb{Q}^+$ is the aggregate load of controller c.

Definitions: w_c and R_n are calculated as shown below in Equation 3.4 and 3.5.

$$R_n = \beta \cdot |H_n| + \gamma \cdot |F_n| + \omega \cdot \frac{B_n}{\sum_{n \in N} B_n}, \forall n \in N. \tag{3.4}$$

Equation 3.4 calculates the number of requests for each node n using H_n, F_n, and B_n where $|\cdot|$ represents the cardinality of a set. This is the key element of our proposed mathematical model. It calculates the instant load of network nodes in order to use for load-distribution.

$$w_c = \sum_{n \in N} x_{n,c} R_n, \forall c \in C. \tag{3.5}$$

Equation 3.5 defines the aggregate load for each controller c considering the instant load of the network nodes that are assigned to c. This quantity is used as the empirical variable representing the instant load on the controller itself due to control plane functions.

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

$$\min \frac{\sum_{c \in C} (w_c - \mu_c)^2}{|C|} \tag{3.6}$$

which minimizes the variance of the total number of requests for controller c where μ_c denotes the average number of requests for each controller. In this way, our mathematical model provides load-balancing among the controller instances and gain robustness against catastrophic failures that can be caused by capacity overload.

Constraints: There are two types of constraints in our model: *assignment-related* and *capacity-related*. The first one prevents the conflicts in the controller-switch assignment, and the latter ensures that the controller capacities are not exceeded while achieving the objective function.

The first three constraints that are described in Equation (3.7, 3.8, 3.9) are *assignment-related* constraints and they provide the correctness of controller-switch assignments.

$$\sum_{c \in C} x_{n,c} = 1, \forall n \in N. \tag{3.7}$$

Equation 3.7 ensures that each node n will be controlled by exactly one controller c. This is the first constraint of the controller-switch assignment problem as long as a different management mechanism such as collective mastership is not employed.

$$\sum_{n \in N} x_{n,c} \geq 1, \forall c \in C. \tag{3.8}$$

Equation 3.8 guarantees that each controller c will control at least one node n. This is an additional constraint that we proposed to guarantee that no idle controllers will remain. In this way, all controllers will be used in their efficient capacity.

$$x_{n,c} \geq 1 - f_{n,c}, \forall c \in C, \forall n \in N. \tag{3.9}$$

Equation 3.9 provides that each node n will not be assigned to the controller c if it is forbidden for the node. Prohibition matrix f is a binary matrix that is generated according to the forbidden controller(s) ($\widetilde{C}$) described in Section 3.1.

The other constraint that is explained in Equation 3.10 is a *capacity-related* constraint and guarantees that the capacity of the controllers will not be exceeded by the controller-switch assignment.

$$\sum_{n \in N} x_{n,c} R_n \leq (1 - \alpha_c) \cdot U_c, \forall c \in C. \tag{3.10}$$

Equation 3.10 provides that the controller capacity will not be exceeded taking into account the backup capacity. This backup capacity ensures that the control plane is able to operate smoothly when any controller is broken or new network nodes are added to the topology.

4. PERFORMANCE ANALYSIS

In this chapter, in addition to the simulation environment and experimental scenarios, the performance evaluation of RAFRES is explained. First, we describe the hardware and software components of our testing environment. The distributed controller that we use, and the controller application development process is described. Then, how the network topology is constructed and traffic is simulated between hosts are presented. We explain the run-time analysis of algorithms that we operate to identify our expectations from the proposed framework. Last, experimental scenarios are described and performance metrics are listed and explained in detail. Therefore, the overall process to analyze the performance of the RAFRES framework is clearly laid out.

4.1. Experimental Environment and Parameters

To examine the proposed framework, we use a machine with Intel® Core™ i7-4790 (3.60GHz × 8), 16GB RAM and 1TB HDD space to host four virtual machines (VMs). Each VM has 4GB memory and 100GB HDD space, and is running a controller instance. Three of them are used for the distributed controller mechanism. We use ONOS Nightingale 1.13.1 as SDN controller and the controller application is developed using Java. Mininet 2.2.1 [36] is used for creating network topologies on Ubuntu 14.04 LTS with Open vSwitch 2.0.2 virtual switches. The optimization engine works on the same computer but uses a separate disk space which is a 256GB SSD space. Gurobi Optimizer 8.1.0 [37] is chosen as the ILP solver. Distributed Internet Traffic Generator (D-ITG) 2.8.1 [38] to generate traffic from hosts and monitoring agents are used in our experiments.

For the experiments, a Wide Area Network (WAN) topology is used. It is based on ULAKNET academic network in Turkey as shown in Figure 4.1. 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

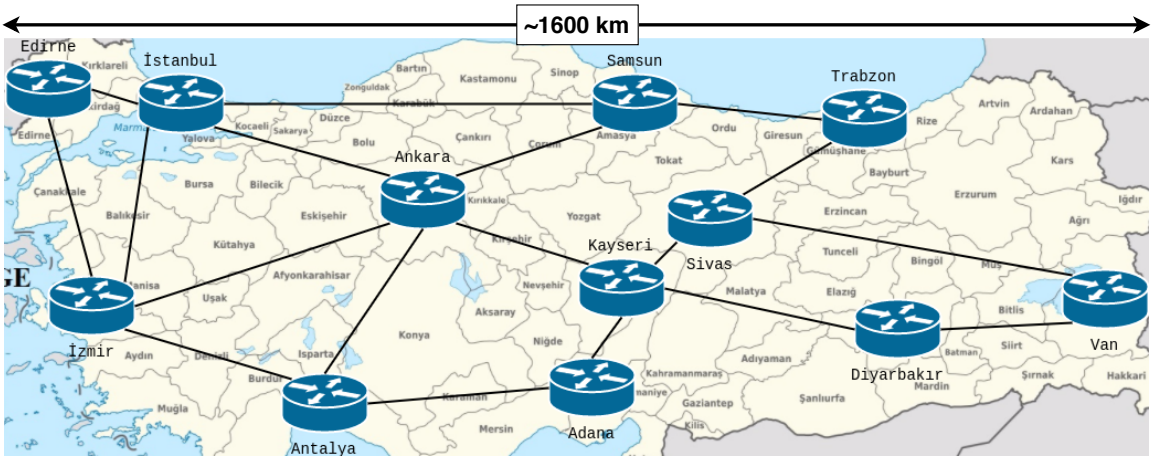

Figure 4.1. WAN of major cities in Turkey.

connections between these cities and is changed by creating redundant links and extra network nodes for more complicated test cases in the Mininet simulation environment.

Monitoring agents are also simulated as hosts in Mininet. Every monitoring agent is run as a software beacon on the hosts at the network edge. Beacons are initialized using a Bash script and configurable according to experiment scenarios.

Table 4.1. Traffic Parameters.

Traffic Type	**Packet per Second**		**Packet size (bytes)**	
	Distribution Type	**Average**	**Distribution Type**	**Average**
VoIP	Exponential	65	Uniform	[65,128][1]
Video	Constant	1000000	Poisson	480
Racing	Exponential	1015	Exponential	691
Sport	Exponential	915	Exponential	597
FPS	Exponential	975	Exponential	667
Flight	Exponential	955	Exponential	722
Puzzle	Exponential	722	Exponential	459

[1]These are the minimum and maximum packet sizes for uniform distribution.

Generating network traffic considering actual traffic characteristics is crucial for realistic experiments. Therefore, we use the parameters of different types of traffics which are obtained from actual traffic flows. For the network traffic, we first decide on the traffic characteristics according to our experimental scenarios, then generate the traffic flows of VoIP, video and five different types of online games with D-ITG. The parameters of these traffic types are obtained from the literature which rely on actual traffic flows [39, 40] and are listed in Table 4.1.

We comply with the system parameters used in the experiments in Table 4.2.

Table 4.2. System Parameters.

Parameter	**Value**	**Explanation**
α	0.8	SA cooling factor
T_0	1000	SA starting temperature
α_c	0.1	Percentage of backup capacity for controllers
β	0.5	Weight coefficient for hosts
γ	0.5	Weight coefficient for monitoring agents
ω	5	Weight coefficient for instant loads
U_c	200	Controller capacity

For SA, five different (M,t_s) tuple is used in the tests, and they are shown below:

(M,t_s) = {(1,1),(1,3),(1,5),(3,5),(5,5)}

4.2. Algorithmic Run-time Analysis

To evaluate the framework, run-times of three controller-switch assignment algorithms on topologies of various sizes are examined. The computational complexity of random assignment is $O(N)$ due to the assignment of each network node to a controller instance one by one, where N stands for the number of network nodes in the topology. SA and ILP run with $O(M \times N^2)$ and $O(2^N)$ according to their working

steps, respectively, and M represents the number of *move_state*s. These theoretical time complexities render run-time behavior expected from RAFRES algorithms and guide us to establish a trade-off between run-time and network performance.

During these run-time experiments, we use eight different topology sizes, i.e. 5 to 1000 switches. These topologies are generated artificially in different sizes after examining our topology in terms of traffic, the number of hosts and the link structure of the network nodes. For SA, different numbers of iterations (t_s) and numbers of *move_state*s per iteration (M) values are tested. All algorithms were run 100 times for each size of topologies and the mean values were reported in Figures 4.2 and 4.3. For topology with five switches, SA has no results because it needs at least two switches for each controller instance to be able to operate. For simplicity in the legends of the graphs, we label the results of SA according to different parameters as SA(i, j) where $M = i$ and $t_s = j$.

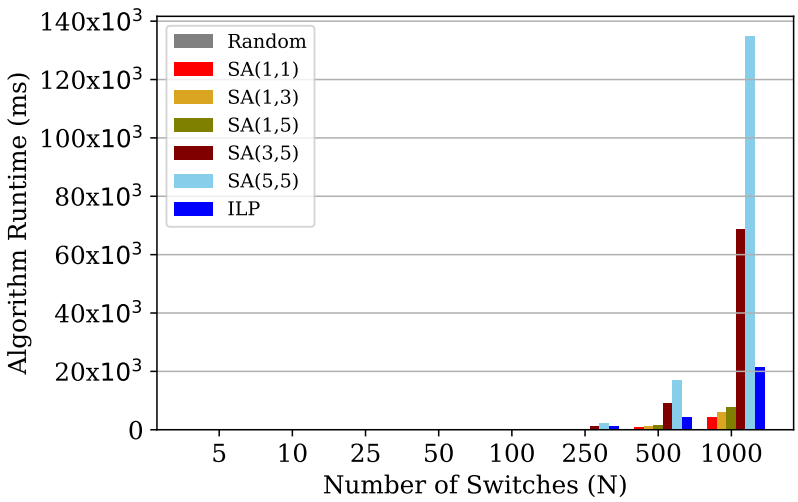

Figure 4.2. Run-times of RAFRES algorithms.

As shown in graphs, random assignment is the quickest algorithm to find an assignment solution as expected. The results show that the algorithms give the anticipated 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 multi-

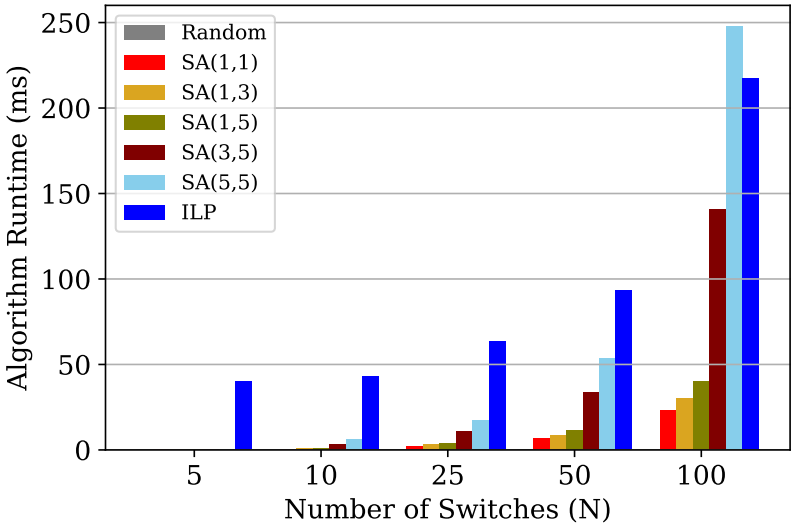

Figure 4.3. Run-times for up to 100 switches (zoomed from Figure 4.2).

plies exponentially due to the increasing number of iterations based on t_s 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 as general guideline.

4.3. Experimental Scenarios and Performance Metrics

In our experiments to evaluate our proposed framework, we have used two primary scenarios to examine the effects of the network failures and the benefits of the monitoring agents.

Scenario 1: In this scenario, we trace the performance of RAFRES without using the monitoring agents as network beacons. Monitoring agents basically provide the information that they are alive and running. We use a topology that consists of 15 switches and all the links between the switches have 10 Gbit bandwidth. After starting the base traffic to establish the background load on the network throughout the entire scenario, different types of traffic flows are generated at various intervals over certain periods of time among the randomly selected users for a dynamic traffic behavior.

A network failure (e.g. switch, link or controller) happens at a specific point in the timeline. The failure type is decided before the scenario starts, and the failure occurs at a randomly selected element in the network.

Scenario 2: In this scenario, we analyze the effect of the monitoring agents when they are used as network beacons. Monitoring agents send also the RTT values for this scenario unlike the first one. We adopt a network topology that consists of 20 switches and all the links between the switches have 1 Gbps bandwidth to force RAFRES into a more constrained regime. After starting the base traffic, the same general chain of events as described above occur.

For dynamic traffic behavior, traffic flows are generated according to two different timelines in our experiments. The first timeline takes 360 seconds and the traffic flows take place at more scattered times. The events in the first timeline are listed in Table 4.3.

The second timeline is generated to measure the impact of the network failure on the current traffic flows. Therefore, the traffic flows take place more densely and this timeline is only 150 seconds long. This timeline has three groups of traffic flows as *before the failure*, *during the failure* and *after the failure* and the network failure happens 75 seconds after the events start. The second timeline can be seen in Figure 4.4.

We simulate switch, link and controller failure as network failures in the scenario timelines. Each failure type is generated using different procedures, and a list of events that occur after the failures is different from each other. We explain all three event sequences in detail below:

- *Switch Failure:* To simulate switch failure, one of the switches in the topology is stopped at the specified time in the scenario. Besides the network node itself, all the links to other nodes are also severed and the network access of the hosts connected to that switch is lost. After that failure incident, RAFRES calculates

Table 4.3. First Timeline of Scenario 2.

Starting Time(sec)	**Traffic Type**	**Duration(sec)**
0	Background	360
30	FPS	30
	Video	20
	VoIP	5
	Racing	12
40	Video	20
	Racing	10
	VoIP	5
65	Video	20
	Racing	10
	VoIP	5
80	Sport	20
	Puzzle	60
	VoIP	5
	Racing	12
110	Video	20
	FPS	10
	VoIP	5
	Racing	12
150	Racing	15
	Sport	25
	VoIP	5
180	Flight	35
	Video	45
	VoIP	5
	Racing	12
200	**Network Failure**	
240	Video	15
	Racing	12
	VoIP	5
290	Puzzle	30
	VoIP	5

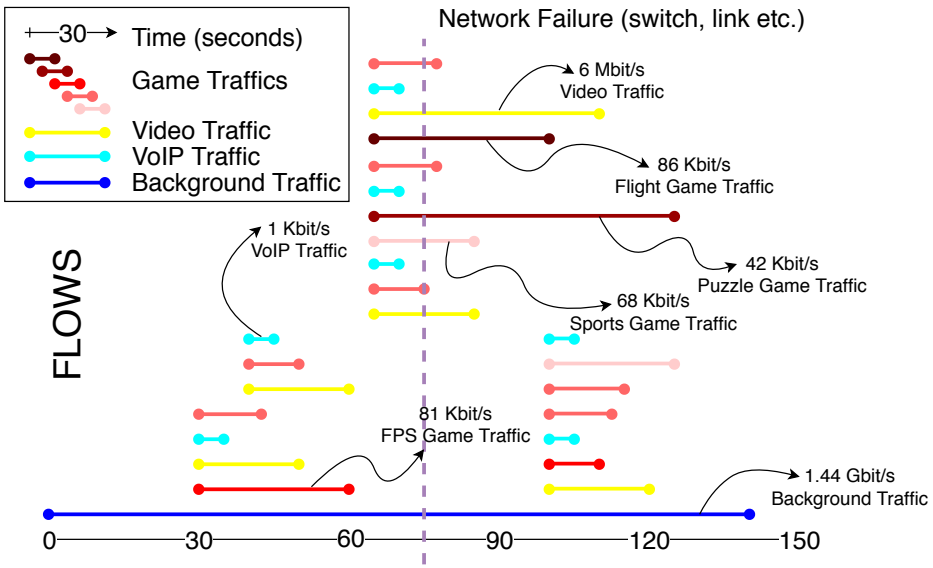

Figure 4.4. Second timeline of Scenario 2.

the controller-switch assignment according to the new topology and current loads, then accordingly traffic continues to flow.

- *Link Failure:* To severe a network link, one of the interfaces connected by the link is shut down from the OF switch and the interface at the other endpoint closes automatically. After the link failure, the assignment is calculated reactively by RAFRES while the operation continues.
- *Controller Failure:* In order to create a controller failure, we remotely shut down the controller instance via remote connection. 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 new master for controller instances and RAFRES reassigns the switches to surviving controllers based on the embedded algortihms. After the reassignment, the scenario events continue to happen.

To assess the RAFRES performance, we first analyze the distribution of the number of `PACKET_IN` messages on controllers to observe the performance of the load-aware ILP model. Latency and throughput of data traffic were also examined under

both of the scenarios with the dynamic RAFRES mechanisms. Average bitrate, average delay, and maximum delay were selected as performance metrics.

All experiments in this section were run ten times and average results are reported with their standard deviation in the following section.

4.4. Results and Performance Evaluation

In this section, we express the results of our experiments for different scenarios under proposed timelines to analyze the performance of the load-aware ILP model, monitoring agents and online controller-switch assignment. First, we evaluate the performance of load-aware controller-switch assignment by measuring the number of `PACKET_IN` messages on each controller. Then, to examine the framework against network failures, we apply switch, link and controller failures in the experiments. While the traffic flows occur according to the dedicated scenario, network failure is triggered at a certain time that is decided in the timeline. After the failure, change in the network is detected by RAFRES and the framework reassigns the network nodes and the scenario continues until completed. For each type of failure scenario, average delay, maximum delay, and average bitrate values are measured as performance metrics after RAFRES performs reassignment as a response to the failure.

4.4.1. Performance of Load-aware Controller Assignment

We measure 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) [41] which is a built-in ONOS application. After performing *Scenario 1* with the events in *Timeline 1* listed in Table 4.3, we calculate the average number of the packets received by the controller instances. In this experiment, any network failure occurs as an exception. The results can be seen in Figure 4.5.

ILP gives the best results and random assignment is the worst among the algorithms. Results of SA are improved by increasing the number of iterations. SA(5,5)

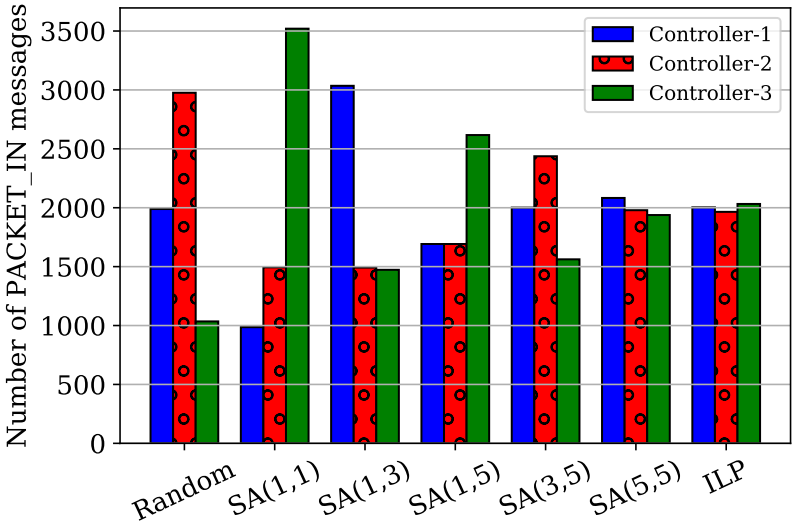

Figure 4.5. Number of `PACKET_IN` messages.

gives nearly the same results with the optimal ILP one, but run-times differ at that point with SA having a very low run-time. However, as seen in Figure 4.2, 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.

4.4.2. RAFRES without Monitoring Agents

In this section, we examine the performance of RAFRES without monitoring agents. To analyze the success of RAFRES, we perform *Scenario 1* with the events in *Timeline 1*. In this scenario, monitoring agents do not send the RTT values and therefore prohibition matrix f is a zero matrix and there is no forbidden controller in the system. All three network failures are generated separately. Performance metrics are calculated for each failure one by one.

The results of switch failure can be seen in Figure 4.6, 4.7 and 4.8. The optimal model achieves an average bitrate of 516 Mbps with a small variance. SA is able to provide a maximum bitrate of 387 Mbps for different values of M and t_s. In terms of

maximum delays, random assignment gives the worst value as 105 milliseconds, while the optimal model has the smallest value with 65 milliseconds. SA managed to achieve 74 milliseconds at best with the parameter values of M=1 and t_s=5. For average delay, all assignment strategies end up with close results like between 35 microseconds and 42 microseconds except from SA(5,5). It has 58 microseconds average delay for switch failure. Moreover, random assignment has the highest standard deviation values and ILP has the lowest for all metrics that we measure. SA is concluded with closer results for each run and gets smaller standard deviation values, as the number of iterations increases.

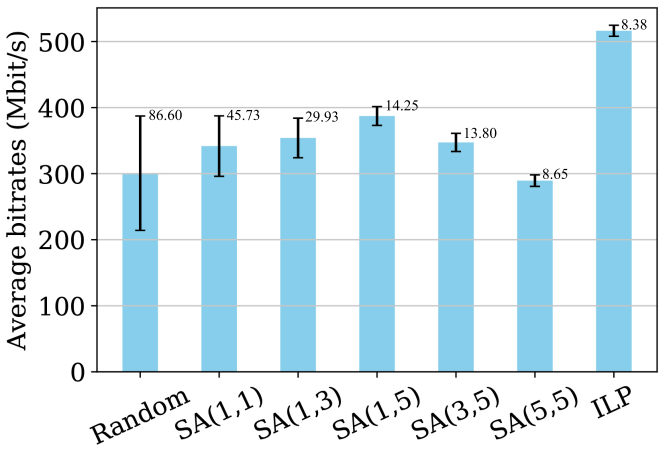

Figure 4.6. Average bitrates for switch failure without monitoring agents.

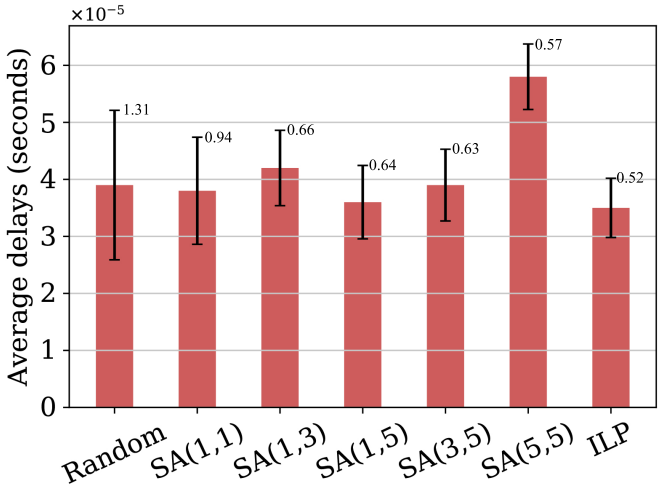

Figure 4.7. Average delays for switch failure without monitoring agents.

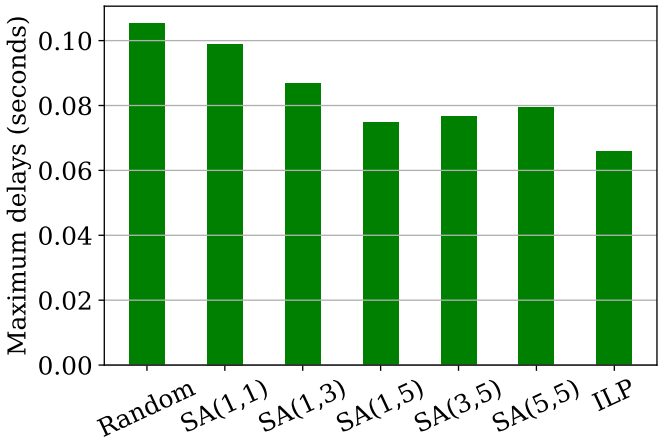

Figure 4.8. Maximum delays for switch failure without monitoring agents.

For the link failure scenario, the optimal model has the best value of average bitrate with the value of 547 Mbps. The second most successful assignment method is SA(1,5) which achieves 375 Mbps. SA(3,5) and SA(5,5) have worse results among all (M,t_s) tuples than expected, but these results show a glimpse of the trade-off between the performance and the number of iterations. Except for them, the other algorithms achieved similar average delays between 32 microseconds and 39 microseconds. During the link failure incident, the optimal model has a maximum delay of 100 milliseconds and there is no better result among the other algorithms. Random assignment ends up with 150 milliseconds of maximum delay, SA managed to achieve 110 milliseconds at best with the parameter values of M=1 and t_s=5. Standard deviations perform like the switch failure case mentioned above. The results can be seen in Figure 4.9, 4.10 and 4.11.

The results of the controller failure can be seen in Figure 4.12, 4.13 and 4.14. Election of the master among the controllers decreases the average bitrates and increases the average and maximum 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 than the other failures. ILP achieves 456 Mbps and SA(5,5) follows the optimal approach with 362 Mbps. Random assignment has the lowest average bitrate with 297 Mbps. In addition, ILP has the best average delay with 42 microseconds and random assignment is the worst assignment methodology with 60 microseconds of

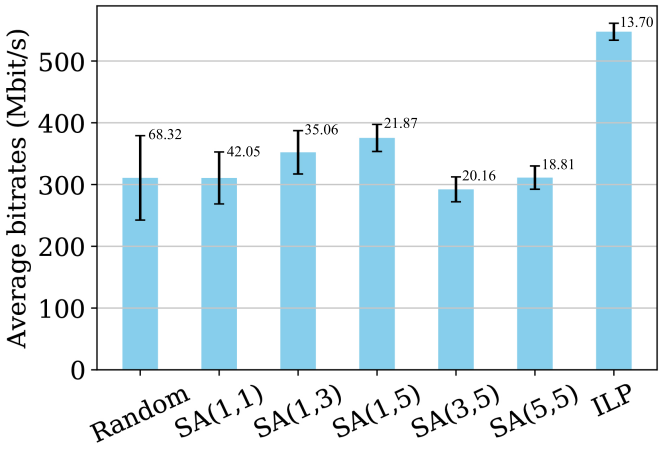

Figure 4.9. Average bitrates for link failure without monitoring agents.

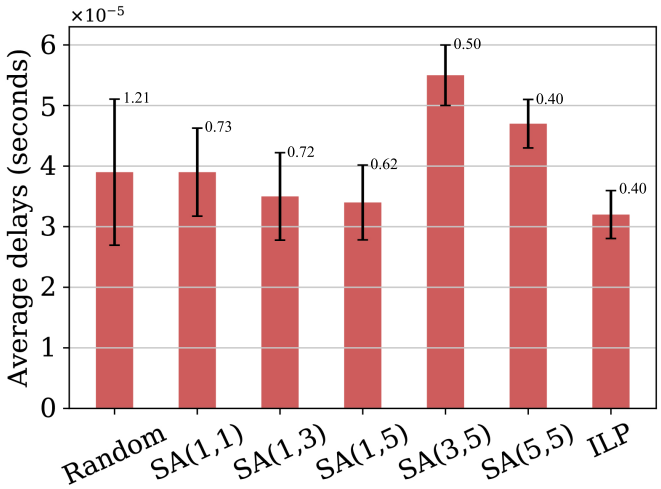

Figure 4.10. Average delays for link failure without monitoring agents.

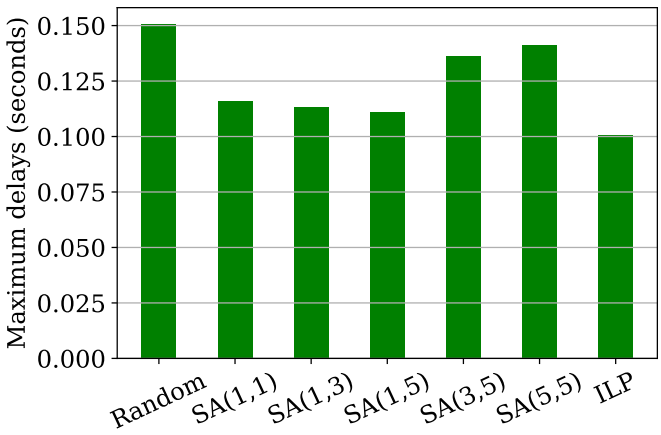

Figure 4.11. Maximum delays for link failure without monitoring agents.

average delay. In terms of maximum delay, ILP has the best result with 129 milliseconds and random assignment causes nearly 195 milliseconds of delay. Average bitrates are lower as well as average and maximum delays are higher than the other scenarios due to the election of the master controller.

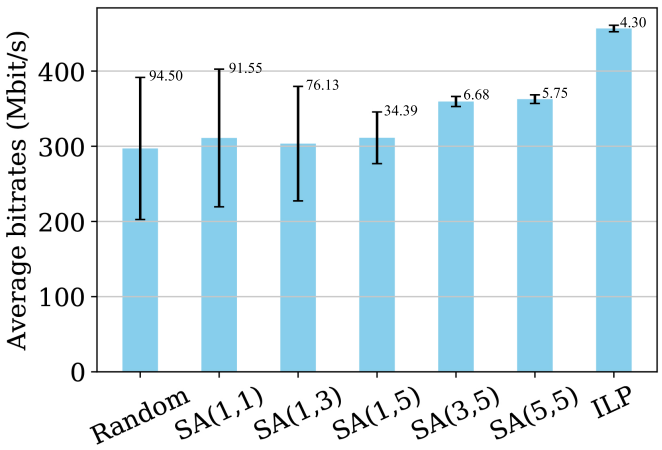

Figure 4.12. Average bitrates for controller failure without monitoring agents.

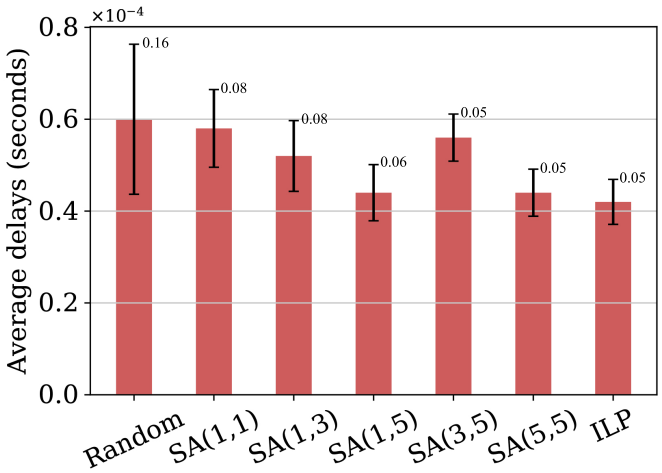

Figure 4.13. Average delays for controller failure without monitoring agents.

4.4.2.1. Analysis of the traffic flows during the failure. We have also examined the performance metrics for only the active flows that continue when the network failures occur. Therefore, we are able to understand the nominative effect of the failures to the system. There are only three flows that match the description above for *Time-*

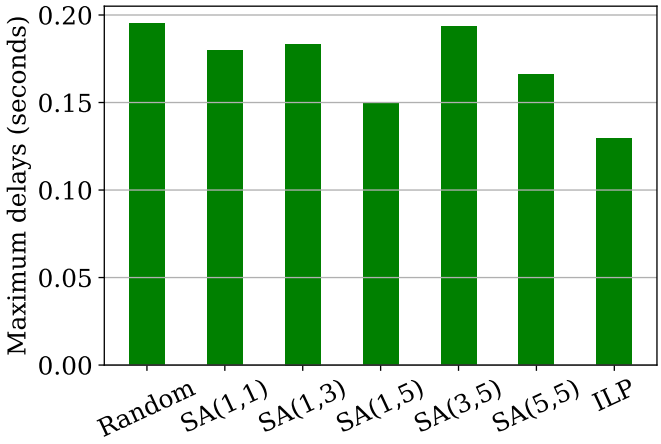

Figure 4.14. Maximum delays for controller failure without monitoring agents.

line 1. To accomplish this, the same flow logs with Section 4.4.2 are used to obtain performance characteristics for these flows.

Experiment results show that maximum delays are caused by the network interruptions, because the same maximum delay values are obtained in the section above. The decrease in the number of examined flows caused the average bitrate to drop. In addition, because all the examined flows are affected by the failure, the average delay values are higher than the results in Section 4.4.2. The graphs of average bitrates and average delays for the network failures can be seen below.

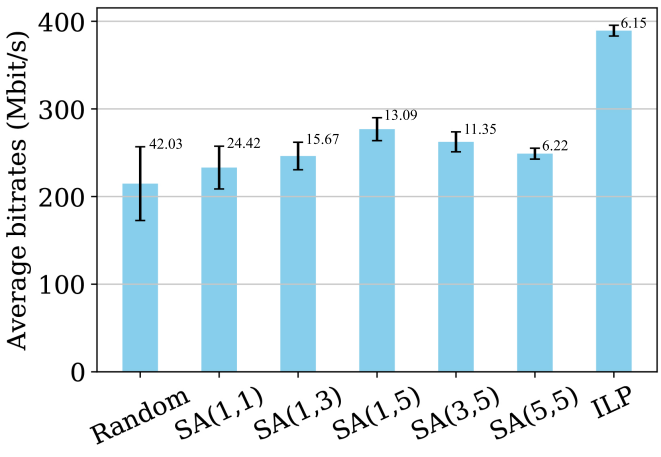

Figure 4.15. Average bitrates for only the flows during switch failure.

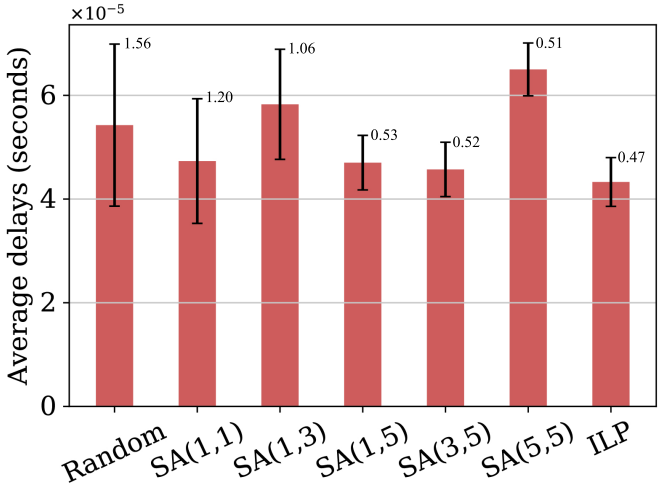

Figure 4.16. Average delays for only the flows during switch failure.

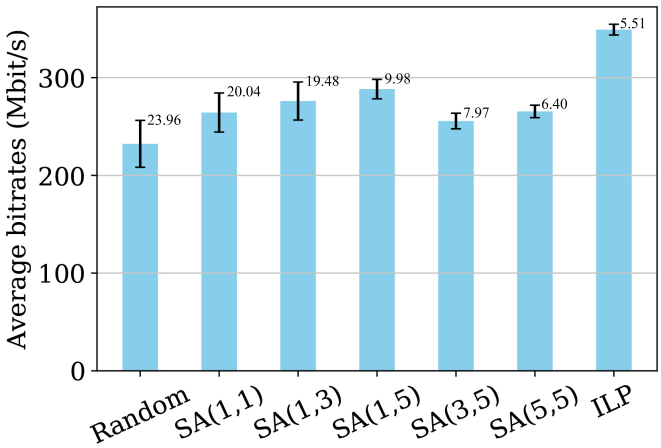

Figure 4.17. Average bitrates for only the flows during link failure.

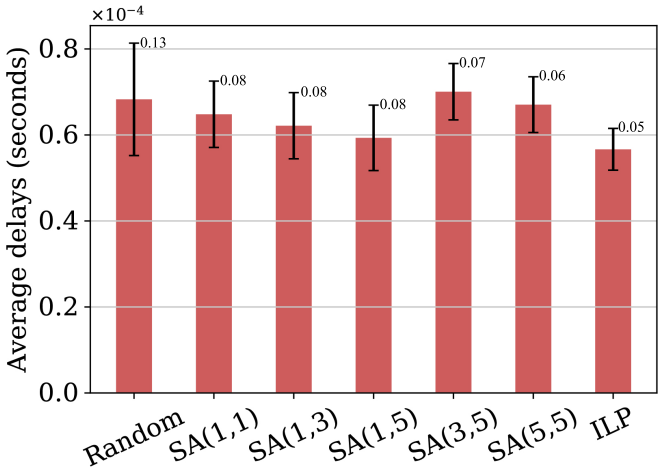

Figure 4.18. Average delays for only the flows during link failure.

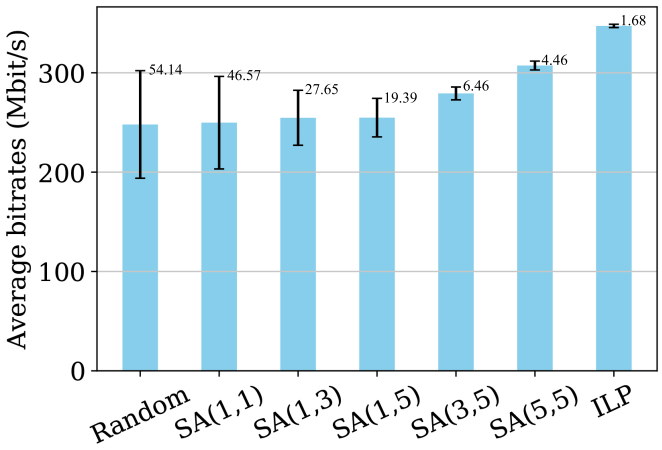

Figure 4.19. Average bitrates for only the flows during controller failure.

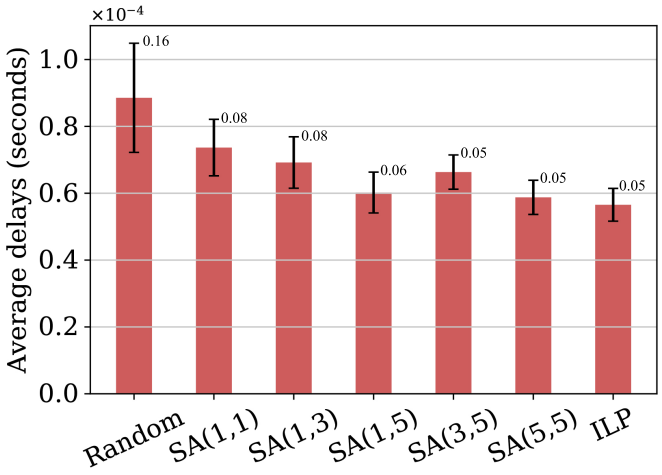

Figure 4.20. Average delays for only the flows during controller failure.

4.4.3. RAFRES with Monitoring Agents

In this section, to analyze the performance of RAFRES with monitoring agents, we perform *Scenario 2* with the events in both *Timeline 1* and *Timeline 2* in order. In this scenario, monitoring agents are used as network beacons so that forbidden controllers can be identified by the controller application. In addition, we challenge RAFRES further by decreasing link bandwidths in the network.

4.4.3.1. Timeline 1 (Sparsely distributed flows). First, we evaluate the system using the events in *Timeline 1*. In this timeline, traffic flows occur more sparsely, and less number of flows are affected by network failures. This experiment can be used to investigate the effect of lower bandwidth on the performance of RAFRES.

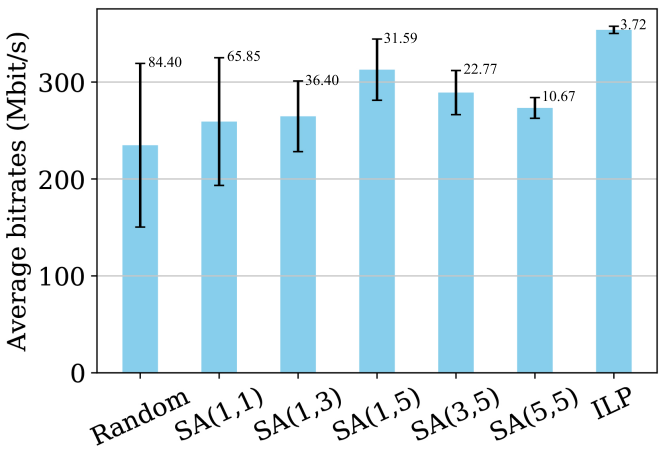

Figure 4.21. Average bitrates for switch failure with sparsely distributed flows.

In this case, ILP provides an average bitrate of 353 Mbps with minimum variance for switch failure. Random assignment can achieve 234 Mbps average bitrate with a high standard deviation. SA can end up with a maximum bitrate of 312 Mbps using the parameter values of M=1 and t_s=5. Random assignment gives the worst performance as 101 milliseconds of maximum delay, while the optimal model has the smallest value with 59 milliseconds. SA managed to achieve 64 milliseconds at best with the same

parameter values mentioned above. For average delays, all assignment strategies end up with results like between 53 microseconds and 73 microseconds. Moreover, random assignment has the highest standard deviation values and ILP has the lowest for all metrics again like Section 4.4.2. The results of switch failure can be seen in Figure 4.21, 4.22 and 4.23. The effects of lower bandwidth can be seen on average bitrates and average delays. It limits the average bitrate at a certain point and causes higher average delays in the system.

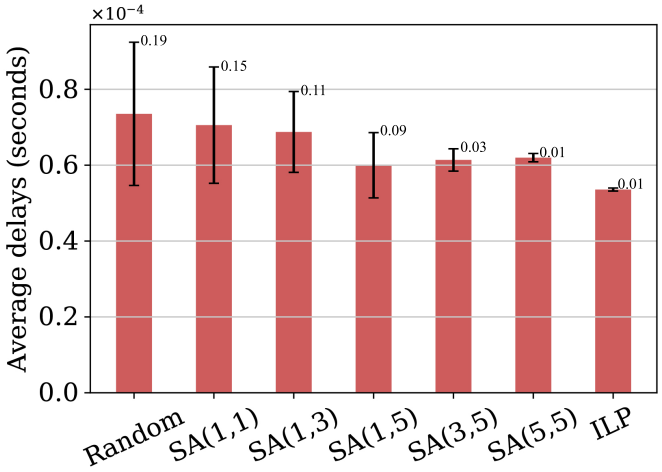

Figure 4.22. Average delays for switch failure with sparsely distributed flows.

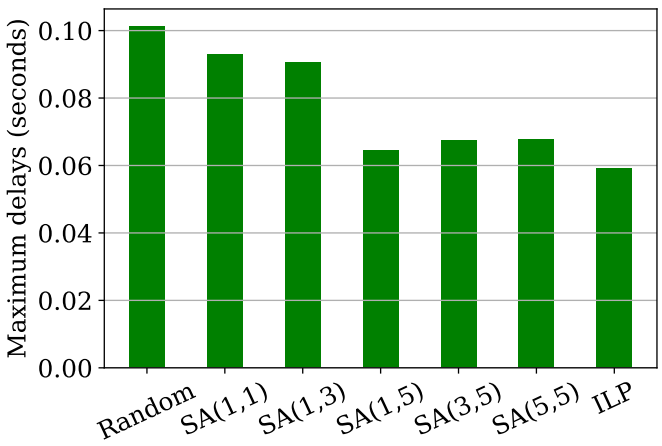

Figure 4.23. Maximum delays for switch failure with sparsely distributed flows.

For the link failure scenario, the optimal model has 302 Mbps average bitrate as the best value. SA(1,5) follows it by achieving 278 Mbps. SA(1,1) and SA(1,3) have

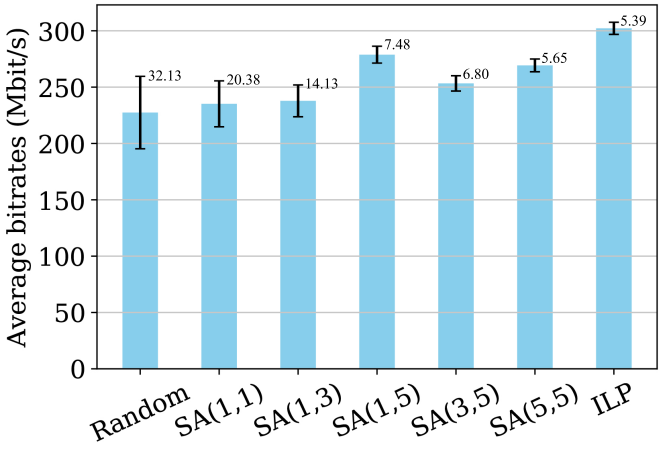

Figure 4.24. Average bitrates for link failure with sparsely distributed flows.

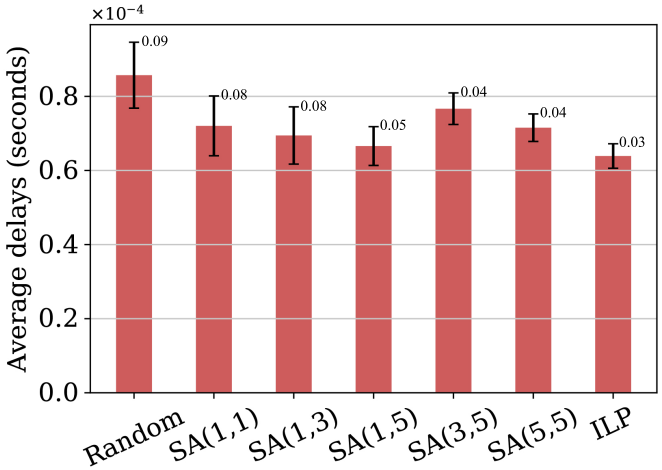

Figure 4.25. Average delays for link failure with sparsely distributed flows.

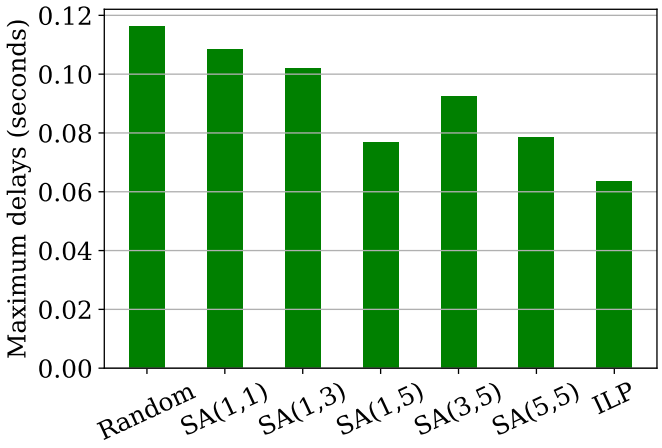

Figure 4.26. Maximum delays for link failure with sparsely distributed flows.

the worst results among all (M,t_s) tuples as expected. Except for random assignment, the other algorithms achieved similar average delays between 63 microseconds and 76 microseconds. Random assignment has an average delay of 85 microseconds. The optimal model has a maximum delay of 63 milliseconds and there is the best result among the other algorithms. Random assignment comes out with 116 milliseconds of maximum delay, SA managed to achieve 76 milliseconds at best with the parameter values of M=1 and t_s=5. Results can be seen in Figure 4.24, 4.25 and 4.26.

For controller failure, the optimal solution achieves the best average bitrate with 270 Mbps and random assignment is at the last place among the assignment algorithms with 246 Mbps. However, there is not much difference between them. It is caused by the coexistence of lower bandwidth situation and controller failure. SA(5,5) follows the optimal with 267 Mbps. In addition, ILP has the best average delay with 61 microseconds and random assignment is the worst one with 91 microseconds of average delay. In terms of maximum delay, ILP has the best result with 67 milliseconds and random assignment causes 122 milliseconds of delay. The election of the master controller impedes the recovery of the system, therefore average bitrates are lower and the average and maximum delays are higher than the other scenarios again. The results of the controller failure can be seen in Figure 4.27, 4.28 and 4.29.

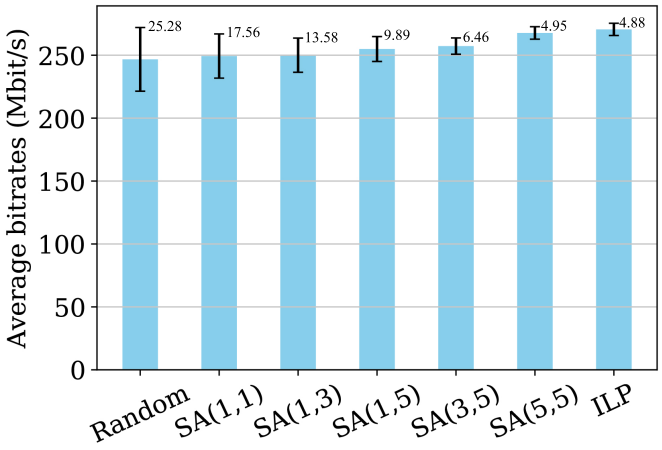

Figure 4.27. Average bitrates for controller failure with sparsely distributed flows.

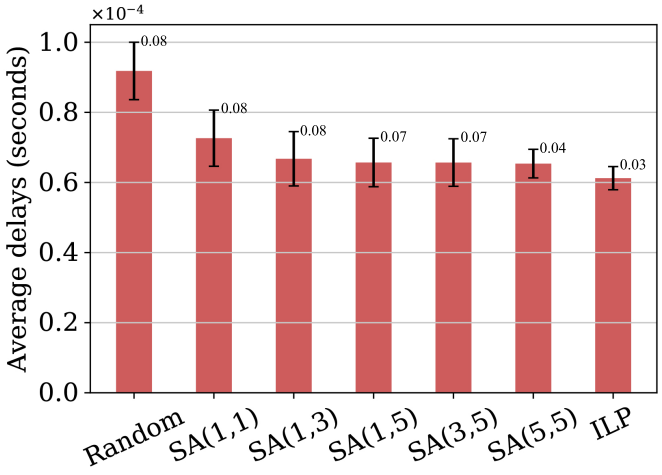

Figure 4.28. Average delays for controller failure with sparsely distributed flows.

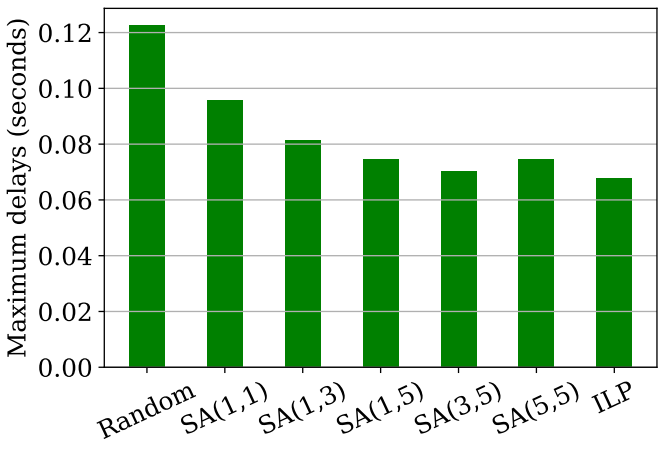

Figure 4.29. Maximum delays for controller failure with sparsely distributed flows.

4.4.3.2. Timeline 2 (Densely distributed flows). We also evaluate the framework using the events in *Timeline 2*. In this case, traffic flows occur more densely, and a large number of traffic flows are affected by network failures according to this timeline. The performance of RAFRES can be evaluated against a network failure in case of heavy traffic by the help of this experiment.

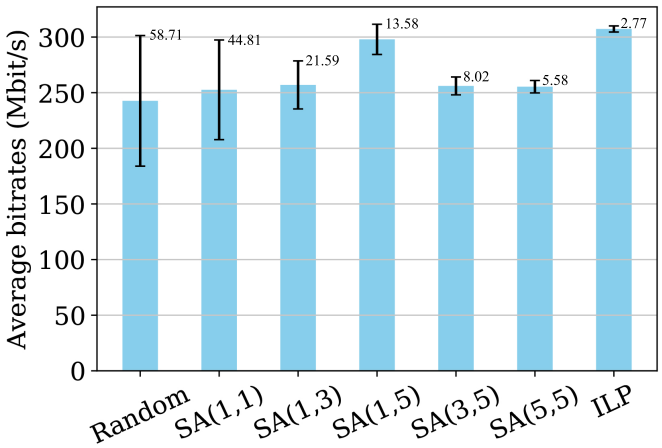

Figure 4.30. Average bitrates for switch failure with densely distributed flows.

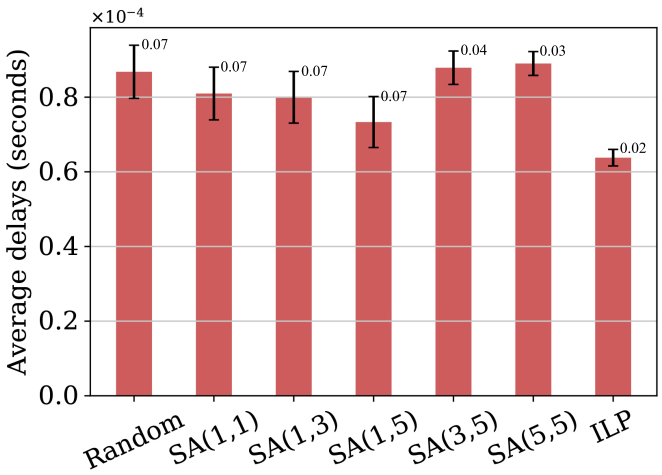

Figure 4.31. Average delays for switch failure with densely distributed flows.

For the switch failure scenario, the optimal model has the best value of average bitrate with the value of 307 Mbps. The second most successful method is SA(1,5)

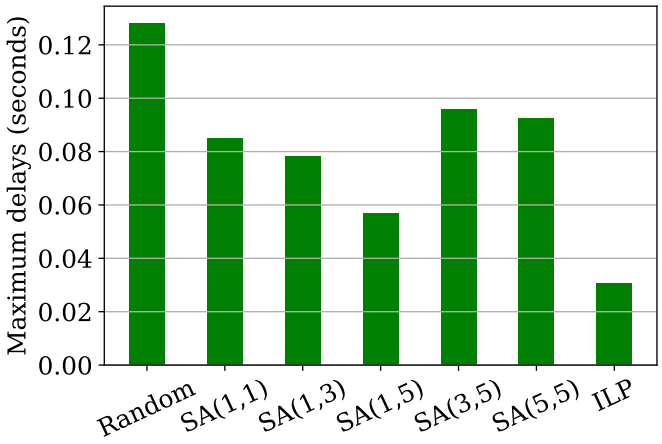

Figure 4.32. Maximum delays for switch failure with densely distributed flows.

which has 297 Mbps average bitrate. SA(3,5) and SA(5,5) have worse results as expected, because of the number of iterations. SA(5,5) achieves 89 microseconds average delay and it is the worst among all algorithms. ILP has the lowest average delay with 63 microseconds. During the switch failure, the optimal model has a maximum delay of 30 milliseconds and there is no better result among the other algorithms. Random assignment ends up with 128 milliseconds of maximum delay and SA managed to achieve 56 milliseconds at best with the parameter values of M=1 and t_s=5. The results of switch failure are provided in Figure 4.30, 4.31 and 4.32.

The link failure results can be observed in Figure 4.33, 4.34 and 4.35. The optimal model achieves an average bitrate of 247 Mbps with a small standard deviation. SA can achieve an average bitrate of 246 Mbps as its best. As a matter of fact, all SA using different (M,t_s) tuples, end up with close average bitrates. Random assignment could not even get close to others with an average bitrate of 183 Mbps. In terms of maximum delays, random assignment gives the worst value as 141 milliseconds, while the optimal model has the smallest value with 54 milliseconds. SA managed to achieve 84 milliseconds at best with the parameter values of M=1 and t_s=5. For average delay, ILP and SA(1,5) achieve 75 microseconds and this is the best result among algorithms. Random assignment has again the worst result with an average delay of 100 microseconds for link failure.

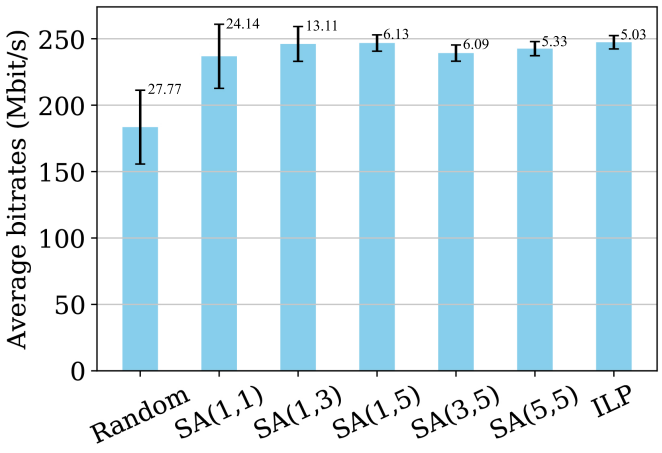

Figure 4.33. Average bitrates for link failure with densely distributed flows.

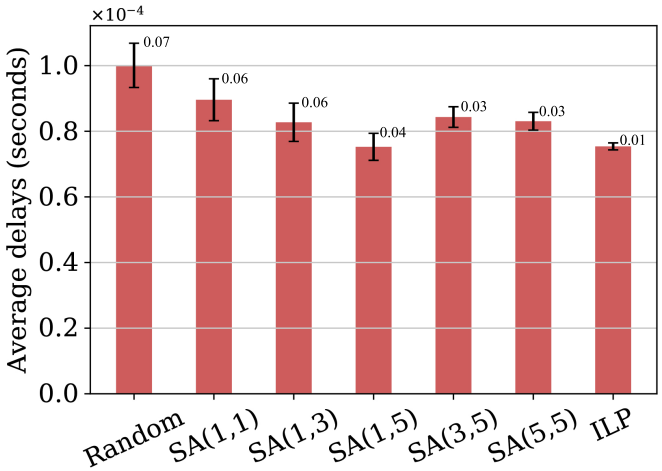

Figure 4.34. Average delays for link failure with densely distributed flows.

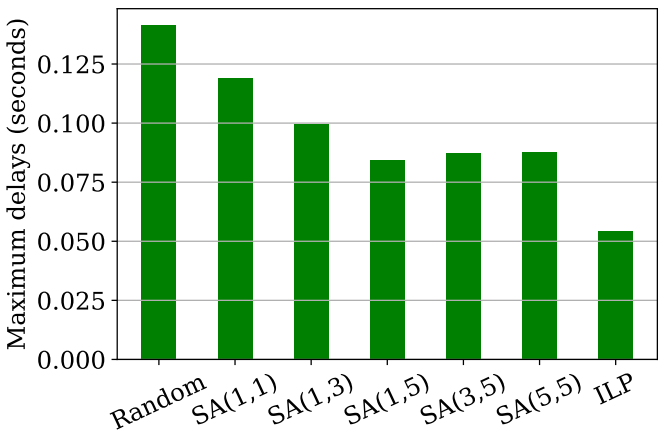

Figure 4.35. Maximum delays for link failure with densely distributed flows.

The optimal solution provides 235 Mbps average bitrate for the controller failure. Random assignment can achieve an average bitrate of 153 Mbps with a higher standard deviation than the other assignment algorithms. SA can provide an average bitrate of 227 Mbps as its maximum using the parameter values of M=5 and t_s=5. In addition, random assignment gives the worst value as 149 milliseconds of maximum delay, while the optimal model has the smallest value with 74 milliseconds. SA achieves 89 milliseconds at best with the same parameter values mentioned above. For average delays, random assignment has the highest value as 489 microseconds, and even ILP is able to achieve 189 microseconds against controller failure when densely distributed traffic flows occur. The results of controller failure can be seen in Figure 4.36, 4.37 and 4.38.

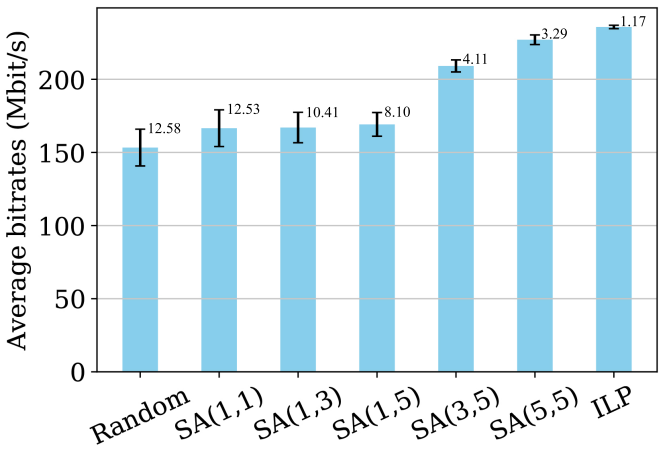

Figure 4.36. Average bitrates for controller failure with densely distributed flows.

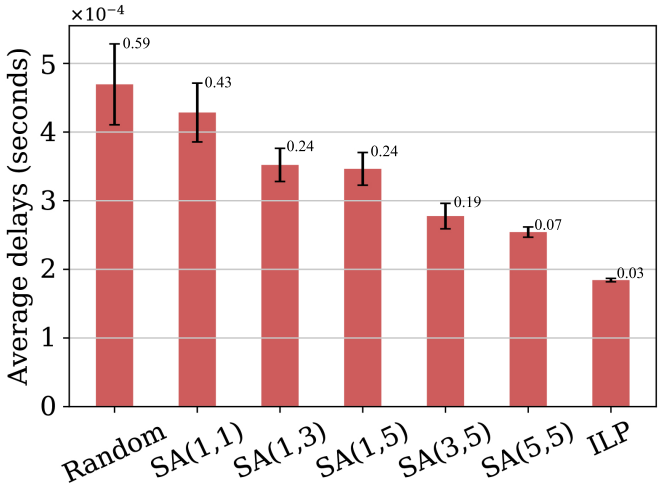

Figure 4.37. Average delays for controller failure with densely distributed flows.

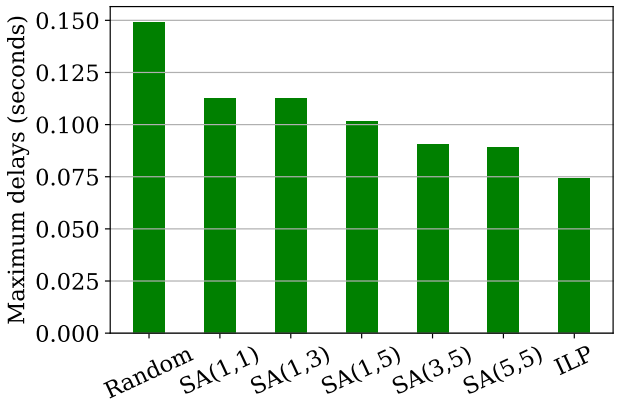

Figure 4.38. Maximum delays for controller failure with densely distributed flows.

4.5. Discussions

Overall, experimental results show that random assignment has the worst results, and the proposed load-aware ILP 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 sluggishness of the assignment calculation which causes higher latency and lower throughput in the network. This degradation is caused by 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.

When 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 the smallest average bitrate 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.

Lastly, in the scenarios, where monitoring agents are used as network beacons, the success of the proposed ILP based model is higher than the scenarios where the monitoring agents are not deployed, when compared to other approaches.

5. CONCLUSION AND FUTURE DIRECTIONS

The adoption of distributed controllers is not solely adequate to achieve robustness and reliability goals in SDN. Such an architecture must employ a dynamic and high-performance controller-switch assignment strategy. In this thesis, we have presented a reactive assignment framework against network failures using integer linear programming based on the load distribution of controllers. We have also proposed a novel strategy to monitor the network by using monitoring agents as network beacons at the network edge thanks to the flexibility of SDN. This kind of aggregated data fusion via edge are not utilized in previous works on the controller-switch assignment problem. Their knowledge about the topology is simply based on the centralized-view of distributed SDN controller. Additionally, we develop an online controller-switch reassignment mechanism considering changes in the network such as switch and controller failures, due to the insufficiency of zero-day assignment. Online reassignment provides robustness against future failures and more efficient utilization of the new topology after changes.

According to the experimental results, RAFRES operates successfully against network failures by the help of the load-aware ILP model. ILP model has achieved a high performance during the experiments. In addition, monitoring agents carry the performance of RAFRES to a higher level. After deploying monitoring agents as network beacons, proposed ILP model has started to achieve much better results than SA and random assignment. RAFRES has also two main advantages in addition to its key contributions:

- RAFRES uses a remote optimization engine to calculate controller-switch assignment for all algorithms that are adopted. Therefore, during the calculation process of the assignment, the server that hosts controller instance does not take a heavy load and is able to continue to respond to switches without a degradation of its performance.

- In RAFRES, network beacons are activated by a software script to send a heartbeat to controller application via REST API. Therefore, there is no need for special devices to deploy as monitoring agents into the network. Any device that supports REST API can be used as a monitoring agent at the network edge.

The first direction is to extend our framework is to operate not only reactively but also proactively. This step will achieve a more robust software-defined network against network events such as failures and security attacks that are caused by faulty users. The proactive mechanism is able to predict the events and pre-arrange the networks to prevent them. The proactive mechanism decreases the number of events that are handled by the reactive mechanism and this can increase the utilization of the network due to the decreasing number of network incidents. In our thesis, the framework runs only reactively, therefore, it intervenes to the architecture just after the network changes or incidents occur.

It is also worth investigating to develop new assignment schemes using different optimization techniques such as additional meta-heuristics. In that regard, RAFRES can easily be extended via its modular architecture and API based integration. This research direction also includes augmenting our optimization model with additional parameters and constraints to have a more intricate system model.

Another interesting addition would be to handle the controller-switch assignment in case of an in-band controller. In this thesis, we just assume that the control traffic is sent using out-of-band management. For out-of-band management, every controller instance is connected to each network node separately, and failure in data-paths does not affect the control traffic. This is possible to achieve for small-sized networks and it provides robustness to the network. On the other hand, it is really difficult to establish in large-sized networks (e.g. with 1000 switches) and using in-band controller is more reasonable for real networks. Therefore, we could extend RAFRES to achieve the controller-switch assignment in case of in-band controller by considering not only switch loads but also failure probabilities of data-paths. The placement of controllers also becomes important when in-band management strategy is used. Hence, a con-

troller placement strategy must also be implemented for zero-day controller-switch assignment.

Lastly, we have devised a novel strategy by using monitoring agents for the controller-switch assignment. For that purpose, we have used RTT values that are provided by the agents. This heartbeat information can be extended with additional sensory data and used for many other problems in the SDN architecture. We believe that using monitoring agents in an edge computing setting is a promising new field of research for SDN resilience.

REFERENCES

1. Nunes, B. A. A., 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. Kalkan, K., 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. McKeown, N., A. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM*, Vol. 38, No. 2, pp. 175–191, Apr. 2008.

4. Benamrane, F., M. Ben Mamoun and R. Benaini, "Performances of OpenFlow-Based Software-Defined Networks: An Overview", *Journal of Networks*, Vol. 10, No. 6, pp. 329–337, Jun. 2015.

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

6. Shalimov, A., D. Zuikov, D. Zimarina, V. Pahkov and R. Smeliansky, "Advanced Study of SDN/OpenFlow Controllers", *Proceedings of the 9th Central Eastern European Software Engineering Conference in Russia*, 2013.

7. Tootoonchain, A., 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.

8. Heller, B., R. Sherwood and N. McKeown, "The Controller Placement Problem",

Proceedings of First Workshop on Hot Topics in Software Defined Networks, 2012.

9. Killi, B. P. R. 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.

10. Hu, T., 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.

11. Bannour, F., S. Souihi and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges", *IEEE Communications Surveys & Tutorials*, Vol. 20, No. 1, pp. 333–353, 2018.

12. Whitmore, A., A. Agarwal and L. D. Xu, "The Internet of Things—A survey of topics and trends", *Springer Science+Business Media*, 2014.

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

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

15. Koponen, T., M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama and S. Shenker, "Onix: a distributed control platform for large-scale production networks", *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pp. 351–364, Oct. 2010.

16. Bondkovskii, A., J. Keeney, S. van der Meer and S. Weber, "Qualitative comparison of open-source SDN controllers", *IEEE/IFIP Network Operations and Management*

Symposium(NOMS), Apr. 2016.

17. Berde, P., M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS", *Proceedings of the Third Workshop on Hot topics in Software Defined Networking*, Aug. 2014.

18. Phemius, K., M. Bouet and J. Leguay, "DISCO: Distributed multi-domain SDN controllers", *IEEE Network Operations and Management Symposium (NOMS)*, May 2014.

19. Soheil and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications", *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 19–24, Aug. 2012.

20. Jain, S., A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart and A. Vahdat, "B4: experience with a globally-deployed software defined wan", *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, Vol. 38, pp. 3–14, Aug. 2013.

21. Gude, N., T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, "NOX: Towards an Operating System for Networks", *ACM SIGCOMM Computer Communication Review*, Vol. 38, pp. 105–110, Jul. 2008.

22. Project Floodlight, *Floodlight SDN Controller*, 2011, `http://www.projectfloodlight.org`, accessed in May 2019.

23. Hazelcast Inc, *Hazelcast Project*, 2009, `http://hazelcast.org`, accessed in May 2019.

24. Hunt, P., M. Konar, F. P. Junqueira and B. Reed, "ZooKeeper: wait-free coordination for internet-scale systems", *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, Jun. 2010.

25. Hu, Y., W. Wendong, X. Gong, X. Que and C. Shiduan, "Reliability-aware Controller Placement for Software Defined Networks", *IFIP/IEEE International Symposium on Integrated Network Management*, 2013.

26. Bo, H., W. Youke, W. Chuan'an and W. Ying, "The controller placement problem for software-defined networks", *2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2435–2439, Oct. 2016.

27. Wang, T., F. Liu and H. Xu, "An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data Center Networks", *IEEE/ACM Transactions on Networking*, Vol. 25, pp. 2788–2801, Oct. 2017.

28. 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 (GLOBECOM)*, Dec. 2014.

29. Manikas, T. W. and J. T. Cain, "Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem", *Computer Science & Eng. Research*, Vol. 1, 5 1996.

30. Wei, Y. 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.

31. Gillani, F., 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.

32. Chu, C.-Y., K. Xi, M. Luo and H. J. Chao, "Congestion-Aware Single Link Failure Recovery in Hybrid SDN Networks", *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1086–1094, May 2015.

33. Savas, S. S., 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.

34. Lin, Y.-D., H.-Y. Teng, C.-R. Hsu, C.-C. Liao and Y.-C. Lai, "Fast Failover and Switchover for Link Failures and Congestion in Software Defined Networks", *IEEE International Conference on Communications (ICC)*, May 2016.

35. van Laarhoven, P. and E. Aarts, "Simulated annealing", *Simulated Annealing: Theory and Applications. Mathematics and Its Applications*, chap. 2, pp. 7–8, Springer, Dordrecht, 1987.

36. Lantz, B., 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.

37. Gurobi Optimization Inc., *Gurobi Optimizer*, 2008, `http://www.gurobi.com`, accessed in May 2019.

38. Universita' degli Studi di Napoli "Federico II", *D-ITG (Distributed Internet Traffic Generator)*, 2004, `http://traffic.comics.unina.it/software/ITG`, accessed in May 2019.

39. He, Q. A., *Analysing the Characteristics of VoIP Traffic*, Master's Thesis, University of Saskatchewan, Saskatoon, 7 2007.

40. Manzano, M., 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.

41. Li, J., 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.