Received: 30 May 2018 | Revised: 27 July 2018 | Accepted: 28 July 2018
DOI: 10.1002/cpe.4953

WILEY

SPECIAL ISSUE PAPER

Software defined service function chaining with failure consideration for fog computing

M.M. Tajiki[1,3] | Mohammad Shojafar[2] | Behzad Akbari[3] | Stefano Salsano[1] | Mauro Conti[2]

[1]Department of Engineering, University of Rome Tor Vergata, Rome, Italy
[2]Department of Mathematics, University of Padua, Padua, Italy
[3]Department of Computer and Electrical Engineering, University of Tarbiat Modares, Tehran, Iran

Correspondence
Mohammad Shojafar, Department of Mathematics, University of Padua, 35131 Padua, Italy.
Email: mohammad.shojafar@unipd.it; mohammad.shojafar@uniroma1.it

Funding information
Horizon 2020 EU project SUPERFLUIDITY, Grant/Award Number: 671566; Cisco University Research Program Fund, Grant/Award Number: 2017-166478 (3696); Silicon Valley Community Foundation; Intel; University of Padua, Italy; Horizon 2020 EU Project TagItSmart!, Grant/Award Number: 688061; EU-India REACH Project, Grant/Award Number: 2017-166478 (3696)

Summary

Middleboxes have become a vital part of modern networks by providing services such as load balancing, optimization of network traffic, and content filtering. A sequence of middleboxes comprising a logical service is called a *Service Function Chain (SFC)*. In this context, the main issues are to maintain an acceptable level of network path survivability and a fair allocation of the resource between different demands in the event of faults or failures. In this paper, we focus on the problems of traffic engineering, failure recovery, fault prevention, and SFC with reliability and energy consumption constraints in Software Defined Networks (SDN). These types of deployments use Fog computing as an emerging paradigm to manage the distributed small-size traffic flows passing through the SDN-enabled switches (possibly Fog Nodes). The main aim of this integration is to support service delivery in real-time failure recovery in an SFC context. First, we present an architecture for Failure Recovery called FRFP; this is a multi-tier structure in which the real-time traffic flows pass through SDN-enabled switches to jointly decrease the network side-effects of flow rerouting and energy consumption of the Fog Nodes. We then mathematically formulate an optimization problem called the Optimal Fast Failure Recovery algorithm (OFFR) and propose a near-optimal heuristic called Heuristic HFFR to solve the corresponding problem in polynomial time. In this way, the reliability of the selected paths are optimized, while the network congestion is minimized.

KEYWORDS

failure recovery, fog computing (FC), network function virtualization (NFV), resource reallocation, service function chaining (SFC), software defined network (SDN)

1 | INTRODUCTION

Network Function Virtualization (NFV) paradigm decouples network functions (NFs) from dedicated hardware equipment and provides more flexibility to the owner of an IT infrastructure. NFV consists in a set of different services running on generic hardware (eg, IDS, proxy, deep packet inspection, and firewall) that are chained using the so called Service Function Chaining (SFC) concept. This concept can be applied to industrial environments; in particular, NFV is used in industrial network design in order to reduce exposure to risks, reduce CAPEX/OPEX costs, and minimize future performance issues by basing the infrastructure upon commodity servers and switches.[1] NFV and SFCs offer a great flexibility in the chaining and placement of NFs. The European Telecommunications Standards Institute (ETSI) is driving the standardization of NFV. Software Defined Networking (SDN) complements NFV and is concerned with the decoupling of control plane functionality from packet forwarding devices, allowing a controller to flexibly configure the networking operations in the infrastructure. NFV and SDN require attention to avoid cascading threats as well as controller protections, especially for the software applications across the SDN switches that interoperate with the server virtualization and virtual machines (VMs).[2] In several scenarios, performance guarantees related to throughput need to be provided. Therefore proper allocation algorithms need to be defined, eg, for allocating network paths and assigning Virtual Network Functions (VNFs) to physical nodes.

In industrial SDN-based NFV environments, centralized controllers have the ability to reprogram the data plane in real time in such a way as to provide high availability and recover the failure that may exist in the SFC traffic flows. Fog computing at the edge can rapidly compute and organize small instance processes locally and move relevant on-demand processing data flows from the local geographical location to core platforms, such as Amazon Web Services.[3] Moreover, nodes that are located geographically near to the users are used to address small-size flows with limited response time SLAs and deliver high user Quality of Experiences (QoEs), like the ones proposed in the works of Peng et al[4] and Liang et al.[5] As a consequence, Fog Nodes are connected with virtualized SDN-enabled switches, which run atop servers or data centers at the edge of the access network. These switches can easily handle such flows within the low latency parameter. However, the way to recover failures in such networks remains nontrivial since it is an NP problem to perform a real-time routing and SFC in case of failure occurrence in the network considering limited processing and link capacity. To cope with these issues, several questions arise, ie, How to route and assign VNFs to flows in case of failure in SDN-based networks? How can we guarantee the elasticity of such solutions in real scenarios? Can we assure that the presented algorithm could swiftly update itself for the dynamic traffic? Recently, several works have been presented in the literature in order to address such issues. Most of these works address the failure-handling process, from failure detection and prevention to failure recovery, but, to the best of our knowledge, none of them addresses the problem of failure recovery and SFC in SDN-based Fog computing environment.

1.1 | Contributions

Motivated by the aforementioned considerations, we addressed the problem of SFC in SDN-based Fog computing environment with reliability considerations. To this end, we proposed a rerouting architecture based on the SDN concept with focus on the failure aspect of network devices. Our main contributions are summarized as follows.

- We proposed a novel failure recovery scheme, which provides service function chaining in SDN-based networks.
- We mathematically formulated the corresponding problem in form of integer linear programming (ILP), which could be easily solved with common ILP solvers.
- In order to make the solution scalable for large-scale networks, we proposed a fast heuristic algorithm to solve the proposed optimization problem. The proposed algorithm is applicable for real-world networks.
- The performances of the proposed solutions were measured through several metrics, ie, (i) computational complexity, (ii) average probability of failure in the selected paths, (iii) link utilization, and (iv) server utilization.

1.2 | Organization

The remainder of this paper is organized as follows. Section 2 presents a holistic literature review. Section 3 presents the problem definition, related assumptions, and overviews, along with the considered architecture and its main components, while Section 4 presents the system model. The problem of jointly managing the network side-effect of rerouting flows triggered by fatigue processes is formulated as an ILP in Section 5. Section 6 details the proposed heuristic algorithm, HFFR, and its computational complexity. The considered scenarios and the setting of the input parameters are detailed in Section 7. The obtained results are detailed in Section 8. Finally, Section 9 concludes this paper with some final remarks, and outlines open research problems.

2 | RELATED WORK

In the following, we briefly discuss the main literature on NFVs/SDNs and service function changing related to our work. We categorized the existing solution into two parts, in which, at first, we address the SFC algorithms,[6-23] and then in the second part, we describe the solutions and handle the failures and faults in SDNs/NFVs.[24-35]

2.1 | SFC solutions in SDNs/NFVs

Consequently, numerous works focus on providing SFC in SDNs. An SFC taxonomy that considers architecture and performance dimensions as the basis for the subsequent state-of-the-art analysis was introduced in the work of Medhat et al.[6] Bhamare et al[7] studied the problem of deploying SFCs over NFV architecture. Specifically, they investigate the VNF placement problem for the optimal SFC formation across geographically distributed clouds. Moreover, they set up the problem of minimizing inter-cloud traffic and response time in a multi-cloud scenario as an ILP optimization problem, along with some other constraints such as total deployment costs and SLAs.

Moreover, in the work of Reddy et al,[8] an optimization model based on the concept of Γ-robustness was proposed. They focus on dealing with the uncertainty of the traffic demand. Zhang et al[9] proposed a heuristic algorithm to find a solution for service chaining. It employs two-step flow selection when an SFC with multiple network functions needs to scale out. Furthermore, AbdelSalam et al[10] introduced a VNF chaining, which is implemented through segment routing in a Linux-based infrastructure. To this end, they exploit an IPv6 Segment Routing (SRv6) network

programming model to support SFC in an NFV scenario. Kulkarni et al[11] proposed a scheme, which provides flexibility, ease of configuration, and adaptability to relocate the service functions with a minimal control plane overhead. Besides, Wollschlaeger et al[12] proposed some time-sensitive networking solutions that applied SDN in such a way to highly configure the real changes in such industrial systems to cover safety, and reduce demand latencies. After deep analyzing the method compared to our solution, we figure out that it does not support Fog-supported SFC and failure SDN-enabled switches.

Besides, Bari et al[14] used ILP to determine the required number and placement of VNFs that optimize network CAPEX/OPEX costs without violating SLAs. In the work of Even et al,[15] an approximation algorithm for path computation and function placement in SDNs was proposed. Similar to the work of Bari et al,[14] they proposed a randomized approximation algorithm for path computation and function placement. In the work of Ghaznavi et al,[16] an optimization model to deploy a chain in a distributed manner was developed. Their proposed model abstracts heterogeneity of VNF instances and allows them to deploy a chain with custom throughput without worrying about individual VNFsâĂŹ throughput. Rost and Schmid[17] considered the offline batch embedding of multiple service chains. They consider the objectives of maximizing the profit by embedding an optimal subset of requests or minimizing the costs when all requests need to be embedded. Jiang et al[19] solved a joint route selection and VM placement problem. They design an offline algorithm to solve a static VM placement problem and an online solution traffic routing. They expand the technique of Markov approximation to achieve their objectives.

Recently, Nayak et al[20] presented a scheduling and routing solution in SDN/NFV time-triggered flows. In detail, they approximate the optimal solution over a corresponding static scheduling problem and solve it using ILP. As in our approach, hard constraints on the overall execution times are considered by Nayak et al.[20] However, we point out that, unlike our approach, (i) the focus of the work of Nayak et al[20] is on the traffic routing and scheduling between SDN-enabled switches per time-flow, so that the resulting flow scheduler does not support, by design, failure, and fault tolerance per link and switch of data time-flow; (ii) the joint flow and computing rate mapping afforded in the work of Nayak et al[20] is, by design, static; (iii) the scheduler in the work of Nayak et al[20] does not perform real-time reconfiguration rerouting and real-time traffic hosted by the serving controller; (iv) the work of Nayak et al[20] does not consider SFC and rerouting; and (v) the scheduler in the work of Nayak et al[20] does not enforce per-flow QoS guarantees on the limited time minimum energy and/or the minimum side-effect. Although the aforementioned solutions are interesting, however, none of them considers the problem of service chaining with respect to the energy consumption of the VMs.

2.2 | Failure recovery and fault-aware solutions in SDNs/NFVs

The available literature ranges from the joint problem of fault-aware distributing and routing the traffic flows/content in SDNs/NFVs infrastructure[24,25] to the problem of fault detection and recovery solutions in SDNs/NFVs.[27,28] In detail, in the work of Kreutz et al,[25] the authors analyzed the fault tolerance over SDN. They present a discussion about fault tolerance and failure happening in the OpenFlow (OF) protocol that is applied in SDNs. Specifically, they propose a link/node failure detection and failure recovery method in the data plane that can be controlled through the controller. However, they do not present any discussion about the application plane side-effect and do not cover the SFC fault-awareness.

In the work of Sharma et al,[33] they presented a controller-based fault recovery solution that covers path-failure detection and preconfigured backup paths. However, we point out that, unlike our approach, (i) the focus of the work of Sharma et al[33] is on presenting the network configuration in order to manage the traffic flows, which is not an effective solution, by design, in real scenarios;(ii) the presented fault prevention method in the work of Sharma et al[33] does not support the SFC over the SDN-enabled switches. van Adrichem et al[34] proposed a solution to quickly detect link failures in order to increase the fault tolerance by combining the flow retrieval, which is achieved through analyzing the protection switching times and using a fast protection method. Interestingly, this paper supports the fault minimization over the links and addresses the end-to-end fault tolerance method per flow, but not radically. Overall, the contribution does not afford, by design, jointly the QoSs in the node and link of SDN and does not support the SFC fault minimization, both of which are adopted in this paper.

Besides, Turchetti and Duarte[29] presented NFV-FD, a fault-tolerant unreliable failure detector that is adapted based on information (it includes communication links states and the flow characteristics) obtained from an SDN. This paper presents flavor of novelties, but it fails to address the SFC traffic flows. Moreover, our solution utilizes a network equipment fault-aware technique that spreads out the fault tolerance process all over the components running in the SDN. In the work of Reitblatt et al,[30] the authors applied novel rule-based programming language presented in the work of Guha et al[32] to talk between the controller and the data plane to manage the adopted in-network fast-fail over mechanisms of incoming traffic flows in FatTire programs. Although this method is an interesting step toward to the fault-aware SDN traffic flow policy management, it suffers from fault recovery and fault prevention that matter in our solution.

3 | THE PROPOSED ARCHITECTURE

In this section, we first define the problem and assumptions, and then we provide a detailed discussion of the proposed architecture and its components.

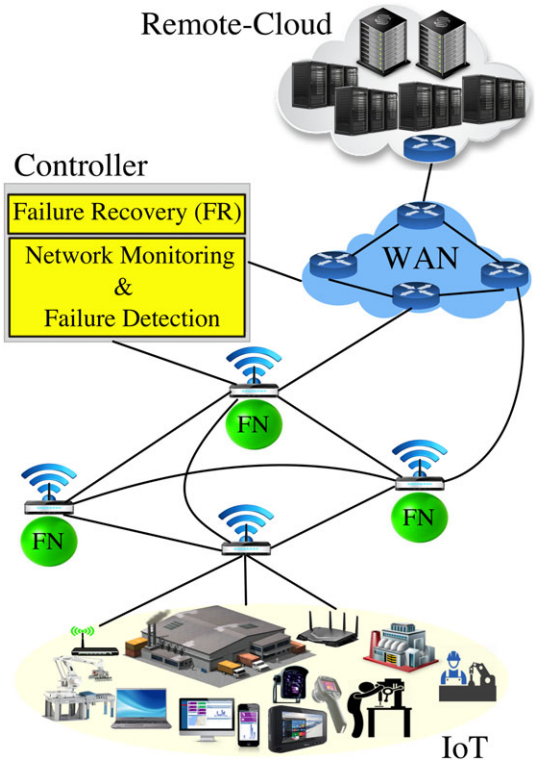

FIGURE 1 The proposed Architecture: this architecture is based on the SDN principles and exploits server virtualization to run several VNFs on a single server. Each Fog Node (FN) contains one or more physical server; IoT= Internet of Thing

3.1 | Problem definition and assumptions

We consider a set of SDN-enabled switches with different failure probabilities that change during time slots. The architecture is defined considering discrete time slots. A logically centralized controller is connected to the switches to fetch the network information and configure them. There are several Fog Nodes in the network that can host VNFs. The Fog Nodes are connected to the edge switches; a maximum of one Fog Node is connected to each switch. We refer to the (Fog Node, switch) pair as a node throughout this paper. For a given Fog Node, the processing load cannot exceed a predefined threshold. Each Fog Node has a given processing capacity and can host several types of VNFs. The set of supported VNFs on each Fog Node is given. Each type of VNF requires a different processing capacity to process a unit of data, and the processing time for a VNF on different Fog Nodes for equal flow rates is the same. It should be mentioned that we suppose some links fails during the time slots. A set of required VNFs are assigned to each flow, which traverses from the source to the destination switch (we refer this set as the *SFC requirement* of flows). The incoming traffic may dynamically change during various time slots.

We define two different problems, ie, (i) recovery of the network in case of failure of one or more nodes, in such a way that the total failure probability of the paths in the network is minimized and the QoS requirement of the flows is satisfied, and (ii) periodic reconfiguration of the network in order to optimize the probability of a failure in selected paths for the active flows.

3.2 | Reference architecture

In our architecture presented in Figure 1, Fog Nodes are interconnected with wired or wireless facilities. The IoT devices are scattered throughout the environment and are connected to the access points of the network. Focusing on the scalability of data processing, Fog Nodes could process the input data with low latency, whereas the Cloud suffers from limitations in this respect. On the other hand, the processing power of the Fog Nodes are limited in compare with the huge processing power of clouds. Hence, the IoT devices receive services on their demands in two ways, ie, (i) Fog Nodes with limited processing power and low latency, or (ii) cloud nodes with high processing power and relatively higher latency. In brief, our presented model can jointly handle Fog and Cloud to support big/small data processing, reduce the data transportation overhead and delay, and finally balance the computation between Fog Nodes and cloud Nodes based on the flows requirements (eg, computation-intensive tasks can be scheduled and executed on the Cloud side to avoid overloading of the processing capacity of the Fog Node). This will facilitate the generation of IoT applications and speed up the processing of large-scale data applications.

Controller Architecture: The architecture is composed of three layers and one centralized Controller (top left box in Figure 1) that manages the flows passing between these layers. The main tasks of the Controller are (i) gather information about the network infrastructure or traffic patterns

TABLE 1 Main Notation

Symbol	Definition	Type - Unit
	Input Parameters	
$\mathcal{N}$	Set of switches, $\lvert\mathcal{N}\rvert \triangleq N$	-
$\mathcal{F}$	Set of flows, $\lvert\mathcal{F}\rvert \triangleq F$	-
$\mathcal{X}$	Set of functions, $\lvert\mathcal{X}\rvert \triangleq X$	-
E	Number of links	Integer - [units]
$\mathcal{T}$	Total number of time slots	Integer - [units]
Ψ	Maximum number of required functions for each flow	Integer - [units]
$B_{(i,j)}$	Matrix of link bandwidth between i-th and j-th switches	Continues - [Mb/s]
μ	Maximum link/server utilization	Continues - [units]
MT	Maximum tolerable joint failure probability	Continues - [ms]
$C^f(t)$	Bandwidth requirement matrix for the f-th flow in time slot t	Continues - [Mb/s]
s^f	Vector of source switch for the f-th flow	Integer - [units]
d^f	Vector of destination switch for the f-th flow	Integer - [units]
FP_x	Required processing for the x-th function	Continuous - [units]
NC_i	Nodes processing capacity for the i-th node	Continuous - [units]
$FN_{(i,x)}$	Function x associated with i-th node	Binary - [units]
$R^f_x(t)$	Requested functions for the f-th flow in time slot t	Binary - [units]
K^f_ψ	Sequence of requested VNFs	Binary - [units]
$p_i(t)$	Failure probability for switch i in time slot t	Continuous - [units]
	Variables	
$\mathcal{P}_r(t)$	Fault probability for path ID r in time slot t	Continuous - [units]
$A^f_{(i,j)}(t)$	Network resource assignment matrix between i-th and j-th switches with the flow f in time slot t	Binary - [units]
$U^f_{(i,x)}(t)$	Used services for the i-th switch with the flow f that runs the function x in time slot t	Binary - [units]
$\mathcal{Z}^f(t)$	Index of selected path for flow f in time slot t	Integer - [units]
$AP^f_{(i,j)}$	rerouting matrix with nodes order	
$J^f_i(t)$	Path allocation vector i used for flow f in time slot t	Binary - [units]

and (ii) force the switches and the Fog Nodes to operate based on the taken decisions. The proposed Controller architecture is shown in Figure 1 and has the following components.

(i) Failure Recovery (FR): This component periodically assigns network and Fog Node resources to flows, in order to optimize the failure probabilities. In addition, if a switch undergoes a failure, this component re-assigns resources to the flows passing through the failed node in real time. To this end, FR considers two aspects, ie, the QoS requirement of the flows and the failure probability of the new assigned path. We formulate this component mathematically in Section 5 and provide a corresponding fast heuristic algorithm in Section 6. Three events activate FR, ie, (i) the arrival of a new flow, (ii) failure in one or more nodes, and (iii) the end of the timer interval (periodically). The input of this component in case of the arrival of a new flow is the specification of the flow (eg, source, destination, rate, and set of required VNFs) and the current state of the nodes (eg, utilization of links and Fog Nodes). In case of a failure, the input is the specification of the flows passing through the failed node(s) and the current state of the network. It should be mentioned that, if a node (switch or Fog Node) fails, the network topology and/or the list of supported VNFs (in each Fog Node) may change.

(ii) Network Monitoring and Failure Detection: This component continuously monitors the network traffic by gathering information from switches. Besides, this component is responsible for the detection of node failures or crashes.

4 | SYSTEM MODEL

In this section, we describe the notations used in this paper. Table 1 defines the symbols, presents their type and units, and provides a brief description of them.

We denote the network topology using matrix $B_{N\times N}$, where N is the number of SDN-enabled switches and $B_{(i,j)}$ is the capacity of the link from the switch i to the switch j. Considering F as the number of flows in the network, vectors s^f and d^f determine the source and destination of flows, respectively. The assignment of network resources (links) to the flows is denoted by matrix $A_{N\times N\times F}(t)$, eg, if $R^f_{(i,j)}(t_1) = 1$, then the flow f passes the link $i \rightarrow j$ in time slot t_1. In our formulation, the links and nodes are not allowed to be used twice in the routing of a flow (No loop exits).

Let X be the different VNFs. The variable $\Psi \leq X$ specifies the maximum number of VNFs a flow can request. The matrix $Req_{F\times X}$ shows the set of requested VNFs for each flow. Therefore, if the VNF x is requested for the flow f in time slot t, then $Req^f_x(t)$ is 1. The sequence of the required VNFs

for all the flows is expressed by matrix $K_{F\times\Psi}$, where K^f_ω specifies the ω^{th} required VNF for flow f. The flow rates in time slot t is specified using the matrix $C^f(t)$. The ith row of this matrix defines the traffic rate requirement of the ith flow. The maximum communication and processing delay that the flow can tolerate is specified by the vector T^f. The processing time of one unit of data over each VNF, eg, $TP_x = 3$ [ms] means that VNF x needs 3 [ms] to process one [unit] of data.

The vector FP_X expresses the required processing capacity of each VNF for a unit of flow rate, ie, the VNF x requires $FP_x \cdot C^f(t)$ processing capacity to process the flow f with rate $C^f(t)$. The processing capacities of each Fog Node is identified by the vector $NC_{1\times N}$. The matrix $FN_{N\times X}$ identifies the VNFs associated with each Fog Node, ie, $FN_{(i,x)}$ specifies whether VNF x is supported by Fog Node i or not. We consider a Fog Node connected to a switch. If no Fog Node is connected to switch i, then $\sum_{x=1}^{X} FN_{(i,x)} = 0$. The assignment of the Fog Nodes and VNFs to the flows in time slot t is denoted by the matrix $U^F_{N\times X}(t)$. If $U^f_{(i,x)}(t)$ is 1, then flow f receives service from VNF x on Fog Node i in time slot t. The variable μ states the maximum link and Fog Node utilization. The end-to-end failure probability of routing in the network for flow f should be less than a predefined threshold MT. The failure probability of switches is stated using $p_N(t)$, eg, $p_i(t_1)$ specifies the failure probability of switch i in time slot t_1.

5 | PROBLEM FORMULATION

In this section, we formulate the problem of path allocation and SFC with the goal of minimizing the total failure probability in the network. In this way, the failure probability of selected path for each flow is guaranteed to be less than a predefined threshold. Besides, the required service functions of each flow is guaranteed to be delivered with an arbitrary ordering. In the following, the problem is formulated.

5.1 | Routing and service function chaining (SFC)

By constraint 1, the assigned traffic to each pair of switches guaranteed to be less than the maximum link utilization

$$\sum_{f=1}^{F}\left(R^f_{(i,j)}(t) \cdot C^f(t)\right) \le \mu \cdot B_{(i,j)}, \forall i,j \in \mathcal{N}. \tag{1}$$

The constraint 2 prevents flows from crossing a node twice. On the other hand, constraint 3 enforces the flow conservation constraint. Formally speaking, a flow leaves its source only once, ie, a flow enters to the destination and does not leave it; ironically, the ingress and egress of each switch are roughly the same except the source and destination nodes

$$\sum_{j=1}^{N} R^f_{(i,j)}(t) \le 1, \quad \forall i \in \mathcal{N}, \quad \forall f \in \mathcal{F}, \tag{2}$$

$$\sum_{j=1}^{N} R^f_{(i,j)}(t) - \sum_{j=1}^{N} R^f_{(j,i)}(t) = \begin{cases} 1 & \text{if } i = s^f \\ -1 & \text{if } i = d^f \\ 0 & \text{Otherwise} \end{cases},$$

$$\forall f \in \mathcal{F}, \quad \forall i \in \mathcal{N}. \tag{3}$$

The constraint 4 expresses that the fth flow meets a valid xth NFV in each traversing switch/hop

$$\sum_{i=1}^{N} U^f_{(i,x)}(t) \ge Req^f_x(t), \quad \forall x \in \mathcal{X}, \quad \forall f \in \mathcal{F}. \tag{4}$$

Moreover, constraint 5 imposes the service delivery only on crossed nodes. Constraint 6 checks whether the requesting function is supported on the specified node

$$\sum_{i=1}^{N} R^f_{(i,j)}(t) \ge U^f_{(j,x)}(t), \quad \forall x \in \mathcal{X}, \quad \forall j \in \mathcal{N} - \{s^f\}, \quad \forall f \in \mathcal{F}, \tag{5}$$

$$U^f_{(i,x)}(t) \le FN_{(i,x)}, \quad \forall f \in \mathcal{F}, \quad \forall i \in \mathcal{N}, \quad \forall x \in \mathcal{X}. \tag{6}$$

It is important to recall that only one VNF function is assigned to fth flow. Such restriction is confirmed in Constraint 7. Inequality 7 enforces each node to ensure its capacity usage while serving various functions

$$\sum_{i=1}^{N} U^f_{(i,x)}(t) = 1, \quad \forall f \in \mathcal{F}, \ \forall x \in \mathcal{X}, \tag{7}$$

$$\sum_{f=1}^{F}\sum_{x=1}^{X}\left(U^f_{(i,x)}(t) \cdot FP_x \cdot C^f(t)\right) \le NC_i, \quad \forall i \in \mathcal{N}. \tag{8}$$

We add constraints 9 to 15 to address the SFC sequence ordering

$$RR^f_{(i,j)} \geq R^f_{(i,j)}, \quad \forall f \in \mathcal{F}, \ \forall i,j \in \mathcal{N}, \tag{9}$$

$$RR^f_{(i,j)} \leq N \cdot R^f_{(i,j)}, \quad \forall i \in \mathcal{N} - \{d^f\}, \quad j \in \mathcal{N}, \quad \forall f \in \mathcal{F}. \tag{10}$$

Since matrix RR and R presents the rerouting and routing matrices, respectively, constraint 9 declares that the value of the ordering matrix (ie, the flow traversing on each crossed link) should be higher or equal to the routing matrix. In addition, inequality 10 designates that the value of ordered rerouting matrix, for each node except the destination one, should be less than traversing of whole switches. Moreover, when the value of routing matrix is zero, no function traversed of link $i \rightarrow j$, the same value is assigned to the ordered rerouting matrix

$$RR^f_{(d^f,i)} = 0, \quad \forall f \in \mathcal{F}, \ \forall \mathcal{N} - \{d^f\}, \tag{11}$$

$$\sum_{j=1}^{N} AP^f_{(i,j)} = \sum_{j=1}^{N} AP^f_{(j,i)} + \sum_{j=1}^{N} A^f_{(j,i)}, \quad \forall f \in \mathcal{F}, \ \forall i \in \mathcal{N} - \{s^f, d^f\}. \tag{12}$$

In a nutshell, according to the constraint 11, the destination node does not have any egress link, so the value of ordering rerouting matrix for this node is zero. Constraint 12 enforces that, if a flow enters to a switch in its ath step, then it would leave that switch in the $(a+1)$th step; however, source and destination switches are exceptions

$$RR^f_{(d^f,d^f)} = \sum_{j=1}^{N} RR^f_{(j,d^f)} + \sum_{j=1}^{N} R^f_{(j,d^f)}, \ \forall f \in \mathcal{F}, \tag{13}$$

$$\sum_{j=1}^{N} AP^f_{(s^f,j)} = 1, \ \forall f \in \mathcal{F}. \tag{14}$$

Since the value of $RR^f_{(d^f,d^f)}$ should be the number of crossed switches, constraint 13 is considered. Focusing on the integrity of the ordering matrix, constraint 14 assures that flows leave the source switch

$$\left(1 - U^f_{\left(i,K^f_{Vf}\right)}\right) \cdot (2N-1) + \sum_{j=1}^{N} RR^f_{(i,j)} \geq \left(U^f_{\left(l,K^f_{Z_{Vf}}\right)} - 1\right) \cdot (2N-1) + \sum_{j=1}^{N} RR^f_{(l,j)}, \ \forall f \in \mathcal{F},$$
$$\forall V^f \in \left\{1, \ldots, \text{len}\,(K^f)\right\}, \ \forall Z_{Vf} \in \{1, \ldots, V^f - 1\}, \forall l, i \in \mathcal{N}. \tag{15}$$

Constraint 15 guarantees the sequence of VNF chaining. To this end, for each VNF, constraint 15 checks whether the VNFs with higher ordering (lower index in K^f) are delivered to the flow in one of the crossed servers or not. In this way, the constraint exploits (i) variable V^f and (ii) sets Z_{Vf}. Variable V^f states the index of each required VNF for flow f. The set Z_{Vf} contains all required VNFs with a higher order (lower index) than V^f. For example, if $K^f = [3\ 2\ 1\ 6]$, then $V^f \in \{1,2,3,4\}$. If we consider $V^f = 3$, then Z_{Vf} is a member of $\{1,2\}$. In constraint 15, If the server i hosts the VNF K_{Vf}, ie, $U^f_{(i,K_{Vf})} = 1$, then the left-hand side of 15 considers the step of the server i and it must be greater than the step of all servers I hosting a VNF with index less than the index of VNF K_{Vf} in K^f. Consider Z_{Vf} as the index of any VNF in K^f with an index less than VNF K^f_{Vf}, ie, flow f must pass VNF $K^f_{Z_{Vf}}$ before K^f_{Vf}.

If the server I hosts the VNF $K^f_{Z_{Vf}}$, ie, if $U^f_{(i,K^f_{Z_{Vf}})} = 1$, then the right-hand side of 15 considers the step of the server I and it must be greater than the step of all servers i hosting a VNF with index greater than the index of VNF $K^f_{Z_{Vf}}$ in K^f. Since the value of $\sum_{j=1}^{N} RR^f_{(i,j)}$ is always less than $(2n-1)$, if one of $U^f_{(i,K_{Vf})}$ or $U^f_{(l,K_{Zf})}$ is zero, then the equation is true.* As we mentioned before, the destination has a flow to itself with a step of at most $N+1$; therefore, in cases that both $U^f_{(i,K_{Vf})}$ and $U^f_{(l,K_{Zf})}$ are equal to one, the constraint is met *if and only if* the value of $\sum_{j=1}^{N} RR^f_{(i,j)}$ is greater than $\sum_{j=1}^{N} RR^f_{(l,j)}$. This means that a server, which delivers the lower index VNF, is crossed before servers that deliver higher index VNFs.

5.2 | Failure probability

To minimize the impact of failure, we formulate the failure probability of selected path for each flow. In this way, we calculate the survival probability of the path. After that, this value is exploited to compute the failure probability. Let $p_i(t)$ denote the failure probability for switch i in time slot t under independent failure assumptions.

*The value of $\sum_{j=1}^{N} RR^f_{(i,j)}$ and $\sum_{i=1}^{N} RR^f_{(i,j)}$ are always less than $2N-1$ because in the worst case, the flow crosses all switches which means that the value of $\sum_{j=1}^{N} RR^f_{(i,j)}$ is at most $(N-1)+N$. A column can have at most two elements and they can be at most N and $(N\text{-}1)$.

We generate an ID to differentiate between different paths. The paths with the same failure probabilities have the same ID. Focusing on the path selection, $Z^f(t)$ identifies the ID that is assigned to the path selected for flow f in time slot t. If the path, which is selected for flow f includes switch number i, then $2^{(i-1)}$ is added to $\mathcal{Z}^f$. Consequently, the paths, which contain the same set of switches, are considered to have the same ID. Indeed, similar ID paths lead to the $\mathcal{Z}^f(t)$ to differentiate between paths with different failure probability and those paths that have the same set of switches have similar failure probability. As an example, if flow f_1 passes throw switch i in time slot t but flow f_2 does not pass throw that switch in time slot t, then $Z^{f_1}(t) \neq Z^{f_2}(t)$.

Let $ID_i^f(t) \triangleq \sum_{j=1}^{N} R_{(i,j)}^f(t) \cdot 2^{(i-1)}$ denote the ID number sets for SDN-enabled switch i for flow f in time slot t. In another word, if flow f passes throw switch i in time slot t, then the $ID_i^f(t)$ will be $2^{(i-1)}$, otherwise it will be zero. Therefore, $Z^f(t)$ can be calculated as follows:

$$\mathcal{Z}^f(t) \triangleq \sum_{i=1}^{N} \left(\sum_{j=1}^{N} R_{(i,j)}^f(t) \cdot 2^{(i-1)} \right) + 2^{(d^f-1)}, \quad \forall f \in \mathcal{F}, \forall t \in \mathcal{T}. \tag{16}$$

According to Section 4, if the destination of flow f is d^f, then $R_{(d^f,j)}^f(t)$ will be 0. Hence, $ID_{d_f}^f(t)$ is always zero for destination switch. Because of this, $2^{(d^f-1)}$ is added to constraint 16 to include the impact of the failure probability of the destination switch on the selected path r for flow f. In brief, for a network with N switches, the value of $\mathcal{Z}^f(t)$ can be a number between 0 to 2^N

$$\sum_{r=1}^{2^N} J_r^f(t) = 1, \quad \forall f \in \mathcal{F}. \tag{17}$$

Variable $J_r^f(t)$ specifies whether $\mathcal{Z}^f(t)$ is r or not, eg, $J_r^f(t) = 1$ means $\mathcal{Z}^f(t) = r$. Note that there are several different paths that have equal ID value (ie, all paths that have same set of switches (with different ordering) have identical path ID). Equation 17 guarantees that, in each time slot, only one ID is assigned to flow f.

$$\sum_{r=1}^{2^N} \left(J_r^f(t) \times r \right) = \mathcal{Z}^f, \quad \forall f \in \mathcal{F}. \tag{18}$$

Since $Z^f(t)$ is the ID of the selected path, which is captured from matrix $R(t)$, constraint 18 checks the consistency of the formulation. Finally, constraint 19 states the condition under which end-to-end failure probability of the selected path is *lower* than a predefined threshold

$$\sum_{r=1}^{2^N} \left(J_r^f(t) \times \mathcal{P}_r(t) \right) \leq MT, \quad \forall f \in \mathcal{F}. \tag{19}$$

5.3 | Overall formulation

The proposed failure recovery problem which aims at minimizing the total probability of failure in the networks is formulated as follows:

$$\min \left[\sum_{f=1}^{\mathcal{F}} \sum_{r=1}^{2^N} \left(J_r^f(t) \times \mathcal{P}_r(t) \right) \right], \tag{20}$$

subject to

Routing and SFC	(1)-(15);
Failure.	(16)-(19);

6 | HEURISTIC FAST FAILURE RECOVERY ALGORITHM (HFFR)

The optimal solution, which is discussed in the previous section, is very computationally complex to be solved even for small-size networks. Therefore, we propose a heuristic algorithm, called HFFR, in Algorithm 1 to solve the corresponding optimization problem. In line 1 of the algorithm, it is checked whether the failure probability of the selected path is less than the predefined threshold. In the second line, if the flow has met all required services and the flow is the destination switch, then the algorithm is stopped. Lines 3 to 25 are executed if the flow receives all of the requested services, but it is currently in a switch, which is not the destination of the flow. If the flow still needs some services to receive, then the lines 26 to 64 are used.

Algorithm 1 HFFR

INPUT: CN, d, K, B, NC, FP, p, PR,CP

`CP:Chosen Path, CN:Current Node, PR:Path Survival probability`

OUTPUT: <CP, PR>

if PR≥ 1-MT or PR<0 **then return** <CP, -1>
else if K=∅ and CN=d **then return** <CP, PR>
else if K=∅ **then**
  if B(CN,d)≥C_f **then return** <Add(CP,d), PR×(1-p_d)>
  end if
  PCN= ∅; ▷ PCN: previously chosen nodes
  do ▷ greedy selection of next hub
    NH= ∅; ▷ NH: next hob
    MFP=1; ▷ MFP: minimum failure probability
    for $i \in \{\mathcal{N}-PCN\}$ **do**
      if $B_{(CN,i)} \geq C_f$ & MFP≥p_i **then**
        [NH, MFP]=[i, p_i];
      end if
    end for
    add NH to PCN
    B′=B;
    for $i \in \mathcal{N}$ **do** ▷ Remove incoming links to NH
      $B'_{(i,NH)}$=0;
    end for
    PR′=PR×(1-p_{NH});
    CP′=Add(CP,NH);
    [CP′, PR]=HFFR(NH, d, ∅, B′, NC, FP, p, PR′, CP′);
    if PR≠ -1 **then return** <CP′, PR′ >
    end if
  while NH≠ ∅
else
  while K≠ ∅ & FN(K(1)) & FP(K(1))≤NC(CN) **do**
    NC(CN)=NC(CN)−FP(K(1));
    K=Remove(K,K(1));
  end while
  if K= ∅ **then**
    [CP′, PR′]=HFFR(CN, d, ∅, B, NC, FP, p, PR, CP);
  end if
  PCN= ∅; ▷ PCN: previously chosen nodes
  HS=K(1); ▷ HS: request service with highest priority
  K=Remove(K,K(1));
  do ▷ greedy selection of next hub
    [NH,NH2]=[∅, ∅];
    [MFP,MFP2]=[1,1];
    for $i \in \{\mathcal{N}-PCN\}$ **do**
      if B(CN,NH)≥C_f **then**
        if $FN_{(i,HS)}$ ς FP(K(1))×C_f ≤NC(NH) ς MFP≥p_i **then**
          [NH, MFP]=[i, p_i];
        else if MFP2≥p_i **then**
          [NH2, MFP2]=[i, p_i];
        end if
      end if
    end for

```
49:        if NH≠ ∅ then
50:            NC'_NH=NC_NH−C_f×FP_HS;
51:        else
52:            NH=NH2;
53:        end if
54:        add NH to PCN
55:        B′=B;
56:        for i ∈ 𝒩 do
57:            B'_(i,NH)=0;
58:        end for
59:        PR′=PR×(1-p_NH);
60:        CP′=Add(CP,NH);
61:        [CP′, PR′]=HFFR(NH, d, K, B′, NC′, FP, p, PR′, CP′);
62:        if PR≠ -1 then return <CP′, PR′ >
63:        end if
64:    while NH≠ ∅
65: end if
66: return <CP, -1>
```

In lines 8 to 13, the next hub is selected based on the failure probability of the switches in a greedy manner. More in details, the algorithm removes all links that have the free capacity less than the required bandwidth of the flow and selects a switch, which directly connected to the current switch.[†] In line 15, the selected node is added to the set of previously chosen nodes to prevent the algorithm from selecting this node twice for the current state of the flow. In lines 16 to 19, the consumed resources by the current flow are reduced from the network resources and all incoming links to the selected node is removed to prevent from returning to the selected node. In line 20, the failure probability of the selected path is increased based on the selected path. The selected node is added to the selected path in line 21. In the next line, the algorithm recursively invokes HFFR and sends the NH (selected next hub) as CN (current node). If the algorithm could find a path from NH, which has a failure probability of less than the maximum tolerable failure probability, then in line 23, the algorithm is stopped. Otherwise, a new node will be selected as the next hub (in lines 8 to 13).

As mentioned earlier, the services are delivered to the flow via lines 26 to 64. In lines 27 to 30, all services that are supported in the CN and they are requested by the flow are delivered to the flow (if CN has enough processing capacity to serve the flow). If flow received all of the requested services, then HFFR is invoked to route the flow toward its destination. In lines 38 to 53, a node supports a requested service with the highest priority based on the failure probability of switches. In this way, only the nodes that are directly connected to the current node are considered. If there is no such a node, then a directly connected node with the minimum failure probability is selected (lines 44, 45, and 49 to 53). Lines 54 to 64 are the same as lines 15 to 25. Finally, in line 66, if no route with a failure probability less than the maximum tolerable failure probability is found, then the algorithm is stopped.

7 | SCENARIO DESCRIPTION

The following sections detail the pursued scenarios and methodology for the applied scenarios.

7.1 | Simulation setup

In this paper, we consider Abilene[36] as the network topology. We consider that the processing power of a Fog Node is appropriate to the input bandwidth and the capacity of all links are equal to 1 [Gb/s]. In the simulation, geometric distribution is used to generate the traffic flows. To test the failure recovery, we fail a link with the highest degree (ie, to preserve the availability and reliability in the network[37]) in each time slot (eg, at the $TS_k, k - 1$ nodes were failed). For the comparisons, we focus on the number of the flow violations happens in shortest path service chaining (SPSC) compared to our solutions.

7.2 | Scenarios

To investigate the impact of the different traffic patterns and network resources, we evaluate the performance of the proposed solutions over different traffic scenarios, which are presented in Table 2. We generate the traffic demands based on two main characteristics, ie, (i) flow size and

[†]Current switch is the switch where the flow is in the current state.

TABLE 2 Different scenarios

	Abilene Topology				Simple Topology			
Scenarios	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**
Avg. Flow Rate [Mb/s]	1	1	10	1	1	10	1	10
Avg. Number of Functions	2	3	3	4	2	2	4	4
Number of Flows	50	50	50	50	20	20	20	20

(ii) number of required VNFs. In order to evaluate the impact of the flow size, two different values for average flow rate (ie, 1 [Mb/s] and 10 [Mb/s]), the average number of functions per flow are ranged between 2, 3, and 4, and 50 number of flows are considered (see the four first scenarios, ie, S1, S2, S3, and S4, in Table 2). Finally, to investigate the impact of the optimum solution, OFFR, another network topology, which is named as *simple topology* with different values, are considered (see the remain four scenarios).

8 | SIMULATION RESULTS

In this section, we investigate the impact of the proposed algorithm on the failure probability and flows QoS.

8.1 | Failure probability

Figures 2A to 2D present the average and maximum failure probability of the network in different time slots. These figures compare the results of HFFR and SPSC in Abilene network topology. As can be seen, HFFR outperforms SPSC in all test cases. It should be mentioned that SPSC is unaware of the switches failure probability. Therefore, the failure probability of path allocation based on SPSC has oscillation during different time slots. On the other hand, since HFFR is a suboptimal solution, the optimality gap during different time slots may vary based on the different inputs. However, it guarantees the maximum failure probability to be less than a predefined threshold.

In our simulation, during the time slots, some links fail; therefore, in average, the failure probability in more in the higher time slots. Therefore, we compare HFFR with the optimal solution in various scenarios (ie, S5 to S8) that is presented in Figures 3A to 3D. Since the failures rate increases by increasing the time slots, the average and maximum failure rates rising; the interesting point is that HFFR gains an average and maximum failure probability near to the optimal solution.

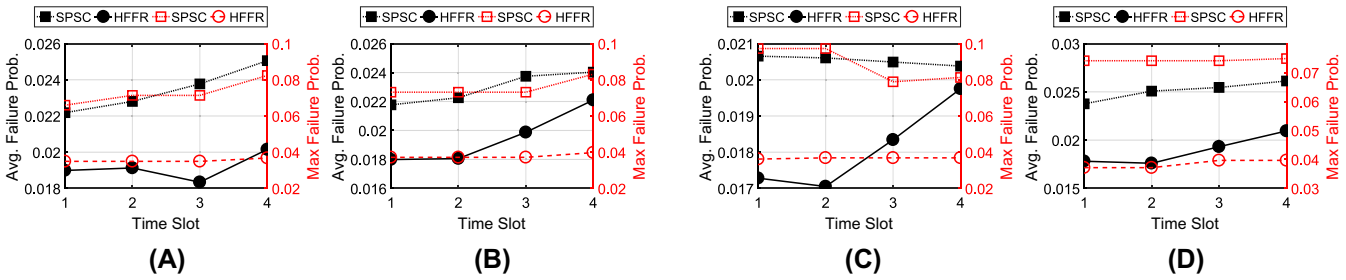

FIGURE 2 Comparison between shortest path service chaining (SPSC) and our proposed heuristic algorithm (HFFR): average and maximum failure probability using Abilene network topology. A, Scenario S1; B, Scenario S2; C, Scenario S3; D, Scenario S4

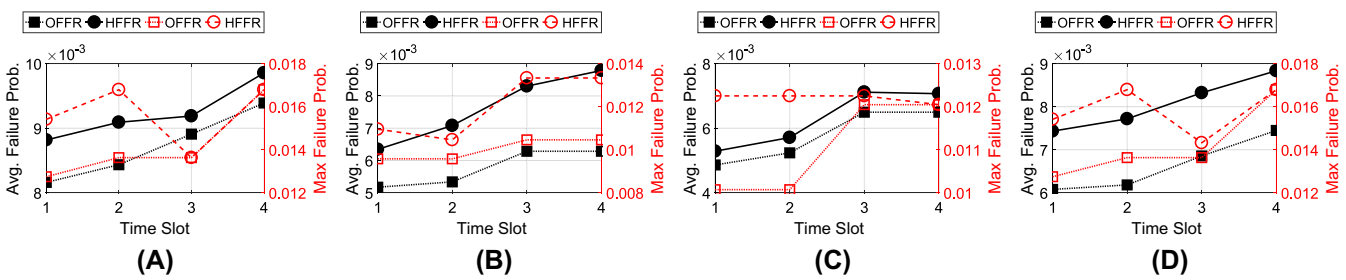

FIGURE 3 Comparison between Optimal Fast Failure Recovery algorithm (OFFR) and Heuristic Fast Failure Recovery algorithm (HFFR): average and maximum failure probability. A,Scenario S5; B, Scenario S6; C,Scenario S7; D, Scenario S8

TABLE 3 Number of allocated paths that violate the maximum tolerable failure probability in SPSC

Scenarios	TS1	TS2	TS3	TS4
S1	4	4	4	4
S2	2	2	2	1
S3	4	4	4	3
S4	4	6	5	5

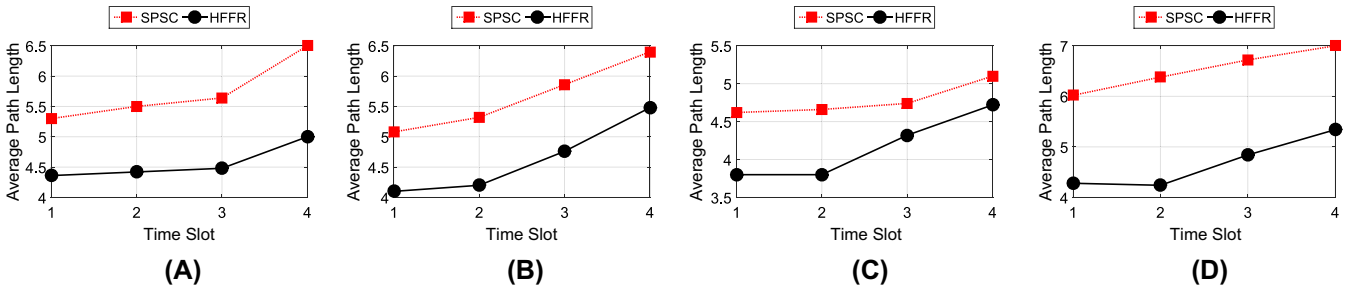

FIGURE 4 Comparison between shortest path service chaining (SPSC) and our proposed heuristic algorithm (HFFR): average path length using Abilene network topology. A, Scenario S1; B, Scenario S2; C, Scenario S3; D, Scenario S4

8.2 | Flow satisfaction (QoS)

In the following step, it is worth to note that neither OFFR nor HFFR do not violate any flows out of 50 flows in the Abilene topology while SPSC violates some of them in each time slot (TS). In Table 3, we list the number of flows that violate the maximum tolerable failure probability in SPSC in the Abilene topology in various scenarios. We have experienced that, (i) when the number of functions increases the average number of violated flows increases in SPSC while this value in our methods is zero, it means our solutions completely satisfy the SLA for the required flows compared to SPSC; (ii) SPSC has fluctuation when the link failures increases. It is due to the fact that SPSC can be considered as unaware-failure shortest path routing method, so it may select some SDN-enabled switches with higher failure probability due to decrease the path length and satisfy the SFC, and (iii) the existing variances of violated flows between the scenarios are related to the differences of sources and destinations, average flow rates, and the number of VNFs, which are implemented on FNs.

8.3 | Path Length

In Figure 4, we investigate the impact of the proposed algorithm on the average length of the selected paths. As can be seen, the average path length of SPSC is lower than HFFR in all test cases. This happens because HFFR tries to minimize the failure probability while SPSC aims to select hobs based on path length. Therefore, the average path length of SPSC is lower in comparing with HFFR.

9 | CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have proposed a network architecture based on SDN concept to handle the event of the failure in Fog-supported network nodes. We first mathematicàlly formulate the problem of service function chaining for the failure probability of paths, which covers fog nodes (FNs) and normal nodes (which is called OFFR). Furthermore, a suboptimal heuristic algorithm called HFFR is proposed to reduce the computational complexity of the solution and make it a real-time scheme. To evaluate these solutions, we compared these solutions with modified shortest path (SPSC) algorithm from three different metrics, ie, (i) failure probability, (ii) path length, and (iii) number of QoS violations. In the future, we plan to analyze the fast coverage lower bound of approximation algorithms that consider a failure of FNs/servers with the presence of the queuing delay of the switches for Fog-industry architecture.

ACKNOWLEDGMENTS

This work has received funding from the Horizon 2020 EU project SUPERFLUIDITY (671566). This work was also partially supported by the grant (2017-166478) (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation, and by the grant "Scalable IoT ManagementandKey security aspects in5Gsystems" from Intel. Moreover, this work was supported by the project "Adaptive Failure and QoS-aware Controller over Cloud Data Center to Preserve Robustness and Integrity of the Incoming Traffic" funded by the University of Padua, Italy. This work is partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896), and the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation.

ORCID

M.M. Tajiki http://orcid.org/0000-0002-7614-9528
Mohammad Shojafar http://orcid.org/0000-0003-3284-5086
Stefano Salsano http://orcid.org/0000-0003-3040-3559
Mauro Conti http://orcid.org/0000-0002-3612-1934

REFERENCES

1. Bari F, Chowdhury SR, Ahmed R, Boutaba R, Duarte OCMB. Orchestrating virtualized network functions. *IEEE Trans Netw Serv Manag*. 2016;13(4):725-739.
2. Amoroso EG. Software-defined networking and network function virtualization security. In: Vacca J, ed. *Computer and Information Security Handbook*. 3rd ed. Cambridge, MA: Elsevier; 2017:953-961.
3. Gargees R, Morago B, Pelapur R, et al. Incident-supporting visual cloud computing utilizing software-defined networking. *IEEE Tran Circuits Syst Video Technol*. 2017;27(1):182-197.
4. Peng M, Yan S, Zhang K, Wang C. Fog-computing-based radio access networks: issues and challenges. *IEEE Netw*. 2016;30(4):46-53.
5. Liang K, Zhao L, Chu X, Chen H-H. An integrated architecture for software defined and virtualized radio access networks with fog computing. *IEEE Netw*. 2017;31(1):80-87.
6. Medhat AM, Taleb T, Elmangoush A, Carella GA, Covaci S, Magedanz T. Service function chaining in next generation networks: state of the art and research challenges. *IEEE Commun Mag*. 2017;55(2):216-223.
7. Bhamare D, Samaka M, Erbad A, Jain R, Gupta L, Chan HA. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Comput Commun*. 2017;102:1-16.
8. Reddy VS, Baumgartner A, Bauschert T. Robust embedding of VNF/service chains with delay bounds. Paper presented at: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN); 2016; Palo Alto, CA.
9. Zhang B, Zhang P, Zhao Y, Wang Y, Luo X, Jin Y. Co-scaler: cooperative scaling of software-defined NFV service function chain. Paper presented at: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). IEEE; 2016; Palo Alto, CA.
10. AbdelSalam A, Clad F, Filsfils C, Salsano S, Siracusano G, Veltri L. Implementation of virtual network function chaining through segment routing in a linux-based NFV infrastructure. 2017. arXiv preprint arXiv:1702.05157.
11. Kulkarni S, Arumaithurai M, Ramakrishnan KK, Fu X. Neo-NSH: Towards scalable and efficient dynamic service function chaining of elastic network functions. Paper presented at: 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN); 2017; Paris, France.
12. Wollschlaeger M, Sauter T, Jasperneite J. The future of industrial communication: automation networks in the era of the internet of things and industry 4.0. *IEEE Ind Electron Mag*. 2017;11(1):17-27.
13. Tajiki MM, Salsano S, Shojafar M, Chiaraviglio L, Akbari B. Energy-efficient path allocation heuristic for service function chaining. Paper presented at: 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN 2018); 2018; Paris, France.
14. Bari MF, Chowdhury SR, Ahmed R, Boutaba R. On orchestrating virtual network functions. Paper presented at: 2015 11th International Conference on Network and Service Management (CNSM); 2015; Barcelona, Spain.
15. Even G, Rost M, Schmid S. An approximation algorithm for path computation and function placement in SDNS. In: Suomela J, ed. *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Cham, Switzerland: Springer; 2016:374-390.
16. Ghaznavi M, Shahriar N, Ahmed R, Boutaba R. Service function chaining simplified. 2016. arXiv preprint arXiv:1601.00751.
17. Rost M, Schmid S. Service chain and virtual network embeddings: approximations using randomized rounding. 2016. arXiv preprint arXiv:1604.02180.
18. Tajiki MM, Akbari B, Mokari N. QRTP: QoS-aware resource reallocation based on traffic prediction in software defined cloud networks. Paper presented at: 2016 8th International Symposium on Telecommunications (IST); 2016; Tehran, Iran.
19. Jiang JW, Lan T, Ha S, Chen M, Chiang M. Joint VM placement and routing for data center traffic engineering. In: 2012 Proceedings of the IEEE INFOCOM; 2012; Orlando, FL.
20. Nayak NG, Dürr F, Rothermel K. Incremental flow scheduling and routing in time-sensitive software-defined networks. *IEEE Trans Ind Inform*. 2018;14(5):2066-2075.
21. Liu X, Liu Y, Song H, Liu A. Big data orchestration as a service network. *IEEE Commun Mag*. 2017;55(9):94-101.
22. Yue X, Liu Y, Wang J, Song H, Cao H. Software defined radio and wireless acoustic networking for amateur drone surveillance. *IEEE Commun Mag*. 2018;56(4):90-97.
23. Tajiki MM, Salsano S, Shojafar M, Chiaraviglio L, Akbari B. Joint energy efficient and QoS-aware path allocation and VNF placement for service function chaining. 2017. arXiv preprint arXiv:1710.02611.
24. Sterbenz JPG, Hutchison D, Çetinkaya EK, et al. Resilience and survivability in communication networks: strategies, principles, and survey of disciplines. *Comput Netw*. 2010;54(8):1245-1265.
25. Kreutz D, Ramos F, Verissimo P. Towards secure and dependable software-defined networks. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13); 2013; Hong Kong, China.
26. Tajiki MM, Akbari B, Shojafar M, et al. CECT: computationally efficient congestion-avoidance and traffic engineering in software-defined cloud data centers. 2018. arXiv preprint arXiv:1802.07840.
27. Vilchez JMS, Yahia IGB, Crespi N. Self-healing mechanisms for software defined networks. Paper presented at: 8th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2014); 2014; Brno, Czech Republic.
28. Fonseca P, Mota E. A survey on fault management in software-defined networks. *IEEE Commun Surv Tutor*. 2017.
29. Turchetti RC, Duarte EP. Implementation of failure detector based on network function virtualization. Paper presented at: 2015 IEEE International Conference on Dependable Systems and Networks Workshops (DSN-W '15); 2015; Washington, DC.

30. Reitblatt M, Canini M, Guha A, Foster N. FatTire: Declarative fault tolerance for software-defined networks. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13); 2013; Hong Kong, China.
31. Khoshbakht M, Tajiki MM, Akbari B. SDTE: Software defined traffic engineering for improving data center network utilization. *Int J Inf Commun Technol Res*. 2016;8(1):15-24.
32. Guha A, Reitblatt M, Foster N. Machine-verified network controllers. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation; 2013; Seattle, WA.
33. Sharma S, Staessens D, Colle D, Pickavet M, Demeester P. OpenFlow: meeting carrier-grade recovery requirements. *Comput Commun*. 2013;36(6):656-665.
34. van Adrichem NLM, van Asten BJ, Kuipers FA. Fast recovery in software-defined networks. In: Proceedings of the 2014 Third European Workshop on Software Defined Networks (EWSDN'14); 2014; Budapest, Hungary.
35. Tajiki MM, Akbari B, Mokari N. Optimal QoS-aware network reconfiguration in software defined cloud data centers. *Comput Netw*. 2017;120:71-86.
36. Abilene Network. 2017. https://uit.stanford.edu/service/network/internet2/abilene.
37. Xiang M, Liu W, Bai Q, Al-Anbuky A, Wu J, Sathiaseelan A. NTaaS: network trustworthiness as a service. Paper presented at: 2017 27th International Telecommunication Networks and Applications Conference (ITNAC); 2017; Melbourne, Australia.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of the article.

How to cite this article: Tajiki MM, Shojafar M, Akbari B, Salsano S, Conti M. Software defined service function chaining with failure consideration for fog computing. *Concurrency Computat Pract Exper*. 2018;e4953. https://doi.org/10.1002/cpe.4953