TESI DI DOTTORATO

UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

DIPARTIMENTO DI INGEGNERIA ELETTRONICA
E DELLE TELECOMUNICAZIONI

DOTTORATO DI RICERCA IN
INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

RESOURCE SCHEDULING APPROACHES FOR PERFORMANCE OPTIMIZATION IN IOT-FOG NETWORKS

SAEED JAVANMARDI

Il Coordinatore del Corso di Dottorato	Il Tutore
Ch.mo Prof. Daniele RICCIO	Ch.mo Prof. Antonio PESCAPE

A. A. 2018–2022

“Hurray
Evviva evviva
Hurray.”

Acknowledgments

I'd like to express my heartfelt appreciation to Dr. Valerio Persico for his constructive advice and assistance.

Saeed Javanmardi

Contents

List of Figures

Introduction

The Internet of Things (IoT) has emerged as one of the most notable technologies in recent years for facilitating new interactions between matters and humans in order to improve the quality of life. Because of the rapid growth of IoT, the fog computing paradigm is emerging as an appealing method for processing data from IoT networks. IoT networks run on intermediate computing nodes within the fog, as well as real servers in cloud data centers within the fog environment.

Fog computing, like IoT and SDN, is attracting a lot of attention. Between the cloud data center and user, fog computing adds more fog nodes. user data may be cached on the fog nodes, and cloud servers may offload duties to these fog nodes. For numerous reasons, fog computing can be regarded a viable alternative for IoT implementation. First, because the Internet of Things creates vast amounts of sensor data, transferring it all to the cloud is impractical. The fog devices can preprocess or aggregate sensor data before transmitting it to the cloud because they are significantly closer to the IoT devices. This conserves network bandwidth in the upstream direction. Second, because many IoT services require immediate response, the cloud is unsuitable for such applications due to substantial traffic delay. Some lightweight applications can be moved to neighboring fog nodes in this circumstance, bringing computational resources closer to IoT devices. This cuts down on processing time. Because both SDN and fog nodes are relatively powerful nodes in a typical IoT deployment, they are frequently coupled, which is a great approach to combine SDN with fog computing features.

In IoT-Fog networks, the two most important metrics are security and efficiency. The key factor determining IoT-Fog network performance is resource management. Whereas application scheduling is important in fog computing for managing resources, application scheduling is the ability to map applications to the appropriate resources in IoT-Fog networks. The application is the user request that must be completed as soon as possible. Because fog comput-

ing uses heterogeneous and distributed resources, application scheduling becomes more complicated. To achieve an ideal solution, application scheduling is an NP-hard problem that requires the use of effective application scheduling strategies.

The ability of software-defined networking (SDN) to manage network flows automatically and dynamically is impressive. Furthermore, SDN switches and SDN controllers are typically powerful devices that can serve as fog nodes at the same time. As a result, SDN appears to be a viable option for detecting attacks in IoT-Fog networks. Accordingly, the combination of SDN and IoT-Fog networks has drawn a lot of attention and has become very popular. Network administrators have a significant problem managing such a large number of connections. In addition, IoT objects generally have limited resources. Both are computationally intensive security approaches that you cannot use directly. Therefore, IoT devices must be protected with the support of the network infrastructure. Traditional networks are no longer suitable for IoT under such conditions, while SDN allows owners to automatically manage the entire network in a flexible and dynamic way by providing many new features such as network planning capability, centralized control, etc. Because of these benefits, SDN has the potential to become the IoT's future foundation.

Though employing SDN to construct IoT-Fog networks appears promising, security concerns are unavoidable. Furthermore, because fog and SDN nodes are frequently integrated, attackers may exploit vulnerabilities in fog nodes to compromise the SDN switches/controller they manage. As a result, security methods are required to better monitor and strengthen the security of SDN infrastructure in IoT-Fog scenarios. Moreover, though the major goal of the IoT-Fog network is to provide high performance for all applications, security measures must be addressed as part of the IoT-Fog architecture to ensure the Confidentiality, Integrity, and Authentication of all sorts of data.

For application scheduling, metaheuristic-based techniques have been shown to achieve near-optimal solutions in a reasonable amount of time. We present performance optimization methods for IoT-Fog environments based on three popular metaheuristic techniques in this thesis: Particle Swarm Optimization (PSO), Multi Objective Particle Swarm Optimization (MOPSO), and Nondominated Sorting Genetic Algorithm (NSGA). The creation of a single objective optimization that considers efficiency metrics is the topic of the first portion of this thesis. In this part, we offer a new application scheduling technique in which a fuzzy based PSO algorithm allocates applications to fog resources while striking a balance between application computing requirements

and fog resource attributes in its fitness function. Following that, we present a multi-objective optimization approach that takes security and efficiency into account by employing a MOPSO algorithm. Finally, we use an NSGA-III algorithm to device a framework that consists of a security access control and workflow scheduling mechanism that acts as a firewall to protect scheduling services.

The outline of the thesis is the following:

Chapter 1 lays the groundwork for this thesis, which employs a single objective optimization approach. In this chapter, we presented FPFTS (Fuzzy PSO Task Scheduler), a task scheduler that takes advantage of the benefits of fuzzy logic and PSO. When the FPFTS scheduler assigns a set of application tasks, it reports the optimal fog devices for processing them based on the fog devices' and tasks' attributes as defined by their fitness values.

Chapter 2 is a continuation of Chapter 1. In this chapter, we suggested the FUPE approach for task scheduling in SDN-based IoT-fog networks, in which fog devices are clustered into virtual organizations (each representing a fog region) and connected via SDN switches. FUPE is a firewall that detects and mitigates TCP DDOS attack by malicious IoT/fog devices. Furthermore, by employing a MOPSO technique, it is possible to find a balance between fog device security and efficiency.

Chapter 3 presents S-FoS, a secure workflow scheduling approach in SDN-based IoT-Fog networks. To safeguard its scheduling services, S-FoS is a firewall that identifies and mitigates malicious requests that perform TCP DDOS, UDP DDOS, and port scanning attacks. It also employs NSGA-III algorithm to strike a balance between load balancing and delay when selecting the appropriate fog device.

Chapter 1

FPFTS

In the Internet of Things (IoT) scenario, the integration with cloud-based solutions is of the utmost importance to address the shortcomings resulting from resource-constrained things that may fall short in terms of processing, storing, and networking capabilities. Fog computing represents a more recent paradigm that leverages the wide-spread geographical distribution of the computing resources and extends the cloud computing paradigm to the edge of the network, thus mitigating the issues affecting latency-sensitive applications and enabling a new breed of applications and services. In this context, efficient and effective resource management is critical, also considering the resource limitations of local fog nodes with respect to centralized clouds. In this article, we present FPFTS, fog task scheduler that takes advantage of particle swarm optimization and fuzzy theory, which leverages observations related to application loop delay and network utilization. We evaluate FPFTS using an IoT-based scenario simulated within iFogSim, by varying number of moving users, fog-device link bandwidth, and latency. Experimental results report that FPFTS compared with first-come first-served (respectively, delay-priority) allows to decrease delay-tolerant application loop delay by 85.79% (respectively, 86.36%), delay sensitive application loop delay by 87.11% (respectively, 86.61%), and network utilization by 80.37% (respectively, 82.09%), on average.

1.1 Introduction

The proliferation of sensors, actuators, as well as hand-held devices connected in a communicating-actuating network has generated the so-called Internet of

Things (IoT), wherein smart objects blend seamlessly with the environment around us, allowing the information to be shared across platforms to develop a common operating picture [1]. To support the requirements of these resource-constrained devices, the integration of IoT with the cloud paradigm is critical [2, 3]. Indeed, cloud computing is an enabler of the utmost importance, providing the IoT devices with the means to gather, process, store, and distribute the huge amount of information generated by the IoT ecosystem. On the other hand, the integration with cloud is not trivial. In fact, its inherent performance and availability limitations [4, 5] make cloud support unsuitable for specific classes of applications (e.g., real-time applications, gaming, medical apps, etc.) [6, 7]. Although cloud providers are deploying more and more capillary infrastructures closer to users' access networks, cloud is proven to be unsuitable for latency-sensitive applications whose performance is impacted by cloud-network infrastructural shortcomings.

In order to mitigate cloud limitations, fog computing has been proposed [8]. This paradigm extends cloud to the edge of the network, with location awareness and low latency infrastructure, also to support mobility. Accordingly, this allows to enable a new breed of streaming and real-time applications and services. Fog paradigm makes available nodes close to users to provide computing resources to run applications or store significant amounts of data. In detail, user-application requests generate a set of tasks that can be run by fog devices. Fog task scheduling defines how to *properly* assign tasks to the fog devices to satisfy the needs of the specific applications or to reduce network utilization. In fact, while fog computing provides significant performance benefits by design, it introduces resource constraints with respect to the cloud paradigm and, therefore, the need for effective and efficient resource management, that may results in even complex and sophisticated scheduling strategies [9].

1.1.1 Goal and contribution

In this section, we propose FPFTS, a fog task scheduler in which we jointly employ *fuzzy logic* in the fitness function of a *particle-swarm optimization (PSO)* algorithm to improve its performance. Due to its suitability for global searching and its guided randomness feature, PSO is expected perform well in dynamic environments such as fog computing. Moreover, it has fewer parameters than either genetic algorithm or swarm intelligence. On the other hand, a fuzzy optimization has some unique characteristics which make it an appropriate choice for several control problems and suited for environments where the

settings are not precisely defined or previously unknown. By using Mamdani fuzzy rules, we can make a relationship between some parameters by adding priority to them[10].

This section presents a new approach to optimize the task scheduling problem in fog computing for both delay-sensitive and delay-tolerant applications. The goal of the proposed approach is to optimally use fog resources such to reduce network utilization and application loop delay. To this end, in our proposal we simultaneously consider the *computing features of the fog nodes* such as CPU processing capacity, RAM size, and bandwidth, together with *the features of the tasks* such as CPU need, as similar approaches proved to be suitable for task scheduling in previous works [11, 12]. Moreover, our proposed approach is designed to work with both delay-sensitive and delay-tolerant applications: FPFTS benefits from the information about the class each application belongs to, to refine scheduling decisions in case of fog-layer overloading. We use fuzzy logic in PSO considering the features of resources and tasks to solve fog task scheduling problems without being trapped into a local minimum, and optimally use the fog resources.

The contributions of this paper are as follows: (i) We design a new joint meta-heuristic method called FPFTS combining PSO and Fuzzy methods to tackle the fog task scheduling problem. (ii) We implement and test our method taking into account an IoT-based case study and considering a three-layered fog-computing architecture. (iii) We evaluate FPFTS leveraging *iFogSim*, a widely adopted and well-known fog simulator. To this end, to test our proposal we take into account both delay-tolerant and delay-sensitive applications. We test FPFTS against *First-Come First Served* (FCFS) and *Delay-Priority* strategies (which are classic methods in this area) by varying the number of users benefiting from the fog services, as well as the characteristics of the network infrastructure, in terms of the bandwidth and the latency of its links. The obtained results show that our proposal outperforms FCFS and Delay Priority approaches in terms of application loop delay and also network utilization. In details, FPFTS improves by $\approx 86\%$ application loop delay, on average, while concerning network utilization, FPFTS improves this metric in by $\approx 81\%$, on average.

1.2 Related work

In this section, we delineate the existing task scheduling algorithms applied in the fog systems. Fog task scheduling approaches are divided into two cate-

gories: *dynamic algorithms* and *static algorithms*. Dynamic strategies calculate the task priorities during their execution, while static approaches assume to have prior information about fog resources and tasks. Static scheduling algorithms can be further divided into two main families: *heuristic-based* and *metaheuristic-based* algorithms. To reduce the delay, heuristic algorithms may return a sub-optimal resource allocation, but still proper to satisfy the users' requests. Metaheuristic algorithms perform a random search to find a good solution to an optimization scheduling problem [13]. In this section, we briefly review the static heuristic-based and metaheuristic-based algorithms.

Concerning heuristic-based strategies, Bittencourt et al. [14] design a mobility-aware fog application scheduling framework which uses application classification and mobility of the clients as the nature of decision making. They implement First Come-First Served (FCFS), concurrent strategy, and Delay-priority as the scheduling algorithms of the framework. Afterwards, Mahmud et al. [15] introduce a latency-sensitivity application task approach that considers the different application delivery latency for fog applications. Gill et al. [16] model a PSO-based resource management approach associating with scheduling which is suitable for smart home IoT device. Zeng et al. [17] present a fog-supported resource management framework which consists of a three-step task scheduling and a task image placement algorithm to minimize the application delay. The authors focus on computation time and task computation. Hoang et al. [18] present a task scheduling approach called FBRC, which assigns user tasks to fog-based regions and cloud data centers to minimize the task execution time. The authors try to reduce the application delay by using a heuristic algorithm. By using a heuristic algorithm, the authors try to reduce the application delay.

For what concerns metaheuristic-based strategies, Sun et al. [19] present a two-level task scheduling approach which consists of two steps. In the first step, the algorithm finds the most proper fog device cluster, and after that it finds the most proper fog device to execute the applications' tasks. This approach implements an improved non-dominated sorting genetic algorithm II (NSGA-II) to reduce the delay, execution time, and also increase scalability. The algorithm designed by Bitam et al. [20] implements an optimization algorithm inspired by bees swarm intelligence method which targets run-time tasks and the allocated memory. Also, the algorithm presented by Luo et al. [21] forms a fuzzy load balancing strategy associating with real-time scheduling. Rafique et al. [22] describe a hybrid bio-inspired algorithm, which is a combination of modified PSO and modified cat swarm optimization (MCSO) named

MPSO. Their MPSO schedules the tasks among fog devices and manages the availability of resources at the fog device level.

Overview on background methods and comparison with FPFTS. The mentioned works refer to various IoT scenarios. Moreover, two main strategies for realizing collaboration among nodes can be identified:

peer-to-peer (P2P) and *master-slave* [23]. peer-to-peer (P2P) collaboration among the fog devices is very common in fog computing infrastructure. Through P2P collaboration, a fog device can use the processed output of the other fog device. Moreover, fog devices can share virtual computing instances between each other. In master-slave mode, a master fog device controls underlying slave fog devices such as processing load, resource management, data flow, etc. [23]. All of the mentioned methods except the methods presented by Bitam et al. [20], and Luo et al. [21] implement P2P as the nodal collaboration strategy between the fog and IoT devices.

Regarding the nature of observations, differently from all of the mentioned works, we analyze the computing features of resources and tasks altogether. Therefore, for FCFS and delay-priority strategies [14], if fog devices do not have free space capacity, they offload tasks to the cloud data center. They consider availability and CPU capacity of fog node resources for the scheduling. The presented algorithm by Mahmud et al. [15], is based on the placement time and the percentage of task deadlines. The presented algorithm by Bitam et al. [20], considers two computing criteria namely, *time* and the *allocated memory* needed by applications for the algorithm decision. The algorithm which is presented by Luo et al. [21], considers task arrival time, task deadline time, and task size metrics for their scheduling algorithm. The presented algorithm by Gill et al. [16] uses various quality of service criteria such as response time, energy, latency, and network bandwidth. The algorithm presented by Sun et al. [19], considers the CPU performance and the battery lifetime, the resource utilization of fog clusters, and also the distance between fog device clusters and users for decision making. The algorithm presented by Zeng et al. [17], considers task completion time as the main metric for decision making. Their algorithm aims to minimize the computation latency of user requests. The algorithm presented by Hoang et al. [18], considers the maximum delay as the constraint condition to set the upper bound of delay which increases the complexity of the algorithm. In their work, computation Time and transmission time are the main parameters for making a scheduling decision. Finally, the algorithm presented by Rafique et al. [22] considers the least loaded device as a criterion of availability of resources for making a decision.

In our proposed approach, differently from the works by Gill et al. [16] and Rafique et al. [22] we use fuzzy theory in the PSO fitness function. These works use some weights to prioritize the components of the PSO fitness function. In this paper—unlike the works by Sun et al. [19], Bitam et al. [20], Zeng et al. [17], and Hoang et al. [18]—we use iFogSim for simulations and testing on various metrics. Besides, unlike the works by Mahmud et al. [15], Bitam et al. [20], and Gill et al. [16], FPFTS uses mobility-aware scenario in performance evaluation. The algorithm presented by Sun et al. [19], considers the distance between the fog resources and the service requester. They assume considering the principle of proximity effects on application delay. It means that for instance, if a fog device is far away from the requester, the application delay will become high. The demerit of this strategy is that they do not consider link bandwidth. Differently than this work, FPFTS considers link bandwidth which is a more proper metric than the distance proximity parameter to reduce delay. Differently than the work by Zeng et al. [17], FPFTS has low memory consumption. Differently than the work by Hoang et al. [18], FPFTS has low time complexity and high efficiency in application processing rate. Finally, unlike the works by Bitam et al. [20] and Luo et al. [21], in FPFTS, we use the P2P structure for nodal collaboration.

Table 1.1 presents the comparison of existing task scheduling methods compared with our proposed FPFTS strategy in summary.

Table 1.1: Comparison of existing scheduling approaches against FPFTS. NC:= nodal collaboration; P2P:= peer-to-peer; M-s: master-slave; Ctr:= centralized; Adv.:= Advantages; Disadv.:= Disadvantages.

Refs.	Algorithms	Tool	Observations	NC	Adv.	Disadv.
[14]	FCFS/Delay-priority	iFogSim	Resources availability and CPU capacity	P2P	+ Considering movement based scheduling at cloudlet level	- No consider fog device processing capacity
[15]	Heuristic method	iFogSim	Deployment time, deadline, #fog node	P2P	+ latency-aware IoT application + Reducing the amount of deployment time + Deals with varying application	- No Real case study - No Mobility
[20]	Bees swarm	C++	CPU execution time, Allocated memory	M-s	+ Managing allocated memory + Low CPU execution time	- Only fog devices are used - Static scheduling
[21]	Fuzzy-NN	iFogSim	Arrival Time, Deadline Time, Task size	M-s	+ Reliable real-time applications	- High cost
[16]	PSO	iFogSim	Response time, energy, latency, and network bandwidth	P2P	+ Optimizes energy, latency, response time and network bandwidth	- No reliability assurance due to a simple fog device
[22]	Hybrid MPSO	iFogSim	least loaded fog device	P2P	+ Consider communication latency & computation latency	- No network latency
[19]	NSGA-II	Matlab	CPU performance Battery lifetime Distance between clusters and users	P2P	+ Low application delay + high scalability	- High cost
[17]	Heuristic-based algorithm	Gurobi tool	Task completion time	P2P	+ Low computation complexity	- High memory consumption
[18]	Heuristic-based algorithm	NA simulator	Computation time Transmission time	P2P	+ Low latency rate	- Low efficiency in service processing rate - High time complexity
FPFTS	Hybrid (PSO-Fuzzy)	iFogSim	CPU, RAM bandwidth of fog devices and task CPU length	P2P	+ Mobility-aware scenario + Considering computing capacity of resources + Low application loop delay/network utilization	- Reliability relies on fog gateway fault tolerance

1.3 FPFTS: Fuzzy PSO Fog Task Scheduler

In this section, we describe our proposed scheduler called FPFTS. This section presents the proposed architecture, the details of FPFTS, and the system sequence diagram.

1.3.1 Architecture

In this work, we refer to a cloud-fog-device structure [24] which consists of three layers. The front-end layer (*device layer*) consists of user devices (such as smartphones, laptops, tablets, etc.). They run user applications and may send requests to fog devices. The *fog layer*, which is composed by a set of fog devices (i.e. servers), is located at the edge of the access network. It receives and processes users' requests. The *cloud layer*, which consists of one or more cloud data center, provides virtually unlimited resources to execute the tasks which are offloaded from the fog devices [25]. The architecture is presented in Fig. 1.1. In our architecture, we employ decentralized broker management strategy [26] for task scheduling. FPFTS runs in the broker (i.e. the fog gateway). However, based on different scenarios, the broker can be located in either the cloud gateway or the fog gateway, or any other intermediate location between the devices and the cloud. Changing the location of the broker may impact the performance of the architecture, which in turn depends on network performance.

According to our model, each fog region has its own fog gateway which is responsible to coordinate the fog devices located in fog region. The broker is a node which has the information of the fog devices. It is located between the fog devices and the users, and acts like a portal for users applications. As for the most of practical scenarios the cloud data center is far away from the users' devices, in this work, firstly, FPFTS uses fog devices for task scheduling, and only in case of fog device overloading, to overcome the fog devices computing limitations, it offloads tasks to the cloud data center by using *offloading agents*. These agents are located in fog devices.

1.3.2 Problem Statement

In this paper, we focus on the task scheduling problem which has to be effectively and efficiently addressed in fog platforms, in order to distribute application tasks among fog nodes. Task scheduling is a critical part in both cloud and fog resource management which has significant effects on the performance

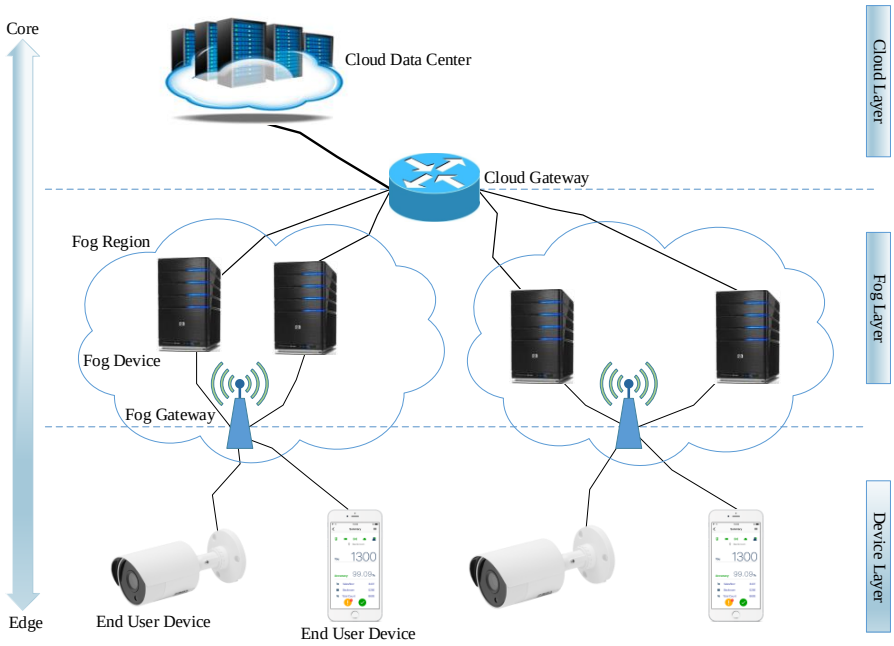


Figure 1.1: Proposed architecture

of the overall architecture [26]. The aim of task scheduling is assigning tasks generated by users' applications to resources available at the fog layer. Resource management strategies may also implement cloud data offloading so as to overcome resource constraints (e.g., limited computational capacity at fog nodes): in case of fog device overloading, cloud resources can be leveraged.

In more details, jobs generated by applications/services are decomposed into a set of atomic tasks to be assigned to fog devices. If the tasks composing a job are mutually independent, they can be executed on separate fog devices, with no need of explicit synchronization [20].

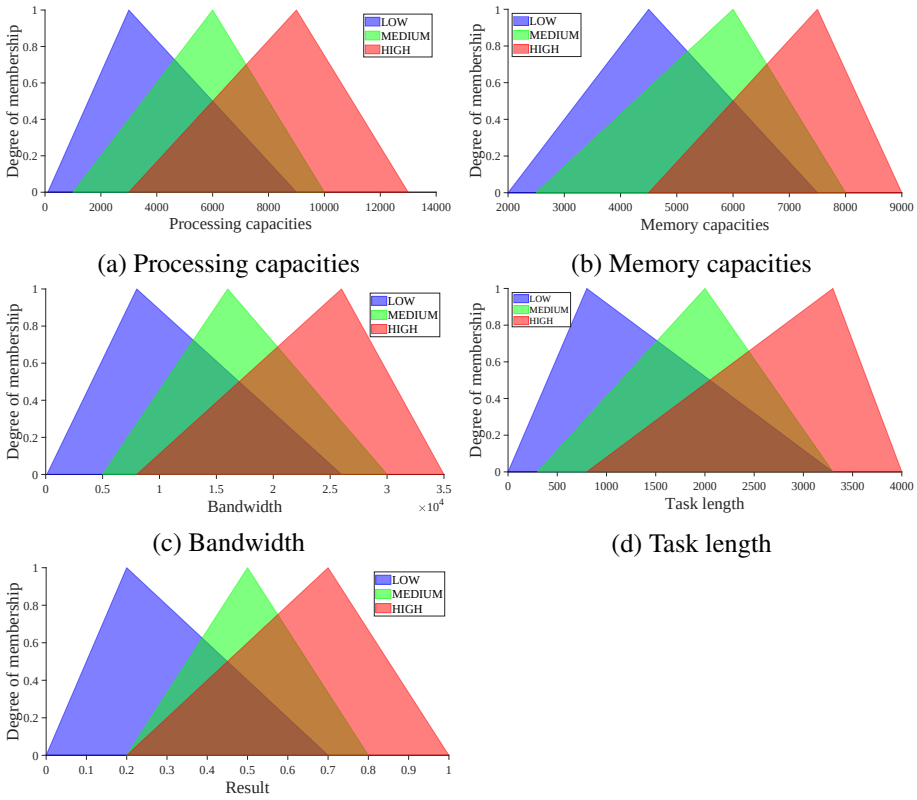
The goal is to find the most adequate fog devices to run each application task so as to have a proper outcome in terms of application loop delay and also network utilization.

1.3.3 Proposed Scheduler

This subsection describes the presented approach, and the way it assigns applications' tasks to the resources. FPFTS implements a bio-inspired approach that uses fuzzy logic in the PSO fitness function for fog task scheduling.

When the scheduler assigns a set of applications' tasks, it returns the most adequate fog devices for executing them, based on both the characteristics of the tasks and of the fog devices. In detail, FPFTS uses task CPU need, fog-device processing capacity, fog-device RAM capacity, and fog-device bandwidth as the input parameters of the fuzzy-based fitness function. In other words, FPFTS considers the features of resources and tasks *simultaneously*. This enables FPFTS to achieve a proper distribution of the task across devices such to reduce network utilization and application loop delay.

Fuzzy Based Fitness Function. Most of the PSO-based schedulers use some predefined weights to prioritize the parameters in the fitness function [16]. Using fuzzy logic is an efficient alternative. We use a Mamdani fuzzy inference system to reap the benefits deriving from fuzzy logic in the prioritization of parameters. Mamdani fuzzy inference engine is one of the common fuzzy inference engines [27] which uses some fuzzy rules. These fuzzy rules can be based on either former experience or some assumptions [28]. Concerning Mamdani fuzzy inference engine, we define some overlapping fuzzy sets—which make the input parameters possibly lie in several sets at the same time. Accordingly, an input parameter can have more than one value with different membership degrees. For example, an input parameter can be defined both as *low* and *medium* with a different membership fitness value (that is a number between 0 and 1 which is obtained via membership fitness functions). As a



(a) Processing capacities

(b) Memory capacities

(c) Bandwidth

(d) Task length

(e) Result

Figure 1.2: Fuzzy set parameters.

consequence, the input parameters may trigger multiple fuzzy rules. As each fuzzy rule is associated to a specific priority, it increases the degree of prioritization. For instance, for task CPU length parameter, a value equal to 1000 belongs to low, medium, high fuzzy sets with the membership value 0.8, 0.4, and 0.2 respectively.

Figs. 1.2a, 1.2b, 1.2c and 1.2d report fuzzy sets for the fog device processing capacities and RAM and bandwidth and task CPU length parameters, respectively. Fig. 1.2e indicates the fuzzy set for the result parameter.

Table 1.2 reports the rules we used. Based on the input parameters of the fitness function, the fuzzy inference engine triggers some of them. We refer to them as the *fired rules*. These fuzzy rules are then integrated (creating a fuzzy output graph). Finally, the the fuzzy inference engine defuzzifies this

fuzzy output into a numeric value via the cetroid method [28]. The obtained value is the output of fitness function which determines the suitability degree of resources for assigning them to the tasks.

Table 1.2: Mamdani fuzzy rules.

Computing Capacity	Memory	Bandwidth	Task Length	Result
Low	Low	Low	Low	High
Low	Medium	Medium	Medium	High
Medium	High	Low	High	Medium
Medium	Low	High	Low	Medium
High	Medium	Medium	Medium	High
High	High	Low	High	High
Medium	medium	High	medium	Medium
High	Low	Low	High	Low
High	Medium	Medium	High	medium

PSO Based Fog Task Scheduling Algorithm. In our approach we use the PSO algorithm [29] to find a proper solution. Each particle is an instance in the search space, and in this work we consider tasks as particles. To start this algorithm, a population of particles is generated randomly. The position of the particles (fog devices in this work) represents a potential solution. Each particle has a position vector and a velocity vector which determines the direction of movements in the search space. The search space represents the fog regions which contain fog devices. We define *pbest* and *gbest* as the best position for each task and the best position for the tasks among group based on the fitness value of all the tasks, respectively.

The PSO algorithm uses Eq. (1.1) for updating position vector, and Eq. (1.2) for updating velocity vector.

$$X_{k+1}^{i} = X_{k}^{i} + V_{k+1}^{i} \tag{1.1}$$

$$V_{k}^{i+1} = W_k V_k^i + c_1 r_1 (pbest_k^i - X_k^i) + C_2 r_2 (gbest - X_i^k) \tag{1.2}$$

C_1 and C_2 are the acceleration coefficients, and PSO considers them as learning factors. Moreover, PSO uses r_1 and r_2 to control the randomness of the particles movements in the search space. Besides, W is inertia weight which indicates the balance between local and global search. This parameter determines the repeat rate for finding the best solution. In FPFTS number of iterations is equal in value to the number of fog devices so as to keep the

computation time low. FPFTS improves the fitness value of the applications' tasks in every generation. It rejects a new solution if its fitness value which is calculated by the fuzzy-based fitness function is less than the current solution. The final result is the best *pbest* value (i.e. *gbest*).

Algorithm 1 FPFTS: proposed algorithm.

INPUT: M, P, v
M: population size
ITE: Iteration number
P: population of particles
v: speed of each particle
OUTPUT: $gbest$

1: **for** $i = 1$ to ITE **do**
Initialize $P[i]$ randomly
2: **Initialize** $V[i] = 0$;
3: $pbest[i] = P[i]$
4: $F = ComputeFitness(P[i])$;
5: $gbest =$ best particle found in F;
6: **for** $i = 1$ to M **do**
7: $pbest[i] = F$;
8: **end for**
9: **repeat**
10: **for** $i = 1$ to M **do**
11: $V[i] = W \times V[i] = C_1 \times rand1(pbest[i] - P[i] + C_2 \times rand2(gbest[i] - P[i])$;
12: **update** $V[i]$
13: Set W, $C_1 >= 0$ and $C_2 >= 0$;
14: $P[i] = P[i] + V[i]$;
15: **if** a particle goes outside the predefined hypercube **then**
16: it is reintegrated to its boundaries;
17: **end if**
18: $F = Computefitness(P[i])$;
19: **if** new population is **better then**
20: $pbest[i] = F$;
21: **end if**
22: $gbest$=Best particle found in $P[i]$;
23: **end for**
24: **until** stopping criterion is satisfied
25: **end for**
26: **return** $gbest$

FPFTS Algorithm Alg. 1 Description. In Alg. 1, first, FPFTS determines the initial positions and velocity for each particle in the search area randomly (lines 1, and 2). We save the current solution in *pbest* (line 3). Then FPFTS computes the fitness function for all particles (line 4), and we save it as the best solution (line 5). Then, after defining the fitness value for all of the parti-

cles, FPFTS compares them, and if the fitness value of the current solution is better than *pbest*, it replaces it with the *pbest* (lines 6 till 8). In FPFTS, Each task's movement is influenced by its best known position (pbest). Besides, it is guided toward the best known positions (gbest) which are found by other particles. To this end, FPFTS uses Eq. (1.1) and Eq. (1.2). FPFTS updates *gbest* value by comparing the *pbest* value of the current iteration to the value which is stored in the *gbest*, and if its satisfaction is more than *gbest*, the PSO algorithm replaces it (lines 9 till 24). We repeat these steps to reach the maximum number of iterations (line 25). Finally, the updated *gbest* value is the final solution and output of this algorithm (line 26).

FPFTS Process Diagram. Fig. 1.3 indicates the process diagram of the presented approach. In the beginning, users (devices such as smartphones or cameras) send their requests (applications) to the nearest broker (fog gateway) which is located in the fog layer and is responsible for task scheduling. FPFTS instances are located in fog gateways which are responsible to decompose the applications into a set of tasks, schedule the tasks, and assign the tasks to the fog devices inside the fog regions. Fog devices execute the tasks, and in case of fog device overloading, fog devices move tasks to the Cloud Data Center which is located in the cloud layer (Cloud data offloading). Fog devices and cloud data center execute the tasks. They send results to the instances of FPFTS, and these instances send the results back to the users. When a fog device becomes overloaded, two different possible data offloading scenarios can take place: *fog device to fog device data offloading*, and *fog device to cloud data center data offloading* [30]. In this work, we only use fog device to cloud data center data offloading. Task scheduling and task offloading are the major parts of resource management approaches. Task schedulers need data offloading to deal with system failure in case of fog device overloading [26]. In this paper, we implement task scheduling unit and data offloading unit, separately. In order to move tasks to the cloud data center, first, *offloading agents* sends delay-tolerant applications to the cloud data center. Then, they offload delay-sensitive applications to the cloud data center.

Space Complexity of FPFTS. For calculating the space complexity we consider the space FPFTS instances require for task scheduling. We name this parameter S, and we consider it (S parameter) in the computational complexity formula. As a result, in general, the total space complexity of the FPFTS is in order of $\mathcal{O}(S \times (I \times F \times T))$. For a small scale of networks, in which there is only one fog region (there is only one instance of FPFTS), S parameter is 1, and as a result, the total space complexity of FPFTS is in order of

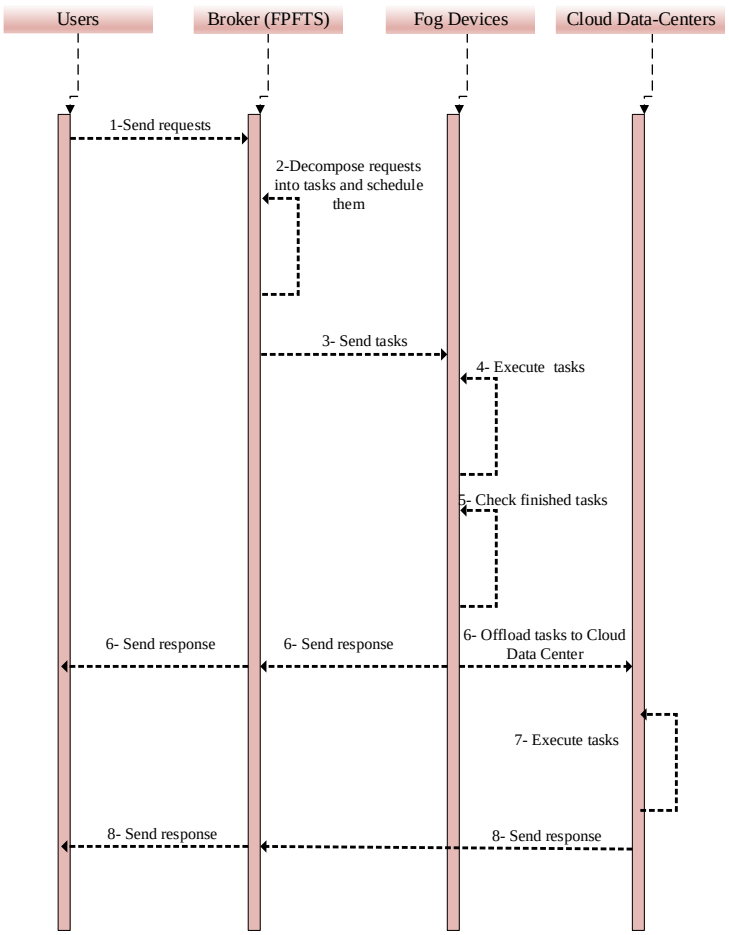


Figure 1.3: The sequence diagram of FPFTS activities.

$\mathcal{O}(I \times F \times T)$. For a large scale of networks, in which there is one instance of FPFTS in each fog region, S parameter is considered equal in value to the number of fog regions.

1.3.4 Case Study: Smart City

In this subsection, we provide a simple smart-city case study to exemplify the task scheduling strategies. The presence of IoT devices allows the monitoring of different activities of the smart city system. In a smart city, improper task scheduling strategy results in a high delay that is unacceptable to user satisfaction. Using a proper task scheduling strategy, a significant number of smart city requests can be performed by nearby Fog devices, resulting in lower delay and much more user satisfaction [31]. In a smart city, some of the applications are delay-sensitive while the others are delay-tolerant. For example, a set of smart camera observations data is collected and saved so that we can prepare a database about urban traffic. We consider data saving and retrieval activities as delay-tolerant applications because delay does not harm their results. On the other hand, in some activities like online gaming, fast processing, and low response times are required to generate real-time experience for users. We consider these real-time interactions as delay-sensitive applications, as user satisfaction is impacted by the delay user perceives. As there is no priority in FCFS, this strategy assigns applications to the resources based on the arrival time of the related requests. As this strategy is not oriented to preserve real-timeliness, users may experiment with bad quality of experience in real-time gaming. Suppose some online gaming and smart camera applications arrive at the fog gateway, in this way, priority aware strategies (i.e., delay-priority and FPFTS) first consider online gaming in both assigning tasks to the resources and data offloading steps. After that, they consider smart camera applications in both mentioned steps. These strategies enhance online gaming real-time experience.

To give a better insight of various strategies, let $T = \{T_1, T_2, \ldots, T_n\}$ and $S = \{S_1, S_2, \ldots, S_n\}$ be *two* sets of delay-tolerant and delay-sensitive applications, respectively. We assume T_1 and T_2 arrive at time t_1 and t_2 to the fog gateway, respectively. Besides, we assume S_1 arrives at time t_3 to the fog gateway. Finally, we assume each of the applications has three tasks as follows: $T_1(1), T_1(2), T_1(3), T_2(1), T_2(2), T_2(3), S_1(1), S_1(2), T_1(3)$. FCFS strategy assigns applications' tasks to the nearest fog device to the fog gateway with no priority based on their arrival time. It also does not consider other fog devices in the fog region. Although S_1 is a delay-sensitive applica-

tion, this strategy does not consider it's priority over T_1 and T_2. FCFS, first, schedules T_1 tasks and after that T_2 tasks, and finally it schedules S_1 tasks. Accordingly, based on this strategy, the nearest fog device executes tasks as follows: $T_1(1), T_1(2), T_1(3), T_2(1), T_2(2), T_2(3), S_1(1), S_1(2), T_1(3)$. In the case of fog device overloading, it moves applications to the cloud data center based on their arrival time. Suppose in the middle of $T_2(1)$ execution, fog device overloads. Hence, this strategy moves the rest of the tasks to the cloud data center as follows: $T_2(2), T_2(3), S_1(1), S_1(2), T_1(3)$. Delay-priority strategy schedules applications based on their priority, but it does not consider other fog devices. This strategy schedules T_1 tasks, and after that, it starts to schedule T_2 tasks. Suppose in the middle of T_2 tasks execution, S_1 arrives at the fog gateway, and we require to schedule it. Since it is a delay-sensitive application, fog device halts the execution of the rest of T_2 tasks and starts to execute the arrived delay-sensitive application tasks. After executing S_1 tasks, this strategy continues to execute the rest of T_2 tasks. Hence, in this strategy, the nearest fog device executes tasks as follows: $T_1(1), T_1(2), T_1(3), T_2(1), S_1(1), S_1(2), T_1(3), T_2(2), T_2(3)$.

For cloud data offloading, suppose in the middle of $T_2(1)$ execution, fog device overloads; this strategy moves the rest of tasks to the cloud data center as follow: $S_1(1), S_1(2), T_1(3), T_2(2), T_2(3)$. When it comes to the FPFTS, unlike the delay-priority strategy, FPFTS considers all of the fog devices in the same fog region for applications' task scheduling. Like delay-priority strategy, FPFTS considers application priority for both task scheduling and cloud data center data offloading steps. Suppose there are two fog devices in the fog region, FPFTS decomposes applications into a set of tasks as follow: $T_1(1), T_1(2), T_1(3), T_2(1), T_2(2), T_2(3), S_1(1), S_1(2), T_1(3)$. Suppose FPFTS makes the first fog device responsible for executing $T_1(1), T_1(2), T_2(1), S_1(1)$; suppose the first fog device is in the middle of executing $T_1(2)$; at this time FPFTS assigns $T_2(1)$ and $S_1(1)$ to the first fog device. As the priority of $S_1(1)$ is higher than $T_2(1)$, the first fog device execute it before $T_2(1)$. So, the first fog device execute the tasks as follow: $T_1(1), T_1(2), S_1(1), T_2(1)$. Suppose FPFTS makes the second fog device responsible for executing $T_1(3), T_2(2), T_2(3), S_1(2), S_1(3)$; suppose the second fog device is in the middle of executing $T_2(2)$; at this time FPFTS assigns $T_2(3)$, $S_1(2)$ and $S_1(3)$ to the second fog device. As the priority of $S_1(2)$, and $S_1(3)$ are higher than $T_2(3)$, it executes the tasks as follow: $T_1(3), T_2(2), S_1(2), S_1(3), T_2(3)$. Suppose in the middle of executing $S_1(2)$, the second fog device overloads; *Offloading agent* which is located

in the second fog device moves $S_1(3)$, and $T_2(3)$ to the cloud data center. Fig.1.4illustrates tasks arrival and assignment to fog devices based on FCFS strategy and priority-based strategy. Execution time is the time required by a task for executing a fog device. We assume T_1 comes at time 0, T_2 comes at time 3, and finally S_1 comes at time 9 to the fog gateway.

Tasks	Arrival times (ms)	Execution time (ms)	Priority
$T_1(1)$	0	1	2
$T_1(2)$	0	3	2
$T_1(3)$	0	2	2
$T_2(1)$	3	3	2
$T_2(2)$	3	2	2
$T_2(3)$	3	2	2
$S_1(1)$	9	3	1
$S_1(2)$	9	1	1
$S_1(3)$	9	1	1

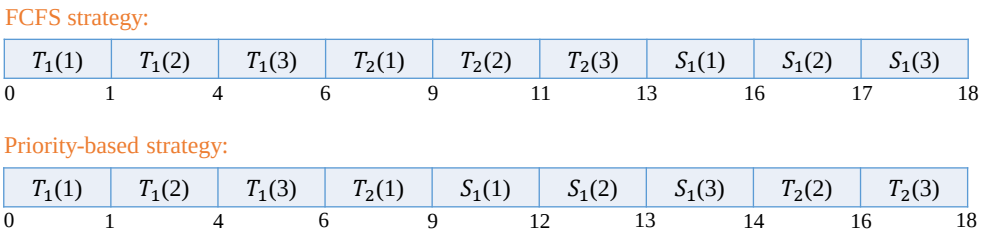


Figure 1.4: An example of task arrival and assignment to fog devices.

1.3.5 Simulation Setup

To analyze the performance of the FPFTS algorithm, we run the simulation under the iFogSim testbed[32, 33]. Besides, we use JFuzzyLogic [34] for implementing fuzzy-based fitness function of FPFTS. We evaluate the proposed method via a city automation experiment case study [14] in which we reuse delay tolerant and delay sensitive applications. For the former, we use a video application video surveillance/object tracking (VSOT) application, and for the latter, we use a game application (electroencephalography (EEG) tractor beam game) or EEGTBG.

Performance comparison. To evaluate the performance of the proposed FPFTS, we compared it against First Come-First Served (FCFS) and Delay-priority strategies[14]. In detail, FCFS is a classic method which is located in the second fog device. In FCFS, the system schedules the tasks based on their arrival time till the fog device becomes overloaded. When it becomes overloaded it offloads the applications to the cloud data center. On the other hand, Delay-priority strategy (which is located in the second fog device) first schedules the delay sensitive applications. Then, it schedules the delay tolerant applications. Besides, for cloud data offloading, delay sensitive applications we consider more priorities for scheduling.
Mobility Scenario. In an urban mobility scenario, inside the Smart City model, several mobile users move to other places. These mobile users send delay-sensitive applications to fog resources. Initially, the delay-sensitive applications run on the nearest fog devices. We assume in a city center several smart cameras run the delay-tolerant applications; we also assume a fog gateway is located in the city center. These smart cameras send delay-tolerant applications to the fog gateway. Moreover, mobile users move to the city center and send delay-sensitive applications to the fog gateway. We put the FPFTS scheduler in the fog gateway. FPFTS schedules the tasks of applications that arrive at the city center. Fig.1.5presents the mobility scenario.

1.3.6 Problem Statement

This scenario helps us to study the effects of different scheduling strategies for a mobile scenario. We assume during rush hours, mobile users move towards the city center. First, we consider application instances in the fog devices. Then, we move the users from the fog devices outside the city center to the fog devices inside the city center. Therefore, the fog devices outside the city center become underutilized, while the fog devices inside the city center become overloaded. To overcome this problem, When mobile users arrive at the city center, they connect to the fog gateway and FPFTS assigns the incoming tasks to the fog devices in the entire fog region. In this study, we locate the user interface in the cloud data center, while the client and motion detector modules in mobile devices are located in the device layer. Besides, we place the object detector, object tracker, concentration calculator and coordinator in fog device or cloud data center based on the scheduling strategy. Table 1.3 presents the features of application modules.
Fog device setup and their communications. In this paper, we consider *three* fog devices [14]. According to [14], the first fog device has a 5,000 millions

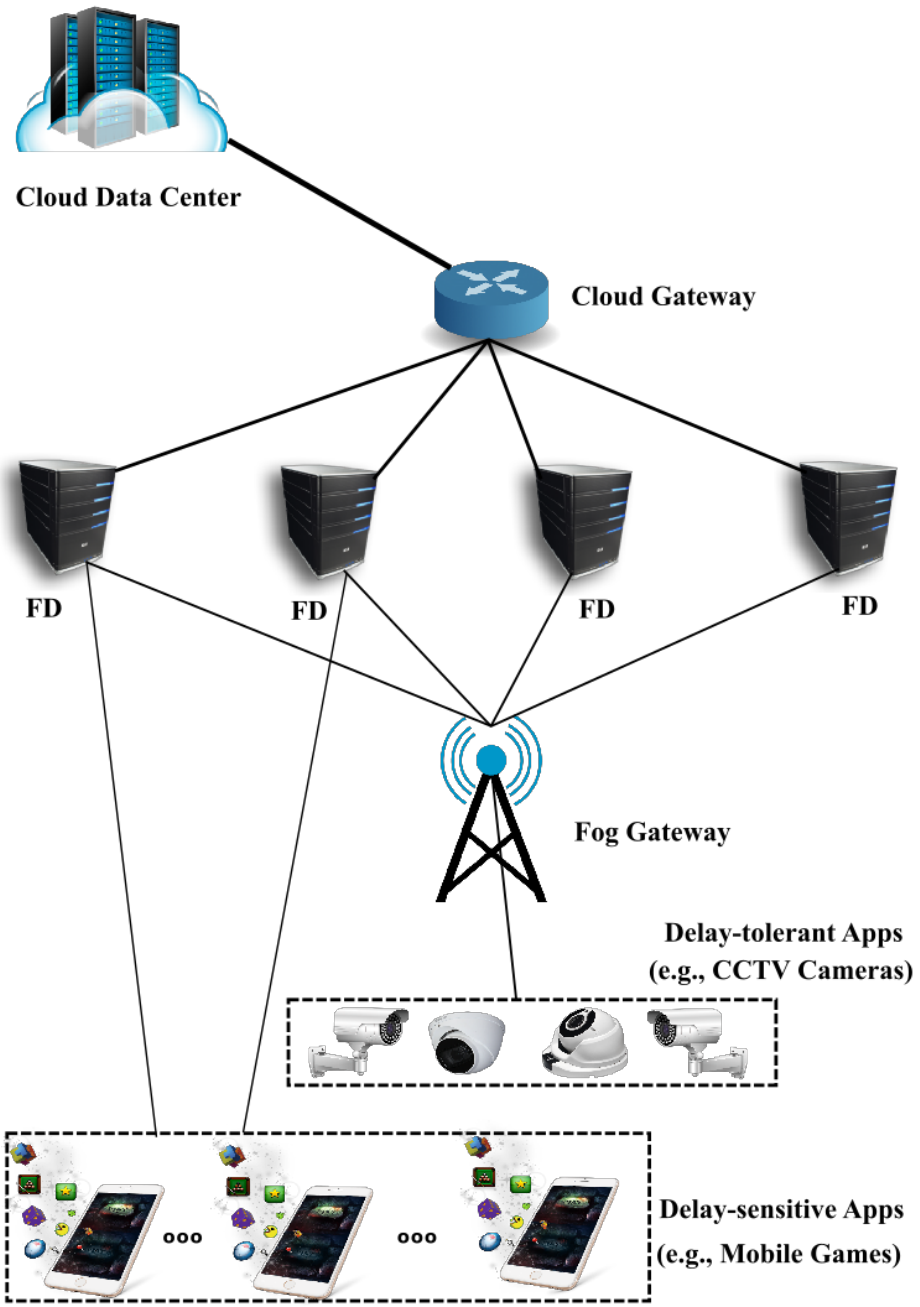


Figure 1.5: Mobility scenario. FD:= Fog Device. It consists of the delay-tolerant applications (e.g., apps instantiated on each CCTV device) that are connected to Fog Gateway and delay-sensitive apps which are directly connected to FDs.

Table 1.3: Maximum CPU requirements of the application modules in (MIPS). OD:= Object Detector; MD:= Motion Detector; OT:= Object Tracker; UI:= User Interface; CC:= Concentration Calculator; CO:= Coordinator.

	VSOT				EEGTBG		
Module Name	**OD**	**MD**	**OT**	**UI**	**Client**	**CC**	**CO**
Maximum CPU Requirements	550	300	300	200	200	350	100

of instructions per second (MIPS) computing capacity, 5,000 (GB) RAM, and 9,000 (Kbps) bandwidth. The second fog device has a 4,000 (MIPS) computing capacity, and 4,000 (GB) RAM, and 10,000 (Kbps) bandwidth. The third fog device has a 5,000 (MIPS) computing capacity, and 5,000 (GB) RAM, and 11,000 (Kbps) bandwidth. The link latency between the cloud gateway and cloud data center, fog devices and cloud gateway, fog gateway and second fog device, fog gateway to the first/third fog device and devices connected to the first/third fog device and fog gateway are 100 (ms), 4 (ms), 2 (ms), 12 (ms), and 2 (ms), respectively. Table 1.4 indicates simulation setup of fog devices.

Table 1.4: Simulation setup for each fog device and cloud data center. CDC:= Cloud Data Center.

Fog Devices	**CPU Capacity**	**RAM**	**Bandwidth**	**Latency to cloud gateway**
First	5,000 (MIPS)	5000 (GB)	9,000 (Kbps)	4 (ms)
Second	4,000 (MIPS)	4000 (GB)	10,000 (Kbps)	4 (ms)
Third	5,000 (MIPS)	5000 (GB)	11,000 (Kbps)	4 (ms)
CDC	44800 (MIPS)	40000 (GB)	10000 (Kbps)	100 (ms)

We select these fog devises for our experiments due to some reasons. First, we set the CPU and RAM capacity of the first/third fog devices higher than the second device to study the situation in which the second fog device becomes overloaded and the others are getting underutilized. Second, we set the link bandwidth of the first fog device lower than the second one. Also, we set the link bandwidth of the third fog device higher than the second fog device. The reason is that we want to study the effects of different fog devices links bandwidth. It also causes fog devices features to be more different to each other. The benefit of this configuration is to *fire more Mamdani fuzzy rules in PSO fitness function*. Third, the delay of communication between the fog gateway to the first/third fog device and fog devices among each other are about 2 (ms) and 12 (ms), respectively. It is because of the case by moving from the fog gateway to the first/third fog nodes it requires at least 2+4+4 (ms) which

equals 10 (ms) link latency. In other words, it takes 2 (ms) to communicate between fog gateway and second fog device, 4 (ms) for communication between second fog device and cloud gateway, and finally takes 4 (ms) to communicate between cloud gateway and second gateway. As these two links are only used in FPFTS, to make the simulation setup comparable and fair, we set 12 (ms) for these links latency.

Evaluation Metrics

To provide a comprehensive evaluations of our algorithm, we use the following metrics:

- **Application Loop Delay:** There is a processing loop for running the tasks of each application (here application is a collection of the same task types), and the time for executing the application tasks is called *the application loop delay*. As the application modules are in user devices, fog devices and cloud data center, implementing a proper scheduling strategy reduces this metric. Link latency, link bandwidth and also computing abilities of the resources impact on the application loop delay.
- **Network Utilization:** It indicates total amount of data which are transmitted in the links between network nodes. For calculating this parameter, we use network latency between the devices that are the origin and the destination of the requests multiplied by the size of the tasks. *Task size* is the file size (in byte) of the task. Each task is characterized by two attributes as follow: first, processing requirements which is defined as million instructions (MI); second, the length of data encapsulated in the task. Task size is the length of data encapsulated in each task. The network utilization is obtained as the sum of the network usage generated by the requests which is sent during the simulation time. Eq. (1.3) illustrates how iFogSim calculates network utilization metric.

$$NetworkUtilization = \sum_{x=1}^{Requests} (Latency \quad (milliseconds) * TaskSize \quad (byte))/ \\ SimulationTime \quad (milliseconds) \tag{1.3}$$

1.3.7 Results

In this section, we test our presented algorithm against Delay-priority and FCFS methods over various moved users, variant of the bandwidth, and various link latency.

Comparing methods based on number of users moved

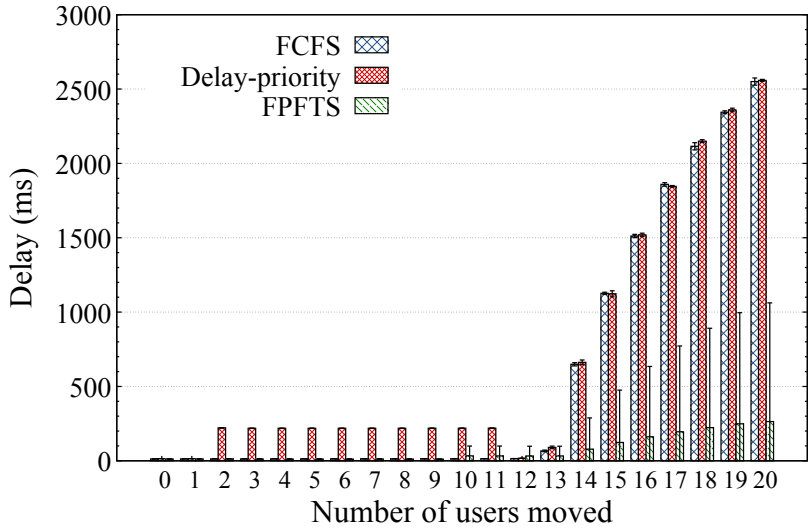
This experiment indicates the mobility-aware scenario of performance evaluation for various number of moved users (IoT tier) in the network. We assume In the second fog device, there are 4 delay-tolerant applications. We also assume 10 mobile delay-sensitive applications running on the first fog device and 10 mobile delay-sensitive applications running on the third fog device, move one by one to the second fog device [14].

Loop delay vs.- number of users moved: Fig. 1.6a and Fig. 1.6b indicate VSOT and EEGBTG applications loop delay for the FPFTS approach compared task scheduling strategies based on the number of users moved from the first/third fog device to the second fog device, respectively. Both FCFS and Delay-priority implement a module merging mechanism in which all of the same kinds of modules are scheduled in the same device. It means they can not schedule the same kind of modules on different devices. From the arrival of the second EEGBTG player till the 11^{th}, Delay-priority has the highest delay in the VSOT loop, while it has a low delay in EEGTBG loop. On the contrary, FCFS has the highest delay in the EEGBTG loop, while it has a low delay in the VSOT loop. When 12^{th} EEGBTG player arrives, because second fog device does not have enough processing capacity, the Delay-priority strategy moves EEGTBG modules to the cloud data center and keeps the VSOT modules in the second fog device. The reason for this behavior is that Delay-priority schedules all the same kind modules in a specific device. It can not keep some of the same kind of modules in the second fog device and move the rest of them to the cloud data center. At this point, both FCFS and Delay-priority have the same outcomes for both applications. From the arrival of 13^{th} EEGBTG player till the last one, both FCFS and Delay-priority have almost the same results. Both these strategies, move the EEGBTG modules to the cloud data center, while they keep the VSOT modules in the second fog device. As the bandwidth of the second fog device links is low for moving this amount of data to the cloud data center, both applications loop are increased dramatically. We study the effects of bandwidth on application loop delay in the next experiments. Despite the fact that from the arrival of 13^{th} EEGBTG

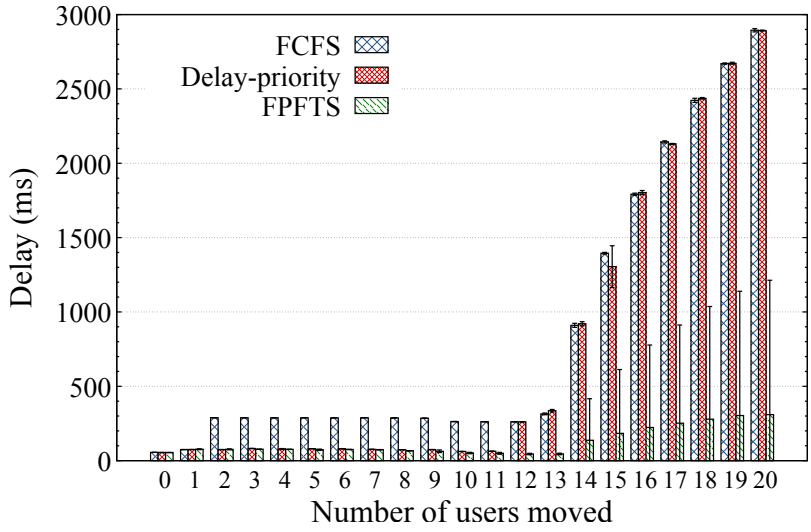
player to the last one, both the strategies keep object tracer and object detector VSOT modules in the second fog device. Because *user interface* VSOT modules are located in cloud data center, it increases the VSOT application loop delay. When it comes to FPFTS, As it uses the features of resources and tasks, it has good results in both application loops. Error bar is a representation of the variability of data and used on graphs to show the uncertainty in a reported measurement. As FPFTS uses the randomness features of the PSO algorithm, we run the FPFTS on average 10 times, for several users moved variable values from 0 to 20. We observed from the arrival of 10^{th} EEGBTG player to the last one, it moves modules to the cloud data center, on average, about 10% of simulator running, which causes having a more standard deviation. To put it another way, because of the randomness features of PSO, for each number of users moved, FPFTS requires a simulator to run in different numbers to offload modules to the cloud data center. On average, it takes one time Cloud data offloading process per 10 times running the simulator.

Network utilization vs.- number of users moved: In Fig. 1.7, we present network utilization among the scheduling algorithms. As FPFTS has the least amount of data transmitted to the cloud data center, it has a reasonable outcome for network utilization. In other words, FPFTS transmits more data in the second layer (fog layer) rather than the first layer (Cloud layer), decreasing the network utilization. Focusing on FCFS and Delay-priority methods, these approaches could schedule tasks either in the second fog device or cloud data center, without considering the another fog device, hence, they increase network utilization. Also, when number of moved users$\geq$ 2, FCFS algorithm moves the EEGBTG modules to the cloud data center. Hence, it increases the network utilization gradually. Moreover, from the arrival of the third EEGBTG player to the 11^{th} moved user, Delay-priority strategy has a high total network usage. The reason is that it moves the VSOT modules to the cloud data center and raises the network usage in this algorithm. Finally, from the arrival of the 12^{th} EEGBTG user to the last one (20^{th} moved user), it keeps $objecttracker$ and $objectdetector$ VSOT modules in the second fog device, it has the same results as FCFS.

Precisely, when the second EEGBTG application arrives, the Delay-priority method moves four $objecttracker$ of VSOT application modules to the cloud data center. Then, by entering the third VSOT application until the 11^{th} one, it maintains four $objecttracker$ and four $objectdetector$ VSOT application modules in cloud data center. Moreover, by arriving at the 9^{th} EEGTBG application until the $11th$ one, because of the limitation in second



(a) VSOT application



(b) EEGBTG application

Figure 1.6: Application loop delay in (ms) for a) VSOT applications, and b) EEGBTG applications for various number of moved users among FCFS, Delay-priority, and FPFTS algorithms.

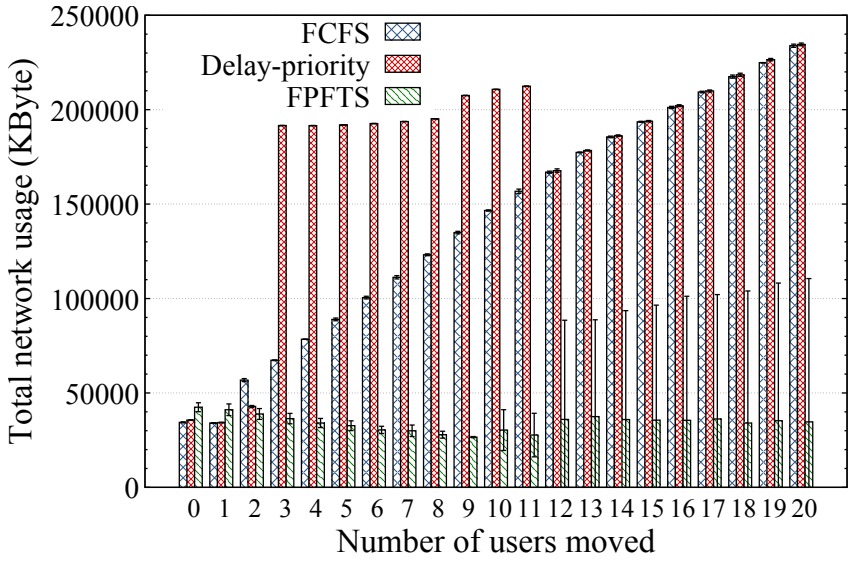


Figure 1.7: Network utilization in (KBytes) for various number of moved users among FCFS, Delay-priority, and FPFTS algorithms.

device computing capacity, schedules the $concentrationcalculator$ modules in the second fog device, and offloads $coordinator$ modules to the cloud data center. It results from having around 34,330 and 42,820 (KBytes) network utilization by arriving the first and the second one respectively, and around 191,600 (KBytes) network utilization by arriving the third one till the 8^{th} one. Besides, the network utilization by arriving the 9^{th} one till 11^{th} one is around 210,700 (KBytes).

Comparing methods based on various bandwidth ranges

This experiment indicates the mobility-aware scenario of performance evaluation for various fog bandwidth capacities. Bandwidth indicates the maximum data transfer rate of network links. It shows how much data can be sent over the links.

Application loop delay vs.- various bandwidth ranges: In this part, we discuss the effects of bandwidth on application loop delay, which we detailed in the following experiment. In this experiment, we move 20 users from the first and third fog device to the second one. Fig. 1.8a and Fig. 1.8b indicate how bandwidth effects on both VSOT and EEGBTG applications loop de-

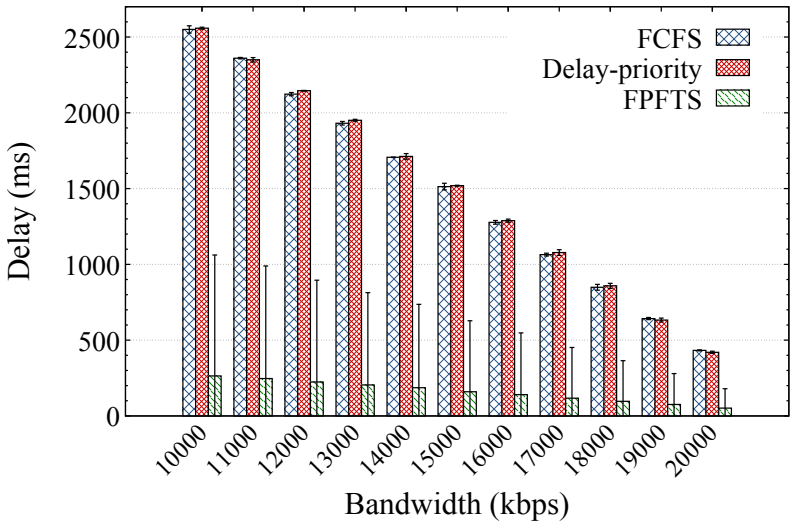
lay, respectively. By changing the second fog device link bandwidth from 10,000 to 20,000 (Kbps), both VSOT and EEGBTG applications loop delay in both FCFS and Delay-priority strategies decrease significantly. The reason is that 19 modules of $concentrationcalculator$ and 19 modules of $connector(Coordinator)$ are offloaded to the cloud data center. Thus, increasing the bandwidth causes to decrease the amount of time for moving them to the cloud data center. When it comes to FPFTS, we can observe that the effects of bandwidth on FPFTS are much lower than other methods. As FPFTS uses all fog devices, and only in case of fog device overloading it moves modules to the cloud data center, link bandwidth has the lowest effects on FPFTS. It experienced a modest decrease in both VOST and EEGTBG loop delay.

Network utilization vs.- various bandwidth ranges: network utilization indicates how much bandwidth is used in a specific time period. Thus, changing link bandwidth has a slight effect on network utilization. Fig. 1.9 indicates how bandwidth variances influence on network utilization. From this figure we conclude that there is a moderate rise for both FCFS and Delay-priority strategies, while there is a very slight rise for FPFTS. We run the FPFTS on average 9 times for variable bandwidth values from 10,000 (Kbps) to 20,000 (Kbps). Based on our observations, FPFTS could offload modules to the cloud data center on average 11% of simulator running, which causes having a more standard deviation. To put it another way, because of the randomness features of PSO, for each bandwidth value, FPFTS requires a simulator to run in different numbers to offload modules to the cloud data center.

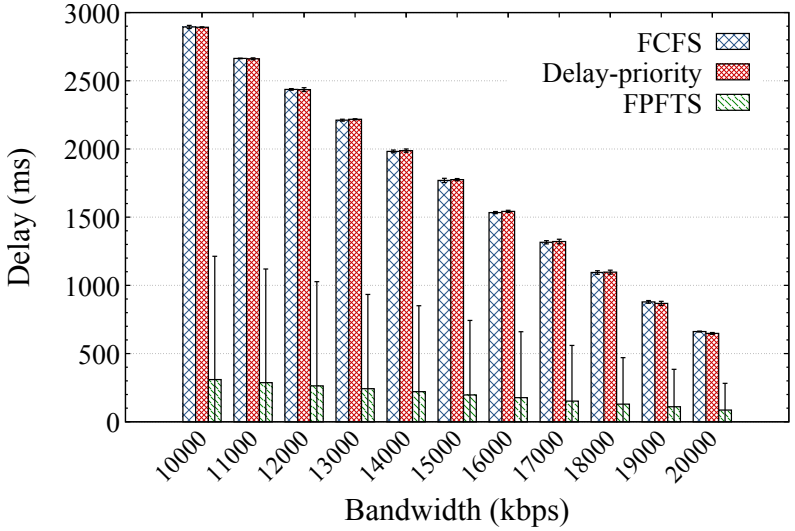
Comparing methods based on various link latency ranges

This experiment indicates the mobility-aware scenario of performance evaluation for various link latency thresholds. Link latency shows the total amount of time it takes to send data.

Application Loop delay vs.- various link latency ranges: In this experiment, we consider a situation in which 20 users moved from the first and third fog device to the second one. This experiment aims to study the effects of changing fog device link latency on application loop delay. Besides, the second fog device link bandwidth is 20,000 (KByte). In this way, Fig. 1.10a and Fig. 1.10b illustrate the effects of link latency on both VSOT and EEGBTG applications loop delay, respectively. By arriving 20 users, both FCFS and Delay-priority keep VSOT applications in the second fog device, and they only move EEGTBG modules to the cloud data center. Considering this experiment, we realize that changing link latency only affects on EEGTBG loop



(a) VSOT application



(b) EEGBTG application

Figure 1.8: Application loop delay in (ms) for a) VSOT applications, and b) EEGBTG applications for various bandwidth ranges in (Kbps) among Delay-priority, FCFS and FPFTS algorithms.

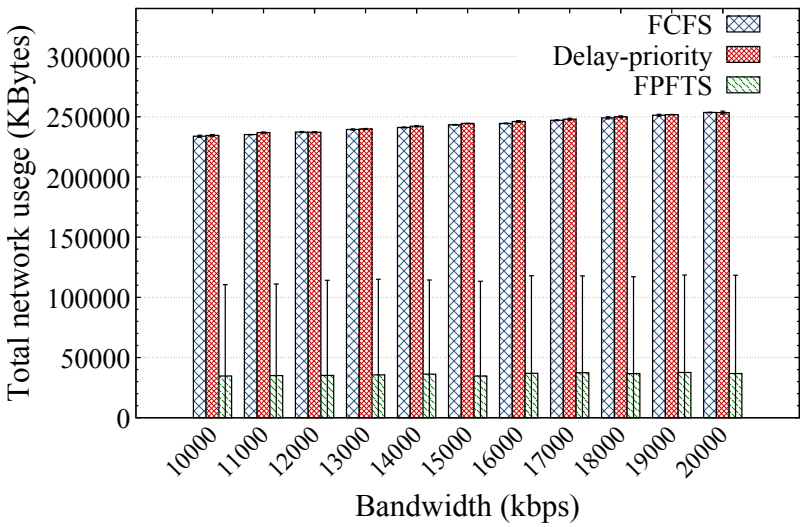


Figure 1.9: Total network usage in (KByte) for various bandwidth ranges in (Kbps) among Delay-priority, FCFS and FPFTS algorithms.

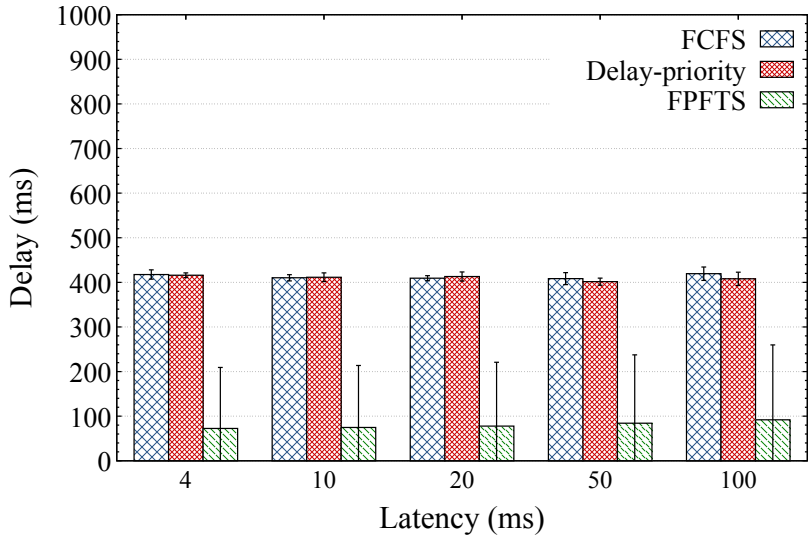
delay. The effect of a link latency from 4 (ms) to 10 (ms) on the EEGTBG loop delay is almost negligible, while one from 10 (ms) to 50 (ms) has almost a considerable effect. Moreover, a link latency ranging from 50 (ms) to 100 (ms), has a more considerable effect on the EEGTBG loop delay which then sees almost a sharp increase. During this period (see link latency 4 (ms) to 100 (ms) in the figures), the VSOT application loop delay remains constant. However, as FPFTS offloads modules to the cloud data center in case of fog device overloading, changing latency has negligible effects on both VSOT and EEGTBG loops delay for the proposed approach.

Network utilization vs.- various link latency ranges: Link latency has a considerable effect on network utilization. In Fig. 1.11, we evaluate this matter among the scheduling algorithms. In this figure, by arriving the 20^{th} EEGTBG user, both FCFS and Delay-priority algorithms move EEGBTG modules to the cloud data center, so considerably raises the network utilization. On the contrary, as FPFTS has the lowest amount of cloud data offloading, changing link latency has the smallest effect on it, and it increases network utilization marginally. We run FPFTS on average 16 times for variable link latency values from 4 (ms) to 100 (ms). As a result, we observe that FPFTS offloads modules to the cloud data center on average, 12.5% of simulator running, which causes

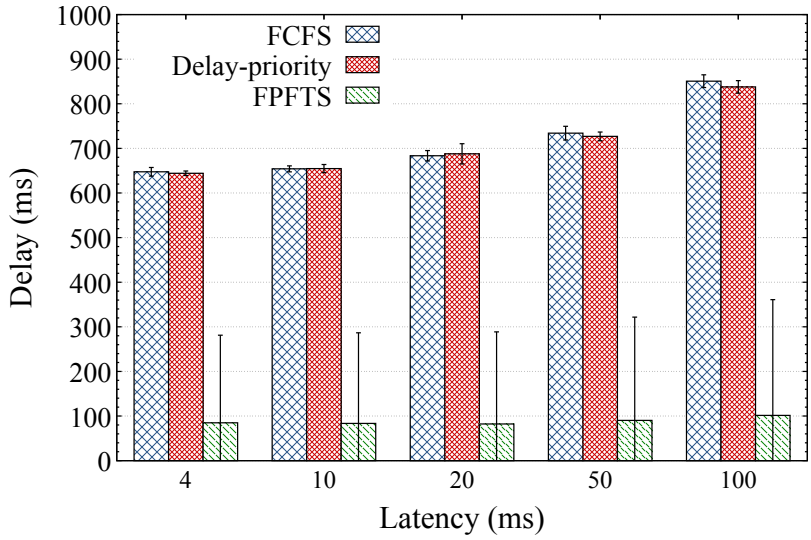
having a more standard deviation. It means for 16 times simulator running, FPFTS offloads modules to the cloud data center two times. These twice cloud data offloading processing selects a high application loop delay compared to the fourteen times FPFTS used fog devices. As there is a huge difference between them, standard deviation of the average of them are high. To put it another way, because of the randomness features of PSO, for each link latency value, FPFTS requires simulator to run in different numbers to offload modules to the cloud data center. On average it takes twice Cloud data offloading per 16 times simulator running.

1.4 Discussion

The major reason for developing fog computing is to reduce delay, network utilization, and amount of data transferred to the cloud data center for processing. Resource management plays an essential role in achieving these goals. The process of scheduling and allocating fog resources to users' applications is called *resource management*. Adopting proper task scheduling is the main challenge of fog computing, which causes improving efficiency and also user satisfaction. In this work, we consider application classification and user mobility features for devising the proposed fog task scheduler. Based on the scenario that we used, we only consider two types of user applications and scheduled them to run users' requests. We argue that the task scheduler should strike a balance between the mentioned two types of applications. It is of interest we indicate that FPFTS can use in different scenarios by considering different policies. We can place some instances of the FPFTS method to locally manage the service requests for various applications, which can enhance abilities such as computing capacity, reliability, and availability in the network. In this paper, we paid close attention to the user mobility, and the designed algorithm mainly addresses the task scheduling on moving applications. The extended case can consider local scheduling. FCFS and Delay-priority methods do not consider the computing capacities of the other fog devices, and the difference is that FCFS does not consider application classification while Delay-priority does. We map and run the instances of applications which arrived at the first and third fog device, respectively. We put one instance of FPFTS in the fog gateway, which is close to the second fog device, and schedules the tasks to all fog devices to use the computing capacities of fog layer resources in a fog region. Only in case of fog device overloading, *Offloading agents* moves data to the cloud data center. Although we used a small scale network scenario in



(a) VSOT application



(b) EEGBTG application

Figure 1.10: Application loop delay in (ms) for a) VSOT applications and b) EEGBTG applications for various latency ranges in (ms) among Delay-priority, FCFS and FPFTS algorithms.

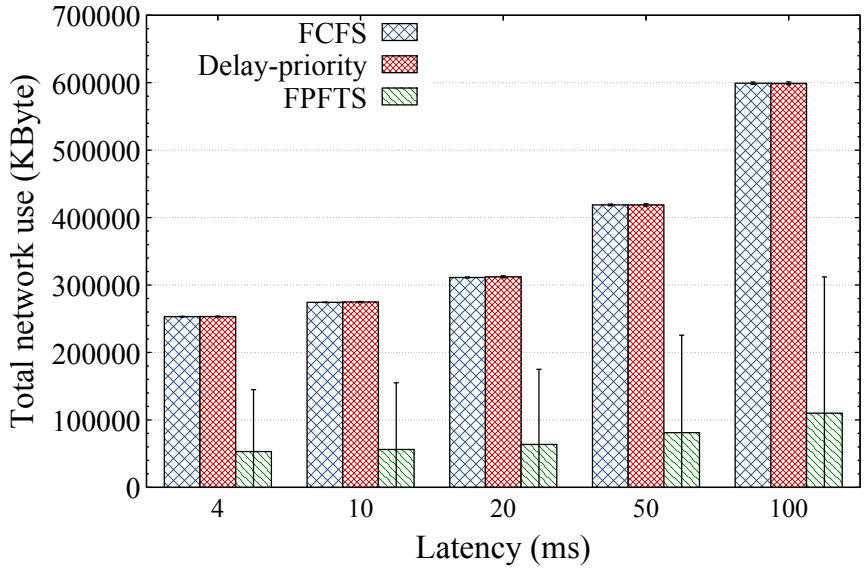


Figure 1.11: Total network usage in (KByte) for various latency ranges in (ms) among Delay-priority, FCFS and FPFTS algorithms.

the experiments, we argue that we can use FPFTS in large scale networks with the aid of peer-to-peer (P2P) technology. P2P technology could utilize distributed systems like Web Operating Systems [35], Semantic P2P Grids [36], Cloud environment [37], and also fog computing [38], and Internet of Things environment [39]. It has two major benefits. First, FPFTS instances can collaborate to assign tasks to the fog devices in the other fog regions. Second, in the case of fog device overloading, we can use a fog device to data offloading either in the same fog region or in the different fog regions.

In this work, we only consider fog device to cloud data center data offloading. The reason is that the resource capacities of the fog devices are much lower than the cloud data center, and the approach requires checking the free space of the other fog devices. It needs some suitable strategies indeed, which causes the proposed method more complicated [30]. Moreover, VM live migration is another good approach to overcome the computational limitation of the fog devices to improve the load balancing of the network. Because it consumes a large number of CPU cycles and network bandwidth, it is considered as a resource-intensive strategy. Even though it brings some advantages in terms of load balancing, transparent mobility, pro-active fault tolerance, and green computing, it requires sufficient network bandwidth capacity to move

the virtual machines from one fog device to another [40]. If we consider fog device to fog device data offloading, and VM live migration to the FPFTS, it will be changed as follow: first, the scheduler should be aware of the real-time changes in other fog devices and the links. It includes the changes in the available computing capacities, links latency and the availability of links bandwidth. The instances of the scheduler in different virtual organizations should collaborate to find the best fog device for fog device to fog device data offloading/VM live migration. Second, the instances of the scheduler should decide which fog device is the most suitable target for data offloading/VM live migration. This decision is made by the availability degree of the fog devices computing capacity, link latency, and link bandwidth.

The IoT is the foundation of so many states of the art technologies. Adopting a better resource management strategy in relevant technologies increases their performance and user satisfaction. For instance, the Industrial Internet of Things (IIoT) [41] presents novel applications for powerful industrial systems. It is the implementation of IoT technology in manufacturing systems. IoT enables interaction and cooperation of physical objects such as sensors, machines, cars, buildings, and other items to reach specific goals in the industry. Fog computing provides near user processing capability, which has a low delay to actuators, sensors, and robots in the manufacturing industry. Besides, as most of the time, industrial big data are unstructured, near user resources perform the process they need before sending it to the cloud data center[42]. As IIoT machines must connect and communicate in a real-time manner, we can use a proper fog task scheduling approach to reduce application delays. To take another example, we can point to smart cities [43]. The smart city uses different types of Internet of Things devices to collect data and then use them to manage the resources and services efficiently. To do so, it requires related skills and the capacity to make the users satisfied. In IoT based smart cities, the devices (street cameras for observation street traffic, sensors for urban transportation systems, etc.) interconnect and communicate with each other through a network infrastructure [44]. *Smart surveillance'* is another usage of smart cities that provides the ability to monitor human activities. The strength of real-time video analysis in surveillance applications is the necessity of this goal [45]. To this end, adopting a proper fog task scheduling approach decreases the network utilization and application loop delay. Cognitive IoT (CIoT) is another example of the usage of IoT networks in relevant cutting-edge technologies. Nowadays, so many healthcare, agriculture, environment monitoring, and smart metering scenarios are implemented by using Cogni-

tive Networks (CNs) [46]. The CIoT is the combination of current IoT and cognitive and cooperative mechanisms to achieve intelligence. Intelligence sensing is a novel research field. We implement the Intelligence sensing information to make people able to contribute data samples for CIoT captured by sensors, which often performs by smartphones. As in CIoT, sensing information shortages/absences results in loss of human life and social unrest, it needs a sufficient fog resource management approach [47][48].

1.5 Conclusions

In this paper, we proposed FPFTS, a mobility-aware fog task scheduler designed to efficiently assign IoT user's tasks to fog devices. FPFTS considers the computing capacity of the resources as well as user's tasks requirements as input parameters and implements an algorithm jointly based on PSO and fuzzy theory to assign applications' tasks to fog resources. We apply FPFTS within an IoT based scenario. To evaluate the proposed approach, we assess the performance of the FPFTS using the iFogSim simulator, considering different numbers of moved users and several configurations for fog-device link bandwidth and link latency. Results show that FPFTS outperforms both FCFS and delay-priority algorithm. Compared against the former, FPFTS improves by 85.79% (resp. 87.11%) delay-tolerant (resp. delay-sensitive) application loop delay and network utilization by 80.37%. On the other hand, FPFTS improves the performance of delay-priority algorithm, in terms of delay-tolerant application loop delay by 86.36%, delay-sensitive application loop delay by 86.61%, and network utilization by 82.09%.

Chapter 2

FUPE

Fog computing is a paradigm to overcome the cloud computing limitations which provides low latency to the users' applications for the Internet of Things (IoT). Software-defined networking (SDN) is a practical networking infrastructure that provides a great capability in managing network flows. SDN switches are powerful devices, which can be used as fog devices/fog gateways simultaneously. Hence, fog devices are more vulnerable to several attacks. DDoS is an abbreviation for Distributed Denial of Service, and it refers to a method in which cybercriminals flood a network with so much malicious traffic that it is unable to operate or communicate regularly. Motivated by this, in this section, we apply SDN concepts to address TCP DDOS attacks in IoT-Fog networks . We propose FUPE, a security-aware task scheduler in IoT-fog networks. FUPE puts forward a fuzzy-based multi-objective particle swarm Optimization approach to aggregate optimal computing resources and providing a proper level of security protection into one synthetic objective to find a single proper answer. We perform extensive simulations on IoT-based scenario to show that the FUPE algorithm significantly outperforms state-of-the-art algorithms. The simulation results indicate that, by varying the attack rates, the number of fog devices, and the number of jobs, the average response time of FUPE improved by 11% and 17%, and the network utilization of FUPE improved by 10% and 22% in comparison with Genetic and Particle Swarm Optimization algorithms, respectively.

2.1 Introduction

Recent advances in smart devices and communication technologies are fueling the Internet of Things (IoT) paradigm [49], which is characterized by the pervasive presence around people of (interconnected and uniquely addressable) things, able to measure and modify the environment and communicate with each other. Accordingly, technology leaders, governments, and researchers are putting serious efforts to develop solutions enabling wide IoT deployment in order to support a variety of applications impacting a number of different scenarios (e.g. healthcare [7], industry 4.0 [50], smart home[16]). Often, resource-constrained things are required to interact with service platforms in order to benefit from their computation, storage, and networking capability on request. While on the one hand referring to cloud services is the natural choice [2], on the other hand, some classes of applications may suffer from the performance figures that cloud platforms may guarantee in terms of provided QoS (e.g. due to latency to reach far-away cloud data centers).

To address this issue, fog computing has been proposed [51], which provide the tools to mitigate cloud QoS shortcomings at the expenses of not having available virtually infinite resources of cloud data centers. In fact, fog computing is a cutting edge solution, which leverages near-user (possibly cooperating) edge devices (fog devices) rather than a single (far-away) cloud data center to supply computing services to IoT applications. [23]. Hence, fog infrastructures allow for supporting IoT application requirements to reduce both delay and network utilization [25]. Usually, cloud data centers still take part in composing the overall fog architectures, but are accessed only in case of need (i.e. when the fog nodes capabilities are not enough). In the context of fog computing, resource management is a challenging issue to be considered. Task scheduling is the major part of a resource management unit that aims to assign a set of tasks to fog devices [52]. To this end, the task scheduler—possibly located at different places in a fog architecture, such as fog gateways, fog devices, cloud gateway or cloud data center—decompresses the jobs into a set of (independent) tasks, and then it assigns them to the fog devices [26].

Besides their requirements in terms of QoS, IoT and related fog infrastructures are prone to security and privacy concerns due to the critical nature of the contexts the applications are deployed in and the generated data (e.g., smart home, healthcare etc.). These concerns potentially derive from low-cost hardware and software design choices of IoT devices (e.g. unsafe update mechanism, outdated component, etc.) or lack of adequate security protection [53]. Thus, malicious users have shifted the main target of daily-released malware,

now pointing to infect IoT services and make them unavailable, leveraging the manifold and significant weaknesses generated by the IoT context. IoT and fog devices are both susceptible to be hacked by malicious users [54]. These devices may be even incorporated in botnets, thus unwillingly participating to the attack after they have been compromised [55] (e.g., taking part to Distributed Denial of Service TCP DDoS attacks which aim at disrupting targeted server/service by overwhelming the target with a flood of malicious Internet traffic [56]). As the Software-defined networking (SDN) provides flexible network programmability and logically centralized control (through a global view of the network), this paradigm is able to provide the tools for effectively detecting and containing network security problems that recently is used in IoT-fog networks [57]. Indeed, SDN represents a good fit for the fog environment as fog devices and gateways have the capabilities to implement this paradigm [58].

A possible solution to detect and mitigate TCP DDOS attack is using firewalls deployed at fog gateways. With the firewall filtering out malicious requests, the scheduler can put away the attacker nodes. Firewalls filter the traffic by using some predefined rules and act like a checkpoint. Firewalls detect flood attacks in IoT-Fog networks by using different mechanisms such as a predefined threshold, specifying port addresses, or defining rules to filter protocols, ports, IP addresses, or network traffic. For instance, in a threshold-based firewall if the number of TCP SYN exceeds 10 (the justification can be applied), the firewall detects the node as an attacker. The major limitation of threshold-based approaches is that they do not provide the sensitivity and specificity required for precise classification. A threshold-based approach can not distinguish the flash traffic (i.e., sudden traffic from a legitimate source node to a particular destination node) from malicious traffic. Moreover, the difference between 9 and 10 is very smidgen; but a threshold-based firewall detects 9 as a benign request, and 10 as a malicious one. Threshold realization is a classic technique for firewall malicious behavior detection. However, different firewalls nowadays utilizes some cutting edge methods. Recent works on cutting-edge firewall use hybrid techniques to mitigate malicious traffic using pattern of machine learning models [59]. On the other side, they have several gaps especially for real-time stream traffic applications. A typical method is to use a regular scheduler plus a firewall. With the firewall filtering out malicious requests, a regular scheduling algorithm can schedule the remaining benign ones. While this approach is legitimate, in this section we investigate the feasibility and the performance of an integrated solution to meet the requirements

of real-time applications. In detail, we make use of fuzzy logic capabilities and consider security in the scheduler deployed at fog gateways. FUPE can be implemented in firewalls as well.

In accordance with the context above, intelligent and efficient solutions are required to detect and protect against TCP DDoS attacks in their IoT-fog networks [60]. However considering task scheduling efficiency and security poses a non-trivial challenge to task scheduling algorithms [61] that are required to strike a balance between these two distinct objectives. As fog environments have dynamic features in which the characteristics of elements change continuously, techniques to jointly preserve trust and security in such dynamic environments are required [62]. Since IoT-fog network provides a shared and distributed infrastructure for the users, establishing security for the fog devices must be considered during the scheduling process. Existing IoT scheduling algorithms consider efficiency and disregard security concerns of resources. Recently, more attention has been paid to security-aware IoT-fog task scheduling algorithms. However, these studies did not touch TCP DDOS attacks issues. Besides, as security and efficiency are considered separately in their frameworks, these studies are not suitable for real-time applications. Motivated by these challenges, in this section we integrate TCP DDoS attack detection techniques in a scheduling algorithm. To the best of our knowledge, FUPE is the first attempt to perform the security-aware task scheduling in IoT-fog networks considered TCP DDOS attack through a multi objective optimization algorithm. FUPE finds the best place for users' applications based on security consideration and resource performance.

2.1.1 Contribution of the paper

In this section, we propose a security-aware task scheduler algorithm tailored for IoT-fog networks called *FUPE*, which provides a proper security level. This scheduler considers the dynamic behavior of distributed systems and uses two trust degrees obtained from Threshold Random Walk with Credit-Based connection rate-limiting (TRW-CB) and Rate Limiting algorithms—i.e. one of the most prominent source-based attack mitigation strategies available [63]—to deal with TCP DDoS attacks [64]. Our proposal jointly merges security issues and task scheduling with the aid of multi-objective PSO [65] and multi-criteria decision-making [66] algorithms. FUPE solves multi-objective task scheduling problem to jointly maximize security and efficiency of quality of services (QoS) such as delay in IoT-fog networks. It also leverages SDN programmability and centralized network features to enforce

attack mitigation mechanisms against the TCP DDoS attacks: in case of detection of requests from devices identified as malicious, the requests are not taken into account. In other words, FUPE addresses security issues inside the scheduler. More specifically, FUPE assigns the most suitable resources to the tasks based on the current status of the resources and incoming tasks. FUPE focuses on assigning applications' tasks among fog devices, considering the trustworthiness of fog devices, and users' devices. Thus, FUPE strikes a balance between efficiency and security objectives.

We evaluate FUPE leveraging Matlab [20], a widely adopted and well-known programming platform. To this end, we test FUPE against two well-known metaheuristic approaches *Genetic algorithm (GA)* and *particle swarm optimization (PSO)* strategies, by varying attack rates, number of fog devices, and number of jobs. The obtained results indicate that our proposal outperforms GA and PSO approaches in terms of Response time, and network utilization. In detail, FUPE improves by $\approx 14\%$ Average Response time, $\approx 49\%$ Maximum Response time, while concerning network utilization, FUPE improves this metric in by $\approx 16\%$, on average. Thus, the contributions of the paper are listed below.

- The paper presents an architecture which integrates the SDN and Fog technology in the presence of the IoT services and tackles the TCP DDoS attacks.
- We design FUPE, a security-aware task scheduler algorithm, dealing with TCP DDoS attack.
- FUPE combines fuzzy logic and a multi-objective particle swarm optimization (MOPSO) algorithms supporting secure IoT task demands in the network.
- FUPE uses Mamdani fuzzy inference system helping the scheduler using a relationship between the metrics and priority of the IoT demands [67].
- FUPE implemented and tested on various scenarios and compared against the state-of-the-art.

2.2 Related work

In this section, first, we list fog task scheduling approaches (Section 2.2.1). Then, we explain the approaches focusing on TCP DDOS attack in IoT/SDN networks (Section 2.2.2), and finally, we explain the security-aware fog task

scheduling algorithms (Section 2.2.3). In the end, Table 2.1 summarizes the comparison of these categories.

2.2.1 Fog task scheduling approaches

In this subsection, we review some of the fog task scheduling algorithms. Bittencourt et al. [14] put forward *three* scheduling approaches, namely concurrent, first-input first-output (FIFO) and delay priority in IoT-Fog network. In all of these methods, the scheduling algorithms run in fog devices (FD) such that when a new request arrives at the FD, the scheduler decides either runs it on the FD or send it to the cloud data center. These algorithms utilize resource availability and CPU capacity for decision-making. They validate their strategies in terms of application loop delay, network usage, and the number of applications moved to the cloud data center. Afterward, Mahmud et al. [15] introduced a latency-aware application scheduling algorithm in FD to assure deadline-satisfied service delivery for users' demands. They use deployment time, deadline, and some fog devices as the nature of observation and reduces the applications' service delivery latency. Similarly, Bitam et al.[20] performed a bio-inspired method using the Bee life scheduling algorithm to discover an optimal trade-off between CPU execution time and allocated memory of the users' requests in an FD and provides low execution time.

Gill et al. [16] designed a task scheduling in FDs using particle swarm optimization (PSO) algorithm to mutually reduce network bandwidth, response time, latency, and energy consumption. Most recently, the same authors of this paper (i.e., FUPE) [68] proposed FPFTS, a meta-heuristic method merging PSO and fuzzy methods to tackle the fog task scheduling problem. They implement and test FPFTS on different scenarios like on various mobile devices and FD characteristics when faced with various link delays. Nevertheless, none of these approaches except FPFTS, mutually consider the FD and application task characteristics to find efficient fog scheduling methods. The proposed FUPE comparing with FPFTS has *three* main differences. First, FUPE uses the remaining amount of RAM and CPU capacity for task scheduling. Second, FUPE uses the trust degree of FDs and users' devices for the scheduling that enables it to select trustworthy devices in the exact time of utilizing the devices. Third, FUPE imposes a balance between FDs' trustworthiness and efficiency with the aid of multi-objective PSO (MOPSO) features. Sun et al. [19] devised a two-level application scheduling approach in fog-IoT networks called RSS-IN. RSS-IN schedules the applications among the fog regions and then performs scheduling within the same fog region. This approach achieved

proper stability and end-to-end delay through a non-dominated sorting genetic algorithm (NSGA-II).

2.2.2 TCP DDOS attack in IoT/SDN approaches

In this part, several surveys summarize recent attack and defense techniques targeting TCP DDoS attack, especially TCP DDOS attack in IoT and SDN [60, 69, 70]. In particular, Evmorfos et al. [71] designed a lightweight technique to detect TCP DDOS attack in IoT network. To this end, the authors apply the long-short-term-memory and the random neural network to develop the predictive model mitigating TCP DDOS attack detection. In another work, Kolias et al. [72] designed Mirai-like bots as the IoT-based TCP DDoS attacks real examples. The Mirai is a malware that causes severe disruptions in the IoT network services owing to its sophisticated spreading mechanism. Mirai uses bot instances that attack the target computational resources with TCP DDOS attack.

Some other related works mainly focus on TCP DDOS attack/defense in the SDN network. For example, Yan et al. [73] proposed a TCP DDOS mitigation multi-level framework with the aid of SDN to control the Industrial IoT network. To accamplish this, their method monitors the SDN gateways, gather the network traffic data and detect the TCP DDoS attack. To confirm their method, they evaluate it based on time-delay of normal users in the context of no defense mechanisms is used, and in the context of the presence, SDN to mitigate the attacks. Alternatively, Kumar et a. [74] proposed, SAFETY, an approach to detect and mitigate TCP DDOS attack in SDN networks. SAFETY considers destination IP and a few attributes of TCP flags. The authors implement SAFETY as a module in the Floodlight OpenFlow controller. Mohammadi et al. [64] presented an approach to mitigate TCP DDOS attack in SDN called *SLICOTS*. SLICOTS supervises the ongoing TCP connection attempts and detects malicious hosts. It utilizes the programmability features of the SDN to tackle the TCP DDOS attack. Later on, they [75] proposed, SYN-Guard, a countermeasure approach to detect and prevent TCP DDOS attack in an SDN network. They implement SYN-Guard as an extension module on the SDN controller to monitor the incoming TCP connection attempts. SYN-Guard is evaluated based on average response time. Recently, Zhou et al. [76] developed a three-way approach to mitigate TCP DDOS attacks in industrial IoT-fog networks. This approach considers the prevailing edge devices' capacities, such as local proxy servers, firewalls, IDS, etc. They take TCP DDOS attacks into consideration in the implementation and show that their method has

a low detection time. In this paper, unlike the solutions above in this category, FUPE aims to detect and mitigate TCP DDOS attack by mutually considering malicious users' devices and compromised fog devices in task scheduling. It enhances the balanced task scheduling on available and benign FDs.

2.2.3 Security-aware fog task scheduling approaches

This part is devoted to summarize the fog task scheduling approaches considering security challenges. Sujana et al. [61] designed a platform that helps to achieve convincing secure scheduling relying on a stochastic behavior of workflow in cloud/fog environment. They emphasize direct trust and indirect reputation metrics for task scheduling to preserve security in the cloud/fog. This scheme assures that the assigned resource only retain the trusted VMs. In their implementation, they consider a distinct security guaranteed level to specify the percentage of security ratio for each VM. Also, they address various related attacks, such as VM theft, VM escape, hyper jacking, data leakage attacks. Besides, Daoud et al. [77] offered a security model for fog resource management and task scheduling. Their security model integrates access control to ensure secure cooperation between diverse resources and tasks. The scheduler assigns user tasks to the resources based on the trust degree of each users' devices and the availability of resources. They compute the trust degree of users' devices based on the behaviors of each user, and the access control considers these degrees of trust to permit trustworthy users to access the FDs.

Lately, Auluck et al. [78] introduced a secure fog task scheduling algorithm considering deadline and security constraints of IoT applications. This algorithm assigns *three* security labels to the tasks as follow, namely private, semi-private, and public. Similarly, it assigns *three* security labels to the resources. The scheduler assigns tasks to the resources based on these labels and the application deadline. It uses jobs deadlines, spare capacity available on the resources and job privacy for assigning tasks to them. Unlike these methods, in this study, FUPE uses the available amount of CPU and RAM space of FDs, tasks CPU requirements, and also trust degree of devices altogether as the nature of observations. In this way, the scheduler is aware of the current state of resources' trust degrees.

2.2.4 Evaluating of the background methods

FUPE, differently from the works by Gill et al. [16] and Li et al. [79], makes use of fuzzy logic in the MOPSO fitness function. These works utilize the

weighted sum approach to prioritize the objectives in the PSO fitness function. RSS-IN [19] takes the distance between the requester and the fog regions into account. Unlike this work, FUPE considers link bandwidth to reduce response time. In this paper, unlike our former work [68] in which we only considered efficiency objective, we take security and efficiency objectives into consideration. The works by Sujana et al. [61], Daoud et al. [77], Rjoub et al. [80], Li et al. [79], and Auluck et al. [78] did not propose multi-objective optimization solutions. These works proposed single objective optimization algorithms for response time minimization with security constraints. They utilized optimization models to define security levels for tasks and resources. Based on the tasks' security requirements, these algorithms assign resources to applications' tasks. The work by Gill et al. [81] has presented a single-objective resource management approach that offers self-protection against security attacks. Different from these works, FUPE utilizes a multi-objective algorithm. The work by Bittencourt et al. [14] did not consider the processing capacities of the other fog devices. Different from this work, FUPE considers the processing capacities of the other fog devices in the same fog region. Unlike the work by Mahmud et al. [15] which focused on reducing latency, FUPE main aim is reducing latency and network congestion. Finally unlike the work by Bitam et al. [20] that provides low scalability, FUPE makes use of the P2P structure for nodal collaboration to provide high scalability.

The evaluated background methods in the mentioned works focus on various IoT scenarios except [81, 79, 80] which use cloud task schedulers. All the mentioned works considered security issues except [14, 15, 20, 16, 68, 19], which relied on the security unit. Although the works [78, 77, 61] have touched the security challenges, they are susceptible to be downed by TCP DDOS attack. Unlike all the mentioned works, FUPE integrates TCP DDoS attack detection technique in a scheduling algorithm. This scheduler is more reliable for Scheduling real-time applications. Considering security issues in the implementation of task scheduler has two important advantages: first, it prevents the computation of malicious requests which may lead to overload the fog resources and as a consequence cause the starvation for benign requests. Second, it allows giving priority to the requests from users exposing legitimate behavior. Moreover, as FUPE utilizes a multi objective optimization algorithm, it prioritizes efficiency and security as the two different objectives in the scheduler.

Table 2.1: Comparison of existing scheduling approaches against FUPE.

Refs.	Algorithms	Tool	Observations	Security	Advantages	Disadvantages
[14]	Concurrent/FIFO/Delay-priority	iFogSim	Resources availability and CPU capacity	No	+ Movement based scheduling at fog device level	- High time complexity - Relying on security unit
[15]	Heuristic	iFogSim	Deployment time, deadline, Number of fog devices	No	+ latency-aware IoT application + Reducing the amount of deployment time + Deals with varying application	- No Real case study - Relying on security unit
[20]	Bees swarm	C++	CPU execution time, Allocated memory	No	+ Managing allocated memory + Low CPU execution time	- Only fog devices are used - Static scheduling - Relying on security unit
[16]	PSO	iFogSim	Response time, energy, latency, and network bandwidth	No	+ Energy + Latency and response time + Network bandwidth	- No reliability assurance - Relying on security unit
[19]	NSGA-II	MATLAB	Requester distance to the fog region Service completion time Fog device's capability Fog device's reliability	No	+ Low latency + Improved stability	- Not suitable for complex topology - Relying on security unit
[81]	Machine learning	Java	Anomaly-based/Signature-based detector Execution time, Execution cost	Yes	+ delivers secure cloud services	- Centralized feature and not suitable for IoT
[79]	PSO	CloudSim	Execution cost, deadline and risk rate	Yes	+ Risk rate constraint of workflow in scheduling	- Centralized feature and not suitable for IoT
[80]	Multi-criteria task priority	CloudSim	Resources degree of trust Tasks priority level	Yes	+ Applying resources degree of trust in scheduling	- Centralized feature and not suitable for IoT
[61]	Stochastic	CloudSim	Service Level Agreement (SLA), The earliest execution start time of tasks	Yes	+ Different levels of trust	- Lack of defense mechanisms to deal with attacks - relying on feedback collected from users
[78]	Multi-criteria task priority	iFogSim	job deadlines spare capacity available on the resources Job privacy	Yes	+ Assigning security tag to the resources.	- Lack of defense mechanisms to deal with attacks
[77]	Multi-criteria task priority	iFogSim	jobs trust level jobs arrival time resources availability	Yes	+ Assigning security tag to the resources.	- Lack of defense mechanisms to deal with attacks - relying on feedback collected from users
[68]	Hybrid (PSO-Fuzzy)	iFogSim	Total amount of fog devices' ram size, Total amount of fog devices' CPU capacity, Total amount of fog-devices' link bandwidth, Applications' Tasks CPU need	No	+ Mobility-aware scenario + Considering total computing capacity of resources + Low application loop delay/network utilization	- No fog gateways fault tolerance
FUPE	Hybrid (MOPSO-Fuzzy)	Matlab	Available fog-devices' processing capacity, Available fog-devices' RAM size, Available fog-devices' link bandwidth Applications' Tasks CPU need Trust degree of nodes	Yes	+ Security-aware scenario + Considering available computing capacity of resources + Low average response time/maximum response time + Low network utilization	- No SDN switches fault tolerance

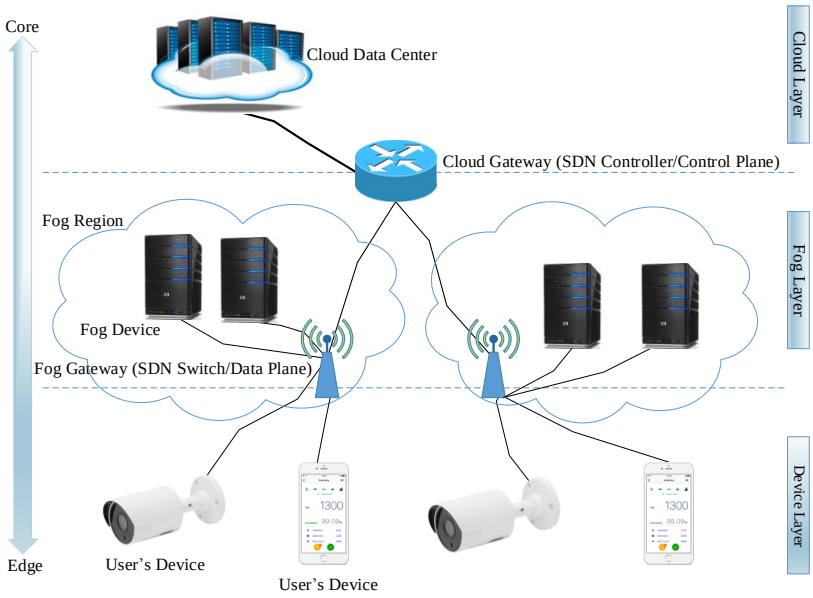
2.3 Proposed approach

In this section, we detail the proposed FUPE. To this end, first we describe the reference architecture in Section 2.3.1. Then, we introduce the security-aware task scheduling problem we address and the resulting organization in modules we adopt for our proposal in Section 2.3.2. In Section 2.3.3, we give an account of TRW-CB and Rate Limiting techniques. Finally, we detail FUPE in Section2.3.4.

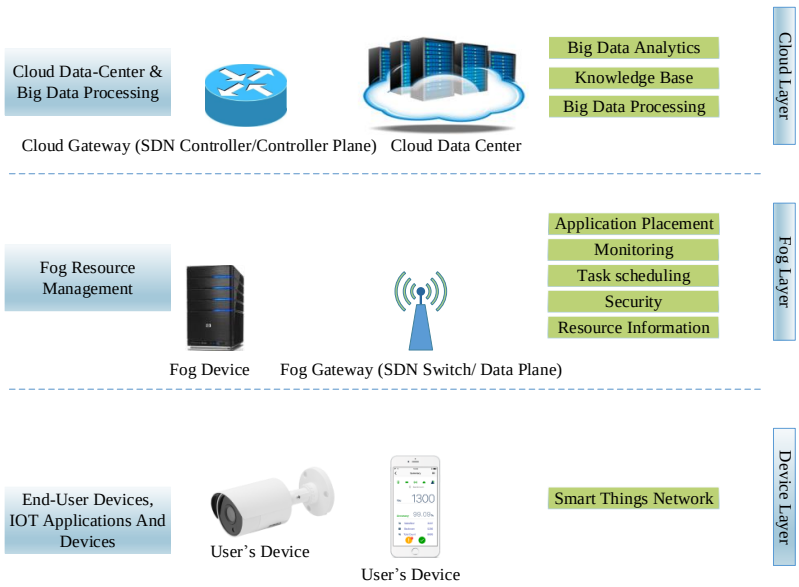
2.3.1 Reference architecture

In this paper, we present a three-layer structure (i.e. consisting of cloud, fog, and IoT device layers) as that adopted in previous works [82, 24]. The device layer consists of users' devices that send requests to fog devices in order to benefit from their computing resources. The fog layer is composed by a set of fog devices which are placed at the edge of the access network. Fog devices are grouped in fog regions: when a fog device becomes overloaded, it can offload requests to other fog devices *fog-to-fog* offloading) inside the same fog region. Finally, at the cloud layer virtually unlimited computing resources are organized in cloud data centers and are available to execute tasks which are passed from by the fog layer (*fog-to-cloud* offloading). Fig. 2.1 presents the architecture of the proposed approach. In our proposed architecture, the fog devices are clustered into virtual organizations (with each of them representing a fog region), and they are interconnected by SDN switches. The central controller of our scheme is located on the Cloud gateway. In this work, we consider the fog gateway and the cloud gateway as SDN switch and SDN controller, respectively.

Fog gateways play a central role as the instances of FUPE are implemented on them. The FUPE instances in fog gateways act as broker [83, 11] between users' devices and fog devices. In fact, they are the interface between users' applications in IoT device layer and resources in fog layer. In a scenario where some of the devices/nodes may take part to attacks against the fog infrastructure, the fog gateways collaborate with the cloud gateway to calculate the trust degree of the nodes for task scheduling. Indeed, TCP DDOS attacks generate a waste of resources, which may lead to denial of service. Accordingly, hereinafter we refer to these attacks as *malicious requests*.



(a) Three-tier IoT-Fog-Cloud architecture.



(b) Task-wise architecture.

Figure 2.1: The considered FUPE architecture.

2.3.2 Problem statement and proposed solution

Security-aware task scheduling is a critical part of IoT-fog networks and significantly impacts the performance of the overall system. In fact, it is a challenging NP-hard problem owing to the large-scale, dynamic, and heterogeneous architecture it relates to[84]. Indeed, users' devices and applications can join and leave the system in a dynamic manner and are both susceptible to be hacked.

The problem we address in this paper consists in efficiently and effectively assigning applications' tasks to fog devices, such that the security level constraints are met. Since various security threats are a big concern of IoT-fog networks, it is mandatory to deploy security mechanisms to protect security-critical users' applications executing on fog devices from being attacked. The aim of security-driven task scheduling is assigning tasks generated by trustful users' applications to the computing resources available at the trustworthy fog devices. In other words, applications which have an adequate level of trustworthiness are decomposed into a set of tasks to be assigned to the fog devices that have proper trust values and adequate computational capacity.

Our solution implements security-aware task scheduling at fog gateways via FUPE and benefits from SDN advantages. SDN infrastructure provides an efficient solution to mitigate security challenges in IoT-fog networks thanks to advanced network programmability, dynamic flow control, and network monitoring through centralized visibility. It enables administrators to automatically manage the entire IoT-fog network flexibly and dynamically (e.g. instantly block network traffic anomaly whenever malicious activity is detected) [85] [86] [87]. As shown in Fig. 2.2, our proposal consists of *five modules*: *Connection Logger*, *Request Validator*, *Job Decomposer*, *Task Assigner*, and *Forwarder*. Each module provides the functionalities as detailed in the following.

- *Connection Logger*: This module monitors the connection attempts from users' devices/ fog devices, logs them, and provides the cloud gateway with this logged information. Based on this information, the cloud gateway updates the *Credit* and *Rate* values for each user's device/ fog device, i.e. the counters related to TRW-CB and Rate Limiting techniques, respectively, which indicate the trustworthiness likelihood ratio of the connection attempt requesters. Finally, these values are sent on request to both Request Validator module and Task Assigner module of the fog gateways.

- *Request Validator*: Authentication is one of the major subjects in the

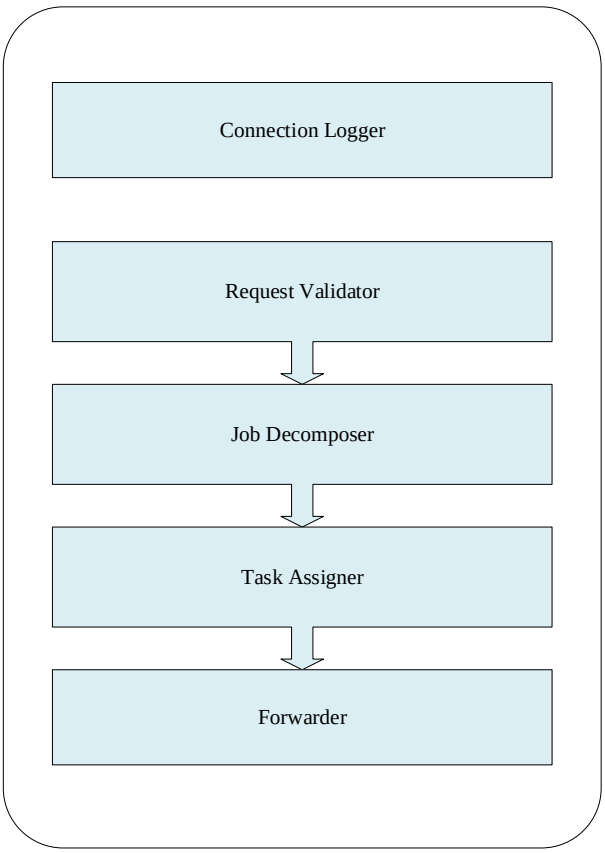


Figure 2.2: Modules composing the proposed solution.

security field. It is the process of verifying the identity of a node (fog device/user's device) that is a prerequisite to allowing access to the fog layer/ cloud layer resources. In this work, we assume the nodes have already been authenticated, and the authentication is not in the area of this research paper. Request Validator module is in charge of validating only users' devices. Upon the fog gateway receives a request from an user's device, this module first checks the validity of the requester. To this end, it asks the Cloud gateway to obtain the *Credit* and *Rate* values for the requester. Then, the module leverages the returned information and implements a Mamdani fuzzy inference system as shown in Fig. 2.5, and Table 2.2. In detail, this module uses the outcome of *security* objective to determine whether the requester is benign or it is an attacker. If the requester is benign then this module passes the job to the Decomposer module. Otherwise, this module rejects the request.

- *Job Decomposer*: This module receives the application jobs sent by the Request Validator module and then decomposes it into multiple separate tasks. After decomposition, the tasks are independent of each other and are ready to process by the assigner module.

- *Task Assigner*: This module assigns the application's tasks originated from the validated users' devices to the fog devices, with the aim of increasing performance securely. To this end, upon receiving application's tasks from the decomposer module, it first gathers required information regarding fog devices computational features together with their security conditions. Therefore, it sends a request to the Cloud gateway and asks the information about the *Credit* and *Rate* values for all fog devices in the fog region. The fog gateways have the information about the fog devices located inside the fog regions. Then, it performs a secure scheduling as described in section 2.3.4. This module uses the outcome of both *security* and *efficiency* objectives to determine a secure and proper fog device to execute the validated application's tasks of user's device. This module provides the application's tasks and designated fog device's ID to the forwarder module.

- *Forwarder*: The forwarder module implements the features of the data plane of the SDN paradigm and, accordingly, provides a set of flow tables that are managed by the controller (Cloud gateway) [88]. These flow tables include flow entries containing a header to match the incoming packets and a set of actions to apply to match packets. Because

preventing an attack is done at the fog gateways, the controller proactively installs flows on the forwarder modules. If a request is recognized as a legitimate by the Request Validator (and the Task Assigner), then the forwarder module forwards it toward designated fog device concerning the flows which the controller has installed on the forwarder module. This ensures more efficiency, because limits the required intervention of the controller.

2.3.3 TRW-CB and Rate Limiting concepts

Based on TRW-CB, the probability of a successful connection attempt is much higher for a trustworthy node than a malicious node. TRW-CB calculates the difference between TCP SYN and TCP ACK packets, just like in a TCP DDOS attack, where a malicious node creates a large number of half-open TCP connections on the targeted nodes. To counteract TCP DDOS attacks, Rate Limiting recognizes a node as malicious when it sends a large number of TCP SYN packets in a short period of time. [87]. Whenever the SDN switch receives a TCP packet, it checks the packet against a list of recently contacted fog devices called the *working set*. If the TCP packet request belongs to a fog device presented in the *working set*, then it forwards the request normally. Otherwise, it enqueues the request in another data structure called the *delay queue*. FUPE separates pairs of working sets and delay queues for every fog device. These connection-based techniques are scalable and can support a large number of rules which are suitable to be used in rule-based fuzzy techniques. It is illustrated in the literature that these techniques, on the one hand, process only a small fraction of the total network traffic; and on the other hand, they attain the same accuracy by inspecting every packet in SDN networks [88, 89, 87]. Rate limiting and TRW-CB do not create additional overhead at the network level. So, our approach does not add a packet or header to the network. It only monitors the network traffic to evaluate the received packets; then decides whether to block them or let them pass.

Figs. 2.3 and 2.4 indicate the process diagram of TRW-CB and rate limiting approaches, respectively. Alg. 2 and Alg. 3 show TRW-CB and Rate limiting algorithms as well.

Algorithm 2 TRW-CB

Output: $Credit$

```
List:= [];
if packet.type!= TCP then
    return;
end if
if packet.flags != SYN or ACK then
    return;
end if
if packet.flags = SYN and not SYNACK then
    if packet.source not in List then
        Install flow for forwarding direction;
        List := packet;
        Else
         Malicious_Counter++;
    end if
end if
if packet is ACK and packet source in list then
    Install flow for backward direction;
     Safe_Counter++;
end if
percentage:= The percentage of normal connections;
Map the percentage to the credit fuzzy set;
return the credit value to the fuzzy inference engine
```

Algorithm 3 Rate limiting

Output: $Rate$

```
if packet.type != TCP then then
    return;
end if
if packet.flags = SYN then
    Install flow for forwarding direction;
    Counter++;
end if
if packet.flags = SYNACK then
    Install flow for backward direction;
end if
if request not in working set then
    delayqueue := request;
    Move the new request from the delay queue to the working set;
end if
percentage:= The percentage of connection attempts;
Map the percentage to the Rate fuzzy set;
return the rate value to the fuzzy inference engine;
```

2.3.4 FUPE: proposed Security-aware Scheduler

As aforementioned, the role of the assigner module is to find the most proper fog device for processing a requested task that has been previously validated by *request validator* module. The designated fog device to the application's tasks should be secure and also has a suitable computing capacity. In this work, we assume both the users' devices and compromised fog devices can perform TCP DDOS attacks against other fog devices. Moreover, we assume the behavior of a benign node can change to become an attacker. We apply TRW-CB in FUPE due to its continuous monitoring ability to detect malicious network traffic. FUPE can instantly detect and block anomalies in network traffic when malicious activity happens. To achieve this, FUPE uses a multi-objective PSO algorithm to increase the efficiency together with security. Multi-objective optimization solutions [90] fall into two main categories: *(i)* those, defining the objective functions separately and combine them by using the weighted sum of them; *(ii)* those, using a single objective function and aggregate the distinct objectives into one synthetic objective by using Pareto Optimality. In

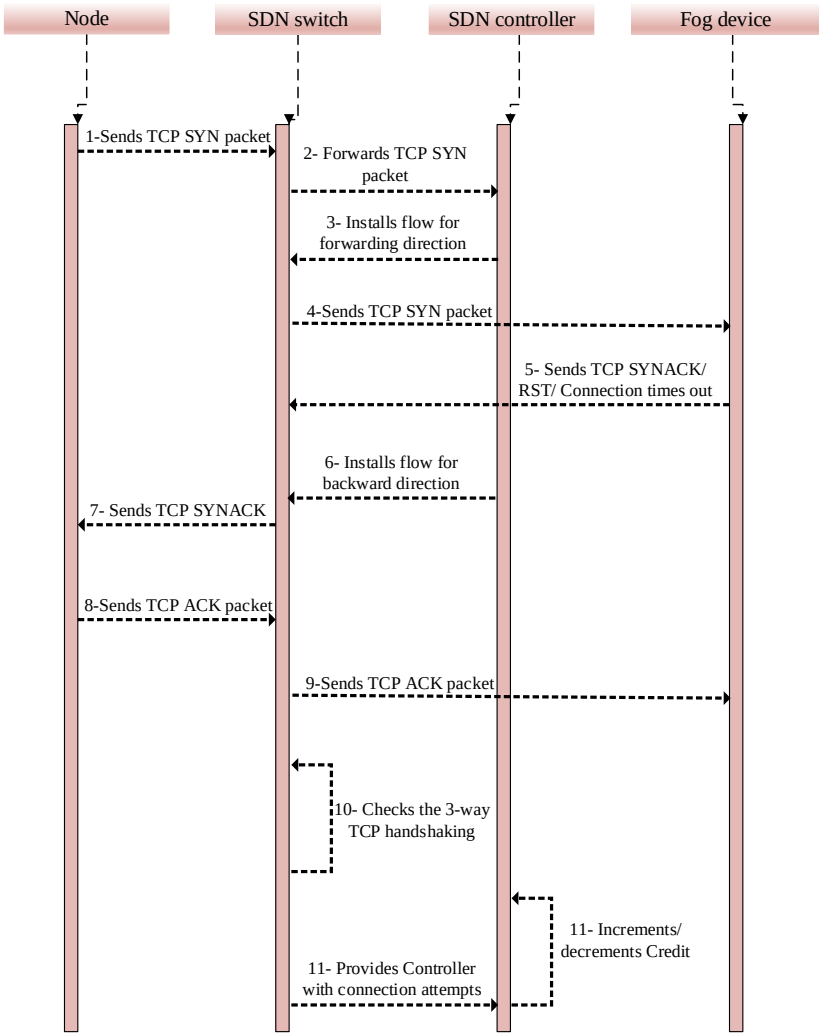


Figure 2.3: The sequence diagram of TRW-CB algorithm.

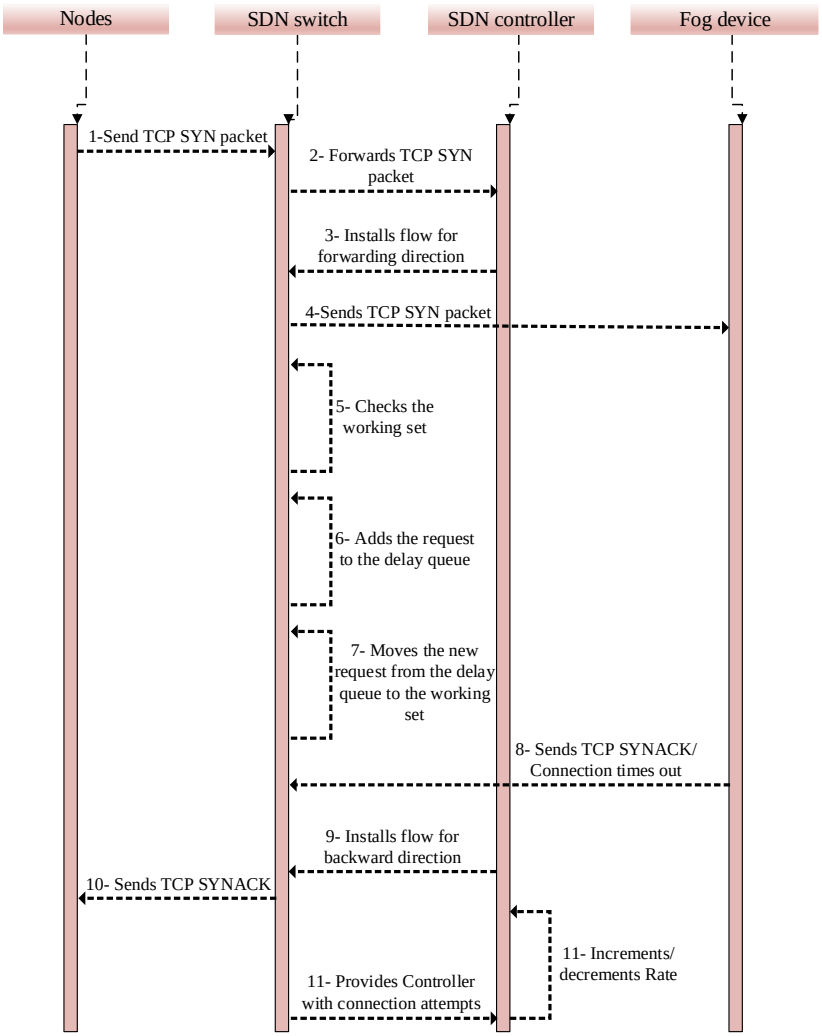


Figure 2.4: The sequence diagram of Rate limiting algorithm.

FUPE, we adopt the former strategy.

Fuzzy Based Fitness Function. As Fuzzy logic is widely used to consider multiple criteria simultaneously, we use it in both fitness functions. Each of the fitness functions determines the rank of the two objectives. Our presented approach has two objectives as detailed in the following.

First Objective (Security): For considering security objective, we use *TRW-CB* and *Rate Limiting* techniques. The cloud gateway gathers the information regarding the amount of exchanged traffic between fog devices/users' devices and the number of successful connections between fog devices/users' devices from all fog gateways. Then, it calculates the *Credit* and the *Rate* values for each fog device/user's device. Finally, it sends these values to fog gateways. We use them as the input parameters of the fuzzy based fitness function for security objective. For instance, *Low Credit* and *High Rate* indicate a malicious fog device/user's device, respectively. According to previous researches such as [87], an anomaly occurs when the value of Credit and Rate exceeds a predefined threshold. This work considers High Credit and High Rate as the malicious activity. In other work [89], the authors use Low Credit and High Rate to indicate malicious nodes. In this paper, we consider Low Credit and High Rate as the anomaly behavior as well. Defining the fuzzy sets and fuzzy rules are based on using either learning methods [91, 92] or former experience /predefined assumptions [10, 89, 11]. We define the range of Credit and Rate fuzzy sets based on the predefined assumptions (i.e., the fuzzy sets and fuzzy rules set are predefined and static). A suitable approach to define the fuzzy sets is in such a way that the end-point of the first fuzzy set is the beginning point of the third fuzzy set; the second fuzzy set has overlap with both the other fuzzy sets. The benefit of this approach is that it increases both of the fuzzy features of the fuzzy sets and the overlapping between the fuzzy sets.

Second Objective (efficiency): It is illustrated in the literature that simultaneous consideration of the computing features of the resources and application's tasks CPU need, is suitable for task scheduling [11, 93, 68, 94]. So, we use application's tasks CPU need, and the features of fog devices (i.e., available processing capacity, available RAM size, and available fog-device link bandwidth) as the input parameters of the fuzzy based fitness function for efficiency objective. FUPE implements a task merging mechanism where the tasks of the same application are assigned to the same fog device. Besides, we use the total amount of tasks CPU needs of an application as the *application's tasks CPU need* parameter.

Mamdani fuzzy inference engine is one of the most common fuzzy infer-

ence engines which uses fuzzy sets and fuzzy rules. These rules are based on the former experience or predefined assumptions. We define three overlapping fuzzy sets that make the input values possibly locate in several sets simultaneously. Accordingly, each of the input values belongs to several fuzzy sets with different membership degrees. For instance, consider the fuzzy sets in Figs. 3 and 4. Low and Medium fuzzy sets are overlapped, but there is no overlap between Low and High fuzzy sets. In this way, an input value which is in the Low fuzzy set interval is considered as both Low and Medium at the same time with different degrees of membership (confidence). Besides, if *B* is the degree of membership of the input value in the Low fuzzy set, then its membership in the Medium fuzzy set is *1-B*. For the sake of clarity, suppose Credit/Rate fuzzy sets in Fig. 2.5. If the input value is 0.2, then the degree of membership for Low and Medium fuzzy sets are 0.8 and 0.2, respectively. In this example, *B* and *1-B* are 0.8 and 0.2, respectively.

As a consequence, each of the input values can trigger several fuzzy rules (i.e. rule firing). Then the fuzzy inference engine aggregates rules and combines the fuzzy sets that represent the outputs of each fuzzy rule into a single fuzzy set. Finally, it interprets the membership degrees of the fuzzy set into a numeric non-fuzzy value which is the output of each of the objectives. The obtained values are the output of the fitness functions (i.e. security and efficiency objective). The fuzzy rules are indicated in Table 2.2 for the first objective and Table 2.3 for the second objective. Figs. 2.5 and 2.6 show the fuzzy sets for the two objectives as well. The rows in Table 2.2 and Table 2.3 are the fuzzy rules. The rules consist of the fuzzy sets, and the values of the fuzzy sets are indicated in Fig. 2.5 and Fig. 2.6. The cells Low, Medium, High, Safe, Potential, and Attack are the fuzzy sets that are used in the fuzzy rules. For instance, the third row in Table 3 means: If Credit is Low and Rate is High then Result is Attack. As it is shown in Fig. 2.5, the values of Credit Low, Rate High, and Result Attack fuzzy sets are 0.0 to 0.5, 0.5 to 1, and 0.0 to 0.5, respectively. To take another example, the first rule in the Table 4 means: If Task Length is Low and Bandwidth is Low and CPU is Low and Ram (memory) is Low then Result is Inappropriate. Fig. 4 shows the values of the Task length Low, Bandwidth low, CPU low, Memory low, and Inappropriate fuzzy sets that are 500 to 2750, 10000 to 15000, 10000 to 25000, 0 to 12300, and 0.0 to 0.5, respectively. Result fuzzy set is the output of each of the rules. Mamdani Fuzzy inference engine utilizes the mentioned fuzzy sets in the aggregation method to obtain the output of each of the objectives.

MOPSO based IoT-fog security driven task scheduling algorithm. The

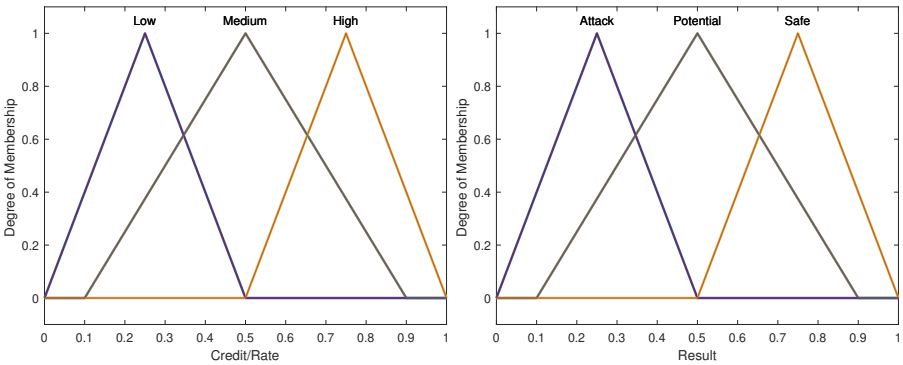


Figure 2.5: Fuzzy sets for security objective.

Table 2.2: Fuzzy rules for security objective.

Credit	**Rate**	**Result**
Low	Low	Potential
Low	Medium	Attack
Low	High	Attack
Medium	Low	Safe
Medium	Medium	Potential
Medium	High	Attack
High	Low	Safe
High	Medium	Safe
High	High	Potential

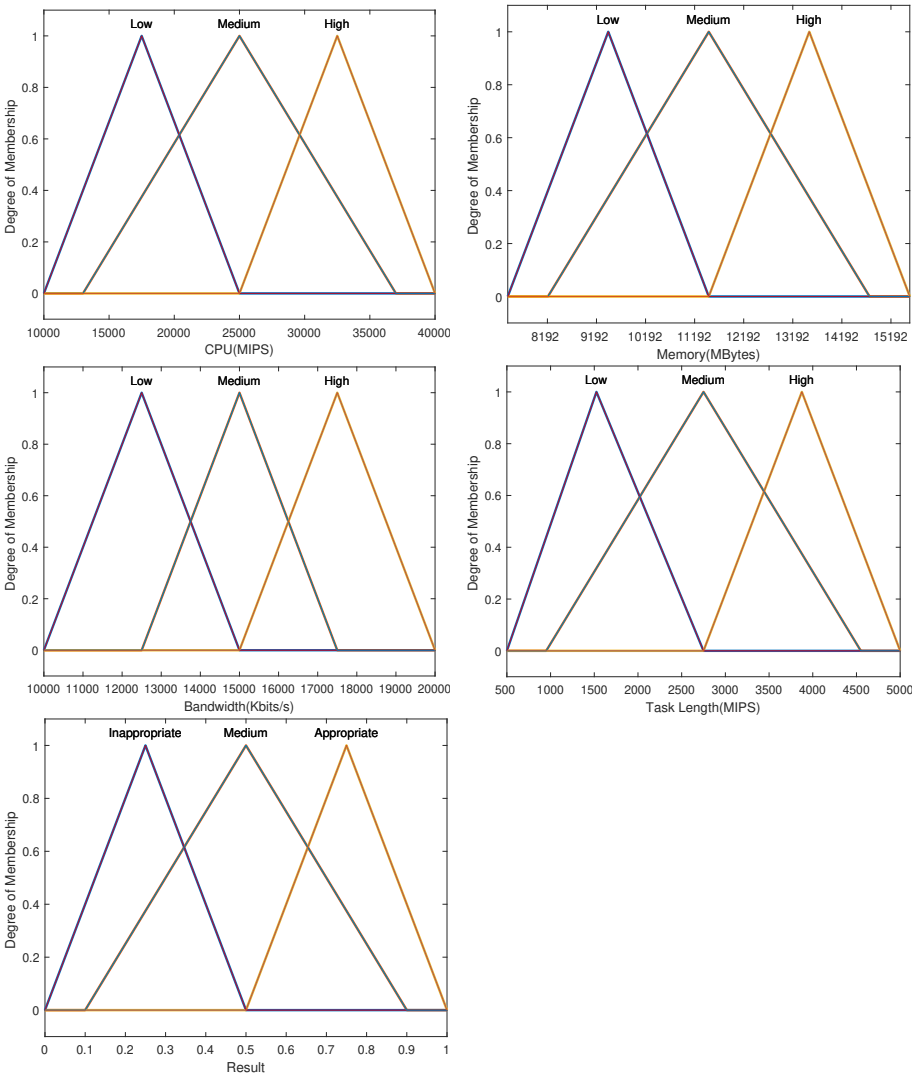


Figure 2.6: Fuzzy sets for efficiency objective.

Table 2.3: Fuzzy rules for efficiency Objective. TL= Task length. C/R = CPU/RAM. I=Inappropriate. A= Appropriate.

TL = Low , BW = Low				**TL= Medium , BW = Low**				**TL = High , BW = Low**			
H	M	L	C/R	H	M	L	C/R	H	M	L	C/R
M	I	I	L	I	I	I	L	I	I	I	L
M	M	I	M	M	I	I	M	I	I	I	M
A	M	M	H	M	M	I	H	M	I	I	H
TL= Low , BW = Medium				**TL = Medium , BW = Medium**				**TL = High , BW = Medium**			
H	M	L	C/R	H	M	L	C/R	H	M	L	C/R
A	M	I	L	M	I	I	L	I	I	I	L
A	A	A	M	A	M	I	M	M	I	I	M
A	A	A	H	A	A	M	H	A	M	I	H
TL = Low , BW = High				**TL = Medium , BW = High**				**TL = High , BW = High**			
H	M	L	C/R	H	M	L	C/R	H	M	L	C/R
A	A	M	L	M	M	I	L	M	I	I	L
A	A	A	M	A	M	M	M	M	M	I	M
A	A	A	H	A	A	M	H	A	M	M	H

PSO algorithm is used to find an optimal solution for optimization problems which is based on the behavior of a flock of birds to find their way to a specific destination. To start this algorithm, a population of particles is generated randomly in which their positions are considered as a potential solution. Each particle has a position vector and a velocity vector which determine the direction of movements in search space. In the PSO algorithm, the particles are initialized randomly. In this work, we consider a request as a particle and the state of the particle as the particle's position. We also consider *pbest* and *gbest* as the best position of a job request attains among the job request list, and the best position of job request as job request attains, respectively. The *gbest* is the global optimal state (i.e., the best fog device for each request). For example, if there are 8 fog devices and 3 requests in the fog region, FUPE assigns the fog device to the requests based on the *gbest* value for that request. A possible solution could resemble $\{2, 3, 2, 1, 2, 3, 2, 1\}$. For instance, the first element of this combination means the first fog device is the best optimal venue for the second request, and the second fog device is the best optimal venue for the third request.

In each iteration, fuzzy-based fitness functions are used to assess each particle's values so as to replace the position of particle with *pbest* variable. FUPE rejects a new solution if its *pbest* value is less than the current solution. In fact, it improves the fitness value of the applications' tasks in every iteration. *pbest* of all particles are used to update the *Gbest*. The current *pbest* is compared to

Gbest, and if its satisfaction is more than *Gbest*, it will be replaced. The final result is the best *pbest* value (i.e. *gbest*). General PSO is defines as follows:

$$X_{k+1}^{i} = X_{k}^{i} + V_{k+1}^{i} \tag{2.1}$$

In this equation, i, X, and V are the particles, the position of the particles, and the velocity of the particles, respectively. The formula for updating velocity vector is:

$$V_{k}^{i+1} = W_k V_{k}^{i} + C_1 r_1 (Pbest_{k}^{i} - X_{k}^{i}) + C_2 r_2 (Gbest - X_{i}^{k}) \tag{2.2}$$

Where C_1 and C_2 are the acceleration coefficient which is considered as learning factors, r_1 and r_2 are used to control the randomness in the movements of particles in the search space, and W is inertia weight which indicates the balance between local and global search. This parameter determines the repeat rate for finding the best solution. To aggregate the outcome of the two objectives into one synthetic objective to obtain a single proper answer, we use Eq. (2.3).

$$Aggregation = ((Security * \alpha) + (Efficiency * \beta)) * (-1) \tag{2.3}$$

Where $Security$ and $Efficiency$ are the first and second objective outcomes respectively. We assign priority to the objectives by α and β which are predefined weights. We set these weights in a way that the sum of them equals 1. In this work, we assume the priority of the two objectives is the same. So, we assign 0.5 to both α and β. PSO is a minimization problem and FUPE is a maximization problem. As the goal of the PSO algorithm is to minimize the objective and we aim to maximize the outcome of FUPE, we multiply the outcome by -1. It is worth highlighting that we use the two fuzzy-based fitness functions to update the values of *security* and *efficiency* variables in Eq. (2.3) to calculate *pbest*. Algorithm 4 indicates the proposed algorithm which leverages the benefits of the MOPSO algorithm in order to find an optimal solution regarding the two aforementioned objective functions.

Algorithm 4 FUPE- Proposed Algorithm

Output: $gbest$

1: **for** $i = 1\ toM = Iterationnumber$ **do**
`Initialize` $P[i]$ `randomly;` P `is population of particles`
2: **Initialize** $v[i] = 0$;
v=speed of each particle
3: $F = ComputeFitness(p[i])$;
4: $gbest =$ best particle found in F;
5: **for** $i = 1\ toM$ **do**
6: $pbest[i] = F$;
7: **end for**
8: **repeat**
9: **for** $i = 1\ toM$ **do**
10: Set W, $C_1 >= 0$ and $C_2 >= 0$;
11: $v[i] = W \times v[i] = C_1 \times rand1(pbest[i] - p[i] + C_2 \times rand2(gbest[i] - P[i])$;
12: **update** speed of each particle
13: $P[i] = P[i] + v[i]$;
14: **if** a particle goes outside the predefined hypercube **then**
15: it is reintegrated to its boundaries;
16: **end if**
17: $F = Computefitness(p[i])$;
18: **if** new population is **better then**
19: $pbest[i] = F$;
20: **end if**
21: $gbest$=Update it by the best particle found in $p[i]$;
22: **end for**
23: **until** stopping criterion is satisfied
24: **end for**
25: **return** $gbest$

Alg. 4 Description. In Alg. 4, first, FUPE defines the initial position and velocity vector for each particle (tasks) in the search area randomly (line 1). We save the current solution in *pbest* (line 2). In lines 3 to 4, FUPE computes the two fitness functions for the particles. In lines 5 to 7, after assigning the fitness value for all of the particles, FUPE compares them, and if the fitness value of the current solution is higher than *pbest*, it replaces it with the *pbest*. In lines

9 to 23, after each iteration, FUPE updates *gbest* value by comparing the *pbest* value of the current iteration to the value which is stored in the *gbest*. FPFTS calculates the request's position and request's velocity by using Eq. (2.1) and Eq. (2.2). In each iteration, FUPE calculates *pbest* for the particles with the aid of Eq. (2.3). FUPE repeats these steps to reach the maximum number of iterations. Finally, FUPE sets the best *pbest* value as the *gbest* value. *Gbest* is the output of this algorithm which indicates the most proper fog device for the application's tasks. Each fog device has a numerical value as the *gbest* for the incoming job tasks. FUPE assigns a fog device which has the greatest *gbest* value for the job tasks.

Computational Complexity of FUPE: Computational complexity shows the number of resources that an algorithm needs to be executed. We make reference to the work by Arora et al. [95] in which the authors set out a clear mathematical illustration of computational complexity. In FUPE, as we use computing features of resources and tasks and the parameters of the TRW-CB and Rate Limiting simultaneously, we consider the two objectives' computational complexity and then multiply them as follow: Firstly, the computational complexity of efficiency objective is in the order of $\mathcal{O}(I \times R \times P)$ where I is the number of iterations, R is the number of resources (i.e. fog devices), and P is the number of particles. Besides, both of the Rate Limiting and TRW-CB techniques take the same amount of time to obtain the result irrespective of the input size. They take the exact time to process one item as well as for example ten items. So, they have a constant growth rate run time. As we apply both TRW-CB and Rate Limiting techniques at the same time for the same nodes, we can say the computational complexity of security objective is in order of $\mathcal{O}(1)$. Finally, FUPE uses Mamdani fuzzy inference engine, so the complexity of the fitness function is in order of $\mathcal{O}(1)$. As Mamdani fuzzy inference engine searches, fires and aggregates the adequate rules in a parallel manner, the complexity of it is in order of $\mathcal{O}(1)$ [36]. As a result, the total computational complexity of the FUPE is in order of $\mathcal{O}((I \times R \times P))$.

Space Complexity of FUPE: For calculating the space complexity of FUPE, we take the spaces that FUPE needs to be executed into consideration. As the instances of FUPE are located in fog gateways, the number of fog regions plays a central role in the algorithm space requirements. So to obtain the space complexity of FUPE, we use the number of fog regions and we name this parameter S. As a result, the total space complexity of the FUPE is in order of $\mathcal{O}(S \times (I \times R \times P))$.

An example of scenario for FUPE complexity: As FUPE uses MOPSO

algorithm, more particles, fog devices, and iterations increase complexity. Defining the number of iterations depends on the environment. For instance, we can define the number of fog devices or applications as the iteration number. Suppose a scenario in which the number of particles is 10, the number of resources is 5, and we define the number of particles as the iteration. Moreover, we have 2 fog regions in the network. In this way, the Computational Complexity is 500 $(10 \times 5 \times 10)$, and the space complexity is 1000 $(2 \times 10 \times 5 \times 10)$.

2.4 Performance Evaluation

In this section, we conduct a comprehensive simulation study and analyze the results to compare the performance of FUPE in a fog environment to a single-objective PSO algorithm [96, 97] and a Genetic algorithm [19] that do not take security into account. This comparison allows to show the overhead of security mechanisms in FUPE.

For the evaluation, for each scenario, we run the algorithms ten times. Application scheduling in distributed systems is an NP-complete problem [98]. Evolutionary optimization algorithms are proper approaches to find the optimized solutions that generate several solutions. The final solution is the best answer among the generated solutions. So, in FUPE we have a set of applications, and FUPE runs while an adequate amount of applications are on the scheduling list. We performed the simulation study in MATLAB R2018a [99]. Besides, we used Xfuzzy 3.5, which is a fuzzy logic design tool to implement fuzzy-inference-based systems [100]. Since Matlab does not have inbuilt features to support the OpenFlow protocol directly, we extended the Matlab classes to add the OpenFlow protocol specification. We added routing or decision making functionality to Matlab classes. In detail, we defined a node (SDN switch) with a data structure similar to the flow tables. Then, another node (SDN controller), fills the flow tables.

iFogSim [33] is a widely accepted IoT-Fog simulator which we also used to implement our former work, FPFTS [40]. This simulator supports edge processing, edge communication, IoT devices, and also cloud processing. The big disadvantage of this tool is the lack of SDN support, SDN-WAN support, network communication, also network protocols and the missing possibility to implement the proposed security objective (e.g., 3-way handshaking). However, standard network simulators (e.g., ns [101], and OMNeT [102]) or SDN-aware network simulators (e.g., mininet [103], and SDN-Sim [104]) are not suitable to implement the presented approach at all. For instance, NS-3 and

OMNeT++ support network communication and network protocols. Although they provide the users with proper programming capabilities, SDN and SDN-WAN features, edge processing, edge communication, also fog computing facilities and IoT devices are not natively supported by them. To take another example, Mininet quantifies SDN performance within different network structures and routing protocols. It only supports SDN-WAN facilities, network communication, and network protocols. The focus of SDN-Sim is to deploy SDN-based policies, such as channel modeling, traffic shaping, and QoS demands. The disadvantage of these tools is the lack of edge processing, edge communication, also fog computing facilities and IoT devices.

FUPE does not consider the actual medium and stack of protocols. Hence, we do not change or modify the TCP/IP layers to observe the effect of FUPE on them. We did not use actual network parameters in Matlab. We can define propagation delay as distance, and not consider Queue delay, transfer delay, and processing delay. Attacks can occur in both wired or wireless ways. We are concentrating on the behavior of node requests. It makes no difference whether they are transmitted via a wired or wireless link. We basically study the behavior of malicious nodes. In any network environment setting, we expect the malicious nodes to behave differently than the benign nodes. Our method catches that abnormality in task scheduling phase. Therefore, using any network environment for packet request generation does not show any significant difference in results than those reported. Finally, it is illustrated in the work [26] that about 15% of the resource management approaches in fog computing have utilized Matlab simulator. There are also several works in SDN networks and TCP DDOS attack detection systems that are implemented by Matlab [105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118]. As there is no corresponding simulator that covers the minutiae of FUPE, we simplify it into a trade-off problem, and use MATLAB for evaluation. We implement a scalable IoT-fog framework in MATLAB in order to simulate FUPE as much as possible in a real scenario.

2.4.1 Simulation Setup

To simulate fog devices, we assume each fog device's configuration can be one of the items specified in 2.4. Upon creating fog devices, one of these configurations is selected at random. We considered the same random generated scenarios for each algorithm under test.

To simulate task requirements, we assume that each separate task needs resources in terms of RAM, CPU, and Bandwidth. Once a task is generated,

Table 2.4: Configuration of fog devices.

Configuration	Config 1	Config 2
RAM	8 GB	16 GB
CPU	5000 MIPS	10000MIPS
Core in CPU	2	4
Bandwidth	10 Mbps	20 Mbps

it randomly selects one of the four requirement configurations specified in the Tab. 2.5.

Table 2.5: Task Requirements.

Requirements	REQ 1	REQ 2	REQ 3	REQ 4
RAM	256 MB	512 MB	512 MB	1 GB
CPU	500 MIPS	1000 MIPS	2000 MIPS	5000 MIPS
Bandwidth	5 Mbps	10 Mbps	10 Mbps	10 Mbps

Simulation metrics

To evaluate the proposed solution against the selected baselines we consider the set of metrics in the following:

- *Response time:* the amount of time elapsed between the time a task is sent by a user to fog device and the time the related result is delivered to the user. It is worth noting that, this time includes network delay, processing delay, and task scheduling mechanism delay. We focus on both average and maximum performance figures for this metric.
- *Confidence interval (CI):* denotes a range of values that is determined by the margin of error. A confidence interval for the results depends on sampling the distribution of a corresponding estimator. The goal of this metric is to indicate the variability of the results. For the performance evaluation, we use confidence intervals for plots calculated with 95% confidence level. Focusing on the confidence interval, we refer the readers to reference [119].
- *Network Utilization:* It provides a quantification for the data transmitted in the network links between network nodes (i.e. how much link bandwidth is used) during a time interval. We use Eq. (2.4) to calculate this

metric. In this equation, the length of data that are encapsulated in each task is calculated in *KByte*. Moreover, Interval is set to 1000 ms (i.e. we calculate network utilization every second).

$$NetworkUtilization = (\sum_{x=1}^{Requests} Task\_length)/Interval \quad (2.4)$$

Simulation scenarios

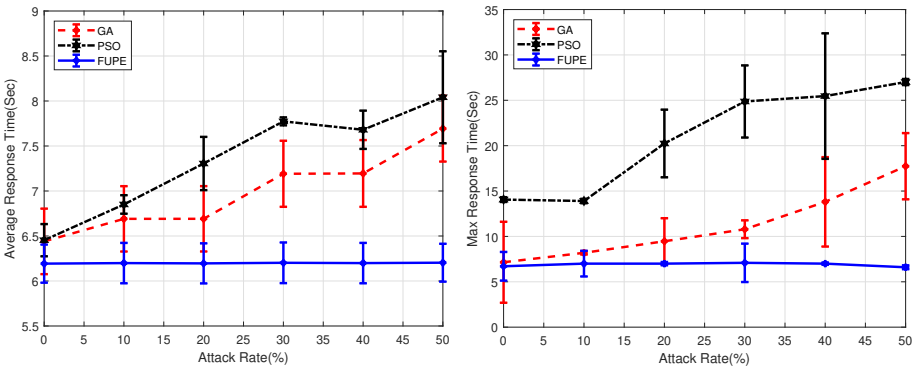
To evaluate all aspects of FUPE in terms of performance and security, we have considered various scenarios with different number of attack rates, different number of fog devices, and different number of jobs. There are listed below

- *Scenario-1:* In this scenario, the number of jobs is 60 per second and the number of fog devices is 10. The attack rate in this scenario varies from 0% to 50% (e.g. 0 to 50 percent of requests are malicious and TCP DDOS attack, while the other requests are benign and valid requests). The aim of this scenario is to investigate whether FUPE can detect attacks and does not assign tasks to malicious fog devices, and also does not process the task coming from attacker users. Moreover, this scenario illustrates the overhead of the FUPE itself.
- *Scenario-2:* In this scenario, the number of fog devices varies from 5 to 25, and the number of jobs is 60 per second. The attack rate is fix and equal to 30%. In fact, the goal of this scenario is to investigate the performance of FUPE when increasing the fog devices in the presence of a moderate attack.
- *Scenario-3:* In this scenario, the number of fog devices is 60, and the attack rate is 30%. The number of jobs varies from 20 to 100 per second. The goal of this scenario is to investigate the scalability of FUPE when increasing the number of jobs in the presence of a moderate attack.

In all simulation scenarios, we assume one cloud gateway and two fog gateways.

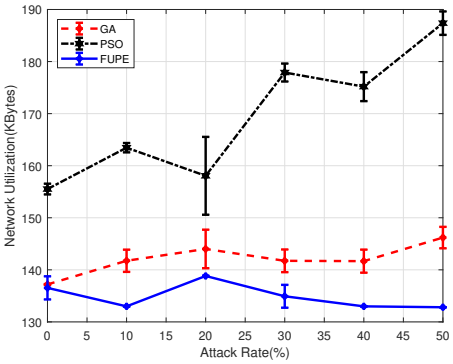
2.4.2 Experimental Results

In this section, we discuss and analyze the simulation results for the FUPE and other approaches in different scenarios. First of all, we investigate the results



(a) Average response time.

(b) Maximum response time.



(c) Network utilization.

Figure 2.7: The comparison results for the first scenario in the presence of different attack rates in (%).

for *Scenario-1*. Fig. 2.7a and Fig. 2.7b show the average and maximum response time against different attack rates for the methods, respectively. In these two figures, it is obvious that by increasing the attack rate, the response time grows linearly for Genetic and PSO algorithms. The reason is that in these two methods, there is no mechanism to detect and mitigate the effect of the attacks. Therefore, the response time considerably grows. On the other hand, in FUPE, increasing attack rate does not affect the response time. This is because of considering *Credit* and *Rate* for each request in fog gateways. Applying these security parameters leads to FUPE significantly outperforms other approaches and improve better response time for users' requests. In addition to security consideration, as aforementioned, FUPE uses a multi-objective model in order to make efficient use of the network resources by assigning tasks to suitable

fog devices. This behavior also leads to less response time.

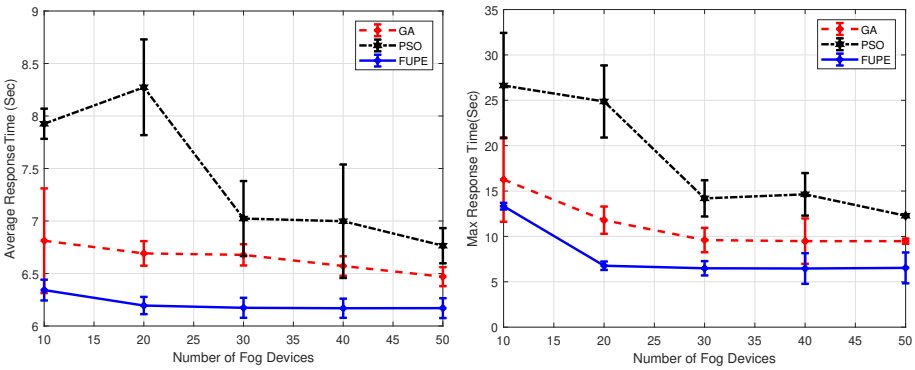
In addition to these achievements, another important benefit of using FUPE is to reduce resource network bandwidth exhaustion. Fig. 2.7c proves that using security parameters in FUPE leads to a significant prohibition of the adversary effect of different rates of TCP DDOS attack. In the other two methods, there is no security mechanism to prevent network utilization by malicious behaviors, and as a result the network utilization has been linearly increased.

Fig. 2.7 also indicates the CI for plots as indicators over the results. As all the methods use randomness features, they have almost the same tolerance in the results. However, the tolerance of the results for FUPE is a bit less. More specifically, it is more obvious in the 2.7c. The reason behind this is FUPE uses security parameters in scheduling the tasks to fog devices. Hence, it does not assign the tasks to malicious fog devices. Consequently, the volume of assigned tasks for fog devices is not equal. The quota of compromised fog devices is zero and the quota of non-compromised fog devices is high. This behavior causes more tolerance in the results.

Fig. 2.8a and Fig. 2.8b illustrate the average and maximum response time with different number of fog devices in Scenario-2, respectively. From these two figures, it can be concluded that, in the presence of a 30% attack rate, increasing the number of fog devices (i.e. computational capacity) decreases the response time in Genetic and PSO algorithms. This result is obvious because the computational capacity of the network has been increased. Moreover, in comparison with Fig. 2.7b, the maximum response time for FUPE also has been decreased. These two figures prove that, unlike Genetic and PSO algorithms, FUPE has not undergone significant changes in terms of response time. The main conclusion from this result is that since FUPE uses the security mechanism together with the multi-objective model to assign tasks optimally, it can provide reasonable response time even in the presence of attacks, and there is no need to add extra fog devices. In nutshell, FUPE utilizes network and computational resources optimally and also prohibits TCP DDOS attack to exhaust the resources.

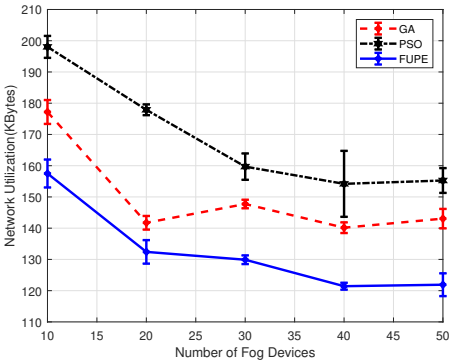
Fig. 2.8c shows that as the number of fog devices increases the network utilization in all three methods decreases. The reason behind this is that, increasing the size of the network while the load of the network is constant, leads to low network utilization. It can be concluded from Fig. 2.8c that, because of applying performance parameters in task scheduling in FUPE, it assigns the task in a fair manner in compared to the other methods.

Fig. 2.8 also shows that the Tolerance of FUPE results in task assignment



(a) Average response time.

(b) Maximum response time.

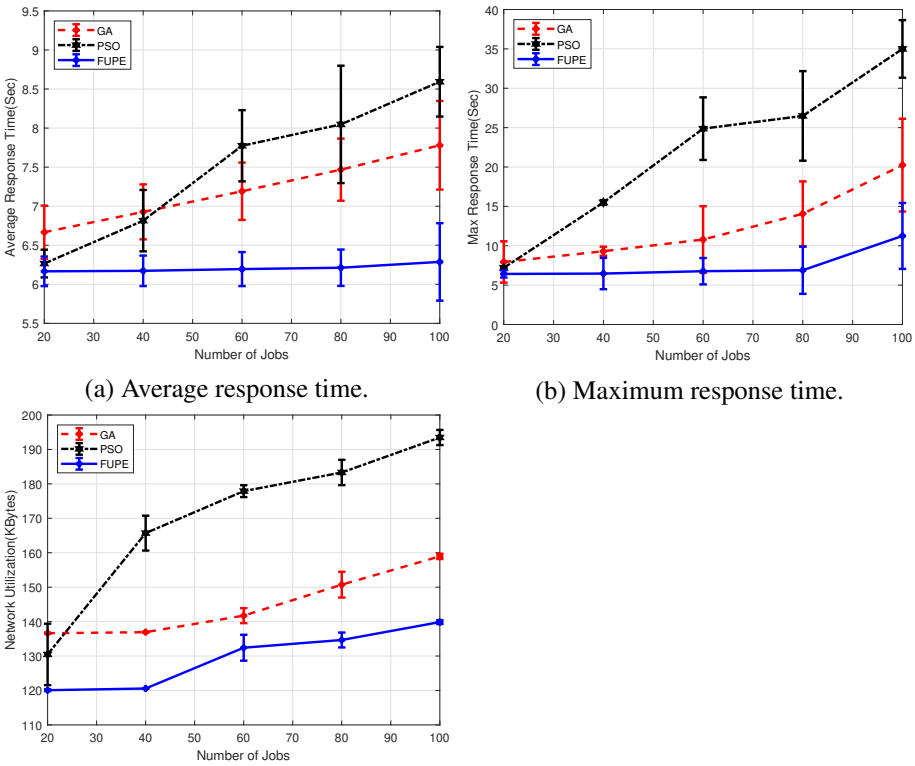


(c) Network utilization.

Figure 2.8: The comparison results for the second scenario in the presence of different number of fog devices.

for FUPE improves when the number of fog devices increases. In fact, using a large number of fog devices leads to spreading tasks to more devices, and as a result the difference between the upper and lower values for CI decreases. The reason behind this is FUPE uses fog devices features and jobs' requirements in the efficiency objective.

Fig. 2.9a and Fig. 2.9b depict average and maximum response time with different number of jobs in Scenario-3, respectively. From these two figures, it can be concluded that, in the presence of a 30% attack rate and a fixed number of fog devices (10 devices), increasing the number of jobs considerably increases the response time both Genetic and PSO algorithms. This increase is also true for FUPE. But, in FUPE, the growth of response time is significantly less than the other two methods. Once more, we note that due to applying the



(a) Average response time.

(b) Maximum response time.

(c) Network utilization.

Figure 2.9: The comparison results for the third scenario in the presence of different number of jobs.

security mechanism together with the multi-objective model in FUPE, assigning the tasks is performed optimally. Therefore, network and computational resources have been used efficiently.

Fig. 2.9c illustrates the impact of increasing the load (number of jobs) in network utilization. As can be seen in this figure, it is obvious that the network utilization is increased by holding the network resources constant. But, this figure clearly shows that FUPE considerably achieves better results in terms of network utilization. This improvement comes from two mechanisms: First, using performance parameters such as CPU, RAM, and network bandwidth in choosing the optimal fog device. Second, FUPE considers the applications' CPU needs. Moreover, it considers the security conditions of requester and fog devices and prohibits malicious requests to be processed. It also blocks future

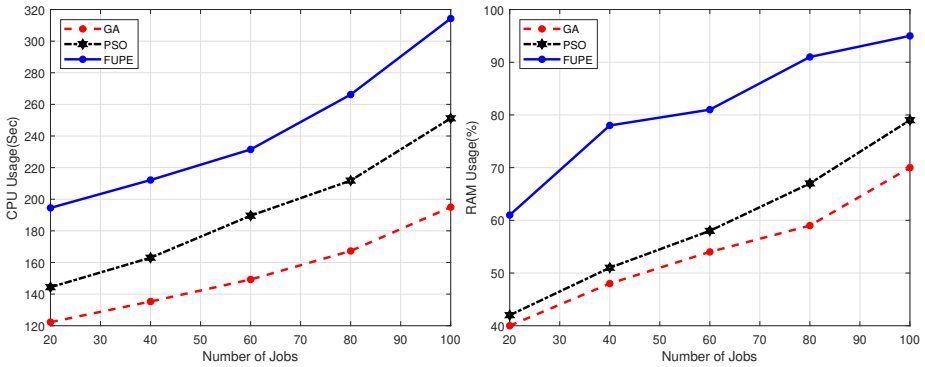
requests from attacker nodes. Unlike FUPE, the other two methods have not any performance and security mechanism to mitigate TCP DDOS attacks.

Fig. 2.9 also illustrates that the results tolerance in task assignment for the approaches. In FUPE, assignment of tasks to compromised fog devices is prohibited and the tasks are assigned to other fog devices. This behavior separates fog devices into two groups and only non-compromised fog devices can process the tasks. As a result, the FUPE results tolerance is less than the other two approaches. AS Genetic algorithm considers resource stability, it improves the stability of the task execution. As the result, it has a better results tolerance compared to the PSO algorithm.

In this scenario, we want to show the overhead to execute the proposed algorithm. We indicate the required CPU and RAM compared to the other solutions. In this paper, we use an approximation algorithm to mimic (approximate) the behavior of the malicious nodes, and also catch them in the scheduling phase. The approximation method gathers information based on the mode status and system capabilities, so we need to access the system characteristics. Consequently, it can take time to gather information unless we ask up it in the model generations. There is no specific hardware (RAM, CPU) requirement for FUPE, and It works for any system without any problem. However, it needs more processing capacity compared to the other solutions. Fig. 2.10a and Fig. 2.10b indicate the required processing capacities for the algorithms. Both Genetic and PSO methods do not have security assurance. For FUPE, we have some time consumption for the security part (i.e., the security cost). It includes the number of connection requests, and the number of flows installed in the fog gateways. Moreover, as MATLAB consumes much CPU and RAM for fuzzy functions, the overhead of FUPE is more than the other two methods.

According to Fig. 2.7 to Fig. 2.9, it can be concluded that, after FUPE, Genetic algorithm achieves better results in comparison with pure PSO. This is because of, in Genetic approach, the objective function is optimized to consider both latency and stability to assign tasks to the fog devices. Furthermore, Fig. 2.7 to Fig. 2.9 obviously shows that the only side effect of FUPE is that, in the presence of attacks, it can not equally assign tasks between fog devices. In fact, this issue is not the weakness of FUPE, because it prohibits compromised fog devices to process tasks. As a result, the burden of workload in compromised fog devices is zero while in non-compromised nodes is high.

Finally, we present the accuracy evaluation of the presented approach. For this evaluation, there are 700 benign requests, and 300 malicious requests (attack). As Genetic and PSO algorithms do not have attack defence mechanisms,



(a) CPU Usage. (b) RAM usage.

Figure 2.10: The comparison results for the required CPU and RAM compared to the other solutions.

for this evaluation, we only evaluate FUPE in terms of accuracy, precision, and recall.

$$Accuracy = ((TP + TN)/(TP + TN + FP + FN)) \tag{2.5}$$

$$Precision = (TP)/(TP + FP)) \tag{2.6}$$

$$Recall = (TP)/(TP + FN)) \tag{2.7}$$

where TP denotes true positive, and FN denotes false negative, TN denotes true negatives, and FP denotes false positive rates, respectively. Based on Eq. (2.5), Eq. (2.6), and Eq. (2.7), *Accuracy*, *Precision*, and *Recall* are 0.9820, 0.9608, and 0.9800, respectively. Tab. 2.6 indicates the accuracy detection of FUPE.

Table 2.6: Accuracy evaluation.

	TN	**TP**	**FN**	**FP**	**Accuracy**	**Precision**	**Recall**
Value	688	294	6	12	0.9820	0.9608	0.9800

2.5 Discussion

The scheduling problem of users' applications is the major challenging research topic in IoT networks. Due to the distributed feature of IoT-fog re-

sources, the computational resources are more susceptible to come under attack. FUPE investigates the security-driven scheduling policy. As it is illustrated in Section 2.4, multi-objective PSO method is a proper way to achieve a tradeoff between security and efficiency. We found Mamdani fuzzy inference system, a method to strike a balance between distinctive parameters in the *two* MOPSO fitness functions. Because of randomness feature of PSO, FUPE has more uneven load distribution among fog nodes. However, from security point of view, this indicates that FUPE has the ability to detect malicious fog devices. Although in this work we take TCP DDOS attack into consideration, we argue that FUPE can be extended to tackle the other TCP DDOS flooding attacks such as UDP flood and ICMP flood attacks.

FUPE can prioritize the distinct objectives based on the organizational policy. It means it can increase/decrease the objectives' priorities. For example, if the priorities of the objectives are the same, we multiply the fitness function outcomes by the same weight value (see Eq. (2.3) in Section 2.3.4). Thus, FUPE chooses a computational resource based on the same trustworthiness and computational capability. In another example, suppose the organizational policy is to adjust either the security or efficiency of the computational resources. We can simply increase or decrease the objectives' weight values which are a number between 0 and 1. Suppose the organizational management model is to increase the security level. Thus, we simply adjust the security objective's weight value.

In both TRW-CB and Rate Limiting techniques, FUPE uses the request sender's ID as their identity. This ID can be the IP address or Mac address of the request sender. To mask their identity in a TCP DDOS attack, compromised fog devices/malicious users' devices use IP spoofing/MAC spoofing. To counteract the aforementioned attacks, we can include IP spoofing and MAC spoofing detector modules in SDN switches. For the former, the IP spoofing detector module includes a table that stores the IP and MAC addresses of fog devices and users' devices. If the detector module detects an IP address associated with more than one MAC address, the fog device/end-user device is considered an attacker [120]. Some works consider the sequence number field in the link-layer header of each flow for the latter. The MAC spoofing detector module computes the difference between the current frame's sequence number and the last frame received from the same fog device/end-device. user's We can use either a threshold-based [121] or a fuzzy-based approach [122] to make decisions.

Improving accuracy metrics is a critical issue in intrusion detection and

prevention systems. The work [87] illustrates that by tuning the threshold parameters of TRW-CB and Rate Limiting, they have proper performance in terms of accuracy evaluation metrics. The performance of these algorithms relies on the threshold settings. As FUPE makes use of fuzzy logic for security objective, we eliminate the problem of threshold approaches (i.e., choosing an adequate threshold based on the aim of the approach). On the other hand, it relies on applying an appropriate fuzzy rules set for proper performance. If we use appropriate fuzzy rules in the Mamdani fuzzy rules set, FUPE can provide better performance in terms of accuracy metrics for obtaining security objective. As FUPE utilizes fuzzy logic for obtaining security objective, the results are perceived based on assumptions for defining fuzzy rules set. So, the accuracy of FUPE TCP DDOS attack detection is based on these assumptions that are tunable. It must be customized and fine-tuned to get the desired accuracy.

FUPE is the first security-driven task scheduler that integrates TCP DDoS attack detection techniques in a scheduling algorithm in IoT-fog networks. Response time and network utilization are the two most critical issues in IoT-fog networks which affects on users' satisfaction. To reduce these important metrics we make use of fog computing in IoT networks. That is the reason why they are the two common metrics for evaluating IoT-fog scheduling algorithms. To evaluate the security aspect of FUPE, we varied the attack rates to obtain the mentioned metrics. For evaluating Trust/ security aware big data task scheduling approaches, it is common to evaluate the methods by considering the efficiency metrics and varying a security variable. For instance, in the work [80], response time is calculated by varying the number of untrusted resources. To take another example from [79] execution cost is calculated by varying the number of risk rates. So, as FUPE is a security aware IoT-fog task scheduler, for evaluation we focus on the metrics of time by varying a security variable.

We argue that FUPE is suitable to be implemented in real examples of disturbed systems such as Web Operating Systems [35] with the aid of peer-to-peer (P2P)/semantic P2P Grid architectures [36]. It has *two* substantial benefits. First, the process of the user applications is done at the fog devices which are close to the users' devices. Accordingly, response time is decreased which causes more user satisfaction. Second, as security is a critical concern for users' applications in web OS, FUPE provides a suitable security level for them on fog devices.

2.6 Conclusions

This study introduced a security-aware fog task scheduler, called *FUPE*, that considers security issues to effectively assign IoT end-user tasks to fog devices in SDN architecture. FUPE induces the remaining amount of RAM size and CPU capacity, end-user tasks CPU need, and trust degrees as input parameters and implements a multi-objective approach jointly based on PSO and fuzzy theory to assign applications' tasks to fog devices. FUPE tackles the TCP DDOS attack as the most common denial of service attacks. Our experiments using an IoT-based scenario illustrate the effectiveness of the presented approach. Results show that FUPE outperforms both the metaheuristic methods, i.e., GA and PSO. Compared against the former bio-inspired method, FUPE improves average response time by 11% and network utilization by 10%. It compared against the latter method and improves average response time by 17% and network utilization by 22%.

Chapter 3

S-FOS

Fog computing aims to provide Cloud data center at the edge resources to support time-sensitive Internet of Things (IoT) applications with low latency requirements. Software defined networking (SDN) is a novel networking paradigm that decouples the control plane from the data plane, resulting in a high level of programmability and manageability. In SDN-based IoT-Fog networks, SDN switches and controllers can serve as fog gateways/cloud gateways. SDN switches and controllers, on the other hand, are more vulnerable to a variety of attacks, making the SDN controller a bottleneck and thus vulnerable to control plane saturation. IoT devices are inherently insecure, leaving the Internet vulnerable to a range of attacks. In this chapter, we present S-FoS, an SDN-based security-aware workflow scheduler for IoT-Fog networks. Our framework is a software application that detects the source of attacks and monitors IoT devices for malicious activity. The proposed approach defends scheduling services against Distributed TCP/UDP Denial of Service (DDoS) and port-scanning attacks. S-FoS is a joint secure and performance optimization approach that employs fuzzy-based anomaly detection algorithms to identify the source of attacks and block malicious requesters. Furthermore, it employs an NSGA-III multi objective optimization approach to the scheduler in order to consider load balancing and delay simultaneously. We do comprehensive simulations on IoT-based scenarios to show that the S-FoS outperforms state-of-the-art algorithms by a significant margin. The simulation results show that by varying the attack rates, the number of fog devices, and the number of flows, the response time of S-FoS improved by 31% and 18%, the network utilization of S-FoS improved by 9% and 4%, and the energy consumption of S-FoS improved by 16% and 9%, respectively, when

compared to the NSGA-II and MOPSO algorithms.

3.1 Introduction

With the rapid growth of Internet of Things (IoT) applications, many novel approaches to meeting the various performance and security requirements of IoT networks have been proposed. Because of the importance of IoT security and application scheduling in the face of challenges imposed by inherent IoT features such as computation or energy consumption limitations, IoT security and application scheduling have gotten a lot of attention. Because of these constraints, the use of strong security protocols to protect scheduling services is limited [49, 123]. As the number of generated IoT workflows increases, time-sensitive processing necessitates resources with sufficient computational capacity. Furthermore, the computational capacity of the cloud data center is nearly limitless, but it necessitates a higher communication latency. Although the cloud data center is a powerful and frequently misunderstood silver bullet for application execution, its silver lining begins to fade at the network's edge, where lower latency, more time-sensitive capabilities, and decentralized resource management are desired. Response time is critical in time-critical environments such as traffic control systems. So, in order to reduce response time, the workflow scheduler should assign applications to network resources at the network's edge. Fog computing is a promising IoT paradigm that improves the QoS of time-sensitive applications by redirecting some resources from cloud data centers to closer edge resources (fog devices). There is no guarantee that fog devices will always be available in such a distributed environment. If the fog devices are not protected by defense mechanisms after they begin processing time-sensitive applications, they are vulnerable to attack and hacking. As a result, it is critical to ensure that time-sensitive application schedulers are reliable for successful execution in the resources, regardless of any attack on fog devices. [78].

The user applications are assigned to the fog devices by a workflow scheduler. Following that, fog devices decompose the applications into small tasks and schedule them to be executed on the resources of the fog devices. The fog devices, as well as the workflow scheduling services of IoT-fog networks, are the most vulnerable to attacks such as TCP/UDP DDoS or port scanning [124, 125, 60, 87]. A TCP/UDP DDoS attack launched by a malicious IoT device can significantly reduce the performance of scheduling services, causing user applications to lag. When a fog devices is attacked, it can be brought

down or even used to aid the attacker. Another important attack in SDN-based IoT-fog network is port scanning. Generally, this attack is performed in the reconnaissance step and its goal is to determine which ports in the network are open and may be receiving or sending requests from users. Open ports on a host may be exploited by attackers to compromise the host in order to launch future malicious behaviors.

In line with the preceding context, IoT-fog networks require efficient approaches to security and efficiency requirements. As a result, these approaches are the most critical in terms of IoT-fog network performance. Security and performance are two distinct issues. Typically, they are addressed separately. It is still difficult to use IoT-fog resources securely and efficiently, so solving this problem requires the use of collaborative efficient algorithms. Furthermore, we can use a multi objective optimization (MOO) algorithm to consider Quality of Service (QoS) parameters such as load balancing and delay for performance metrics at the same time. These algorithms have been widely used in IoT-fog networks. [126, 127, 19].

3.1.1 Motivation

Without considering the security of the fog devices, the IoT-fog network cannot properly execute the users' time-sensitive applications. The reason for this is that, as a result of a failure in fog devices or SDN controller caused by an attack, the user experiences high rates of delay, causing the execution time to exceed the service-level agreement (SLA). As a result, the time-sensitive application execution fails. The requirement for a defense mechanism to protect scheduling services prompted us to apply security defense mechanisms to the presented approach in order to consider fog devices and SDN switches security. In the control plane, network functions such as routing, traffic management, security and resource scheduling are offloaded to the SDN controller. The data plane SDN switches forward traffic according to the flow rules provided by the SDN controller. SDN-based IoT-fog networks, while beneficial in many respects, are subject to a variety of security issues [60, 87]. TCP/UDP DDOS and port scan attacks on fog devices affect the controller and can bring the entire network down due to the SDN controller's logically centralized architecture. The malicious IoT device sends TCP/UDP packets to various ports on the fog devices and evaluates the response packets to determine the availability of the service on the fog devices in a TCP and UDP port scan attack. Furthermore, malicious IoT devices flood fog devices with SYN requests in order to overwhelm them with open connections. In March 2013, a TCP/UDP

DDoS attack against Spamhaus caused significant network congestion, resulting in a significant loss. The github servers were brought down by TCP/UDP DDoS assault traffic in February 2018. It also happened once against Amazon's online service, which caused a 209 million USD loss by disrupting all of its services for two hours. Approximately 17 TCP/UDP DDoS assaults were launched against the University of Albany website during February and March 2019. The above examples demonstrate the need to countermeasure these attacks.

Current application schedulers use the security unit to validate fog devices [16, 128, 14, 126, 20, 19, 68]. The schedulers assign applications to fog devices that have both been approved by the security unit. While this technique is valid, time-sensitive processes may encounter unforeseen challenges such as the security unit going down or IoT devices being compromised after they have been validated by the security unit and before the applications are executed. In other words, application execution must take place within a specific time frame. The applications have time-sensitive requirements that specify the amount of time the applications must be executed. The failure of the security unit has disastrous consequences for the workflow scheduling unit. On the one hand, if it fails, the scheduler considers the malicious IoT devices requests. When a fog device is attacked, it may engage in other malicious activities such as abusing users and stealing critical data. Attempting to return to a stable state, on the other hand, will result in time loss and user dissatisfaction because time is a critical factor in time-sensitive workflow. A possible solution to overcome this problem is using security aware application scheduling. Some studies have been conducted in this area [78, 61, 77]. They primarily attempted to shorten the makespan while keeping security in mind. Consideration of security and scheduling together is better suited for real-time applications than considering them separately. The reason for this is that a delay in communication between the security unit and the scheduling unit, or even a failure of the security unit, can result in the user application being executed after the deadline. To that purpose, these works model the application in such a way that it is not allowed to execute on all fog devices because those fog devices do not match the application's security requirements or are not risk-free. They provide the level of security for each application that will be run on fog devices that meet the security requirements. These works still rely on the security unit, even though they allocate the applications to fog devices with the specified security level. The attacks can jeopardize the system's time-sensitive capabilities as well as bring the fog device down if the scheduler fails to address security.

3.1.2 Contribution of the chapter

Because IoT-fog architecture contains users' essential data, it necessitates QoS and security considerations. To our knowledge, S-FOS is the first attempt to perform secure workflow scheduling approach for performance optimization in SDN-based IoT-Fog networks that considers TCP/UDP DDOS, and port scanning attacks. To secure the IoT-fog scheduling services, it automatically disables IoT devices that begin to engage in malicious activity. S-FOS validates users' applications using the results of the security mechanism. Then it chooses fog devices that meet the load balancing and delay requirements of the users' applications using one of the recently investigated MOO solutions, the non-dominated sorting genetic algorithm (NSGA-III) [129]. Because it outperforms the other MOO algorithms on large-scale problems with a large number of users, this algorithm is widely used in IoT-fog networks. Furthermore, by using reference points for the next-generation population, it first reduces the time it takes to find the final solution; second, it performs better for problems with more than two objective functions [130, 131]. S-FoS requests that NSGA-III take load balancing and delay into account for performance optimization. In this chapter, we assume that the attackers are IoT devices capable of attacking fog devices, SDN switches and controllers.

We assess S-FoS using IoTSim-Osmosis [132]. To that end, we compare S-FoS to two well-known metaheuristic approaches, NSGA-II [133] and Multi objective particle swarm optimization (MOPSO) [124], by varying attack rates, fog device count, and flow count. According to the results, our proposal outperforms the NSGA-II and MOPSO approaches in terms of response time, network utilization, and energy consumption. In particular, S-FoS improves response time by $\approx 25\%$, network utilization by $\approx 7\%$, and energy consumption by $\approx 13\%$ on average. The following are our primary contributions:

- We present a novel framework for workflow scheduling that considers load balancing and delay while meeting the security requirements of IoT devices.
- When scheduling the workflow, we only take into account the applications of validated users as determined by our proposed security mechanisms. A proper flow assigner modules on SDN controllers, as well as some fog devices, are required for efficient application execution.
- The NSGA-III-based scheduling algorithm takes into account fog devices that can be dynamically acquired and released, and each application is then mapped into a corresponding fog device while load balancing and delay

are taken into account. We propose an NSGA-III approach for workflow scheduling to solve the multi-dimensional and multi-constraint optimization problem.

- We conduct experiments to demonstrate the efficacy of the proposed secure workflow scheduler.

3.2 Related work

In this section, first, we discuss defense mechanisms against the attacks in IoT/SDN networks. Following that, we describe the fog application scheduling methods. Then, we investigate security-aware fog task scheduling algorithms. Table 3.1 summarizes the comparison of the related application scheduling approaches.

3.2.1 DDOS and port-scanning attacks in IoT/SDN approaches

In this section, we discuss recent defense mechanisms in IoT-fog and SDN networks against DDOS, and port scanning attacks [60, 70]. Kolias et al. [72] cited the Mirai botnet and its variants as real-world examples of DDoS attacks in IoT networks. These botnets demonstrate how DDoS attacks can take advantage of the lack of security mechanisms in IoT devices, resulting in equipment failure, production downtime, and reputation damage for IoT systems. The authors discussed lessons learned from previous real-world DDoS attacks, but they did not present mitigation strategies for the future. Spilios Evmorfos et al. [71] conducted research on the mechanism of SDN against SYN flood attacks. Such a paper investigates long-short-term memory and random neural network as two lightweight techniques for mitigating SYN flood attacks in IoT systems. Zhou et al. [76] proposed a three-level DDoS attack mitigation strategy in industrial IoT (IIoT)-fog networks. This research provides a low-latency communication for the cloud center and monitors firewall devices running real-time traffic filtering, filtering signature botnet DDoS attack packets. The anomaly traffic is filtered based on the known traffic signature. According to the results of the tests, this work reduces the number of false alarms and the time it takes to detect an attack compared to using only the local fog level. Yan et al. [73] proposed a DDoS attack mitigation framework with multiple levels for the IIoT. It uses SDN technology to manage a large number of IIoT devices as well as network flows. As they prove, the intrusion detection system can protect against DDoS attacks in IIoT much more quickly and

easily by leveraging SDN capabilities. To accomplish this, SDN controllers work with SDN switches to collect traffic data. Then, it detects DDOS attacks using network traffic data. After this, it restrains DDOS attacks based on detection and finally perceives network state using honeypots. Kumar et al. [74] proposed SAFETY, a method for detecting and mitigating DDOS attacks in SDN networks. The authors employ the SDN's flexible programmability and global visibility in a Normalized Entropy (NE) detector algorithm. To measure network flows, SAFETY uses Shannon entropy as the NE algorithm. It monitors the entropy value of the packets and detects a DDOS attack if the value falls below a certain threshold. The authors include SAFETY in the Floodlight OpenFlow controller, which uses the destination IP address, port, and TCP flags to calculate NE. SAFETY reduces false positive and false negative rates by dynamically increasing the thresholds to auto-adapt to the normal fluctuations of the SDN link traffic. When compared to approaches that consider multiple features, entropy detector methods that only consider a single feature from the packet header (i.e., source IP or destination IP) do not provide a reasonable detection accuracy rate. To overcome the limitations of single packet features, Mao et al. [134] employ multiple packet header features based on the joint-entropy method for DDoS attack detection and mitigation in SDN. The authors use information theory to achieve greater scalability, detection accuracy, and less complexity. Furthermore, the joint entropy approach covers flow duration, source IP address, packet length, and destination port to detect DDOS attacks. Using packet length and source IP, the authors conducted experiments to compare the joint entropy approach with a single entropy method. In terms of detection accuracy and false-positive rate, the joint entropy method outperforms the single entropy method. The disadvantage of this work is that it is not capable of early detection because it takes longer to detect an attack. SLICOTS, a DDOS attack mitigation approach in SDN, is proposed by Mohammadi et al. [64]. Because of a bottleneck in the control plane, the authors believe it is vulnerable to saturation DDOS attacks. The authors run SLICOTS as a dynamic, flexible SDN programmable mechanism. They use SLICOTS in the SDN controller to validate ongoing TCP connection attempts in order to determine whether the hosts are benign or malicious. SYN-Guard, a DDOS attack mitigation approach in SDN, is proposed by Mohammadi et al. [75]. In SYN-Guard, the authors use the SYN proxy technique and implement it as a module in the SDN controller. The controller keeps track of the number of SYN packets with the same source MAC address to apply a forwarding rule or a block rule to the packet on the switches. Birkinshaw et al. [87] proposed

an intrusion detection and prevention system (IDPS) in the SDN that detects and mitigates malicious flows. It countermeasures port-scanning and denial-of-service (DoS) attacks. As defense mechanisms against DOS attacks, they employ TRW-CB and rate limiting, and a QoS strategy based on flow statistics in the IDPS. Furthermore, they propose Port Bingo, a port-scanning detection technique for port-scanning attacks. Hamza et al. [135] proposed an intrusion detection system (IDS) for the IoT paradigm that is a combination of Manufacturer Usage Description (MUD) and SDN. This work demonstrates the shortcomings of approaches that use MUD-derived flow-rules. They get around it by sending the exception packets to off-the-shelf intrusion detection systems. Port scanning, DDoS attacks, and TCP/SSDP/SNMP reflection attacks are all addressed in this work. Ujjan et al. [136] proposed a snort IDS for SDN-based IoT networks that use flow and adaptive polling-based traffic sampling. The authors use deep learning algorithms to detect DDoS attacks. The presented method results in low processing and network head in SDN switches. To improve DDoS detection accuracy, it analyzes network overhead between the control and data planes with each sampling implementation. Patil et al. [137] developed a port scanning-based model to mitigate malicious TCP traffic in SDN that validates the source IP addresses and port numbers of TCP-SYN established connections using customized TCP-FIN packets. This work employs a separate node with two modules integrated with the POX controller to monitor new SYN packets and scan TCP source ports via which SYN packets are received. To verify the request, a separate node sends two FIN flag packets to the source host, using the requested host's source IP address and port number, to persuade the requester to accept its packet. It determines whether it is an attacker or not based on the requester's response. Based on the NAT penetration system, Tang et al. [138] suggested a probe delay-based Port Scanning approach for IoT devices. It adjusts the frequency and techniques of port scanning to balance the IoT network's performance and security requirements. It primarily focuses on optimizing routing to capture IoT data, allowing for comprehensive probing across a wide IP address range. This method cannot probe sleeping IoT devices, which not only prolongs scan times but also overloads the IoT network. S-FOS detects and mitigates port-scanning and DoS attacks in application scheduling, checking research path. It increases the efficiency of available and safe IoT-fog resources.

3.2.2 Fog task scheduling approaches

In this section, we will explain some of the fog task scheduling algorithms. Gill et al. [16] present ROUTER, a novel resource management approach for smart home IoT devices that uses particle swarm optimization (PSO) to improve QoS parameters such as network bandwidth, response time, latency, and energy consumption simultaneously. The authors use the iFogSim simulator to compare ROUTER with gateway-based fog computing (GFC) and virtualization-based resource provisioning (VRP). Stavrinides et al. [128] proposed a hybrid approach in which workflows requiring low communication overhead are placed on the cloud data center and tasks requiring higher communication overhead are placed on the edge layer. These methods are divided into two stages: task selection and resource selection. Tasks are prioritized based on their earliest deadline. If two or more tasks are assigned the same priority, the one with the highest average computational cost is assigned first. When a task is selected, the scheduler assigns it to the resource with the earliest estimated completion time. All resources in the cloud data center and fog layers are considered. While this method has a low percentage of missed deadlines, it comes at a high cost due to the use of cloud data center resources. They also did not use an IoT use case scenario. Because of the use of cloud data center resources, task execution times can be extremely long. Bittencourt et al. [14] proposed a task scheduling approach that categorizes applications as delay-tolerant or delay-sensitive. They examine the behavior of three task scheduling strategies (Concurrent, First Come-First Served (FCFS), and Delay-priority) in a mobility scenario. They employ a module merging mechanism in strategies that assign tasks from the same application to the same fog devices. The authors consider user mobility to be the leading cause of fog device congestion in this work, so they use policy-based load balancing between the fog devices and the cloud data center to reduce it. Maio and Kimovski [126] presented a novel multi-objective scheduler that uses a Pareto-front method to optimize response time, reliability, and cost for task execution in a fog network. To solve the problem, the authors formulate the fog task scheduling as a multi-objective optimization problem and apply the Non-dominated Sorting Genetic Algorithm II (NSGA-II) to the scheduler. The authors use the NSGA-II algorithm with simulated binary crossover and polynomial mutation to present the solutions as a set of Pareto optimal outcomes. To simplify the algorithm, the authors assume that each task has a deadline without taking into account the division of deadlines in the workflow, which is not consistent with reality. Furthermore, in this work, each task executes

immediately after its predecessors have completed. This situation can occur only when all of the resources are fully utilized, which is difficult to achieve. Bitam et al. [20] presented a novel fog application scheduler using the Bees Life Algorithm, a well-known bio-inspired optimization strategy. As the two performance criteria, they considered time and the allocated memory required by mobile user requests. In order to find the best solution, the authors employ a greedy local finding method. They use the C++ programming language to implement their approach and evaluate it in terms of response time, memory consumption, and cost. Sun et al. [19] presented a Multi-objective fog task scheduler based on an improved non-dominated sorting genetic algorithm-II (NSGA-II). Their method consists of two steps: first, the algorithm selects the most appropriate cluster, then assigns user requests to the most appropriate fog device within that cluster. They simultaneously improve service latency and overall stability as two distinct goals. The algorithm minimizes the former and maximizes the latter. In another work, the authors proposed FPFTS [68], a new joint meta-heuristic task scheduler approach, a combination of the PSO and fuzzy algorithms to address delay and network utilization minimization for IoT-fog networks. FPFTS classified the applications as delay sensitive or delay tolerant. For multi-criteria decision-making, the authors apply fuzzy logic to the PSO algorithm fitness function. Finally, they used the iFogSim simulator to test FPFTS for a smart city scenario with varying numbers of users moved, bandwidth ranges, and link latency ranges.

3.2.3 Security-aware fog task scheduling approaches

IoT-fog networks have seen widespread adoption in everyday applications where efficient resource management and security are critical concerns. Subbaraj et al. [139] proposed a multi-objective meta-heuristic scheduler in the fog environment that employs the crow search algorithm (CSA). In their CSA-based scheduler, they have two distinct objectives: deadline and security. The applications require a security level. If fog devices provide the required security levels for the application, then the fog devices are suitable to execute the application based on the security objective. This work considered application and fog device features such as application CPU and RAM requirements, application deadline, fog device CPU, RAM, and bandwidth to meet the deadline. Auluck et al. [78] proposed (RT-SANE), a non-preemptive security-aware fog task scheduling method that is suitable for real-time user requests (e.g., delay-sensitive applications). The authors' primary focus is on reducing the number of times users must wait for their applications (i.e., the time

it takes to find the most appropriate resources) and providing proper levels of security protection. To that end, they use security and deadline as two types of observations when assigning tasks to resources. They categorize the tasks and resources of applications based on their security level. Private applications are assigned to trusted resources, semi-private applications to semi-trusted resources and public applications are assigned to non-trusted resources by the scheduler. Sujana et al. [61] developed the trust-based stochastic scheduling method, which considers security and task scheduling as the two main fields of fog networks. Their approach prioritizes the tasks of the applications based on the stochastic top-level (STL) to find the best task-resource pair for the application. It creates a trustworthy execution environment by taking both direct trust and indirect reputation metrics into account. This work makes use of user feedback to build a reputation-based scheduling model. Daoud et al. [77] proposed a security-driven task scheduling-based distributed trust access control for IoT-fog networks. The presented framework manages new user access by authorizing and calculating their trust level. The trust level is divided into three categories: highest trust, medium trust, and lowest trust. Following authentication, the algorithm assigns scheduling priorities to trusted users. It prioritizes requests from authorized users based on their trust levels, service type, and resource availability. For example, if a user's a shallow trust value, the algorithm provides a less powerful resource than the requested privileges. Shivi et al. [125] describe the duplicate task detection problem for industrial IoT applications, which reduces storage capacity and latency of fog resources. In terms of security, the authors use an ECC-based HM algorithm to encrypt data in the presented method. Over a four-layer fog architecture, they implement a task allocation strategy and a secure deduplication mechanism. For clustering IoT devices, this scheme employs a multi-objective algorithm that takes into account node degree, residual energy, and distance to other devices. The authors evaluate their method based on average latency, energy consumption, user satisfaction, network lifetime, and security strength. Baniata and colleagues [140] use blockchain as a security solution for task scheduling optimization in an IoT-fog network. As the blockchain node, the authors implement blockchain in the device layer (user layer). The fog layer monitors and controls communications in the form of digital currency between the two types of blockchain nodes and task requester nodes in the device layer. They use the Ant Colony Optimization (ACO) algorithm in their presented task scheduler to protect the privacy of the user while also reducing execution time and network load. The same authors of this paper (i.e., S-FOS) recently

proposed FUPE [124], a security-aware task scheduling algorithm, to protect fog scheduling services from TCP DDOS attack. To distinguish between benign and malicious applications, FUPE employs TRW-CB and rate-limiting algorithms. Based on this, FUPE computes a final solution for task scheduling using multi-object PSO. Using a fuzzy-based multi-objective PSO approach, FUPE can find a balance point between efficiency and security objectives. To manage network flows, the authors apply the SDN paradigm to fog architecture. When malicious activity is detected, SDN allows FUPE to immediately block network traffic anomaly.

3.2.4 Evaluating of the background methods

For the applications in the works [139, 78, 61, 77], the security level is determined by a probability distribution function or a predetermined random value. They choose resources to meet the applications' security level limitation using a specified amount based on an SLA. Based on a QoS agreement, the SLA is the degree of service users anticipate from an IoT provider. They also cluster resources using local, remote, public, semi-private, and private tags. Local or private resources are assumed to have more robust intrusion detection systems (IDSs), higher security requirements (e.g., higher availability, authentication, integrity, and secrecy), and a lower level of accessibility. As a result, their protection is based on different levels of IDSs that are chosen based on probability. Shivi et al. [125] discuss the usage of data encryption to ensure the privacy and security of users' data. In another paper, Baniata et al. [140] employed blockchain to enable identity verification and privacy for IoT applications. S-FoS, in contrast to these works [125, 140], detects and mitigates the source of assault. Unlike our previous article, FUPE [124], which only evaluated one type of attack, this paper considers three type of attacks. FUPE assumes that attackers can compromise fog devices and treats both IoT devices and fog devices as attackers. Both IoT devices and fog devices are considered attackers. It uses a multi-objective optimization to balance the security and efficiency of fog devices. S-FoS, in contrast to previous research, exclusively considers IoT devices as attackers and employs multi-objective optimization to find a compromise between load balancing and delay. In contrast to Bitam et al. [20], Stavrinides et al. [128], and Sun et al. [19], S-FoS analyzes the network level. The suggested methods are designed for application-level use without taking into account the overhead of implementing them at the network level. In addition, unlike these methodologies, S-FoS takes energy usage into account when evaluating. In terms of scalability, these works are likewise

lacking. S-FoS, on the other hand, provided the network with a high level of scalability. It divides the network into many fog areas and allows new fog devices to be deployed. The architecture is also designed in such a way that distributed SDN controllers can be used to augment it. S-FoS, unlike Gill et al. [16], has a scalable architecture. S-FoS, unlike Bittencourt et al. [14], takes into account the computing capabilities of all fog devices in each fog region. S-FoS considers load balancing for scheduling, unlike De Maio et al. [126] and our previous work [68]. There are no security or privacy safeguards in any of these works. They are all reliant on the security unit as well as additional defense devices. Protecting scheduling services from attacks results in removing of malicious requests, resulting in a decrease in response time, network utilization, and energy consumption. Furthermore, the scheduling services are not dependent on the security unit, and any fault in the security unit does not affect on the scheduling services. This method is helpful for time-sensitive applications.

3.3 Proposed approach

In this section, we go over the proposed S-FoS in detail. To that end, we will first describe the reference architecture. The problem statement is then reviewed. Then we will go over the SDN switches and controllers functions. Following that, we will go over the TRW-CB, Rate Limiting, and Entropy methods. Finally, we describe the proposed security-aware Scheduler.

3.3.1 Reference architecture

This section gives an account of the S-FOS architecture, which is used to connect IoT and fog devices and execute user requests with low delay. A three-level architecture is common in IoT-fog networks [82, 24]. In response to this, we use a three-layer architecture to address SDN needs for IoT-Fog networks using S-FoS [141, 142, 124].

It applies SDN features to an IoT-fog network, as shown in fig. 3.1. In the application layer, also called device layer, there are IoT devices (user devices). Users submit their applications for execution to fog devices (fog resources at the edge of the IoT-fog network). The data plane, also known as the infrastructure layer [143], is made up of fog devices and SDN switches that are grouped into fog regions. The execution of user applications is handled by fog devices. IoT devices in the application layer communicate with the SDN switch, which

Table 3.1: Comparison of related works.

Refs.	Algorithms	Tool	Observations	Security	Advantages	Disadvantages
[14]	Concurrent/FIFO/ Delay-priority	iFogSim	fog devices' capacity and applications' priority	No	+ User mobility	- Unclear prioritization method for applications - Relying on security unit
[128]	Hybrid heuristic	C++	Communication required for tasks, The cost of communication between the links, The applications' tasks deadline, The estimated completion time of the resource.	No	+ A proper deadline miss ratio.	- High financial cost, - Delay can be very high, - Not considering network level, - Relying on security unit.
[20]	Bees swarm	C++	CPU execution time, Allocated memory	No	+ Considering allocated memory + Low algorithm execution time	- Cloud computing is not support, - Static algorithm, - Not considering network level, - Relying on security unit.
[16]	PSO	iFogSim	Response time, energy, latency, and network bandwidth	No	+ Energy + Latency and response time + Network bandwidth	- No fault tolerance assurance, - High run time, - Relying on security unit
[126]	NSGA-II	Monte-Carlo	Total response time of the tasks, Reliability of the physical devices, Sum of the execution costs of each task, The deadline of the tasks	No	+ Considering the reliability for executing a workflow, + Considering User cost for a deployment, + Considering workload response time	- Not considering tasks' waiting time, - Not considering the tasks' deadline, - Relying on security unit.
[19]	NSGA-II	MATLAB	Proximity and the resource utilization of fog clusters, The required completion time of a service, Fog devices' calculation capabilities, The degree of a fog device's reliability	No	+ Low execution time, + High scalability, + Low latency	- Not considering network level, - Relying on security unit.
[125]	multi-objective WOA	iFogSim	Node residual energy, Node degree (no. of neighbor connections), Distance between nodes, The hash information of the fog devices, Task length and processing delay	Yes	+ Enabling detecting duplicate tasks, + Reducing the latency of the cloud server, + Enabling to encrypt data, + Increasing network lifetime, + Reducing energy consumption, + Improving security strength	- High computational complexity, - System fails in case of cluster head failure.
[140]	Multi-Objective ACO	Python	Resources' computing power, The length of tasks, Cost of machines	Yes	+ Low latency, + high privacy awareness, + Low network load, + High scalability	- High time complexity, - Execution time is not validated, - Energy conservation is not justified
[61]	Stochastic	CloudSim	Service Level Agreement (SLA), Tasks with the lowest execution start time	Yes	+ multi levels of trust	- Relying on the users' feedback
[139]	Crow search algorithm	iFogSim	Security level applications' features fog devices' features	Yes	+ Considering deadline, + Considering privacy, + Considering efficiency.	- The definition of security level is imprecise.
[78]	Multi-criteria task priority	iFogSim	The deadline of the jobs Available fog devices' spare capacity The privacy of the jobs	Yes	+ Considering data privacy by applying security tags to the resources.	- No attack defence mechanism
[77]	Multi-criteria task priority	iFogSim	Level trust of the jobs Arrival time of the jobs Availability of the resources	Yes	+ Assigning security tag to the resources.	- No attack defence mechanism, - Relying on the users' feedback
[68]	Hybrid (PSO-Fuzzy)	iFogSim	Fog devices' ram capacity, Fog devices' CPU capacity, Fog-devices' link bandwidth, Computational needs of the applications	No	+ Considering the user's mobility pattern, + Considering fog devices' computational capacity, + Reducing application latency, + Reducing network utilization	- Static scheduling, - Relying on security unit.
FUPE	Hybrid (MOPSO-Fuzzy)	Matlab	Available computational capacity of the fog devices, Available memory capacity of the fog devices, Available link bandwidth of the fog devices, Computational needs of the applications, Fog devices' trust degree	Yes	+ Considering Security patterns in evaluation, + Mitigating the attacks by detecting malicious users' device, + Mitigating the attacks by detecting compromised fog device, + Considering available amount of fog devices' computational capacity, + Reducing application latency, + Reducing network utilization	- Relies on fault tolerance techniques, - Relies on spoofing defences approaches
S-FOS	Hybrid (NSGAIII-Fuzzy)	IoTSim-Osmosis	Available computational capacity of the fog devices, Available memory capacity of the fog devices, Available link bandwidth of the fog devices, Computational needs of the applications, Fog devices' trust degree	Yes	+ High scability, + Considering Security patterns in evaluation, + Mitigating the attacks by detecting malicious users' device, + Dynamic scheduling, + Low delay, network usage, and energy consumption	- Relies on fault tolerance techniques, - Relies on spoofing defences approaches

communicates with the fog devices. We assume that no hosts are connected to the SDN switch via a hub or a layer 2 switch. Because we are using SDN abilities, all flows can be managed by SDN switches using flow rules injected through OpenFlow from the controller. Finally, the Control plane, also known as the control layer [143], consists of cloud data centers and an SDN controller. The cloud data center is responsible for running user applications that have been offloaded by fog devices. In the presented architecture, there is a SDN controller (i.e. cloud gateway) with a global view of the entire network. The SDN controller is in charge of the SDN switches' control and management. On a regular basis, the SDN switches query the fog devices' information within the fog region that controls them in order to gather data directly from them. The information includes events such as fog device leave/join, fog device up/down, link up/down, and the status of fog device computational resources (i.e., CPU, RAM, and the link bandwidth utilization). Furthermore, SDN controller collaborates with SDN switches to modify/update their flow tables and flow paths (forwarding and backward paths) via the OpenFlow protocol which is a central aspect in SDN networks.

3.3.2 Problem Statement

This work is based on the assumption that malicious IoT devices can perform attack on fog devices that affects SDN controller in the same fog region to bring the network down. The attack on the infrastructure layer begins with reconnaissance of fog devices to discover port vulnerabilities. Probing a potential open ports clears the way for further attacks and is frequently used as a precursor to TCP/UDP DDOS attacks. The security component of S-FoS, which employs anomaly detection techniques, serves as a deterrent to TCP/UDP DDOS-based port scanning attacks. SDN switches and controllers get resource suffocated as a result of this behavior. Malicious flows reduce network performance, causing user dissatisfaction. Security mechanisms must be deployed to protect security-critical users' applications running on IoT resources from being attacked. The problem we address in this paper is how to assign validated user requests to fog devices while maintaining efficiency. We solve this problem by employing a hybrid strategy, which allows the workflow scheduler to provide optimal performance in both *security* and *efficiency*. Our proposed approach first validates IoT device requests and then blocks malicious ones. The benign ones are then assigned to fog devices while *load balancing* and *delay* are taken into account.

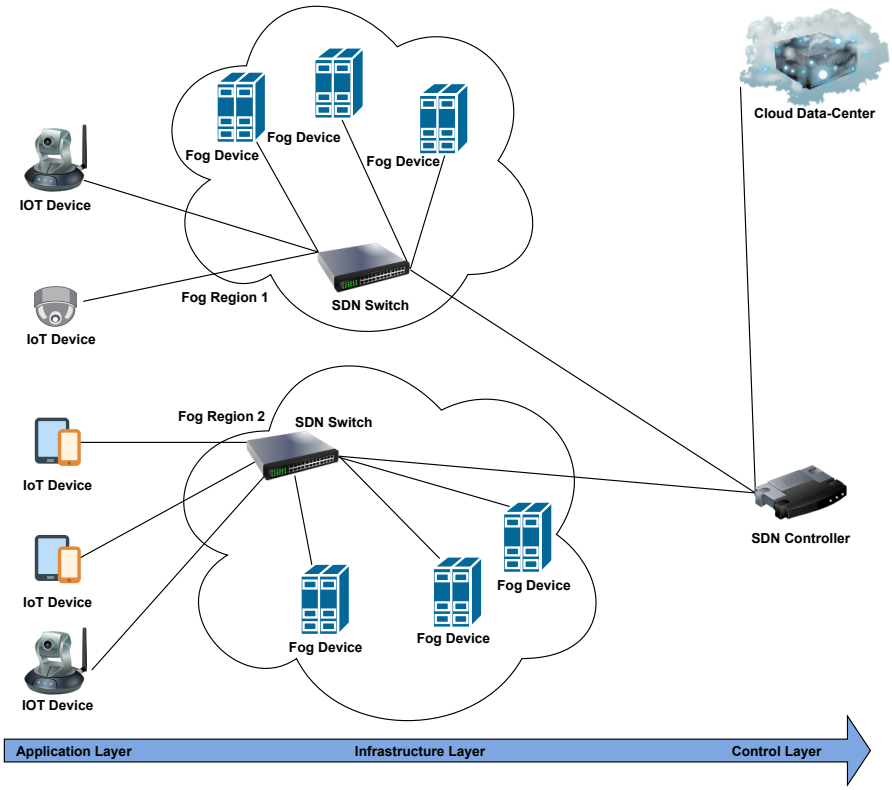


Figure 3.1: S-FOS architecture.

3.3.3 The SDN switches and controllers functions

In order to handle flows, the SDN switches cooperate with the SDN controller. The SDN switches have internal flow tables with their own set of flow rules. The data are forwarded by them in accordance with the flow rule entries specified by the SDN controller. In fact, the SDN controller proactively calculates optimal paths from the SDN switches to the fog devices, and then installs the paths as forwarding rules on the flow table of the SDN switches. Each flow rule contains headers (i.e., source IP, source MAC, destination IP, destination MAC, and TCP/UDP port) as well as a set of actions (i.e., forward, and drop) to handle matched packets. Incoming packets are matched by the packet headers. The SDN controller manages the flow tables in the SDN switches via the OpenFlow protocol. Security and Performance optimization functions are implemented in our plan in the SDN controllers. The attacks are detected by the *Flooding and Port Scanning Detector* module inside the SDN controller. In addition, with the efficiency metric, the *Flow Assigner* module assigns flows to the most proper fog device in the SDN controllers. It is worth mentioning that, at the initiation phase, the SDN controller temporarily installs paths for a short while to enable the flows to be routed through the SDN switches. Moreover, the SDN controller periodically collects some useful statistics about each traffic flow associated with an IoT device connection, such as source and destination IP addresses, source and destination port numbers, the number of requests made by each device, and so on. It detects TCP/UDP DDOS, and port scanning attacks using these values. The *Flooding and Port Scanning Detector* module, uses a Mamdani fuzzy function to detect attacks and disables the offending IoT device. If no attack is detected on the requester nodes, the SDN controller permanently adds the forwarding rule to the SDN switch flow table, and the *Flow Assigner* module considers the requests during the scheduling process. Otherwise, the SDN controller adds the drop rule to the SDN switch flow table to mitigate future attacks. Finally, the *Flow Assigner* module is in charge of assigning validated user requests to fog devices. It employs an NSGA-III method that takes into account both *load balancing* and *delay* when assigning flows to the fog devices.

3.3.4 Anomaly Detection Approaches

The Fuzzy-based functions of the *Flooding and Port Scanning Detector* module in the SDN controllers require the output values of anomaly detection algorithms. They collect raw data from the SDN switches and then run anomaly

detection algorithms such as TRW-CB, Rate Limiting, and Entropy [88, 62]. The functionalities of these three algorithms are as follows:
TRW-CB algorithm To countermeasure TCP DDOS attack, this algorithm uses the rate of successful/unsuccessful IoT device TCP connection attempts to the fog devices to detect anomalous behavior [87]. This method detects TCP DDOS attack when the number of *TCP SYN packets* sent from an IoT device (Requester) to a fog device in the infrastructure layer (Requested) is significantly greater than the number of *TCP ACK packets*.
Rate Limiting algorithm To countermeasure UDP DDOS attack, this algorithm is based on the assumption that an IoT device does not send a large number of UDP packets in a short period of time, whereas a malicious IoT device does [87]. In threshold-based approaches, this technique detects UDP DDOS behaviour if the rate of UDP packets exceeds a predefined threshold value. The attack is determined by a *high fuzzy set* in fuzzy-based approaches such as S-FoS.
Entropy algorithm In the baseline, this algorithm estimates the maximum/minimum entropy for benign traffic detection [88, 74]. S-FOS employs Claude Shannon's Entropy method[144] to assess the randomness of destination port numbers associated with random variables [145]. S-FOS generates output with a uniform distribution in the [0, 1] range using normalized entropy [74] to detect the probed port numbers. Figs. 3.2 to 3.5 indicate the process diagram of the TRW-CB, Rate Limiting and Entropy approaches.

3.3.5 S-FOS: the security-aware Scheduler

This section describes the approach that is being presented.

Fuzzy approach

In the security part of our approach, we used the Mamdani fuzzy logic.
Flooding and port scanning attacks S-FOS uses the entropy technique to prevent port-scanning, which occurs when an attacker sends packets related to a large number of different ports in order to detect TCP/UDP port probes that are accessible. For port scanning attacks, attackers typically probe the most commonly used ports that provide well-known services. The entropy of destination port numbers reveals a diverse set of streams (variety of ports). To determine the variety of ports probed by requesters, we use the entropy of destination port numbers. During a port scanning attack, the entropy distribution

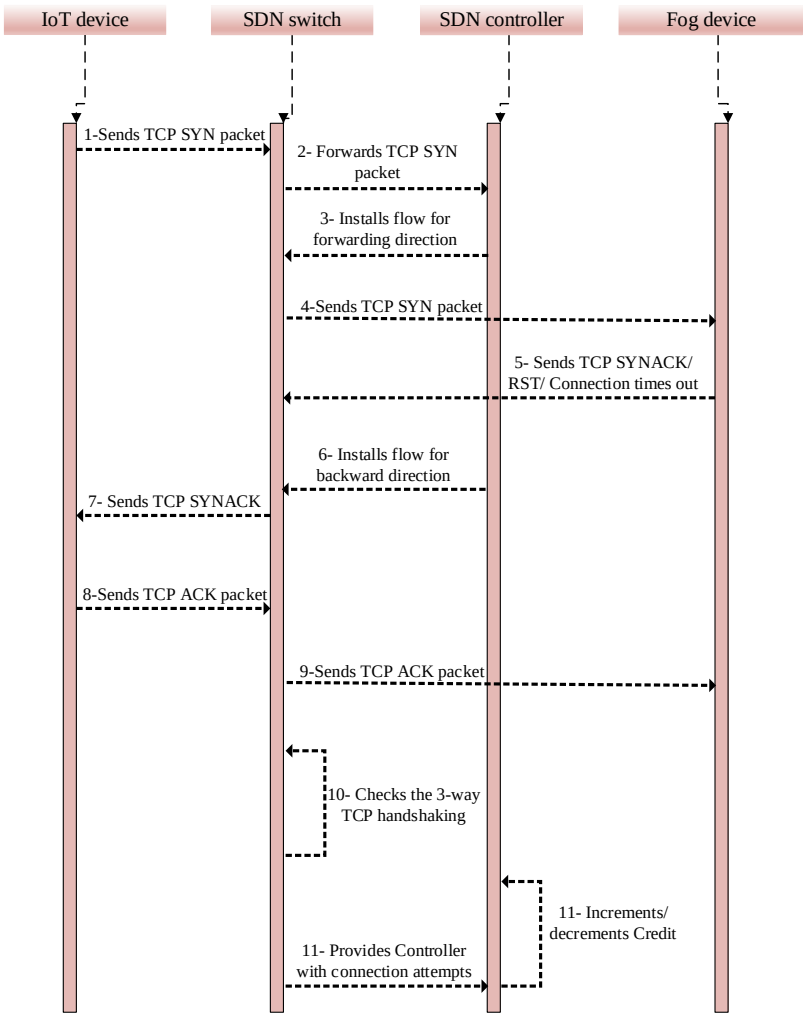


Figure 3.2: The sequence diagram of TRW-CB algorithm.

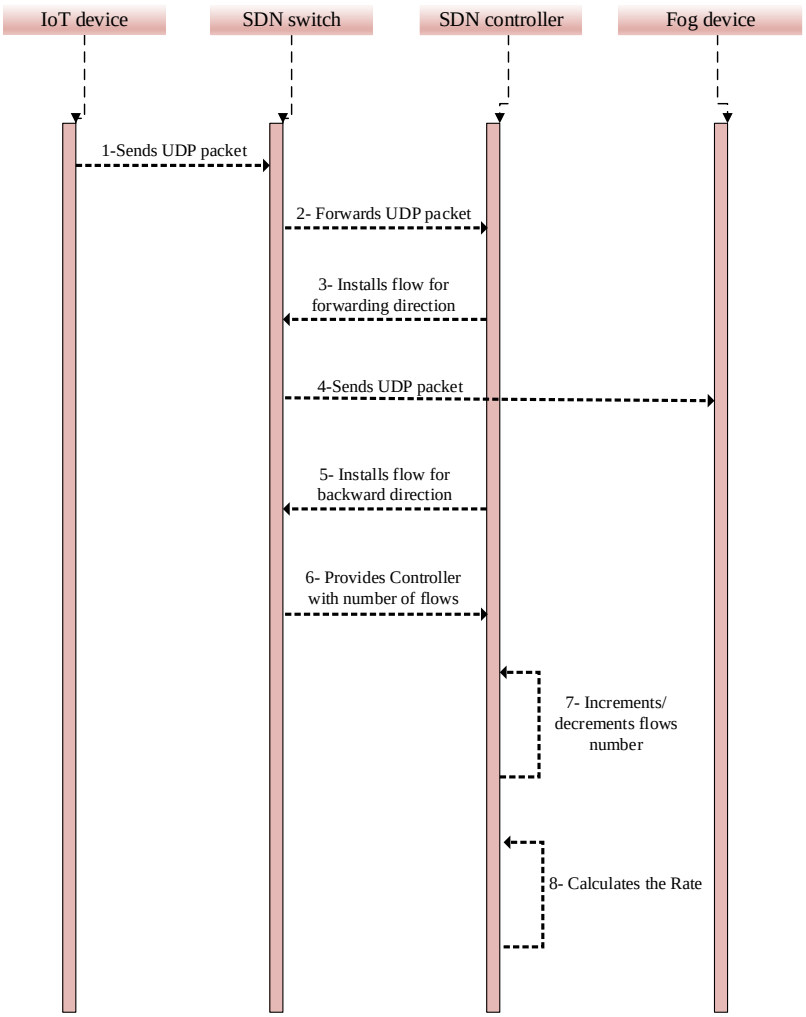


Figure 3.3: The sequence diagram of Rate limiting algorithm.

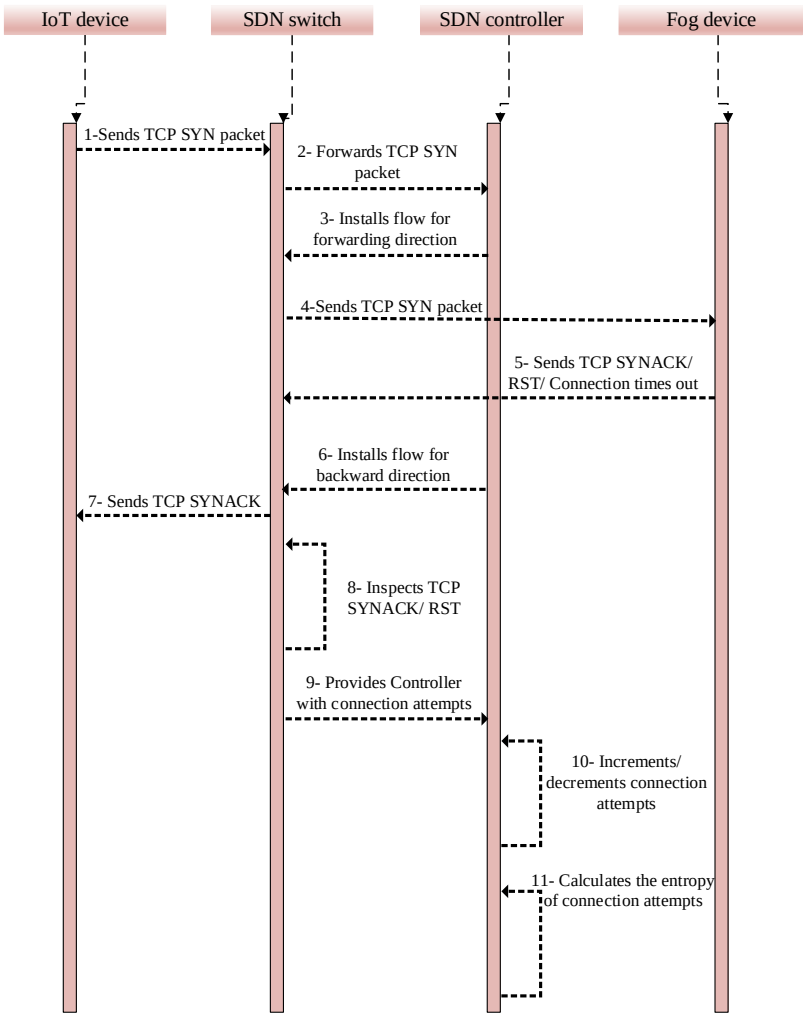


Figure 3.4: The sequence diagram of Entropy algorithm for TCP packets.

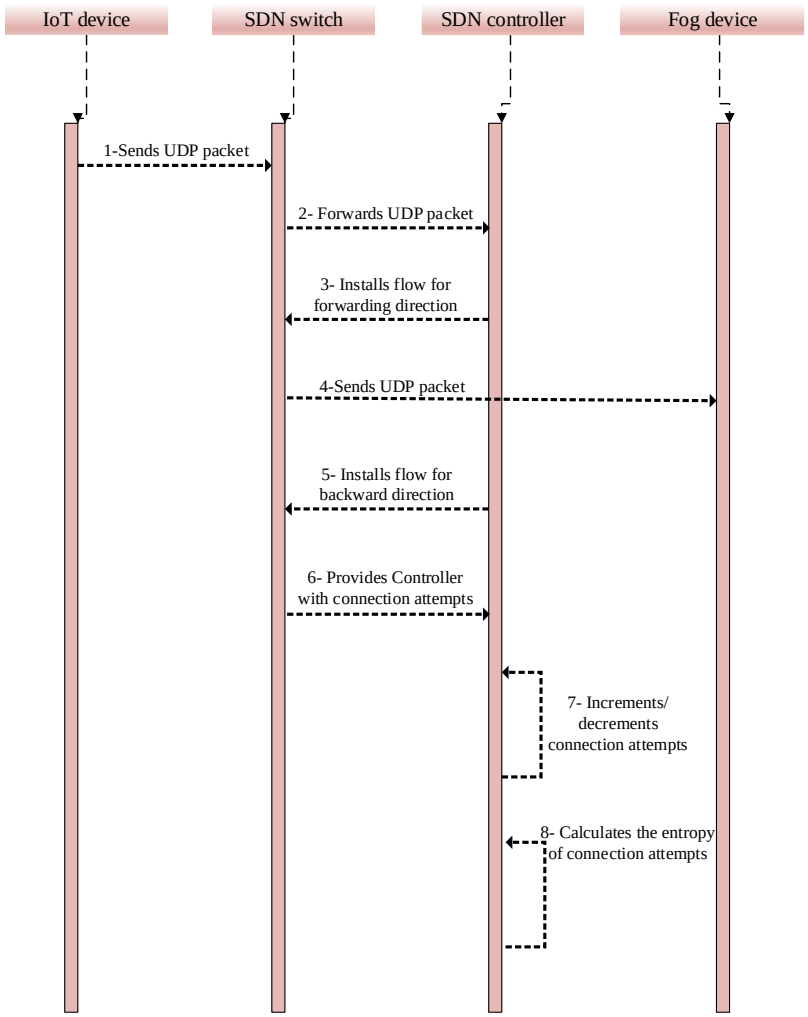


Figure 3.5: The sequence diagram of Entropy algorithm for UDP packets.

of destination port numbers becomes more random, whereas the randomness of destination port numbers is very low for benign traffic.

TRW-CB is used to counter TCP DDOS attacks because it is a proper mechanism for connection-oriented protocols. Furthermore, unlike TCP DDOS attack, UDP DDOS has a high attack rate, so we use Rate Limiting to mitigate UDP DDOS attack. For this purpose, the *Flooding AND Port Scanning Detector* module examines the packet type. It feeds the TRW-CB outputs and the entropy of the destination port numbers into a Mamdani fuzzy inference system for TCP packets. Furthermore, for UDP packets, it uses the outputs of the entropy of the destination port numbers and Rate Limiting as input parameters to the Mamdani fuzzy inference system. These are the input parameters for the fuzzy function, and the output of the fuzzy function is used to remove malicious IoT devices. It should be noted that the fuzzy feature is used to countermeasure both flooding and port scanning attacks at the same time. The following are the fuzzy sets for the aforementioned parameters and the result parameter:

- $Credit$: $\in$ {Low; Medium; High}
- $Rate$: $\in$ {Low; Medium; High}
- $Entropy$: $\in$ {Low; Medium; High}
- $Result$: $\in$ {Attack; Potential; Safe}

In the above fuzzy rules, we use *Credit*, *Rate*, and *Entropy* to represent the TRW-CB, Rate Limiting, and SDN switch port numbers entropy, respectively. Tables 3.2 and 3.3 show the fuzzy rules used in the *Flooding And Port Scanning Detector* module. Moreover, figs. 3.6a, 3.6b show the fuzzy sets used for the Credit/Rate and Entropy parameters, as well as the results.

NSGA-III approach

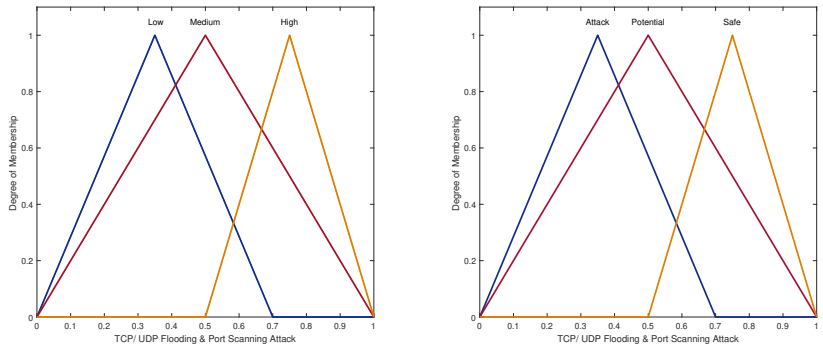
In the performance optimization part of our approach, we used an NSGA-III algorithm. In the *Flow Assigner* module, S-FOS employs the NSGA-III approach to implement an evolutionary algorithm that optimizes both *delay* and *load balancing*. The *Flow Assigner* module is in charge of allocating flows to the best fog device for executing validated user requests. The fog device that runs the users' flows should provide a proper response time. The NSGA-III method is a novel approach to solving multi-objective optimization problems. The main advantage of NSGA-III over NSGA-II is a significant improvement

Table 3.2: Fuzzy rules for TCP flows.

Credit	**Entropy**	**Result**
Low	Low	Potential
Low	Medium	Attack
Low	High	Attack
Medium	Low	Potential
Medium	Medium	Potential
Medium	High	Potential
High	Low	Safe
High	Medium	Safe
High	High	Potential

Table 3.3: Fuzzy rules for UDP flows.

Rate	**Entropy**	**Result**
Low	Low	Safe
Low	Medium	Safe
Low	High	Potential
Medium	Low	Potential
Medium	Medium	Potential
Medium	High	Potential
High	Low	Potential
High	Medium	Attack
High	High	Attack



(a) Input parameters for Flooding And port scanning detector module.

(b) Output parameter for Flooding And port scanning detector module.

Figure 3.6: Fuzzy sets

in its next generation selection mechanism. NSGA-III selects the next generation chromosomes using a series of reference points rather than the crowding distance operator used by NSGA-II (i.e., the distance between a solution and reference points). The crowding distance is the distance between two solutions (i.e., the distance between two neighboring solutions [130, 131]). Security-aware workflow scheduling is defined by S-FOS as a multi-objective optimization problem that takes into account both the *load balancing* and *delay* of fog devices in IoT-fog networks at the same time. S-FOS encodes the workflow scheduling strategy first. It then employs the multi-objective fitness function to achieve both *load balancing* and *delay* goals. Then, using crossover and mutation operations, it generates new schedule solutions. Furthermore, it employs a well-distributed set of reference points in the selection process.

We propose a method in which virtual machines on the resources are represented as genes, S-FOS assigns user requests to the genes, and sets of genes form chromosomes. For example, if there are 8 user requests and 5 fog device VMs in a fog region, a possible chromosome might look like this: $\{J_1(1), J_2(3), J_3(2), J_4(1), J_5(1), J_6(3), J_7(3), J_8(2)\}$. Assume that the parent population of each generation is P_j, and the child population created from P_j is C_j, with both having N members. NSGA-III uses $B_j = P_j \cup C_j$ to make the best choice among N members of the parent and child populations, allowing the elite members of the parent population to be retained. To this end, NSGA-III first sorts the B_j population at various levels $(LS_1, LS_2, LS_3, \cdots)$.

$(LS_1, \cdots)$ is the Various levels of non-dominantly sorted members where LS_1 is the Non-dominated members of the last level sorted. Then, for the New Population of G_j, each non-dominant level is generated, starting with LS_1, until the size of G_j is equal or first exceeds N. Consequently, all B_j population solutions from the $L+1$ level on are refused. G_j is the new genuine non-dominant population. We refer to the last level as level L.

The length of the chromosome was determined by the number of VMs. The multi-objective fitness function in S-FOS determines the optimality of a possible solution. Each chromosome is unique, and each chromosome represents a scheduling strategy that represents a solution to the multi-objective optimization problem. The goal of S-FOS is to find the best scheduling strategy for each fog device in order to minimize the two objectives. The fitness of a solution is determined by achieving the best trade-off between the two objectives. Eq. (3.1) and Eq. (3.2) define the *load balancing* and *delay* objectives, respectively [146, 147].

$$Load\_Balance = (\sqrt{\sum_{j=1}^{Resources} (U_j^{CPU} - \overline{U^{CPU}})^2} + \sqrt{\sum_{j=1}^{Resources} (U_j^{BW} - \overline{U^{BW}})^2})/2 \tag{3.1}$$

Where U^{CPU}, $\overline{U^{CPU}}$, U^{BW}, and $\overline{U^{BW}}$ represent the amount of CPU used for each resource, the average amount of CPU used for resources, the amount of bandwidth used for each resource link, and the average amount of bandwidth used for resource links.

$$Delay = ((Flows\_MIPS)/(Resource\_MIPS)) \tag{3.2}$$

Where $Flows\_MIPS$ and $Resource\_MIPS$ represent the application's (flow's) CPU requirement and the resource's available CPU, respectively.

Eq. (3.3) is used to combine the results of the two fitness functions into a single answer solution..

$$Outcome = (Load\_Balance * \alpha) + (Delay * (1 - \alpha)) \tag{3.3}$$

Where $Load\_Balance$ and $Delay$ are the outcome of the Eq. (3.1) and Eq. (3.2). We use the two predefined weights α and $1 - \alpha$ to prioritize the

fitness function results. We assume that the fitness functions have the same priority and assign 0.5 to both of them.

In principle, S-FOS begins by generating a set of chromosomes known as the population. It generates several random populations of chromosomes, then randomly assigns fog devices to their genes, and computes the *load balancing* and *delay* fitness values for all of the chromosomes. The fitness function of the genes is the *weighted sum* of the *load balancing* and *delay* functions. We employ the weighted sum approach to arrive at a single optimal solution that determines the suitability of the current fog device for the user's request. Then it generates reference points and associates them with the solutions. To that end, it compares the solution to a candidate solution's reference point. The algorithms replace them if the value of the candidate solution is greater than the value of the current solution. In this step, a normalized scale on a hyperplane of the $(M-1)-dimension$, that exists equally on all objective axes, is assigned by NSGA-III. The number of objective functions is denoted by M. The total number of reference points (N_R) in the M-objective problem is provided by placing the number of divisions (N_D) next to each objective by means of eq. (3.4).

$$N_R = \binom{M + N_D - 1}{N_D} \tag{3.4}$$

The perfect population point S_t is determined by calculating the perfect point $\overline{z} = (z_1^{min}, z_2^{min}, z_M^{min})$ and the minimum value of z_j^{min} for each objective function $j = 1...M$ in $U_{\xi=0}^{t} S_t$. Normalization is performed for each chromosome based on the fitness function value. Each objective value of S_t is translated by subtracting the objective f_j from the objective z_j^{min}, so that the ideal point of translated S_t is a zero vector. Eq. (3.5) indicates the *translated objective*.

$$f_j'(x) = \ f_j(x) - z_j^{min} \tag{3.5}$$

Following that, using the solution $(x \in S_t)$ that minimizes the achievement scalarizing function generated by $f\ '_j(x)$ and a weight vector in close proximity to the *j*th objective axis, an extreme point $(Z^{j,max})$ in the *j*th objective axis is discovered. Using eq. (3.6), the algorithm normalized each objective fitness function.

$$\begin{aligned} &\forall j \in J, \quad J = \{1, 2, ..., M\} \\ &f_j^n(x) = \ f_j'(x) / \ a_j \end{aligned} \tag{3.6}$$

The reference points are the averages of CPU, RAM, and storage space in fog devices, taking into account the associated average (i.e. it determines the fog devices which have adequate amount of computational and storage capacity in each generation). S-FOS then compares current solution genes to reference points that are candidate solutions (possible fog devices). S-FOS replaces candidate solutions (reference points) if their value is greater than the current solution. In other words, we filter the fog devices based on a criterion (i.e., resource capacity) and use them as reference points, and in this way, we use more suitable fog devices in the initial step and in the next generation.

There are many random populations of chromosomes in each generation. Each of the genes contains fog devices that are linked to a specific request. We compare the weighted sum answer obtained by two fitness functions to the possible request for a specific fog device. The sum of the fitness functions of each chromosome's genes yields the overall fitness function. The algorithm continues by keeping 20% of the chromosomes and using the remainder for the cross over step. It employs a series of two distinct chromosomes for each generation based on their overall fitness value. S-FOS performs a single-point crossover operation to combine the two chosen chromosomes and create a new individual. Because the length of the two chromosomes and the order of the genes are the same (e.g., the first genes of both chromosomes contain the same request, but each has a different fog device), the crossover step produces a set of one chromosomes. Fig. 3.7 indicates the single-point crossover. In this figure, the latter part of the chromosomes exchange their indexes. As a result, the latter gens of chromosome *A* contain fog devices 2, 3, 2, and 1. While fog devices 3, 1, 1, and 3 are found in the later gens of chromosome *B*.

The final operation in each generation is mutation, which changes one of the gens at random. It causes the gen to be equipped with a different fog device. Due to the rarity of mutations in genetics, we apply the mutation operator to 10% of the cross over output chromosomes. The mutation operation is depicted in fig. 3.8. The result chromosomes from the previous figure are used in this figure (i.e., cross-over step). We assume that the mutation occurs on both outcome chromosomes. As a result, fog devices 2 (R2) are found on chromosome *A*'s second gen, while fog devices 3 (R3) are found on chromosome *B*'s seventh gen. S-FOS reintroduces these chromosomes into the current generation, creating a new population for the next generation. S-FOS perpetuates all processes on the population for future generations. We set the breaking condition (i.e., iteration number) to a predetermined number or whenever both of the crossover's selected chromosomes are homologous (i.e., the genes of the chro-

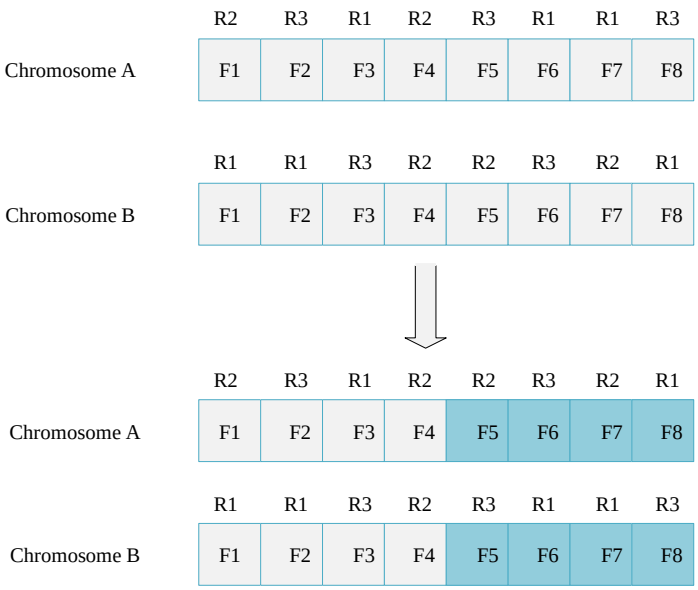


Figure 3.7: The single-point crossover method (i.e., F = 8; numbers of flows and R = 3; number of fog devices)

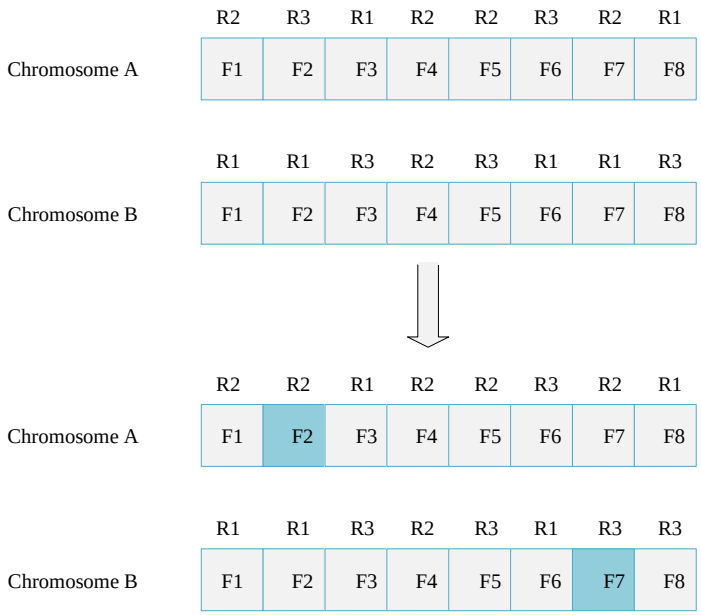


Figure 3.8: The mutation method

mosomes contain the same fog devices). The steps of the S-FOS algorithm are depicted in 5.

Algorithm 5 NSGA-III based algorithm in Flow Assigner module.

Input: $N(PopulationSize), M(IterationNumber),$
$J(Workflows)$
Output: $Best - Chromosome$

1: $P \leftarrow$ GeneratePopulation()
2: $P \leftarrow$ InitializePopulation()
3: $S \leftarrow$ Non-Dominated-Sort(P)
4: $RF \leftarrow$ Generate-Reference-Point()
5: $RF \leftarrow$ Initialize-Reference-Point()
6: **repeat**
7: **for** $i = 1$ *to* M **do**
8: $S \leftarrow$ Non-Dominated-Sort(P)
9: $O \leftarrow$ Normalize-objectives(S)
10: Associate(S, O, RF)
11: Compare(RF, S)
12: **if** RF is better than S **then**
13: Replace(RF, S)
14: **end if**
15: $SCHR \leftarrow$ Select-chromosomes(S)
16: $CHR \leftarrow$ Cross-Over$(SCHR)$
17: $BestChromosomes \leftarrow$ mutation(CHR)
18: $NextGeneration \leftarrow BestChromosomes()$
19: **end for**
20: $BestChromosome \leftarrow BestChromosomes()$
21: **until** $stoppingCriterion()$
22: **return** $BestChromosome$

Alg. 5 Description. In Alg. 5, S-FOS first generates a non-dominated population of chromosomes and randomly initiates them. Following that, it sorts the population at different levels (lines 1 to 3). Then, for each iteration, it generates the reference point chromosome based on the mean of the fog devices' CPU, RAM, and storage (lines 4 to 5). It sorts the population at different levels and normalizes it using eq. (3.6) in each iteration for the current population. The population is then subjected to non-dominated sorting (lines 6 to 10). Thereafter, It compares the current chromosome's genes to the reference

point's genes for each population, and the one with the higher value is considered the current solution. It employs eq. (3.3) to accomplish this (lines 11 to 14). Following that, it performs a crossover operation to select a set of two chromosomes from the current generation (iteration). Thereafter, it executes the mutation operator and adds the result back to the next generation (lines 15 to 19). S-FOS continues to make the same progress until the stopping criterion is met. The stopping criterion is the point at which either the input chromosomes for the crossover step are homologous (i.e., their genes are completely identical) or the algorithm reaches the maximum number of iterations (lines 20 to 21). Finally, as the solution, it chooses a chromosome with the best fitness function outcome (line 22).

3.4 Performance Evaluations

In this section, we present the experiments that were carried out in order to assess the performance of the proposed S-FoS approach. We compared S-FoS to an improved version of FUPE, which uses a MOPSO algorithm to detect and mitigate TCP and UDP DDOS attacks [124], as well as a standard NSGA-II algorithm that does not take security into account. We used IoTSim-Osmosis [132], a java-based simulator built on the CloudSim framework [148], to implement our approach. In addition, we used a fuzzy logic tool called Xfuzzy [100] to implement the fuzzy-based functions found in the SDN controller.

3.4.1 Simulation Setup

To simulate the methods, we applied the specific features to the scenarios that is indicated in the Tab. 3.4. To generate entity requirements, we assume that each of them requires RAM, CPU, and bandwidth.

Table 3.4: Entity Requirements.

Requirements	Cloud data center hosts	Fog device	Flow
RAM	1000000 MB	20000 MB	10000 MB
CPU	8000000 MIPS	80000 MIPS	250 MIPS
Bandwidth	10000 Mbps	200 Mbps	100 MbPS

Table 3.5: Other Configurations.

Cloud hosts number	Hosts VM number	IoT Device flow size	flows' tasks size
2-6	2-6	50-200 Mb	200-500 MI

Simulation metrics

We use the following metrics to compare the performance of S-Fos to the other two methods:

- *Response time:* The total amount of time it takes to respond to a flow is referred to as response time. It begins when IoT devices send flows and ends when they receive a response. It's the sum of the flows' total running time.
- *Network utilization:* This metric refers to the amount of data that is sent back and forth across links as a result of flows, cloud data center hosts, SDN controller, fog devices, SDN switches, and IoT devices. Eq. (3.7) is used to calculate network utilization in *Mb*. In this equation, flow size is the length of data encapsulated in the flow, expressed in *Mbps*. Besides, the time it takes for flows to go from the source to the destination, measured in seconds, is known as flow transmission time.

$$Network\ usage = (\sum_{x=1}^{Requests} Flow\ Transmission\ Time * FlowSize) / flownumbers \tag{3.7}$$

- *Energy consumption:* The CPU, RAM, and network interface all contribute to the node's energy usage. Furthermore, when compared to other system resources, the CPU consumes the most energy. As a result, the node's energy usage simply needs to account the CPU's energy consumption [149]. It is illustrated in the work [150], a node's energy consumption can be defined as a linear function of its CPU use. We discovered that the energy consumption is computed using the Eq. (3.8) after reviewing the simulator. N_{idle} and N_{busy} are the average energy consumption of the node when it is idle and fully utilized, respectively. The CPU utilization of the j_{th} node is represented by U_j^{CPU}. This metric obtains the total energy consumption of cloud data center hosts, fog

devices, SDN controller, SDN switches, and IoT devices.

$$Energy\ consumption = N_{idle} + (N_{busy} - N_{idle}) * U_j^{CPU} \quad (3.8)$$

- *Confidence interval (CI):* It is a concept used to assess the accuracy of the results. The percentage of error in the results is reflected here. In other words, it is regarded as a measure of correctness in relation to the results. It reveals the range of population values within a certain level of correctness. In this work, we obtained the results with a 95 percent CI. Eq. (3.9) is used to calculate this metric. In this equation, 1.96 equals 95% CI. Besides, n, σ, and $\overline{\eta}$ represent the number of samples, standard deviation, and samples average, respectively.

$$Confidence\ interval = \overline{\eta}\ \pm\ (1.96 * (\sigma/\sqrt{n})) \quad (3.9)$$

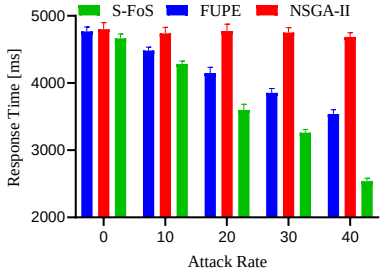
Implementation scenarios

We compare our proposed approach to the FUPE [124] and NSGA-II [133] approach methods over the following attack rates, various IoT devices, and various fog devices:

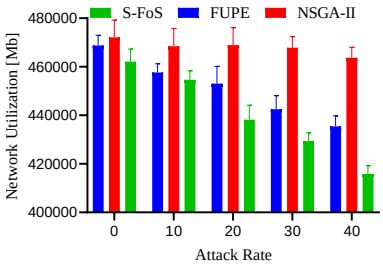
- *Scenario-1: Comparing methods based on various attack rates* The number of IoT devices and fog devices in this scenario is 60 and 20, respectively. The percentage of attacks ranges from 0% to 40%. This percentage represents the proportion of malicious flows in the workflow versus benign flows.
- *Scenario-2:Comparing methods based on various IoT devices* The number of fog devices and the percentage of attack rates in this scenario are 20 and 30%, respectively.
- *Scenario-3: Comparing methods based on various fog devices* The number of IoT devices and the percentage of attack rates in this scenario are 60 and 30%, respectively.

3.4.2 Experimental Results

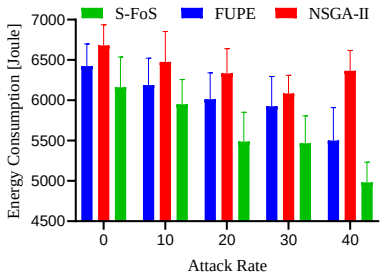
In this section, we show the implementation results for the presented approach as well as other approaches in the scenarios mentioned. First, we look at the results for the first Scenario, which considers the different attack rate. The



(a) Average response time.



(b) Network utilization.

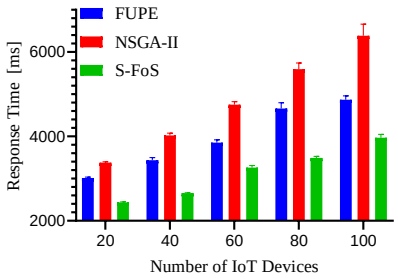


(c) Total Energy consumption.

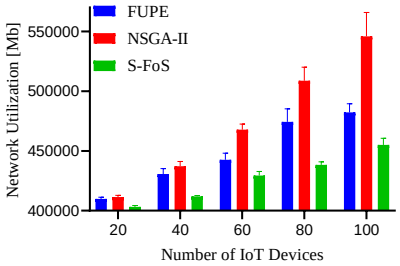
Figure 3.9: The outcomes for various attack rates.

average response time for the methods is depicted in Fig. 3.9a against various attack rates. It is obvious that increasing the attack rate notably decreases the response time for the S-FoS, but at a lower rate for the FUPE. On the other hand, NSGA-II remains constant. The reason for this is that the NSGA-II method lacks a mechanism for detecting the effects of attacks. It assigns all flows to resources, regardless of whether they are malicious or benign. As a result, it considers both malicious and benign flows, resulting in a response time that is roughly unchanged with a moderate fluctuation. FUPE can detect two forms of DDOS attacks: TCP and UDP. As in this scenario, the IoT devices can perform three types of attacks, FUPE exhibits a gradual fall. S-FoS can detect and block all threats, as well as malicious flows. As a result, it only assigns benign flows to resources, significantly reducing response time. S-FoS also reduces network utilization as it is illustrated in Fig. 3.9b. As fewer flows are executed in S-FoS, its declivity becomes more apparent. The reason for this experiment is because the S-FoS strategy assigns less flows to edge resources, resulting in fewer flows being sent via the network links between

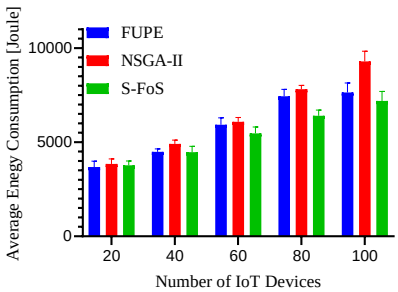
SDN switches and fog devices. As a result, it is gradually decreasing. In comparison to S-FoS, FUPE can detect fewer malicious flows and hence assigns more flows to fog devices, resulting in higher network utilization. NSGA-II assigns all flows to fog devices, resulting in a network utilization that is nearly same. Aside from these accomplishments, another significant advantage of S-FoS is that it reduces energy consumption that is indicated in 3.9c. S-FoS experiences the greatest decrease. In comparison to S-FoS, FUPE witnesses a continuous decrease with a lower rate. However, NSGA-II stays almost uniform with a slight fluctuations. Because S-FoS only assigns benign flows, it blocks malicious requests and schedules fewer flows. As a result, it consumes the least amount of energy.



(a) Average Response Time.



(b) Network Utilization.

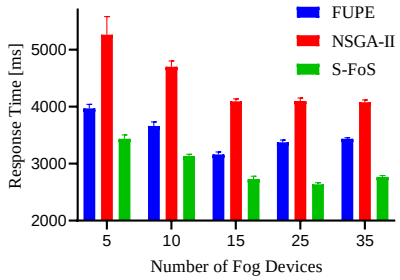


(c) Total Energy consumption.

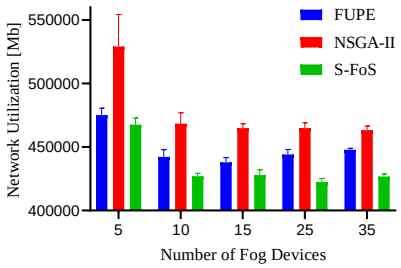
Figure 3.10: The outcomes for various fog devices.

Following the expression of the first scenario, we examine the results of the second scenario, which covers the various IoT devices. As illustrated in 3.10a and 3.10b, there is a clear upward trend in response time and network utilization across all approaches. However, this is a minor trend in S-FoS. By

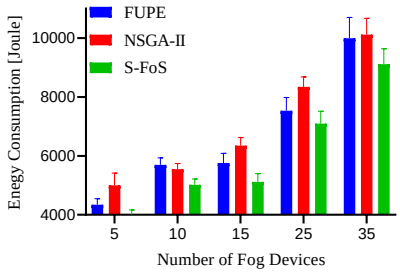
adjusting the number of IoT devices, these two metrics rise as the number of requests rises. Because S-FoS blocks all malicious requests, the increasing trend is occurring at a slower rate. When it comes to energy consumption, S-FoS outperforms the other two approaches. By adjusting the IoT devices, the resources should be able to execute more requests, increasing the number of CPUs that are busy. As a result, they are on the rise. As S-FoS blocks malicious requests, the number of CPUs that are busy decreases, and this trend in S-FoS is slower. The energy consumption of the approaches is depicted in 3.10c.



(a) Average Response Time.



(b) Network Utilization.



(c) Total Energy Consumption.

Figure 3.11: The outcomes for various fog devices.

The final scenario we look at is changing the number of fog devices. The purpose of this scenario is to demonstrate the benefits and drawbacks of increasing the number of fog devices. The response time and network utilization for each of the methods are depicted in 3.11a and 3.11b, respectively. By varying the number of fog devices, all methods experience a consistent decrease. The reason for this is that as the requests spread to more fog devices,

each resource receives fewer requests. S-Fos has the shortest response time because it applies reference points to its NSGA-III fitness function. The other two approaches send more requests to the fog devices, resulting in a longer response time and network usage. As seen in the graphs, increasing the number of fog devices reduces network utilization and response time until a certain point is reached. They almost hit a plateau after that. This steadiness is achieved with a modest fluctuation in the network utilization. This scenario also depicts the IoT network's energy consumption. In comparison with the other two approaches, S-FoS, as shown in 3.11c, has the lowest energy consumption. However, increasing the number of fog devices increases energy consumption for all approaches. The reason is that the consumption of energy is a linear function of CPU usage. According to Eq. (3.8), the idle CPU consumes energy as well. The busy CPU, on the other hand, consumes more energy. S-FoS consumes the least amount of energy because it puts away malicious requests. This scenario demonstrates how increasing the number of fog devices is a double-edged sword. On the one hand, it shortens the response time up to a certain point. After that, keeping the number of requests constant has no effect on response time, and it plateaus. It also has no effect on network utilization. However, there is a minor fluctuation in network utilization. On the other hand, after the specific point, we can see a significant increase in energy consumption due to connecting more switched on fog devices that are idle but still consume energy.

According to 3.9 to 3.10, it can be concluded that, after S-FoS, the FUPE algorithm outperforms the NSGA-II algorithm. The reason for this is that S-FoS prevents all three types of attacks that cause the resource management system to assign fewer requests. Furthermore, S-FoS makes use of the delay and load balancing objectives' parameters. S-FoS considers the flow CPU requirement, available amount of resources' computational capacity and bandwidth in the NSGA-III fitness function to optimize the performance of the scheduler. The CI for plots as indicators over the results are also indicated in the presented implementations. Because all of the methods use randomness features, the results have nearly the same tolerance.S-FoS, FUPE and NSGA-II have uneven load distribution among fog nodes due to the randomness feature of MOPSO and Genetic algorithms.

To wrap up this section, we'll go over the S-FoS accuracy evaluation. To that end, we classified 30% of the flows as benign requests. Because the NSGA-II lacks attack defense mechanisms, we only evaluate S-FoS and FUPE in terms of accuracy, precision, and recall in this evaluation.

$$TPR = (TP/ActualPositive) = TP/(TP + FN) \tag{3.10}$$

$$FNR = (FN/ActualPositive) = FN/(TP + FN) \tag{3.11}$$

$$TNR = (TN/ActualNegative) = TN/(TN + FP) \tag{3.12}$$

$$FPR = (FP/ActualNegative) = FP/(TN + FP) \tag{3.13}$$

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{3.14}$$

$$Precision = (TP)/(TP + FP) \tag{3.15}$$

$$Recall = (TP)/(TP + FN) \tag{3.16}$$

where TP denotes true positive, and FN denotes false negative, TN denotes true negatives, and FP denotes false positive, respectively. Based on Eq. (3.14), Eq. (3.15), and Eq. (3.16), *Accuracy*, *Precision*, and *Recall* of S-FoS are 0.971, 0.960, and 1.0, respectively. Besides, these metrics for FUPE are 0.865, 0.838, and 1.0, respectively. Tab. 3.6 summarizes the accuracy evaluation.

Table 3.6: Accuracy evaluation.

	TNR	**FNR**	**TPR**	**FPR**	**Recall**	**Precision**	**Accuracy**
S-FoS	0.903	0.00	1.0	0.0966	1.0	0.960	0.971
FUPE	0.55	0.00	1.0	0.45	1.0	0.838	0.865

3.5 Discussion

Network information distribution is a new paradigm for extending logically centralized controllers, and it is divided into *two* major categories [151]: *hierarchical structure* and *flat structure*. Because the global network state is only available in the global SDN controller, when the local SDN controller receives an inter-domain request, it first queries the network information from the global SDN controller. For the latter, all of the local SDN controllers can communicate directly with one another. They have a data store to keep track of the overall network state. They distribute parts of their local network state to the other local SDN controllers to update their data-store. The changes within their domains are shared with the other local SDN controllers to update the global network state. Because each local SDN controller only keeps information within its domain in the previous model, the local SDN controllers have

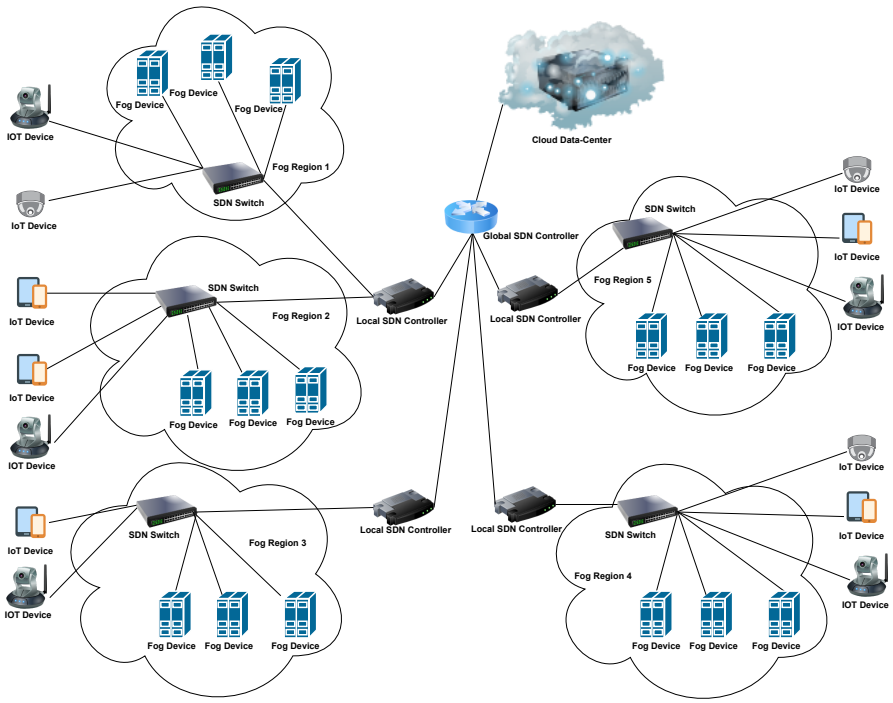


Figure 3.12: Hierarchical network information distribution architecture.

less overload in the later model. If they receive an inter-domain demand, they should first query the global SDN controller, usually located far from the other local SDN controllers.

Logically, single centralized controllers have limitations in terms of scalability and robustness. Thus, the S-FOS architecture can be extended to distribute SDN controller features. As a result, there is a global SDN controller (global cloud gateway) with a global view of the entire IoT-fog network in the extended architecture. The S-FOS architecture can benefit from the hierarchical model (vertical architecture). Controlling and managing the SDN switch(es) in their domains is the responsibility of the local SDN controllers. A domain of a local SDN controller is a collection of SDN switches directly connected to it (i.e., the local SDN controller) and fog devices attached to this SDN switch. For example, nodes in Fog Region 1 and Fog Region 2 are in the same domain because they are managed by the same local SDN controller in fig. 3.12. Finally, each local SDN controller distributes its local network state to the global SDN controller for it to construct a global network state. The global SDN controller then combines the local domain states provided by the local SDN controllers to generate the global network state. The global cloud gateway also serves as a coordinator for the local SDN controllers. The global cloud gateway monitors the overall state of all domains. Local SDN controllers, on the other hand, only save the states of their local domains [152].

In this work, we assume that SDN switches and SDN controllers, respectively, have the features of fog gateways and cloud gateways in the IoT-Fog networks. Cisco Kinetic is a commercial product that consists of two modules: the Edge and Fog Processing Module and the Data Control Module, which work together to securely connect fog devices and then extract, compute, and move data from IoT devices. In IoT-Fog networks, the former module is implemented in the fog gateway, while the latter is implemented in the cloud gateway. In the SDN-based IoT-Fog networks, Cloud/fog gateways are nodes with powerful computational capabilities that can implement SDN characteristics using Open vSwitch, and there are no restrictions for implementing that in a real-world environment. The scheduling component is also implementable in both cloud/fog gateways.

The fog gateway is an ideal location for an IoT scheduler. The reason for this is that the scheduler may make key decisions near the point of action (requests from IoT devices) and make the best use of fog device resources this way. On the other hand, in most SDN networks, the goal is to lower the SDN controller's load. Although we included the assigner in the SDN controller

in S-FOS for SDN-based IoT-Fog networks, we claim that by employing distributed SDN controllers, the centralized controller's load will be minimized, and each of the local SDN controllers will be able to manage the flows within a small fog region. A commercial example of a distributed SDN controller is OpenDaylight. An assigner in another node that can operate as a fog/cloud gateway is another option. Cisco Kinetic, as previously discussed, is a commercial example of data control and edge processing that operates on cloud/fog gateways.

For IoT forensics [153], the information kept by SDN controller can be utilised. The purpose of IoT security is to lower the risk of potential threats and assaults. IoT forensics, on the other hand, seeks to pinpoint the source of the assault. Despite the fact that security and forensics are two distinct professions with different tools and methodologies, there is a lot of overlap between them. As a result, IoT forensics policies in an IoT-fog network can take into account IoT security measures. By incorporating forensics features into security methodologies, IoT security and IoT forensics can be combined to increase security [154]. The local SDN controller can see and evaluate the behavior of the nodes attempting to disable the fog devices. For effective case reporting, the logger module's function is based on learning from experience to continuously carry out network node analysis in the local SDN controllers. The local SDN controller works with SDN switches to identify hostile IoT devices. For judicial investigations, the SDN controllers store the device IDs that are identified as the attacker.

3.6 Conclusions

This chapter proposed S-FoS, a joint secure and performance optimization workflow scheduler that considers security issues to protect scheduling services while effectively assigning user requests to the edge resources in SDN-based IoT-fog networks. S-FoS detects and blocks malicious IoT device requests by using fuzzy functions with TRW-CB, Rate limiting, and Entropy algorithm output as input parameters. Then it employs NSGA-III to consider both load balancing and delay. S-FoS countermeasures TCP/UDP DDOS, and port scanning attacks. The effectiveness of the presented approach is demonstrated by our evaluation using IoT-based scenarios. S-FoS outperforms both metaheuristic methods, NSGA-II and MOPSO, according to the results. When compared to the previous one, S-FoS improves response time by 31%, network utilization by 9%, and energy consumption by 16%. It improves response time

by 18%, network utilization by 4%, and energy consumption by 9% when compared to the latter.

Conclusions

This thesis proposes three resource scheduling algorithms for IoT-Fog networks: FPFTS, FUPE, and S-FOS.

FPFTS is a fog task scheduler in which we use fuzzy logic to improve the fitness function of a particle-swarm optimization (PSO) algorithm. It introduces a novel method for optimizing the task scheduling problem in fog computing for both delay-sensitive and delay-tolerant applications. The proposed approach's goal is to make the best use of fog resources in order to reduce network utilization and application loop delay. To that end, FPFTS takes into account both the computing characteristics of the fog nodes, such as CPU processing capacity, RAM size, and bandwidth, as well as the computing characteristics of the tasks, such as CPU need. Furthermore, it is intended for use with both delay-sensitive and delay-tolerant applications: FPFTS uses information about the class to which each application belongs to refine scheduling decisions in the event of fog-layer overloading. It employs fuzzy logic in PSO, taking into account the characteristics of resources and tasks, to solve fog task scheduling problems without becoming trapped in a local minimum and to make the best use of fog resources.

FUPE is a security-aware task scheduler algorithm designed for SDN-based IoT-fog networks that ensures a sufficient level of security. To deal with TCP DDoS attacks, this scheduler takes into account the dynamic behavior of distributed systems and uses two trust degrees obtained from Threshold Random Walk with Credit-Based Connection Rate-Limiting (TRW-CB) and Rate Limiting algorithms- i.e. one of the most popular source-based attack mitigation strategies available. With the use of multi-objective PSO and multi-criteria decision-making algorithms, our proposal combines security concerns with job scheduling. In IoT-fog networks, FUPE solves a multi-objective task scheduling problem to maximize security and efficiency of quality of services (QoS) such as latency. FUPE focuses on allocating applications' tasks across fog devices while taking into account fog device trustworthiness and IoT devices. As

a result, FUPE achieves a balance between efficiency and security.

S-FOS is the first attempt to implement a secure workflow scheduling approach for performance optimization in SDN-based IoT-Fog networks that takes into account TCP/UDP DDOS, as well as port scanning threats. It automatically shuts IoT devices that begin to participate in harmful activities to secure the IoT-fog scheduling services. In S-FOS, we assume that the attackers are IoT devices capable of assaulting fog devices that influence SDN switches and controllers. The results of the security mechanism are used by S-FOS to validate users' applications. Then, using one of the recently researched MOO solutions, the non-dominated sorting genetic algorithm, it selects fog devices that fulfill the load balancing and latency requirements of the customers' applications (NSGA-III). This algorithm is extensively employed in IoT-fog networks since it outperforms other MOO algorithms on big-scale difficulties with a large number of users. Furthermore, it minimizes the time it takes to discover the final solution by utilizing reference points for the next-generation population; second, it performs better for problems with more than two objective functions by using reference points for the next-generation population. S-FoS requests that NSGA-III consider load balancing and delay when optimizing performance.

Appendix A

Bibliography

[1] Jakob Mass, Chii Chang, and Satish Narayana Srirama. Edge process management: A case study on adaptive task scheduling in mobile iot. *Internet of Things*, 6:100051, 2019.

[2] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, 56:684–700, 2016.

[3] Aaqif Afzaal Abbasi, Almas Abbasi, Shahaboddin Shamshirband, Anthony Theodore Chronopoulos, Valerio Persico, and Antonio Pescapè. Software-defined cloud computing: A systematic review on latest trends and developments. *IEEE Access*, 7:93294–93314, 2019.

[4] Valerio Persico, Pietro Marchetta, Alessio Botta, and Antonio Pescapè. Measuring network throughput in the cloud: The case of amazon ec2. *Computer Networks*, 93:408–422, 2015.

[5] Fabio Palumbo, Giuseppe Aceto, Alessio Botta, Domenico Ciuonzo, Valerio Persico, and Antonio Pescape. Characterizing cloud-to-user latency as perceived by aws and azure users spread over the globe. In *2019 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[6] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6. IEEE, 2012.

[7] Giuseppe Aceto, Valerio Persico, and Antonio Pescapé. The role of information and communication technologies in healthcare: taxonomies,

perspectives, and challenges. *Journal of Network and Computer Applications*, 107:125–154, 2018.

[8] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[9] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019.

[10] Ehsan Pourjavad and Rene V Mayorga. A comparative study and measuring performance of manufacturing systems with mamdani fuzzy inference system. *Journal of Intelligent Manufacturing*, 30(3):1085–1097, 2019.

[11] Mohammad Shojafar, Saeed Javanmardi, Saeid Abolfazli, and Nicola Cordeschi. Fuge: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. *Cluster Computing*, 18(2):829–844, 2015.

[12] Hicham Ben Alla, Said Ben Alla, Abdellah Ezzati, and Ahmed Mouhsen. A novel architecture with dynamic queues based on fuzzy logic and particle swarm optimization algorithm for task scheduling in cloud computing. In *International Symposium on Ubiquitous Networking*, pages 205–217. Springer, 2016.

[13] Pejman Hosseinioun, Maryam Kheirabadi, Seyed Reza Kamel Tabbakh, and Reza Ghaemi. atask scheduling approaches in fog computing: a survey. *Transactions on Emerging Telecommunications Technologies*, page e3792, 2020.

[14] Luiz F Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F Rana, and Manish Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, 2017.

[15] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Latency-aware application module management for fog computing environments. *ACM Transactions on Internet Technology (TOIT)*, 19(1):1–21, 2018.

[16] Sukhpal Singh Gill, Peter Garraghan, and Rajkumar Buyya. Router: Fog enabled cloud based intelligent resource management approach for smart home iot devices. *Journal of Systems and Software*, 154:125–138, 2019.

[17] Deze Zeng, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016.

[18] Doan Hoang and Thanh Dat Dang. Fbrc: Optimization of task scheduling in fog-based region and cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 1109–1114. IEEE, 2017.

[19] Yan Sun, Fuhong Lin, and Haitao Xu. Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii. *Wireless Personal Communications*, 102(2):1369–1385, 2018.

[20] Salim Bitam, Sherali Zeadally, and Abdelhamid Mellouk. Fog computing job scheduling optimization based on bees swarm. *Enterprise Information Systems*, 12(4):373–397, 2018.

[21] Fatma M Talaat, Shereen H Ali, Ahmed I Saleh, and Hesham A Ali. Effective load balancing strategy (elbs) for real-time fog computing environment using fuzzy and probabilistic neural networks. *Journal of Network and Systems Management*, 27(4):883–929, 2019.

[22] Hina Rafique, Munam Ali Shah, Saif Ul Islam, Tahir Maqsood, Suleman Khan, and Carsten Maple. A novel bio-inspired hybrid algorithm (nbiha) for efficient resource management in fog computing. *IEEE Access*, 7:115760–115773, 2019.

[23] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of everything*, pages 103–130. Springer, 2018.

[24] Paola G Vinueza Naranjo, Zahra Pooranian, Mohammad Shojafar, Mauro Conti, and Rajkumar Buyya. Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments. *Journal of parallel and distributed computing*, 132:274–283, 2019.

[25] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of things*, pages 61–75. Elsevier, 2016.

[26] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing*, pages 1–42, 2019.

[27] Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.

[28] Bart Kosko. *Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence.* Number QA76. 76. E95 K86. 1992.

[29] Manitpal S Sidhu, Parimala Thulasiraman, and Ruppa K Thulasiram. A load-rebalance pso heuristic for task matching in heterogeneous computing systems. In *2013 IEEE Symposium on Swarm Intelligence (SIS)*, pages 180–187. IEEE, 2013.

[30] Mohammad Aazam, Sherali Zeadally, and Khaled A Harras. Offloading in fog computing for iot: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems*, 87:278–289, 2018.

[31] Yiqin Deng, Zhigang Chen, Xin Yao, Shahzad Hassan, and Jia Wu. Task scheduling for smart city applications based on multi-server mobile edge computing. *IEEE Access*, 7:14410–14421, 2019.

[32] Redowan Mahmud and Rajkumar Buyya. Modelling and simulation of fog and edge computing environments using ifogsim toolkit. *Fog and edge computing: Principles and paradigms*, pages 1–35, 2019.

[33] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[34] Pablo Cingolani and Jesús Alcalá-Fdez. jfuzzylogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. *International Journal of Computational Intelligence Systems*, 6(sup1):61–75, 2013.

[35] Saeed Javanmardi, Mohammad Shojafar, Shahdad Shariatmadari, Jemal H Abawajy, and Mukesh Singhal. Pgsw-os: a novel approach for resource management in a semantic web operating system based on a p2p grid architecture. *The Journal of Supercomputing*, 69(2):955–975, 2014.

[36] Saeed Javanmardi, Mohammad Shojafar, Shahdad Shariatmadari, and Sima S Ahrabi. Fr trust: a fuzzy reputation–based model for trust management in semantic p2p grids. *International Journal of Grid and Utility Computing*, 6(1):57–66, 2015.

[37] ShengYao Sun, WenBin Yao, BaoJun Qiao, Ming Zong, Xin He, and XiaoYong Li. Rrsd: A file replication method for ensuring data reliability and reducing storage consumption in a dynamic cloud-p2p environment. *Future Generation Computer Systems*, 100:844–858, 2019.

[38] Mohammad Shojafar, Zahra Pooranian, Paola G Vinueza Naranjo, and Enzo Baccarelli. Flaps: bandwidth and delay-efficient distributed data searching in fog-supported p2p content delivery networks. *The journal of supercomputing*, 73(12):5239–5260, 2017.

[39] Sun Park, ByungRea Cha, Kyungyong Chung, and JongWon Kim. Mobile iot device summarizer using p2p web search engine and inherent characteristic of contents. *Peer-to-Peer Networking and Applications*, pages 1–10, 2019.

[40] Opeyemi Osanaiye, Shuo Chen, Zheng Yan, Rongxing Lu, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. From cloud to fog computing: A review and a conceptual live vm migration framework. *IEEE Access*, 5:8284–8300, 2017.

[41] Sabina Jeschke, Christian Brecher, Tobias Meisen, Denis Özdemir, and Tim Eschert. Industrial internet of things and cyber manufacturing systems. In *Industrial internet of things*, pages 3–19. Springer, 2017.

[42] Mohammad Aazam, Sherali Zeadally, and Khaled A Harras. Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Transactions on Industrial Informatics*, 14(10):4674–4682, 2018.

[43] Houbing Song, Ravi Srinivasan, Tamim Sookoor, and Sabina Jeschke. *Smart cities: foundations, principles, and applications*. John Wiley & Sons, 2017.

[44] H Arasteh, V Hosseinnezhad, V Loia, A Tommasetti, O Troisi, M Shafie-Khah, and P Siano. Iot-based smart cities: a survey. In *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, pages 1–6. IEEE, 2016.

[45] Vasileios A Memos, Kostas E Psannis, Yutaka Ishibashi, Byung-Gyu Kim, and Brij B Gupta. An efficient algorithm for media-based surveillance system (eamsus) in iot smart city framework. *Future Generation Computer Systems*, 83:619–628, 2018.

[46] Jiang Zhu, Yonghui Song, Dingde Jiang, and Houbing Song. A new deep-q-learning-based transmission scheduling mechanism for the cognitive internet of things. *IEEE Internet of Things Journal*, 5(4):2375–2385, 2017.

[47] Yuxin Liu, Anfeng Liu, Tian Wang, Xiao Liu, and Neal N Xiong. An intelligent incentive mechanism for coverage of data collection in cognitive internet of things. *Future Generation Computer Systems*, 100:701–714, 2019.

[48] Arun Kumar Sangaiah, Arunkumar Thangavelu, Venkatesan Meenakshi Sundaram, et al. Cognitive computing for big data systems over iot. *Gewerbestrasse*, 11:6330, 2018.

[49] Hanan Elazhary. Internet of things (iot), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *Journal of Network and Computer Applications*, 128:105–140, 2019.

[50] G. Aceto, V. Persico, and A. PescapÃ©. A survey on information and communication technologies for industry 4.0: State-of-the-art, taxonomies, perspectives, and challenges. *IEEE Communications Surveys Tutorials*, 21(4):3467–3501, 2019.

[51] Ashkan Yousefpour, Genya Ishigaki, and Jason P Jue. Fog computing: Towards minimizing delay in the internet of things. In *2017 IEEE international conference on edge computing (EDGE)*, pages 17–24. IEEE, 2017.

[52] Jiafu Wan, Baotong Chen, Shiyong Wang, Min Xia, Di Li, and Chengliang Liu. Fog computing for energy-aware load balancing and

scheduling in smart factory. *IEEE Transactions on Industrial Informatics*, 14(10):4548–4556, 2018.

[53] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[54] Fadele Ayotunde Alaba, Mazliza Othman, Ibrahim Abaker Targio Hashem, and Faiz Alotaibi. Internet of things security: A survey. *Journal of Network and Computer Applications*, 88:10–28, 2017.

[55] Jianbing Ni, Kuan Zhang, Xiaodong Lin, and Xuemin Shen. Securing fog computing for internet of things applications: Challenges and solutions. *IEEE Communications Surveys & Tutorials*, 20(1):601–628, 2017.

[56] Alberto Dainotti, Antonio Pescapé, and Giorgio Ventre. A cascade architecture for dos attacks detection based on the wavelet transform. *Journal of Computer Security*, 17(6):945–968, 2009.

[57] Péter Megyesi, Alessio Botta, Giuseppe Aceto, Antonio Pescapé, and Sándor Molnár. Challenges and solution for measuring available bandwidth in software defined networks. *Computer Communications*, 99:48–61, 2017.

[58] Cheng Li, Zhengrui Qin, Ed Novak, and Qun Li. Securing sdn infrastructure of iot–fog networks from mitm attacks. *IEEE Internet of Things Journal*, 4(5):1156–1164, 2017.

[59] J Ramprasath and V Seethalakshmi. Secure access of resources in software-defined networks using dynamic access control list. *International Journal of Communication Systems*, 34(1):e4607, 2021.

[60] Ruchi Vishwakarma and Ankit Kumar Jain. A survey of ddos attacking techniques and defence mechanisms in the iot network. *Telecommunication systems*, 73(1):3–25, 2020.

[61] J Angela Jennifa Sujana, M Geethanjali, R Venitta Raj, and T Revathi. Trust model based scheduling of stochastic workflows in cloud and fog computing. In *Cloud Computing for Geospatial Big Data Analytics*, pages 29–54. Springer, 2019.

[62] Qiao Yan, F Richard Yu, Qingxiang Gong, and Jianqiang Li. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE communications surveys & tutorials*, 18(1):602–622, 2015.

[63] Narmeen Zakaria Bawany, Jawwad A Shamsi, and Khaled Salah. Ddos attack detection and mitigation using sdn: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, 42(2):425–441, 2017.

[64] Reza Mohammadi, Reza Javidan, and Mauro Conti. Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, 14(2):487–497, 2017.

[65] Amandeep Verma and Sakshi Kaushal. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Computing*, 62:1–19, 2017.

[66] Mohammed Abdullahi, Md Asri Ngadi, Salihu Idi Dishing, Barroon Isma'eel Ahmad, et al. An efficient symbiotic organisms search algorithm with chaotic optimization strategy for multi-objective task scheduling problems in cloud computing environment. *Journal of Network and Computer Applications*, 133:60–74, 2019.

[67] Mohammed Anis Benblidia, Bouziane Brik, Leila Merghem-Boulahia, and Moez Esseghir. Ranking fog nodes for tasks scheduling in fog-cloud environments: A fuzzy logic approach. In *2019 15th international wireless communications & mobile computing conference (IWCMC)*, pages 1451–1457. IEEE, 2019.

[68] Saeed Javanmardi, Mohammad Shojafar, Valerio Persico, and Antonio Pescapè. Fpfts: a joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for internet of things devices. *Software: Practice and Experience*, 51(12):2519–2539, 2021.

[69] Mikail Mohammed Salim, Shailendra Rathore, and Jong Hyuk Park. Distributed denial of service attacks and its defenses in iot: a survey. *The Journal of Supercomputing*, 76(7):5320–5363, 2020.

[70] PJ Beslin Pajila and E Golden Julie. Detection of ddos attack using sdn in iot: a survey. In *Intelligent Communication Technologies and Virtual Mobile Networks*, pages 438–452. Springer, 2019.

[71] Spilios Evmorfos, George Vlachodimitropoulos, Nikolaos Bakalos, and Erol Gelenbe. Neural network architectures for the detection of syn flood attacks in iot systems. In *Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, pages 1–4, 2020.

[72] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[73] Qiao Yan, Wenyao Huang, Xupeng Luo, Qingxiang Gong, and F Richard Yu. A multi-level ddos mitigation framework for the industrial internet of things. *IEEE Communications Magazine*, 56(2):30–36, 2018.

[74] Prashant Kumar, Meenakshi Tripathi, Ajay Nehra, Mauro Conti, and Chhagan Lal. Safety: Early detection and mitigation of tcp syn flood utilizing entropy in sdn. *IEEE Transactions on Network and Service Management*, 15(4):1545–1559, 2018.

[75] Reza Mohammadi, Mauro Conti, Chhagan Lal, and Satish C Kulhari. Syn-guard: An effective counter for syn flooding attack in software-defined networking. *International Journal of Communication Systems*, 32(17):e4061, 2019.

[76] Luying Zhou, Huaqun Guo, and Gelei Deng. A fog computing based approach to ddos mitigation in iiot systems. *Computers & Security*, 85:51–62, 2019.

[77] Wided Ben Daoud, Mohammad S Obaidat, Amel Meddeb-Makhlouf, Faouzi Zarai, and Kuei-Fang Hsiao. Tacrm: trust access control and resource management mechanism in fog computing. *Human-centric Computing and Information Sciences*, 9(1):1–18, 2019.

[78] Nitin Auluck, Omer Rana, Surya Nepal, Andrew Jones, and Anil Singh. Scheduling real time security aware tasks in fog networks. *IEEE Transactions on Services Computing*, 2019.

[79] Zhongjin Li, Jidong Ge, Hongji Yang, Liguo Huang, Haiyang Hu, Hao Hu, and Bin Luo. A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds. *Future Generation Computer Systems*, 65:140–152, 2016.

[80] Gaith Rjoub, Jamal Bentahar, and Omar Abdel Wahab. Bigtrustscheduling: Trust-aware big data task scheduling approach in cloud computing environments. *Future Generation Computer Systems*, 110:1079–1097, 2020.

[81] Sukhpal Singh Gill and Rajkumar Buyya. Secure: Self-protection approach in cloud resource management. *IEEE Cloud Computing*, 5(1):60–72, 2018.

[82] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42, 2017.

[83] Fatin Hamadah Rahman, Thien-Wan Au, SH Shah Newaz, Wida Susanty Suhaili, and Gyu Myoung Lee. Find my trustworthy fogs: A fuzzy-based trust evaluation framework. *Future Generation Computer Systems*, 109:562–572, 2020.

[84] Maurizio D'Arienzo, Antonio Pescape, and Giorgio Ventre. Dynamic service management in heterogeneous networks. *Journal of Network and Systems Management*, 12(3):349–370, 2004.

[85] Ammar Muthanna, Abdelhamied A Ateya, Abdukodir Khakimov, Irina Gudkova, Abdelrahman Abuarqoub, Konstantin Samouylov, and Andrey Koucheryavy. Secure and reliable iot networks using fog computing with software-defined networking and blockchain. *Journal of Sensor and Actuator Networks*, 8(1):15, 2019.

[86] Ivan Farris, Tarik Taleb, Yacine Khettab, and Jaeseung Song. A survey on emerging sdn and nfv security mechanisms for iot systems. *IEEE Communications Surveys & Tutorials*, 21(1):812–837, 2018.

[87] Celyn Birkinshaw, Elpida Rouka, and Vassilios G Vassilakis. Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks. *Journal of Network and Computer Applications*, 136:71–85, 2019.

[88] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *International workshop on recent advances in intrusion detection*, pages 161–180. Springer, 2011.

[89] Sergei Dotcenko, Andrei Vladyko, and Ivan Letenko. A fuzzy logic-based information security management for software-defined networks. In *16th International Conference on Advanced Communication Technology*, pages 167–171. IEEE, 2014.

[90] Jürgen Branke, Jurgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowiński. *Multiobjective optimization: Interactive and evolutionary approaches*, volume 5252. Springer Science & Business Media, 2008.

[91] Nikola K Kasabov. Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. *Fuzzy sets and Systems*, 82(2):135–149, 1996.

[92] Shuang Feng and CL Philip Chen. Fuzzy broad learning system: A novel neuro-fuzzy model for regression and classification. *IEEE transactions on cybernetics*, 50(2):414–424, 2018.

[93] Hicham Ben Alla, Said Ben Alla, Abdellah Touhafi, and Abdellah Ezzati. A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment. *Cluster Computing*, 21(4):1797–1820, 2018.

[94] Simar Preet Singh, Anand Nayyar, Harpreet Kaur, and Ashu Singla. Dynamic task scheduling using balanced vm allocation policy for fog computing platforms. *Scalable Computing: Practice and Experience*, 20(2):433–456, 2019.

[95] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[96] Maurice Clerc. *Particle swarm optimization*, volume 93. John Wiley & Sons, 2010.

[97] Rongbin Xu, Yeguo Wang, Yongliang Cheng, Yuanwei Zhu, Ying Xie, Abubakar Sadiq Sani, and Dong Yuan. Improved particle swarm optimization based workflow scheduling in cloud-fog environment. In *Inter-*

national Conference on Business Process Management, pages 337–347. Springer, 2018.

[98] Vangelis Angelakis, Ioannis Avgouleas, Nikolaos Pappas, Emma Fitzgerald, and Di Yuan. Allocation of heterogeneous resources of an iot device to flexible services. *IEEE Internet of Things Journal*, 3(5):691–700, 2016.

[99] https://www.mathworks.com/products/matlab.html. Matlab.

[100] http://www2.imse cnm.csic.es/Xfuzzy/. Xfuzzy.

[101] Lelio Campanile, Marco Gribaudo, Mauro Iacono, Fiammetta Marulli, and Michele Mastroianni. Computer network simulation with ns-3: A systematic literature review. *Electronics*, 9(2):272, 2020.

[102] Andras Varga. A practical introduction to the omnet++ simulation framework. In *Recent Advances in Network Simulation*, pages 3–51. Springer, 2019.

[103] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6. IEEE, 2014.

[104] Saptarshi Ghosh, SA Busari, Tasos Dagiuklas, Muddesar Iqbal, R Mumtaz, J Gonzalez, Stavros Stavrou, and Loizos Kanaris. Sdn-sim: integrating a system-level simulator with a software defined network. *IEEE Communications Standards Magazine*, 4(1):18–25, 2020.

[105] Juan Wang and Di Li. Adaptive computing optimization in software-defined network-based industrial internet of things with fog computing. *Sensors*, 18(8):2509, 2018.

[106] Ke Xiao, Kai Liu, Xincao Xu, Yi Zhou, and Liang Feng. Efficient fog-assisted heterogeneous data services in software defined vanets. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2019.

[107] Zhihan Lv and Wenqun Xiu. Interaction of edge-cloud computing based on sdn and nfv for next generation iot. *IEEE Internet of Things Journal*, 7(7):5706–5712, 2019.

[108] Quamar Niyaz, Weiqing Sun, and Ahmad Y Javaid. A deep learning based ddos detection system in software-defined networking (sdn). *arXiv preprint arXiv:1611.07400*, 2016.

[109] Xiaoyu Duan and Xianbin Wang. Fast authentication in 5g hetnet through sdn enabled weighted secure-context-information transfer. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[110] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 2015.

[111] Hsin-Hung Cho, Chin-Feng Lai, Timothy K Shih, and Han-Chieh Chao. Integration of sdr and sdn for 5g. *Ieee Access*, 2:1196–1204, 2014.

[112] Leonid M Kupershtein, Tatiana B Martyniuk, Olesia P Voitovych, Bohdan V Kulchytskyi, Andrii V Kozhemiako, Daniel Sawicki, and Mashat Kalimoldayev. Ddos-attack detection using artificial neural networks in matlab. In *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019*, volume 11176, page 111761S. International Society for Optics and Photonics, 2019.

[113] Sarwan Ali, Maria Khalid Alvi, Safi Faizullah, Muhammad Asad Khan, Abdullah Alshanqiti, and Imdadullah Khan. Detecting ddos attack on sdn due to vulnerabilities in openflow. In *2019 International Conference on Advances in the Emerging Computing Technologies (AECT)*, pages 1–6. IEEE, 2020.

[114] Huseyin Polat, Onur Polat, and Aydin Cetin. Detecting ddos attacks in software-defined networks through feature selection methods and machine learning models. *Sustainability*, 12(3):1035, 2020.

[115] Gaganjot Kaur and Prinima Gupta. Hybrid approach for detecting ddos attacks in software defined networks. In *2019 Twelfth International Conference on Contemporary Computing (IC3)*, pages 1–6. IEEE, 2019.

[116] Abdul Fadlil, Imam Riadi, and Sukma Aji. Ddos attacks classification using numeric attribute-based gaussian naive bayes. *International*

Journal of Advanced Computer Science and Applications (IJACSA), 8(8):42–50, 2017.

[117] Jisa David and Ciza Thomas. Efficient ddos flood attack detection using dynamic thresholding on flow-based network traffic. *Computers & Security*, 82:284–295, 2019.

[118] Huangxin Wang, Quan Jia, Dan Fleck, Walter Powell, Fei Li, and Angelos Stavrou. A moving target ddos defense mechanism. *Computer Communications*, 46:10–21, 2014.

[119] Avijit Hazra. Using the confidence interval confidently. *Journal of thoracic disease*, 9(10):4125, 2017.

[120] Mauro Conti, Chhagan Lal, Reza Mohammadi, and Umashankar Rawat. Lightweight solutions to counter ddos attacks in software defined networking. *Wireless Networks*, 25(5):2751–2768, 2019.

[121] Fanglu Guo and Tzi-cker Chiueh. Sequence number-based mac address spoof detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 309–329. Springer, 2005.

[122] D Dasgupta, J Gomez, F Gonzalez, M Kaniganti, K Yallapu, and R Yarramsetti. Mmds: multilevel monitoring and detection system. In *Proceedings of the 15th Annual Computer Security Incident Handling Conference*, pages 22–27, 2003.

[123] Junlong Zhou, Jin Sun, Peijin Cong, Zhe Liu, Xiumin Zhou, Tongquan Wei, and Shiyan Hu. Security-critical energy-aware task scheduling for heterogeneous real-time mpsocs in iot. *IEEE Transactions on Services Computing*, 2019.

[124] Saeed Javanmardi, Mohammad Shojafar, Reza Mohammadi, Amin Nazari, Valerio Persico, and Antonio Pescapè. Fupe: A security driven task scheduling approach for sdn-based iot–fog networks. *Journal of Information Security and Applications*, 60:102853, 2021.

[125] Shivi Sharma and Hemraj Saini. Fog assisted task allocation and secure deduplication using 2fbo2 and mowo in cluster-based industrial iot (iiot). *Computer Communications*, 152:187–199, 2020.

[126] Vincenzo De Maio and Dragi Kimovski. Multi-objective scheduling of extreme data scientific workflows in fog. *Future Generation Computer Systems*, 106:171–184, 2020.

[127] Ismail M Ali, Karam M Sallam, Nour Moustafa, Ripon Chakraborty, Michael J Ryan, and Kim-Kwang Raymond Choo. An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems. *IEEE Transactions on Cloud Computing*, 2020.

[128] Georgios L Stavrinides and Helen D Karatza. A hybrid approach to scheduling real-time iot workflows in fog and cloud environments. *Multimedia Tools and Applications*, 78(17):24639–24655, 2019.

[129] Wiem Mkaouer, Marouane Kessentini, Adnan Shaout, Patrice Koligheu, Slim Bechikh, Kalyanmoy Deb, and Ali Ouni. Many-objective software remodularization using nsga-iii. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):1–45, 2015.

[130] Yuan Yuan, Hua Xu, and Bo Wang. An improved nsga-iii procedure for evolutionary many-objective optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 661–668, 2014.

[131] Elnaz Parvizi and Mohammad Hossein Rezvani. Utilization-aware energy-efficient virtual machine placement in cloud networks using nsga-iii meta-heuristic approach. *Cluster Computing*, pages 1–23, 2020.

[132] Khaled Alwasel, Devki Nandan Jha, Fawzy Habeeb, Umit Demirbaga, Omer Rana, Thar Baker, Scharam Dustdar, Massimo Villari, Philip James, Ellis Solaiman, et al. Iotsim-osmosis: A framework for modeling and simulating iot applications over an edge-cloud continuum. *Journal of Systems Architecture*, 116:101956, 2021.

[133] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[134] Jiewen Mao, Weijun Deng, and Fuke Shen. Ddos flooding attack detection based on joint-entropy with multiple traffic features. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference*

On Big Data Science And Engineering (TrustCom/BigDataSE), pages 237–243. IEEE, 2018.

[135] Ayyoob Hamza, Hassan Habibi Gharakheili, and Vijay Sivaraman. Combining mud policies with sdn for iot intrusion detection. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 1–7, 2018.

[136] Raja Majid Ali Ujjan, Zeeshan Pervez, Keshav Dahal, Ali Kashif Bashir, Rao Mumtaz, and J González. Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn. *Future Generation Computer Systems*, 111:763–779, 2020.

[137] Jitendra Patil, Vrinda Tokekar, Alpana Rajan, and Anil Rawat. Port scanning based model to detect malicious tcp traffic and mitigate its impact in sdn. In *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, pages 365–370. IEEE, 2021.

[138] Fengxiao Tang, Yuichi Kawamoto, Nei Kato, Kazuto Yano, and Yoshinori Suzuki. Probe delay based adaptive port scanning for iot devices with private ip address behind nat. *IEEE Network*, 34(2):195–201, 2019.

[139] Saroja Subbaraj, Revathi Thiyagarajan, and Madavan Rengaraj. A smart fog computing based real-time secure resource allocation and scheduling strategy using multi-objective crow search algorithm. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2021.

[140] Hamza Baniata, Ahmad Anaqreh, and Attila Kertesz. Pf-bts: A privacy-aware fog-enhanced blockchain-assisted task scheduling. *Information Processing & Management*, 58(1):102393.

[141] Samaresh Bera, Sudip Misra, and Athanasios V Vasilakos. Software-defined networking for internet of things: A survey. *IEEE Internet of Things Journal*, 4(6):1994–2008, 2017.

[142] Ola Salman, Imad Elhajj, Ali Chehab, and Ayman Kayssi. Iot survey: An sdn and fog computing perspective. *Computer Networks*, 143:221–246, 2018.

[143] Laizhong Cui, F Richard Yu, and Qiao Yan. When big data meets software-defined networking: Sdn for big data and big data for sdn. *IEEE network*, 30(1):58–65, 2016.

[144] Claude E Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.

[145] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, and Prem Kumar Singh. Feature selection of denial-of-service attacks using entropy and granular computing. *Arabian Journal for Science and Engineering*, 43(2):499–508, 2018.

[146] Nguyen Trung Hieu, Mario Di Francesco, and Antti Ylä Jääski. A virtual machine placement algorithm for balanced resource utilization in cloud data centers. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 474–481. IEEE, 2014.

[147] Guangshun Li, Jiping Wang, Junhua Wu, and Jianrong Song. Data processing delay optimization in mobile edge computing. *Wireless Communications and Mobile Computing*, 2018, 2018.

[148] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[149] Maolin Tang and Shenchen Pan. A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers. *Neural processing letters*, 41(2):211–221, 2015.

[150] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH computer architecture news*, 35(2):13–23, 2007.

[151] Yustus Eko Oktian, SangGon Lee, HoonJae Lee, and JunHuy Lam. Distributed sdn controller system: A survey on design choice. *computer networks*, 121:100–111, 2017.

[152] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354, 2018.

[153] Hany F Atlam, Ezz El-Din Hemdan, Ahmed Alenezi, Madini O Alassafi, and Gary B Wills. Internet of things forensics: A review. *Internet of Things*, page 100220, 2020.

[154] Maria Stoyanova, Yannis Nikoloudakis, Spyridon Panagiotakis, Evangelos Pallis, and Evangelos K Markakis. A survey on the internet of things (iot) forensics: Challenges, approaches and open issues. *IEEE Communications Surveys & Tutorials*, 2020.