Computers, Materials & Continua
DOI: 10.32604/cmc.2023.035602
Article

Tech Science Press

Latency-Aware Dynamic Second Offloading Service in SDN-Based Fog Architecture

Samah Ibrahim AlShathri, Dina S. M. Hassan* and Samia Allaoua Chelloug

Information Technology Department, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh, 84428, Saudi Arabia
*Corresponding Author: Dina S. M. Hassan. Email: DSHassan@pnu.edu.sa
Received: 27 August 2022; Accepted: 14 December 2022

Abstract: Task offloading is a key strategy in Fog Computing (FC). The definition of resource-constrained devices no longer applies to sensors and Internet of Things (IoT) embedded system devices alone. Smart and mobile units can also be viewed as resource-constrained devices if the power, cloud applications, and data cloud are included in the set of required resources. In a cloud-fog-based architecture, a task instance running on an end device may need to be offloaded to a fog node to complete its execution. However, in a busy network, a second offloading decision is required when the fog node becomes overloaded. The possibility of offloading a task, for the second time, to a fog or a cloud node depends to a great extent on task importance, latency constraints, and required resources. This paper presents a dynamic service that determines which tasks can endure a second offloading. The task type, latency constraints, and amount of required resources are used to select the offloading destination node. This study proposes three heuristic offloading algorithms. Each algorithm targets a specific task type. An overloaded fog node can only issue one offloading request to execute one of these algorithms according to the task offloading priority. Offloading requests are sent to a Software Defined Networking (SDN) controller. The fog node and controller determine the number of offloaded tasks. Simulation results show that the average time required to select offloading nodes was improved by 33% when compared to the dynamic fog-to-fog offloading algorithm. The distribution of workload converges to a uniform distribution when offloading latency-sensitive non-urgent tasks. The lowest offloading priority is assigned to latency-sensitive tasks with hard deadlines. At least 70% of these tasks are offloaded to fog nodes that are one to three hops away from the overloaded node.

Keywords: Fog computing; offloading algorithm; latency-aware; software defined networking; SDN

1 Introduction

Traffic generated by smart and Internet of Things (IoT) devices is rapidly dominating the Internet. In 2021, hundreds of Zettabytes were processed by cloud data centers [1–3], and it is expected that this number will double in the coming years. However, approximately 10% of these data are critical [2,3]. Processing latency-sensitive tasks on remote cloud servers without violating their deadlines is a major challenge because all requests, regardless of their type, compete for Internet infrastructure. Furthermore, the vast increase in the number of requests due to unforeseen social, accidental, or environmental causes imposes a greater burden on network resources from a general perspective and on cloud data centers from a focused perspective. Introducing the paradigm of Fog Computing FC in 2012 [4] was an attempt to ensure that data are processed in the most efficient location by local fog servers to minimize latency, provide efficient Quality of Service QoS management, allow scalability, accept heterogeneity, and promote wireless access. These benefits have inspired the integration of FC with other network architectures, such as sensor networks [5–7], IoT [8–11], Internet of Vehicles (IoV) [12–15], blockchain [16–21], and big data analysis [22–25]. Further integration with Software Defined Networking (SDN) [26] enhances performance. SDN applies flow rules to control traffic forwarding according to its features and type. Several studies have addressed the integration of SDN-based networks with FC [27–31]. The results showed that the SDN architecture allowed fog networks to achieve efficient network management and promoted scalability while maintaining cost efficiency [32,33]. These benefits originate from the characteristics imposed by the architecture of SDN networks, which are based on centralized orchestration and traffic programmability. These properties eventually lead to agility and flexibility in network services performance.

Tasks that cannot be fully executed at end devices are offloaded to the fog nodes. Occasionally, fog nodes may become overloaded with a large number of requests. Traffic update requests sent to local fog nodes during traffic congestions or video streaming requests sent during key events are examples of fog node overload scenarios. Furthermore, the continuous growth in the number of users of cloud applications and rapid expansion in the deployment of IoT technologies may overwhelm the resources of fog nodes. Adding more fog nodes to reduce the workload is a static solution that cannot adapt to rapid and unexpected changes in the rate of generated requests. This may also be unjustified, especially if overloading is not the dominant state in all fog nodes. A second offloading was introduced as a dynamic solution that allowed fog nodes to exit the overloading state by redirecting a portion of the workload to other fog nodes. To propose an efficient offloading service, four main questions should be answered: How is the offloading node chosen [34–47], how is the number of offloading nodes decided [37], what tasks are eligible for offloading [35] and how is the number of offloaded tasks determined [38]. The proposed dynamic offloading service attempts to address these questions. The service is implemented in an SDN-based FC network. In a distributed architecture (e.g., non-SDN-based), the overloaded node is usually responsible for sending offloading requests to other fog nodes, collecting replies, and executing the offloading algorithm to decide which node should be selected to offload tasks to. This policy overburdens the overloaded fog node and incurs large communication overhead. Hence, the system throughput and average response time may degrade significantly. In an SDN-based network, the controller is responsible for selecting an offloading node. It benefits from the global network view built based on the collected information sent by fog nodes and network infrastructure components.

In this paper, both fog nodes and SDN controllers participate in determining the offloading nodes and selecting the offloaded task set. To the best of our knowledge, no previous research in the field of offloading algorithms has considered the problem of dynamically determining the type and amount of offloaded tasks in an overloaded fog node and employed this information to select a

number of offloading nodes using an SDN-based logically centralized architecture such that resource requirements and latency constraints of tasks are not violated and the time required to select offloading nodes is minimized. The major contributions of this study are as follows.

- A dynamic latency-aware second offloading service exploits the SDN approach to determine offloading destinations and offloaded tasks by executing one of the following heuristic algorithms:
  - o A greedy fog-to-cloud offloading algorithm selects task instances based on the value of the resources required. Tasks without latency constraints or latency-tolerant tasks are candidates for this type of offloading.
  - o A greedy fog-to-fog offloading algorithm for latency-sensitive non-urgent tasks. It aims to offload tasks to fog nodes with the lowest resource utilization without violating latency constraints. The algorithm is executed if fog-to-cloud offloading cannot be applied.
  - o The third proposed algorithm offloads latency-sensitive hard deadline tasks. A service is activated if this task type is the only one that exists. The algorithm selects the offloading fog nodes that achieve the lowest network latencies.
- A policy for determining the size of the task set to be offloaded is also presented.
- A performance evaluation study was conducted to assess the effectiveness of the proposed algorithm in reducing the time required to select offloading nodes, achieve a fair distribution of workload and guarantee the execution of hard deadline tasks.

The rest of the paper is organized as follows. Section 2 presents the motivation behind the proposed service. Section 3 presents a review of the related literature. Section 4 proposes a novel dynamic second offloading service based on the task type at the overloaded fog node. The performance evaluation is discussed in Section 5. Section 6 concludes the paper and presents future work.

2 Motivations

Innovations in smart city solutions, such as intelligent utilities, buildings, and healthcare management systems, are evolving rapidly. As the complexity of intelligent management systems increases, tasks executed by those systems vary in their types and importance and have different latency constraints. One example is the intelligent transportation system. The system provides several services, such as enhancing public safety, managing traffic flows, enabling vehicle-to-everything communication, controlling bus pick up times, collecting tolls electronically, performing fleet control, and managing emissions. Owing to the perceived benefits of SDN and FC, instead of sending all tasks to cloud nodes, integrating the SDN-based FC paradigm with cloud architecture can improve throughput, reduce delays and enhance the user Quality of Experience (QoE). However, owing to the diversity of tasks, a task queue in a fog node may have tasks belonging to several services. Therefore, these tasks have different resource requirements and latency constraints. When applying the proposed dynamic offloading service to the previous example. If a fog node becomes overloaded, latency-tolerant tasks, such as emission monitoring tasks, will be offloaded before latency-sensitive tasks, such as pedestrian detection tasks. Similarly, the proposed service can be applied to Cloud-edge-based Automated Driving Platforms (ADP) and Advanced Driver Assistance Systems (ADAS). Furthermore, in some systems, the same application may create task instances with different latency constraints depending on the end device that created the task. In cloud-based Virtual Reality (VR), users of this type of VR services engage in a simulated environment rendered by the cloud and then stream to the user's headset. In [48], the authors discussed the effects of adding variable delays to certain responses to user actions in a VR environment. The experiment showed that small delays added

to display updates in response to head movements resulted in more discomfort than relatively large delays added to display updates in response to joystick movements. Although both events activated the display update task, the delay that each task could endure differed. This wide variation in task requirements creates the need for offloading services that can efficiently select the most appropriate tasks for offloading and minimize the time required to select offloading nodes.

3 Related Work

The task offloading technique transfers task execution from one node to another. Offloading algorithms can be categorized as first or second offloading algorithms. The lack of required resources at end devices is the main reason that thrusts the first offloading. The first offloading algorithm transfers a task from the node at which it is created to another node, where the execution of the task resumes as in [34–37], [49,50]. However, a task may be offloaded once more using a second offloading algorithm for several reasons, such as adapting to node mobility [37], achieving load balancing [34,38], avoiding node overloading [44], and achieving QoS optimization [34,35]. Offloading may occur from the end device to the cloud server. However, device-to-cloud offloading increases latency and may result in missed deadlines, which may have serious consequences for hard-deadline, real-time tasks. The FC paradigm improves task response times by offloading them to local fog servers. As the number of offloaded tasks increases, fog nodes may become overloaded, and the overall performance may degrade sharply. Fog-to-fog offloading allows overloaded fog nodes to migrate tasks to other fog nodes that satisfy the requirements. Recently, there has been an evident increase in the volume of research that proposed task-offloading algorithms in FC.

In [34], IoT devices sent task offloading requests to fog nodes. The fog layer consists of a master fog node that collects offloading requests and performance data from the remaining fog nodes. The master fog node schedules offloaded tasks to fog nodes based on the collected information. The offloading algorithm was developed using an Ant Colony Optimization (ACO) probabilistic technique. The network architecture permits device-to-fog and fog-to-cloud offloading. Ordinary fog nodes do not enter an overloading state because the scheduling service is performed by the master node. Hence, fog-to-fog offloading is not required. Performance evaluation showed that ACO has a better response time than Round Robin (RR) algorithm and Particle Swarm Optimization (PSO) algorithm [34]. However, this model lacked scalability. As each task created in the IoT layer issues an offloading request, the master fog node, which is a single point of failure, may become a bottleneck as the number of requests increases.

In [35], the authors proposed a QoS optimization approach that triggers task computation offloading from end devices to fog nodes if better QoS can be achieved. The objective is to determine the optimal allocation of energy for real-time applications without missing task deadlines. The end device consists of sensors, a MultiProcessor System-on-Chip (MPSoC), and a low-power transceiver. MPSoC executes real-time applications and performs a local scheduling service that arranges the execution of applications while maintaining optimal energy allocation in two stages. In the first stage, the available energy of the end device is optimally allocated to the real-time task instances. The solution obtained in the first stage is the input for the second stage. In the second stage, the energy allocated to each task is optimally partitioned among the end-device components: the sensors, MPSoC, and transceiver. The output of this stage was sent as the input for the adaptive computation offloading algorithm executed at the fog node. For each locally running real-time task, if computation offloading achieves a better QoS when executed at the fog server, the task is offloaded. Otherwise, it is executed in the end device. If computational offloading occurs, the energy allocation algorithms of the device and

components are recalled to be executed at the end device to reallocate the available energy. However, no offloading service was introduced, assuming that the end device and fog server entered an overload state. Furthermore, the proposed solution does not consider how global optimal energy allocation can be achieved in a network architecture with multiple end devices and several fog servers.

The idea presented in [36] considered a network scenario that includes IoT devices connected through multi-hop connections to fog nodes. The network scenario also incorporated an SDN controller that relies on southbound Application Programming Interfaces (APIs) to control the dynamic task offloading from IoT devices to fog nodes. Moreover, the authors of [36] proposed a bi-objective mathematical model for optimizing the delay and energy while satisfying the energy, flow conservation, bandwidth, and capacity requirements of the access points. The proposed optimization model is non-linear and has been transformed into an equivalent Integer Linear Program (ILP). The proposed solution, called Detour, represents an approximate greedy algorithm that decides for each IoT task whether to offload it or to process it locally according to where improvement in delay and energy consumption is determined. Detour assumes that each device has one task to execute and that all required data units and applications are available locally. However, this may not always be the case in FC. In several scenarios, device-to-fog offloading is required because IoT devices do not have the required data, processing power, or applications. Furthermore, the SDN controller is required to execute Detour for each newly created task. As the number of tasks increases, this may overload the SDN. In addition, determining how the offloading service is performed for concurrent tasks has not been considered.

In [37], the distributed control plan consists of multiple SDN controllers that communicate with each other using the east-west API. Each SDN controlled a set of fog nodes. The Mobile Node (MN) executes a delay-sensitive service to locally compute the feasibility of satisfying a task within the time limit. This was achieved by executing the local scheduling part of the hybrid scheduling algorithm proposed in [37]. If the service cannot be fully executed locally, the MN sends an offloading request along with its QoE requirement to the corresponding fog node. The fog node relays a request to its corresponding SDN controller. The controller begins the execution of the fog-scheduling part of the hybrid algorithm. The network status information was collected, and the mobile trajectory of the MN was predicted. The algorithm then uses Yen's k shortest path algorithm to determine the shortest path between the source and destination fog nodes. The data file related to the application was partitioned into file blocks. The SDN controller tests the feasibility of stratifying requests by assigning file blocks to fog nodes along that path. If the deadline cannot be met, the iteration continues until the optimal path that satisfies the request is determined. However, partitioning files into blocks that maintain the control flow of the program is a significant issue, and its cost should be considered.

The model introduced in [38] assumes that a fog node can offload a task to a neighboring fog node based on the decision of the SDN controller, which may allow a certain number of tasks to be offloaded while balancing the load between the deployed fog nodes. The solution adopted in [38] relies on Markov Decision Process (MDP) to consider the dynamic behavior of fog nodes as well as the uncertainties in terms of task demands. More specifically, the algorithm presented in [38] is a Quality (Q)-learning algorithm, which has been proposed for solving MDPs that consider dynamically changing task demands. The idea of the proposed algorithm is based on an agent that observes the environment, makes a decision, observes the new state, and then calculates the reward, which is a function of the utility, delay, and overload probability. However, a time complexity analysis of the algorithm was not provided. Although Q-learning algorithms achieve convergence in polynomial time, if the proposed algorithm cannot guarantee convergence time, it cannot be used to offload latency-sensitive tasks.

The authors in [39] proposed a framework for an IoT network that employs FC by adding a layer of fog nodes controlled and managed by a logically decentralized SDN network. Network architecture adopts a three-tier fog SDN-based architecture. However, it is integrated with blockchain technology to incorporate its benefits. The algorithm starts by authenticating an IoT node using a cloud server. The cloud server then starts the address detection process. In this process, ISP refers to an SDN controller that determines the location of the device. The SDN controller builds a routing table with all possible routes between the device and the IoT cloud. Once the best route is determined, data migration between the IoT cloud and the assigned fog node commences. The fog node is also responsible for synchronizing data with the cloud server. The SDN controller, based on a certain set of parameters [39], decides whether to allow offloaded data from fog nodes to be processed at OpenFlow (OF)-switches that have certain computation resources or to be offloaded to a cloud server. However, the allocation policy from fog to OF-switches was not motioned. Furthermore, fog-to-fog offloading was not considered.

In [40], the authors proposed an offloading algorithm for Energy IoT (EIoT) devices in smart grids. The algorithm aims to achieve energy efficiency without violating the delay constraints. EIoT devices offload tasks to fog nodes if they cannot be executed locally because of limited computation or energy resources. Both EIoT devices and fog nodes create preference lists. The EIoT preference list ranks fog nodes, which can be associated with, in descending order, according to the weighted difference between energy efficiency and delay. Similarly, each fog node applies the same procedure to EIoT devices. Fog nodes accept association requests from EIoT devices if the total number of concurrently executed tasks at the fog node does not exceed the maximum limit. Otherwise, the fog nodes use their preference lists to select the more preferable devices without exceeding the maximum number of concurrent tasks. The remaining devices that failed to associate with fog nodes started a new iteration. The process continues until each device is associated with a fog node or until there are no more fog nodes to which to assign devices. In the latter case, fog-to-fog offloading may solve the problem of unassigned requests and balance the network load. However, fog-to-fog offloading was not considered in the proposed service. Moreover, the algorithm only considers transmission and processing delays when computing the delay model.

The authors in [41] introduced an offloading algorithm based on Evolutionary Genetic Algorithm (EGA), which aims to minimize energy consumption in vehicular networks. Fog nodes execute the offloading algorithm to decide whether to offload requests received from vehicular applications to cloud servers or to execute them locally at fog nodes. The algorithm starts by generating several offloading solutions. In each solution, requests were randomly allocated to fog or cloud nodes. Solutions are then evaluated to exclude infeasible solutions according to a fitness function that results in accepting a solution if it does not violate latency constraints and leads to a total energy consumption that is close to the optimal value [41]. Subsequently, a crossover operation is performed to generate offspring solutions that converge to the optimal solution. After crossover, a mutation is performed to avoid converging to local optima. The algorithm terminates when the maximum number of generations is produced or when the target fitness score is attained. However, the distributed execution of the algorithm at each fog server may not lead to a globally optimal solution because fog-to-fog offloading was not considered. Moreover, the triggering event at which the offloading algorithm started to execute was not mentioned.

In [42], the authors proposed fog-assisted data services for n a heterogeneous vehicular communication environment. Four communication types are permitted between network nodes: device-to-cloud, device-to-fog, fog-to-fog, and device-to-device. The bitwise XOR coding technique was used to encode and decode the data packets. The aim is to encode data items that satisfy the maximum number

of requests with a minimum delay. To achieve this, the authors proposed a greedy heuristic scheduling algorithm to identify the highest priority clique. The proposed scheduling algorithm has three main phases: initializing the clique, updating the clique, and identifying the clique with maximum priority. The first phase starts with an empty clique, searches the graph to identify the vertex with the maximum priority, and then appends it to the clique. In the second phase, the remaining vertices are traversed to add a maximum degree vertex to the clique. The search is completed when there are no more vertices to be found. After identifying all the cliques in the graph, the maximum priority clique was selected. The SDN controller notifies the cloud node, which in turn responds by encoding the corresponding data items, and then broadcasts the encoded packet. Since device-to-cloud communication is unavoidable, executing latency-sensitive tasks may not be feasible in such architectures. In addition, selecting the highest priority clique may lead to starvation. However, no anti-starvation policy has been mentioned.

An SDN-based offloading policy was presented in [43]. This policy exploited underutilized mobile units and parked vehicles as fog nodes. A variant of the conventional three-tier network architecture was presented. The architecture has three layers: a conventional fog-node layer, an SDN layer, and Offload Destination (OLD) layer. The SDN layer consists of an SDN controller and a set of OF-switches. The OLD layer includes the cloud and mobile fog nodes. To overcome the single point of failure deficiency that originates from using one SDN controller, a local SDN agent is installed at the fog nodes and executes two main control functions. The first control function calculates the time required to fully execute the task, and the second determines the feasibility of the request [43]. An SDN-offloading policy was proposed to select the optimal OLD that can satisfy the request. However, the cost of maintaining a dynamic network topology that includes mobile fog nodes was not addressed. Moreover, using vehicles and mobile devices as fog nodes raises several concerns, such as network and endpoint security, energy consumption, protecting user privacy, and remote management.

In [44], the authors proposed a new dynamic fog-to-fog offloading service. They selected logically centralized, physically distributed SDN-based network architecture. In the network, the SDN controller is responsible for monitoring and collecting the status of network infrastructure elements such as SDN switches and fog nodes, while the fog orchestrator controls the operation of fog nodes. Upon receiving an offloading request from an overloaded fog node, the fog orchestrator starts an offloading service that performs two main tasks. First, a list of node IDs that satisfies the request is created. These nodes are then ranked according to their computation and network capabilities. Finally, the best fog node was assigned as the target offloading node. In the worst-case scenario, nearly all the remaining fog nodes may be on the candidate list. For m fog nodes and n candidate nodes, the time complexity for selecting the optimal offloading node is $O(n+m)$. The authors introduced a threshold value to limit the number of candidate fog nodes, particularly in large networks. However, the authors did not provide criteria for selecting a threshold value. Moreover, no conditions were mentioned regarding the tasks selected for offloading from an overloaded node.

In the fog-based architecture proposed in [45], the fog layer is an ad hoc network. If the queuing waiting time at any fog node exceeds an offloading threshold value, the fog node sends an offloading request to the best neighboring fog node with the minimum value for the summation of queuing and propagation delays. The threshold is dynamically updated based on the node workload and availability of the remaining neighbors. If all neighboring fog nodes are overloaded, the request is sent to the cloud server. The proposed offloading service was applied to urgent and non-urgent tasks. However, priority assigning policies and deadline handling of urgent tasks were not mentioned. In addition, it was not specified if offloading was performed between one or k-hop neighbors. Offloading between one-hop neighbors prevents a balanced workload distribution among all the fog nodes. However, offloading tasks between k-hop distant neighbors involve routing costs.

Fuzzy decision-based task offloading management was proposed in [46]. An orchestrator layer is responsible for executing the fuzzy logic rules to determine the target offloading node, which can be a local edge node, neighboring edge node, or cloud node. The offloading decision was made for each new task created by a user device. The task orchestrator management node is also responsible for sending the task to the selected offloading node, collecting the results after execution, and sending the results back to the local edge node. Hence, the orchestrator node is a bottleneck and may become congested as the number of tasks created increases. Authors in [47] proposed an optimization algorithm for resource allocation and load balancing. In the proposed system architecture, the organizer module decides which tasks are offloaded to which fog nodes. The aim is to achieve load balancing and reduce delays. However, the system lacks scalability because all requests have to be managed through a centralized decision-maker module, which represents a performance bottleneck at high network loads. A summary of the reviewed task offloading algorithms is presented in Table 1.

Table 1: Summary of reviewed task offloading algorithms

Ref.	Offloading decision	Offloading service is triggered	Offloading is performed from	Number of offloaded tasks per request	SDN-based	Application	Main contribution	Main issue which was not considered	Complexity analysis derived	Performance evaluation tool
[34]	Centralized	For each new task	-Device-to-fog -Fog-to-cloud	One	No	General	Load balancing	Scalability	No	Simulation using MATLAB
[35]	Centralized	For each new task	Device-to-fog	One	No	IIoT	QoS optimization	Overloading	No	Experimental using MPSoC
[36]	Centralized	For each new task	Device-to-fog	One	Yes	General	QoS optimization	Concurrent tasks	Yes	Simulation using Mininet & POX
[37]	Distributed	For each task cannot be fully executed locally	Device-to-fog	One	Yes	Vehicular	Partitioning the Execution of a single task on multiple fog-nodes	Cost and feasibility of partitioning data file into the required number of blocks	No	Simulation using Mininet
[38]	Centralized	Policy not mentioned According to states of fog nodes	-Device-to-fog -Fog-to-Fog	One Varies	Yes	General	Load balancing	Execution of latency-sensitive tasks	No	Simulation tool not mentioned
[39]	Logically distributed	For each new task For each task cannot be fully executed	Device-to-fog/cloud fog-to-Openflow switches/-cloud	One	Yes	IoT	Reduce data latency	Fog-to-fog offloading	No	Experimental using testbed & simulation using iFogSim
[40]	Distributed	For each new task	Device-to-fog	One	No	EIoT	QoS optimization	Overloading	Yes	Simulation using MATLAB
[41]	Distributed	-For each new task -For a set of tasks	-Device-to-fog -Fog-to-cloud	One	No	Vehicular	Minimize energy consumption	Fog-to-fog offloading	No	Simulation using MATLAB

(Continued)

Table 1: Continued

Ref.	Offloading decision	Offloading service is triggered	Offloading is performed from	Number of offloaded tasks per request	SDN-based	Application	Main contribution	Main issue which was not considered	Complexity analysis derived	Performance evaluation tool
[42]	Centralized	For each new task	Device-to-Cloud/fog/device Fog-to-fog/cloud	Varies	Yes	Vehicular	Minimize total service delay	Delay sensitive constraints were not accounted for in computing the priority	No	Simulation using MATLAB
[43]	Centralized	For each new task	-Device-to-Fog -Device-to-Cloud	One	Yes	Vehicular	Minimize total service delay	network and endpoint security, energy consumption, protecting user privacy and remote management	No	Simulation using MATLAB
[44]	Logically Centralized	Overloaded node, task unspecified	-Device-to-fog -Fog-to-fog	Unspecified	Yes	General	Select optimal offloading fog node	What tasks are selected for offloading	Yes	Mininet
[45]	Centralized	If queuing waiting time at any fog node exceeds an offloading threshold value	-Device-to-fog -Fog-to-fog	One	No	Vehicular	Dynamically Controlling Offloading Thresholds	Priority assigning policy and deadline handling of urgent tasks	Yes	Simulation using iFogSim
[46]	Centralized	For each new task	-Device-to-edge -Device-to-Cloud -Edge-to-Edge	One	No	General	Select optimal offloading node based on fuzzy logic rules	Scalability since, orchestrator node is a bottleneck	No	Simulation using EdgeCloudSim
[47]	Centralized	For each new task	Device-to-fog	One	No	Healthcare	Select optimal fog node based on load-balancing algorithm	Scalability because all tasks are offloaded to fog nodes through single organizer module	No	Simulation using iFogSim
Proposed service	Logically Centralized	Overloaded node, task offloaded according to type	-Device-to-fog -Fog-to-fog -Fog-to-Cloud	-One -Task set -Task set	Yes	General	Set task offloading priority, determine the size of offloaded tasks, and minimize the time required to select offloading nodes	mobility-aware task offloading	Yes	Custom simulation tool

4 Proposed Service

This section presents the proposed dynamic offloading service. Section 4.1 describes the system model. The proposed offloading algorithms are presented in Section 4.2. Section 4.3 discusses the

effect of offloading on task latency and deadline calculations. Section 4.4 discusses complexity analysis.

4.1 System Model

The first layer of the proposed network architecture includes smart and IoT devices. In this layer, nodes are connected via wireless or wired links to the fog nodes. The second layer consists of a set of m identical fog nodes at both the hardware and software levels. These nodes can be deployed in public areas, such as road networks, industrial zones, and airports, and in private areas, such as smart factories and hospitals. The fog nodes are connected to a set of core SDN switches. The third layer consists of similar SDN controllers that run the same set of applications and provide identical services. Network events received by a controller are flooded to the remaining controllers using the east-west interface. Hence, a global network view is created at each SDN node [26,51], and redundancy and scalability are maintained. The SDN architecture is connected to the cloud via backbone routers. In the SDN-based fog architecture shown in Fig. 1, SDN nodes form a fully connected network to share information and distribute workloads. Each controller manages and monitors a set of SDN switches and fog nodes. However, this assignment can change according to network status. The out-of-band control mode is selected to send monitoring and management information, install flow rules in SDN switches, and send offloading decisions to fog nodes via the southbound interface. Open Network Operating System (ONOS) [51] is deployed to support SDN-based fog architecture.

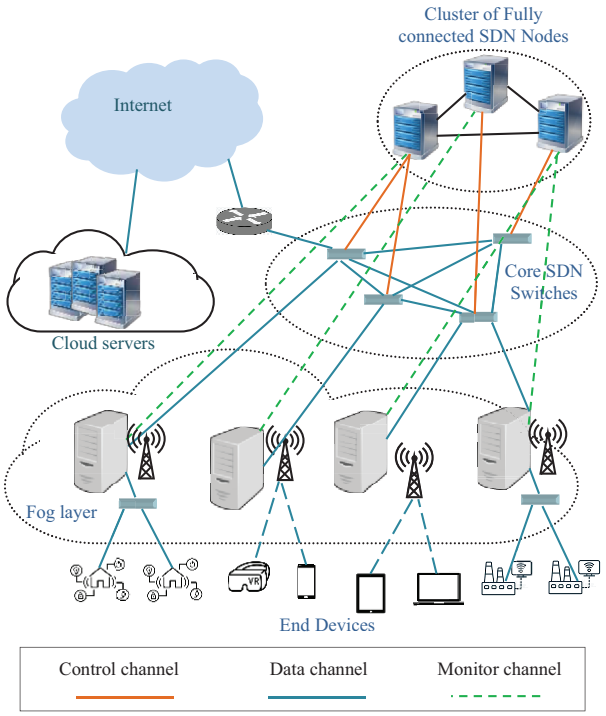

Figure 1: SDN-based fog architecture

Offloading can occur once or twice. The first offloading occurs from device-to-fog. Essentially, tasks are offloaded if they cannot be fully executed at the node that is currently running the task. This occurs when tasks require specific services that cannot be locally satisfied using available resources. However, the device must first send an association request to the nearest fog node. When

the association process completes, the end device may offload the task to the fog node. The second offloading occurs from fog-to-fog or fog-to-cloud. The offloading destinations are determined by the SDN controller. If a fog node is overloaded, it sends an offloading request to the SDN controller. Based on the global network view, the controller selects the offloading nodes which can execute the tasks without violating latency constraints. Then, the IDs of the offloading nodes and information about selected routes are sent to the overloaded fog node, which initiates the offloading process.

Two issues must be addressed before presenting the formulation of the problem. The first issue is to determine when a fog node is considered overloaded. The second is to determine the type and number of offloaded tasks. An overloaded server experiences noticeable throughput degradation. Therefore, servers often have threshold monitoring tools that identify performance issues by applying threshold rules that define acceptable system performance. These rules are usually set during system configuration, and the node is considered overloaded if a threshold value is reached. In this study, it is assumed that a fog node is overloaded if any of its resources has reached the threshold value. That is because; a resource is considered performance-critical if it is over-utilized. In addition, the maximum resource utilization ratio represents the fog node utilization. For example, if the Central Processing Unit (CPU), physical memory, and disk utilization are 20%, 45%, and 10%, respectively. Then, the utilization of the fog node is 45%, which is the maximum resource utilization value calculated at that node. Hence, if node utilization is u, then it is guaranteed that any resource in that node has utilization $\leq u$.

4.2 The Dynamic Offloading Service

In the proposed service, determining which tasks qualify for offloading depends on the offloading source and destination nodes. For device-to-fog offloading, each task that could not be fully executed at the end device was offloaded to the nearest fog node. Hence, in this case, the number of offloaded tasks per request is one, and offloading is performed for any task type, as shown in Algorithm 1. For fog-to-fog and fog-to-cloud offloading requests, an overloaded fog node i is responsible for selecting the type of tasks to be offloaded and sending the list of candidate tasks for offloading T_i^c to the SDN controller. The set of all the tasks running on fog node i is T_i where $1 \leq n(T_i^c) < n(T_i)$. The list is sent to the SDN controller, which invokes a dynamic offloading service that selects the offloaded tasks, and assigns them to fog or cloud nodes without violating the latency constraints of those tasks. Additionally, it installs the flow rules in SDN switches based on the optimized route selected according to network status.

Algorithm 1: Pseudocode for Device-to-Fog Offloading.

```
for a task t at end device e
  L ← getTaskMaxLatency(t);
  Rr ← getRequiredResource(t);
 Ar ← getAvailableResource(e);
 if (Rr cannot be satisfied by Ar)
    sendOffloadingRequest(t, e, i, Rr, L);
 end if
end for
```

The SDN controller has a number of utilization vectors n. Each utilization vector UV_j has a certain threshold U_j, where $j = 0, 1, 2, \ldots, n-1$. The utilization step U_s between any two consecutive

thresholds is fixed, and the value of U_j is determined as follows.

$$U_j = U_s + U_{j-1}, \tag{1}$$

where $U_{-1} = 0$. The number of utilization vectors n depends on the maximum utilization U_{max} and the value of U_s as follows.

$$n = \frac{U_{max}}{U_s} \tag{2}$$

As mentioned in Section 4.1, threshold rules determine system acceptable performance. If the utilization of a fog node exceeds the value of U_{max}, the node is considered overloaded. During network configuration, the default U_s value is determined based on normal network load conditions. Fog node i sends its resource utilization value u_i to the SDN controller. The controller records u_i in the fog utilization array UF. The ID of fog node i is recorded in the corresponding utilization vector UV_j, according to the following condition:

$$ID_i \in UV_j \leftrightarrow UV_{j-1} < u_i \leq UV_j \tag{3}$$

The controller uses the size of the utilization vectors to monitor the status of the fog nodes and determine the next $U_s$value. Because the value of U_s determines the size of offloaded tasks, it should be adjusted to avoid network congestion. At a high load, many fog nodes can become overloaded. Hence, the default U_s value may lead to network congestion owing to offloading large amounts of traffic sent from overloaded nodes. In contrast, at low load conditions where few fog nodes are overloaded, raising the value of U_s reduces the number of offloading requests. The length of the last utilization vector $L(UV_{n-1})$ and a predetermined factor f control the value of the next U_s as shown in Eq. (4), where m is the number of fog nodes, and n_d is the number of utilization vectors based on U_s default value:

$$U_{s(next)} = \begin{cases} \dfrac{U_{s(current)}}{f}, & L(UV_{n-1}) \geq m/n_d \\ U_{s(current)}, & 0 < L(UV_{n-1}) < \dfrac{m}{n_d} \\ U_{s(current)} \times f, & L(UV_{n-1}) = 0 \end{cases} \tag{4}$$

Any fog node that fulfills the condition $u_i > U_{max}$ is considered overloaded, and it starts to execute the task selection algorithm to determine the number and type of tasks to be offloaded. Assume that application al is running on node i where $l = 1, 2, \ldots, A_i$, and A_i is the total number of applications running on fog node i. Application al has a number of instances $I^i_{a_l}$. Hence, the task instance utilization $U^i_{a_{l,},r}$ is given in Eq. (5), and the total utilization $U^i_{a_l}$ of each application a_l is given in Eq. (6):

$$U^i_{a_{l,},r} = \max\left(U^i_{R1}, U^i_{R2}, \ldots, U^i_{Rz}\right)_r \quad where\ r = 1, 2, \ldots, I^i_{a_l}, \tag{5}$$

$$U^i_{a_l} = \max\left(\sum\nolimits_{r=1}^{I^i_{a_l}} \left(U^i_{R1}\right)_r, \sum\nolimits_{r=1}^{I^i_{a_l}} \left(U^i_{R2}\right)_r, \ldots, \sum\nolimits_{r=1}^{I^i_{a_l}} \left(U^i_{Rz}\right)_r\right), \tag{6}$$

where z is the number of resources at fog node i and the value of U^i_{R1} is normalized using min-max normalization. Table 2 lists the main notations used in this section.

Table 2: Main notations

Symbol	Description
M	Number of fog-nodes in the network
A_i	Number of applications running on fog node i
a_l^i	Application a_l running on fog node i
$t_{a_l^i}$	Task instance of a_l^i
$I_{a_l}^i$	Number of instances of application a_i running on fog node i
T_i	Set of all tasks at fog node i
T_i^c	set of candidate tasks for offloading running on fog node i
u_i	Utilization of fog node i
N	Number of utilization vectors
U_j	Utilization threshold, where $j=0, 1, .., n-1$
U_s	Utilization step between any two consecutive utilization thresholds
U_{max}	Maximum utilization for any fog node
UV_j	Utilization vector of U_j
$U_{a_{l,}, r}^i$	Task instance utilization at fog node i
$U_{a_l}^i$	Total utilization of application a_l
Z	Number of resources at fog node i
U_{Rz}^i	Total utilization of resource R_z at fog node i
UF	Fog utilization array
$L\,(UV_{n-1})$	Length of the last utilization vector
$OLTS_i$	Offloaded task set for fog node i
$OLFS_i$	Offloading fog set for fog node i
$Lt_{a_l^i}$	Maximum allowed latency of $t_{a_l^i}$
L_c	Latency of route c from best route matrix
BRM_i	Best-route matrix for fog node i

Each fog node has three task vectors that are continuously maintained. The first task vector V_1 has the task ID and $U_{a_l}^i$ of latency-tolerant tasks such as file storage, file management, and environmental monitoring tasks. In this vector, tasks are ranked according to the value of $U_{a_l}^i$. Tasks in the second and third vectors are ranked according to the value of latency. The second vector V_2 has the task ID, value of $U_{a_{l,}, r}^i$, and latency of delay-sensitive non-urgent tasks such as video streaming and online gaming tasks. The third vector V_3 has the task ID, value of $U_{a_{l,}, r}^i$, and latency of delay-sensitive hard deadline tasks such as autonomous driving tasks. All vectors are ranked in descending order, as shown in Algorithm 2.

Algorithm 2: Pseudocode to Classify Tasks.

```
for each task at i do
    Type← determineTaskType(t);
     if (Type = LatencyTolerant)
        V1← addTask(t);
     else if (Type = LatencySensitive_NotCritical)
        V2← addTask(t);
     else
        V3← addTask(t);
     end if
end for
for set of task instances in V1 do
    V1←descendingSort(U^i_al);
end for
for each task at V2 do
    V2←descendingSort(Lt_ali);
end for
for each task at V3do
    V3←descendingSort(Lt_ali);
end for
```

Once a fog node enters an overloaded state ($u_i > U_{max}$), it sends an offloading request to the SDN controller. There are three types of offloading requests. The first type is fog-to-cloud offloading. It is the only allowed type of offloading in an overloaded fog node if the length of $V_1 \neq 0$. In this case, the SDN controller optimizes a route between the fog node and the backbone router then the offloading process commences, as shown in Algorithm 3 (Fog-to-Cloud Offloading FCO). Then, V_1 is updated. If $u_i > U_{max}$ and V_1 is empty, the second type of offloading (i.e., Utilization-based Fog-to-Fog Offloading UFFO) is activated. The request holds an ordered list according to the latency of all candidates in V_2. Similarly, if the lengths of V_1 and V_2 are equal to zero, all tasks in the overloaded node are critical and latency-sensitive tasks. Hence, the third type of offloading (i.e., Latency-based Fog-to-Fog Offloading LFFO) is activated.

Algorithm 3: Pseudocode for FCO algorithm.

```
At fog node i
If (length(V1) > 0 And ui > Umax )
  sendFogToCloudOffloadingRequest(i, SDN);
end if
At SDN
for each set of task instances in V1 do
  If (U_total < U_s)
    U_total += U^i_al;
     OLTS_i←addOffloadedTask(t_ali);
  else
    Route_t←determineBestRoute (i, gatewayRouter);
```

(Continued)

Algorithm 3: Continued

```
12:     S←getOFSwitches(Route_t);
13:     sendOffloadingReply(SDN, i, OLTS_i, Route_t);
14:     for each element in S do
15:       applyFlowRules();
16:     end for
17:   end for
```

In the proposed network architecture, the SDN controller determines the best route, according to the target metric, from each fog node as the source node to every other node as a possible destination in the network. For each fog node, the SDN controller creates a Best-Route Matrix (BRM), as shown in Algorithm 4, which stores information about routes as follows.

- For a fog node i, each row in the matrix has four elements:
  - o The ID of the destination fog node.
  - o The total transmission and propagation delays along the selected route Lc.
  - o The hop count
- Elements of the matrix are arranged in ascending order according to the value of latency.
- In the case of a tie, the hop count is used to break the tie. If it still holds, the lowest node ID breaks the tie.

Once the SDN controller receives a fog-to-fog offloading request and V_2 is not empty, it executes a greedy heuristic offloading algorithm that allocates the offloaded tasks to the best utilization fog nodes that do not violate latency constraints, as shown in Algorithm 5 (UFFO algorithm). The proposed algorithm returns two values. The first is the offloaded task set $OLTS_i$, and the second is the offloading node set $OLFS_i$. At the start of execution, $OLTS_i$ and $OLFS_i$ are empty. As long as the latency condition in line 4 holds true, the best utilization node is selected as per lines 5–8. If the utilization condition in line 15 holds, then the corresponding entities from lines 16 to 18 are updated. The process terminates if the condition in line 22 holds or if the end of the task list is reached.

Algorithm 4: Pseudocode for Creating Best-Route Matrix (BRM_i) and Utilization Array (UF)

```
1:   At SDN
2:   if real-time network status change
3:     for each i do
4:       for each k ∈ m do
5:         If (i ≠ k)
6:           Lc← determineBestRoute(i,k);
7:           BRM_i←updateRow(Lc, hop-count, ID, ref_d);
8:         end if
9:       end for
10:    end for
11:    for each BRM_i do
12:      BRM_i ← ascendingSort(Lc, hop − count, ID);
13:    end for
14:  end if
```

(Continued)

Algorithm 4: Continued

```
15:   if ( u_i of node i changes)
16:      UF ←updateRow(i, u_i);
17:   end if
```

Algorithm 5: Pseudocode for UFFO algorithm

```
1:  If controller receives fog-to-fog offloading request from i
2:    for each element t^i_{a_l} in V_2 do
3:      for each element x in BRM_i do
4:        If (Lt^i_{a_l} > BRM_i[x][0])
5:          If(ref_d ≠ null)
6:            If (bestU < u_i)
7:              bestU = u_i;
8:              OL = x;
9:            end if
10:         else break;
11:         end if
12:       else break;
13:       end if
14:     end for
15:     If (best U + U_total + U^i_{a_l,r} < U_{n-1})
16:          OLFS_i ←addOffloadingFogNode(BRM_i[OL][2]);
17:         OLTS_i ←addOffloadedTask(t_{a_l^i});
18:         U_total += U^i_{a_l,r} ;
19:       break;
20:     end if
21:     bestU = 0;
22:     If (U_total ≥ U_s) break; end if
23:   end for
24: end if
25: U_total = 0;
26: return OLFS_i, OLTS_i
```

Algorithm 6 (LFFO algorithm) starts by initializing the offloaded task set $\boldsymbol{OLTS_i}$ and the offloading node set $\boldsymbol{OLFS_i}$. To add task instances in $\boldsymbol{OLTS_i}$, the algorithm identifies which tasks from the set of candidates can be offloaded without violating latency constraints. The algorithm compares the maximum allowed latency of the task with the latency recorded in $\boldsymbol{BRM_i}$. If $\boldsymbol{Lt_{a_l^i}} > \boldsymbol{L_c}$, then the task may be offloaded. If the utilization condition in line 6 holds, then $\boldsymbol{OLFS_i}$ is updated with the ID of the offloading fog node and task instances are added to $\boldsymbol{OLTS_i}$. Otherwise, the next $\boldsymbol{L_c}$ is checked. Because the first row $\boldsymbol{BRM_i}$ has the lowest latency, and the first task instance in the offloading request has the highest maximum latency, if the latency condition as per line 4 in LFFO fails, no tasks can be offloaded from that fog node and the comparison stops because all subsequent route latencies are higher than $\boldsymbol{Lt_{a_l^i}}$. If the utilization condition in line 6 does not hold, the latency condition is checked for the next entry in $\boldsymbol{BRM_i}$.

For both algorithms UFFO and LFFO, if $\sum U^{i}_{a_{l},r} < U_s$ the next task candidate is checked and $OLTS_i$ and $OLFS_i$ are updated until $\sum U^{i}_{a_{l},r} \geq U_s$ or the end of the offloading candidate list is reached. The values of $OLTS_i$ and $OLFS_i$ and the information about optimized routes are sent to the overloaded node to start the offloading process.

Algorithm 6: Pseudocode for LFFO algorithm

```
If controller receives fog-to-fog offloading request from i
  for each element t^i_{a_l} in V_3 do
   for each element x in BRM_i do
     If (Lt^i_{a_l} > BRM_i[x][0])
       If(ref_d ≠ null)
         If(u_i + U_total + U^i_{a_l,r} < U_{n-1})
             OLFS_i ← addOffloadingFogNode(BRM_i[x][2]);
             OLTS_i ← addOffloadedTask(t^i_{a_l});
             U_total += U^i_{a_l,r} ;
             break;
         end if
       else break;
       end if
     else break;
     end if
   end for
   If (U_total ≥ U_s) break; end if
  end for
end if
U_total = 0;
return OLFS_i, OLTS_i ;
```

4.3 Computing Task Latency and Deadline

The service discipline is performed using the preemptive Earliest Deadline First (EDF) algorithm, which assigns dynamic priorities to tasks according to their deadlines. The queuing model at the fog node is an M/G/1/./EDF queue which was presented in [52]. Hence, when a task that has the nearest deadline arrives at the ready queue, it is assigned the highest priority and preempts the currently executing task. It has been proven that under normal operating conditions (i.e., when a fog node is not overloaded), EDF is an optimal scheduling algorithm for real-time independent tasks [52,53]. However, offloading a task from one node to another requires recalculations of deadlines. That compensates for the time consumed in the transmission and propagation delays. Consider the following example in Fig. 2, a task with a total of 10 ms of computation time has arrived at the ready queue of an end device at 100 ms. It was executed for 2 ms at the end device and then offloaded to an overloaded fog node and waited in the queue for 1 ms.

Subsequently, it was offloaded again to Node B to complete the remaining 8 ms of its execution time. However, transmission and propagation delays were not considered. Assume that Node B finished task execution after 190 ms (i.e., before the deadline). However, the result still needs 35 ms to return to the end device. It would arrive at 225 ms; hence, the task misses the deadline. To rectify this, the deadlines must be recalculated before the task is offloaded. In the same example, the deadline

at Node A should be no more than 193 ms and at Node B no more than 162 ms. Accordingly, during the first offloading, the end device recalculates the deadline before offloading the task. For the second offloading, the SDN controller recalculates the deadline by subtracting the latency from Node A to B, which consists of transmission and propagation latencies in addition to the time consumed in executing the offloading algorithm. As mentioned in Section 4.1, the network architecture has identical fog nodes; hence, the processing delay is the same for all the nodes.

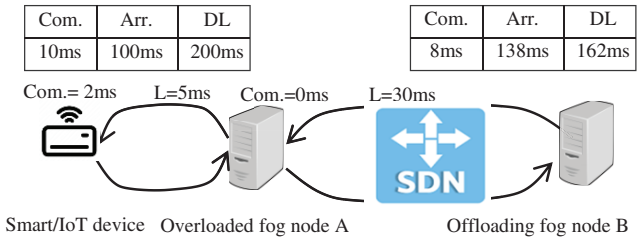

Figure 2: Deadline recalculation example

4.4 Complexity Analysis

The SDN controller executes Algorithm 4 when the network status changes. That may occur because of changes in the device or network configuration: a network link is down, device failure, or network congestion. The worst time complexity of the algorithm is $O\left(m^2\,(1+log\ m)\right)$. The worst time complexity of the FCO algorithm is $O\,(A_i)$, where A_i is the number of running applications in fog node i. Assume that the best utilization node is the last node in BRMi. Accordingly, the worst time complexity of the UFFO algorithm is $O\left(\,n(T_i^c) * m\right)$, where $n(\ T_i^c)$ is the number of candidate tasks for offloading. The worst-case scenario of the LFFO algorithm assumes that the first half of the fog nodes in BRM_i are overloaded and that all elements in BRM_i do not violate the latency condition. Hence, the worst time complexity is $O\left(\,n(T_i^c) * m\right)$.

5 Performance Evaluations

The performance evaluations were performed using Apache NetBeans [54]. The offloading algorithms were written in Java. The custom simulator built in [55] was updated to include the fog and SDN layers. The fog layer has three sizes: 16, 32, and 64 nodes. For the three fog layer sizes, three network dimensions were selected (200 m × 200 m, 400 m × 400 m, and 800 m × 800 m). The numbers of end devices associated with 16, 32, and 64 fog nodes are 400, 800, and 1600 devices, respectively. The mobility mode for end devices was disabled, and transmission ranges were: 40, 60, 80, and 100 m. The SDN network was simulated as a binary tree network in which 16, 32, and 64 fog nodes are connected to sets of 15, 31, or 63 OF-switches, respectively. The root switch was connected to the SDN controller. The tasks were created using a traffic generator. Then, they are randomly assigned to end devices, which offload those tasks to fog nodes. The physical server that ran the simulation had four cores and eight virtual threads with 16 GB of RAM.

5.1 Simulation Setup

The task type and required resources (i.e., CPU, memory, disk, and bandwidth) were randomly assigned for each task. However, when evaluating the impact of offloading on a certain performance metric, task type distribution can be biased. The proposed algorithms were compared with the Dynamic Fog-to-Fog (DF2F) offloading algorithm presented in [44]. The average simulation results

were based on 20 executions for each algorithm for each network size. Because DF2F did not specify the selection methodology of offloaded tasks, all compared algorithms used one task instance per application. The values of simulation parameters are listed in Table 3.

Table 3: Values of simulation parameters

Parameter	Value
Number of fog-nodes in the network	16, 32 and 64 nodes
Number of SDN switches	15, 31 and 63 switches
CPU utilization range	0.25–0.8
Memory utilization range	0.25–0.8
Bandwidth range	15–50 Mbps
Server to switch latency	2 ms
Switch to switch latency	1 ms
End device to fog server latency	1 ms
Latency of V_1 tasks	No limits assigned
Latency of V_2 tasks	50–300 ms
Latency of V_3 tasks	50–100 ms
Task arrival rate according to the number of fog nodes	(200, 400, 800) task/s for (16, 32, 64) fog nodes
$U_{s(Default)}$	0.2
F	2
U_{max}	0.8

Because no delay is assigned to latency-tolerant tasks, all the required resources for these tasks can be selected randomly. The ranges used in the simulation were 1%–10% of the CPU time, 10–300 MB of memory, and 15–50 Mbps of required bandwidth. The corresponding values for latency-sensitive non-urgent tasks were as follows: from 0.1% to 2% of CPU time and from 1 to 100 MB of memory size. For latency-sensitive urgent tasks, the values range from 0.1% to 0.5% of the CPU time and from 1 to 50 MB of memory. The required bandwidth for latency-sensitive tasks was computed according to the assigned latency. If the value exceeded the bandwidth range (i.e., from 15 to 50 Mbps), task CPU time and memory size were reselected.

5.2 Results and Discussion

The comparison with DF2F was performed for fog-to-fog requests because fog-to-cloud offloading was not addressed in [44]. For the proposed algorithms, $\boldsymbol{U}^{i}_{a_l,r}$ for each task was computed based on the required resources. The value of u_i is calculated based on the resource consumption of all tasks at fog node i. The average time required to select an offloading node was improved by 33.34% when compared to DF2F, as shown in Fig. 3a. For the three proposed algorithms, increasing the number of nodes increased the selection interval, as shown in Fig. 3b.

The proposed algorithms outperformed the offloading node selection time achieved by DF2F for two main reasons. First, for every offloading request, the DF2F algorithm executes two main functions. The first function finds all available nodes that can satisfy the offloading request. The output of the first function is the input for the second function, which selects the node with the highest final

resource score [44]. As mentioned in Section 3, the time complexity for offloading a task using DF2F is $O\ (n+m)$. However, the time complexity of UFFO or LFFO is $O\ (m)$ when one task is offloaded per request, as in DF2F. Second, to compute the final resource score in DF2F, the second function calculates the individual score for each resource in each fog node in the list of available nodes. Thus, the optimal offloading node is determined. Each offloading request requires the computation of the final resource score before selecting the node with the highest score. Hence, the algorithm adopts a sequential execution. However, in the proposed service, it is assumed that all fog nodes are identical. Hence, the resource requirements of a task at any fog node are the same. The service selects optimal tasks for offloading according to their type and latency constraints. The selection of offloading node is determined based on the offloading service type. In the UFFO algorithm, the optimal offloading node is the node with the lowest utilization value that does not violate the latency constraints of the task. In the LFFO algorithm, the optimal offloading node is the one that has the lowest route latency, which does not violate utilization constraints. In both algorithms, no calculations are performed on individual resources. Both algorithms used the available data in the UF and BRM_i matrices. Because of the SDN centralized architecture that supports the global network view, these matrices are updated continuously regardless of the presence of offloading requests.

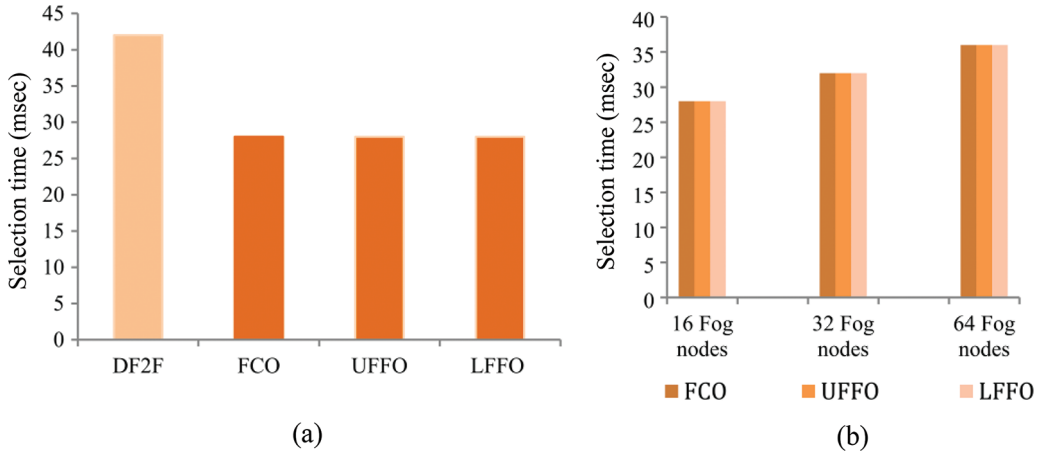

Figure 3: Average time required to select an offloading node

Figs. 4a and 4c show utilization snapshots of two 32-fog node networks before offloading starts. The main requirement is to evaluate the impact of offloading on the fairness of utilization distribution for fog nodes. Fig. 4b shows the utilization after performing fog-to-fog offloading for latency-sensitive non-urgent tasks. Nodes with high utilization but not yet overloaded are grey-colored, as shown in Figs. 4a and 4b. These were not selected to be offloading destinations. Except for these nodes, the utilization of nodes converges to be a uniform distribution. Fig. 4d shows the utilization of fog nodes for latency-sensitive urgent tasks after executing the LFFO algorithm. The utilization of Node 1 changed from 29% to 74% because it offered the best latency to Nodes 4 and 5. However, offloading tasks from Node 5 to Node 6 was avoided because the best latency route may not be the one with the shortest hop count.

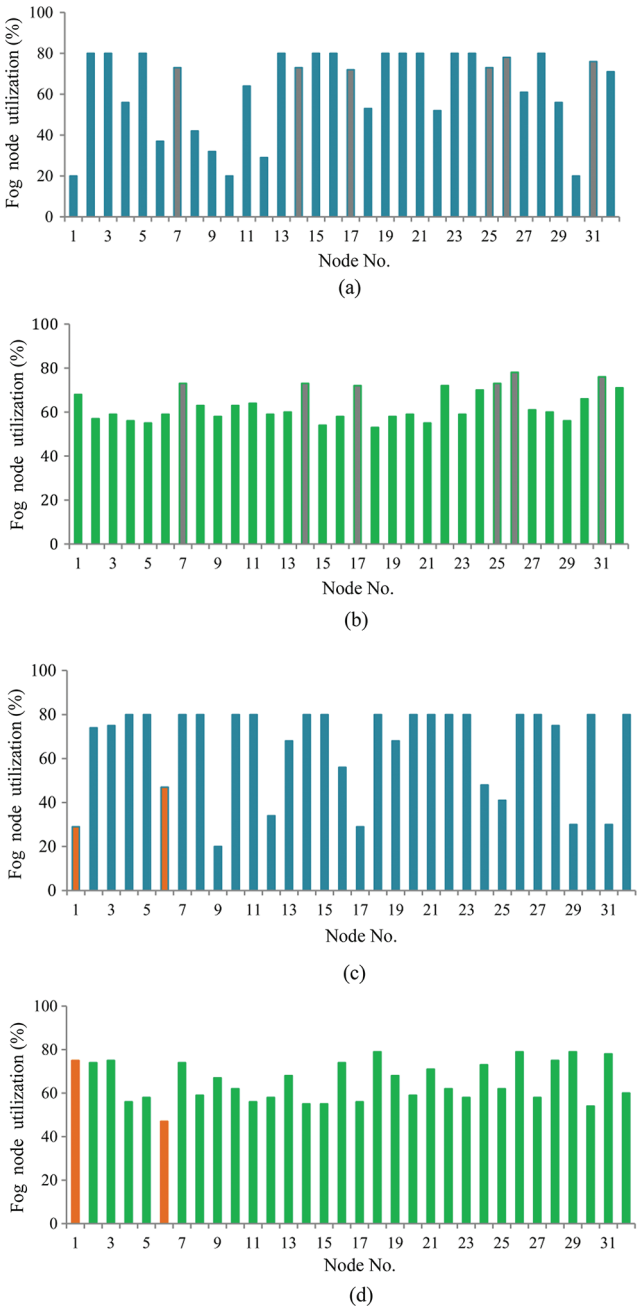

Figure 4: Utilization snapshot comparison (a) utilization of fog nodes before offloading, (b) utilization of fog nodes after executing UFFO, (c) utilization of fog nodes before offloading, (d) utilization of fog nodes after executing LFFO

The distribution of the average hop distance between overloaded and offloading nodes is shown in Fig. 5. A comparison of the average hop distance between LFFO and DF2F algorithms is shown in Fig. 5a. In both algorithms, most offloading nodes are selected from the set of neighbors three hops away from the overloaded node. As depicted in Fig. 5b, the average hop distance distribution has a

descending pattern regardless of the network size. Fog nodes three hops away from the overloaded nodes have the highest ratio of assigned offloading requests. The average hop distance ratio decreases as the hop count increases because the LFFO algorithm selects offloading nodes with the lowest latency. A fog node is selected if all fog nodes at smaller hop distances cannot satisfy the request. However, the distribution of hop distance has an opposite pattern for nodes with one and three hop counts. That is greatly affected by the binary tree architecture because the number of possible offloading nodes doubles as the test for these nodes moves up one level in the tree. That doubles the probability of choosing an offloading node at the current level if no offloading nodes are selected in the previous level. For example, in a 16-fog node network, assume that Node 7 is overloaded. If Node 8 is not selected as an offloading node, the probability that one of the Nodes 5 or 6 is selected doubles. However, this probability is reduced to half of its value each time the network size doubles. That explains the gradual increase in the ratio of offloading nodes selected from the set of nearest neighbors.

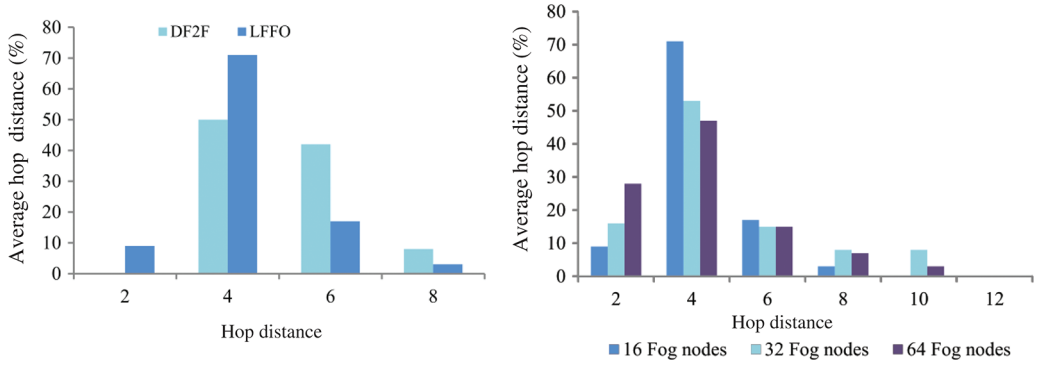

Figure 5: The distribution of average hop distance between overloaded and offloading nodes; (a) The distribution of average hop distance comparison between DF2F and LFFO; (b) The distribution of average hop distance comparison of LFFO algorithm for 16, 32, and 64 fog nodes

As previously mentioned in Section 1, the four main questions that govern the design of offloading algorithms shall be addressed during the design of the offloading service. The proposed dynamic offloading service answered these questions as follows.

- Tasks selected for offloading are determined based on their types. The selection of tasks and offloading type is determined according to data stored in V_1, V_2 and V_3.
- The number of offloaded tasks depends on the value of U_s, which is determined using Eq. (4), and the value of $U^i_{a_l}$, for the FCO algorithm, or the value of $U^i_{a_{l,}, r}$, for Algorithms UFFO and LFFO.
- The selection of offloading fog nodes was based on the type of offloading service provided by FCO, UFFO, or LFFO algorithms. FCO offloads tasks to cloud nodes, whereas algorithms UFFO and LFFO select offloading fog nodes, as mentioned earlier.
- The number of offloading nodes varied according to the offloading algorithm. FCO does not determine the number of offloading nodes because it is beyond the scope of this study because all offloaded tasks are assigned to cloud servers according to allocation decisions determined by the cloud resource allocation scheduler. In algorithms UFFO and LFFO, the number of offloading nodes is determined according to the utilization of fog nodes and the value of U_s.

6 Conclusion and Future Work

This paper proposes three offloading algorithms, each of which targets a specific type of task. FCO algorithm offloads latency-tolerant tasks from an overloaded fog node to the cloud server. The UFFO algorithm selects offloading nodes for non-urgent latency-sensitive tasks. The LFFO algorithm determines offloading nodes for latency-sensitive urgent tasks. Candidate tasks for offloading are selected by the overloaded node; subsequently, the offloading request is sent to the SDN controller. The main motivation behind this approach is to increase the interval in which a fog node remains in the non-overloaded state. This is achieved by determining the maximum number of tasks that can be offloaded without causing network congestion. The highest offloading priority is given to latency-tolerant tasks, while latency-sensitive urgent tasks get the lowest offloading priority. The participation of fog nodes and SDN controllers in choosing offloaded tasks reduces the time required to select offloading nodes by 33%. However, the effect of mobility on the proposed service was not considered. It is assumed that end devices remain associated with the same fog node during the entire interval of task execution. Hence, in the future, we will consider developing a mobility-aware task offloading algorithm in a clustered SDN-based FC network. In addition, developing an energy consumption optimization approach that allocates energy based on the task type, level of importance, and latency constraints, will also be considered.

Acknowledgement: The authors would like to thank the support of the Deanship of Scientific Research at Princess Nourah bint Abdulrahman University. This research project was funded by the Deanship of Scientific Research, Princess Nourah bint Abdulrahman University, through the Program of Research Funding after Publication, Grant No. (PRFA–P–42–10).

Funding Statement: This research project was funded by the Deanship of Scientific Research, Princess Nourah bint Abdulrahman University, through the Program of Research Funding after Publication, Grant No. (PRFA–P–42–10).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] "Cisco annual Internet report (2018–2023) white paper," Cisco, 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] "Redefine connectivity by building a network to support the Internet of things," Cisco-Colt, 2019. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/service-provider/pdfs/a-network-to-support-iot.pdf.

[3] "Cisco global cloud index: Forecast and methodology (2016–2021)," Cisco, 2018. [Online]. Available: https://virtualization.network/Resources/Whitepapers/0b75cf2e-0c53-4891-918e-b542a5d364c5_white-paper-c11-738085.pdf.

[4] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC*, Helsinki, Finland, pp. 13–15, 2012.

[5] H. -J. Cha, H. -K. Yang and Y. -J. Song, "A study on the design of fog computing architecture using sensor networks," *Sensors*, vol. 18, no. 11, pp. 3633, 2018.

[6] A. K. Idrees and A. K. M. Al-Qurabat, "Energy-efficient data transmission and aggregation protocol in periodic sensor networks based fog computing," *Journal of Network and Systems Management*, vol. 29, no. 1, pp. 1–24, 2021.

[7] Q. Yaseen, F. AlBalas, Y. Jararweh and M. Al-Ayyoub, "A fog computing based system for selective forwarding detection in mobile wireless sensor networks," in *Proc. FAS∗W*, Augsburg, Germany, pp. 256–262, 2016.
[8] J. Ni, K. Zhang, X. Lin and X. Shen, "Securing fog computing for internet of things applications: Challenges and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 601–628, 2017.
[9] T. Wang, L. Qiu, A. K. Sangaiah, G. Xu and A. Liu, "Energy-efficient and trustworthy data collection protocol based on mobile fog computing in internet of things," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3531–3539, 2019.
[10] X. Li, Y. Liu, H. Ji, H. Zhang and V. C. Leung, "Optimizing resources allocation for fog computing-based internet of things networks," *IEEE Access*, vol. 7, pp. 64907–64922, 2019.
[11] M. Ahmed, R. Mumtaz, S. M. Zaidi, M. Hafeez, S. A. Zaidi *et al.,* "Distributed fog computing for internet of things (IoT) based ambient data processing and analysis," *Electronics*, vol. 9, no. 11, pp. 1756–1775, 2020.
[12] M. Wazid, P. Bagga, A. K. Das, S. Shetty, J. J. Rodrigues *et al.,* "AKM-IoV: Authenticated key management protocol in fog computing-based internet of vehicles deployment," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8804–8817, 2019.
[13] A. Thakur and R. Malekian, "Fog computing for detecting vehicular congestion, an internet of vehicles based approach: A review," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 2, pp. 8–16, 2019.
[14] T. Abar, A. Rachedi, A. Ben Letaifa, P. Fabian and S. El Asmi, "FellowMe cache: Fog computing approach to enhance (QoE) in internet of vehicles," *Future Generation Computer Systems*, vol. 113, pp. 170–182, 2020.
[15] W. Zhang and G. Li, "An efficient and secure data transmission mechanism for internet of vehicles considering privacy protection in fog computing environment," *IEEE Access*, vol. 8, pp. 64461–64474, 2020.
[16] M. S. Eddine, M. A. Ferrag, O. Friha and L. Maglaras, "EASBF: An efficient authentication scheme over blockchain for fog computing-enabled internet of vehicles," *Journal of Information Security and Applications*, vol. 59, pp. 102802, 2021.
[17] S. Tuli, R. Mahmud, S. Tuli and R. Buyya, "FogBus: A blockchain-based lightweight framework for edge and fog computing," *Journal of Systems and Software*, vol. 154, pp. 22–36, 2019.
[18] D. Wu and N. Ansari, "A cooperative computing strategy for blockchain-secured fog computing," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6603–6609, 2020.
[19] K. Lei, M. Du, J. Huang and T. Jin, "Groupchain: Towards a scalable public blockchain in fog computing of IoT services computing," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 252–262, 2020.
[20] P. Kochovski, S. Gec, V. Stankovski, M. Bajec and P. D. Drobintsev, "Trust management in a blockchain based fog computing platform with trustless smart oracles," *Future Generation Computer Systems*, vol. 101, pp. 747–759, 2019.
[21] N. Islam, Y. Faheem, I. U. Din, M. Talha, M. Guizani *et al.,* "A blockchain-based fog computing framework for activity recognition as an application to e-healthcare services," *Future Generation Computer Systems*, vol. 100, pp. 569–578, 2019.
[22] T. S. Darwish and K. A. Bakar, "Fog based intelligent transportation big data analytics in the internet of vehicles environment: Motivations, architecture, challenges, and critical issues," *IEEE Access*, vol. 6, pp. 15679–15701, 2018.
[23] R. K. Barik, H. Dubey, A. B. Samaddar, R. D. Gupta and P. K. Ray, "FogGIS: Fog computing for geospatial big data analytics," in *Proc. UPCON*, Varanasi, India, pp. 613–618, 2016.
[24] W. Zhang, Z. Zhang and H. C. Chao, "Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 60–67, 2017.
[25] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He *et al.,* "A hierarchical distributed fog computing architecture for big data analysis in smart cities," in *Proc. ASE*, Kaohsiung, Taiwan, pp. 1–28, 2015.
[26] O. Salman, I. Elhajj, A. Chehab and A. Kayssi, "IoT survey: An SDN and fog computing perspective," *Computer Networks*, vol. 143, pp. 221–246, 2018.

[27] Y. Bi, G. Han, C. Lin, Q. Deng, L. Guo *et al.,* "Mobility support for fog computing: An SDN approach," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 53–59, 2018.
[28] A. J. Kadhim and S. A. Seno, "Maximizing the utilization of fog computing in internet of vehicle using SDN," *IEEE Communications Letters*, vol. 23, no. 1, pp. 140–143, 2019.
[29] M. Arif, G. Wang, T. Wang and T. Peng, "SDN-Based secure VANETs communication with fog computing," in *Proc. SpaCCS*, Melbourne, NSW, Australia, pp. 46–59, 2018.
[30] J. Gao, K. O. Agyekum, E. B. Sifah, K. N. Acheampong, Q. Xia *et al.,* "A blockchain-SDN-enabled internet of vehicles environment for fog computing and 5G networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4278–4291, 2020.
[31] A. J. Kadhim and S. A. Seno, "Energy-efficient multicast routing protocol based on SDN and fog computing for vehicular networks," *Ad. Hoc. Networks*, vol. 84, pp. 68–81, 2019.
[32] Y. Xiao and M. Krunz, "Dynamic network slicing for scalable fog computing systems with energy harvesting," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2640–2654, 2018.
[33] M. Arif, G. Wang, V. E. Balas, O. Geman, A. Castiglione *et al.,* "SDN based communications privacy-preserving architecture for VANETs using fog computing," *Vehicular Communications*, vol. 26, pp. 100265, 2020.
[34] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020.
[35] K. Cao, J. Zhou, G. Xu, T. Wei and S. Hu, "Exploring renewable-adaptive computation offloading for hierarchical QoS optimization in fog computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2095–2108, 2020.
[36] S. Misra and N. Saha, "Detour: Dynamic task offloading in software-defined fog for IoT applications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1159–1166, 2019.
[37] C. Lin, G. Han, X. Qi, M. Guizani and L. Shu, "A distributed mobile fog computing scheme for mobile delay-sensitive applications in SDN-enabled vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5481–5493, 2020.
[38] J. Y. Baek, G. Kaddoum, S. Garg, K. Kaur and V. Gravel, "Managing fog networks using reinforcement learning based load balancing algorithm," in *Proc. WCNC*, Marrakech, Morocco, pp. 1–7, 2019.
[39] A. Muthanna, A. Ateya, A. Khakimov, I. Gudkova, A. Abuarqoub *et al.,* "Secure and reliable IoT networks using fog computing with software-defined networking and blockchain," *Journal of Sensor and Actuator Networks*, vol. 8, no. 1, pp. 15, 2019.
[40] Y. Wei, H. Yang, J. Wang, X. Chen, J. Li *et al.,* "Delay and energy-efficiency-balanced task offloading for electric internet of things," *Electronics*, vol. 11, no. 6, pp. 839, 2022.
[41] H. Materwala, L. Ismail, R. M. Shubair and R. Buyya, "Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks," *Future Generation Computer Systems*, vol. 135, pp. 205–222, 2022.
[42] K. Xiao, K. Liu, X. Xu, Y. Zhou and L. Feng, "Efficient fog-assisted heterogeneous data services in software defined VANETs," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 261–273, 2021.
[43] A. A. Khadir and S. A. Seno, "SDN-Based offloading policy to reduce the delay in fog-vehicular networks," *Peer-to-Peer Networking and Applications*, vol. 14, pp. 1261–1275, 2021.
[44] L. A. Phan, D. T. Nguyen, M. Lee, D. H. Park and T. Kim, "Dynamic fog-to-fog offloading in SDN-based fog computing systems," *Future Generation Computer Systems*, vol. 117, pp. 486–497, 2021.
[45] F. Alenizi and O. Rana, "Dynamically controlling offloading thresholds in fog systems," *Sensors*, vol. 21, no. 7, pp. 2512, 2021.
[46] M. D. Hossain, T. Sultana, M. A. Hossain, M. I. Hossain, L. N. Huynh *et al.,* "Fuzzy decision-based efficient task offloading management scheme in multi-tier MEC-enabled networks," *Sensors*, vol. 21, no. 4, pp. 1484, 2021.
[47] S. Khan, I. A. Shah, N. Tairan, H. Shah and M. F. Nadeem, "Optimal resource allocation in fog computing for healthcare applications," *Computers, Materials & Continua*, vol. 71, no. 3, pp. 6147–6163, 2022.

[48] K. Brunnström, E. Dima, T. Qureshi, M. Johanson, M. Anderssonv *et al.,* "Latency impact on quality of experience in a virtual reality simulator for remote control of machines," *Signal Processing: Image Communication*, vol. 89, pp. 116005, 2020.
[49] Y. Tu, H. Chen, L. Yan and X. Zhou, "Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT," *Future Internet*, vol. 14, no. 2, pp. 30, 2022.
[50] S. I. AlShathri, S. A. Chelloug and D. S. M. Hassan, "Parallel meta-heuristics for solving dynamic offloading in fog computing mathematics," *Mathematics*, vol. 10, no. 8, pp. 1258, 2022.
[51] "SDN control plane performance: Raising the bar on SDN performance, scalability, and high availability," ONOS project, 2017. [Online]. Available: https://wiki.onosproject.org/download/attachments/13994369/Whitepaper-%20ONOS%20Kingfisher%20release%20performance.pdf?version=1.
[52] V. G. Abhaya, Z. Tari, P. Zeephongsekul and A. Y. Zomaya, "Performance analysis of EDF scheduling in a multi-priority preemptive M/G/1 queue," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2149–2158, 2014.
[53] Ł. Kruk, J. Lehoczky, K. Ramanan and S. Shreve, "Heavy traffic analysis for EDF queues with reneging," *Annals of Applied Probability*, vol. 21, no. 2, pp. 484–545, 2011.
[54] Apache NetBeans 12.5. [Online]. Available: https://netbeans.apache.org/.
[55] D. S. M. Hassan, H. M. A. Fahmy and A. M. Bahaa-ElDin, "RCA: Efficient connected dominated clustering algorithm for mobile ad hoc networks," *Computer Networks*, vol. 75, pp. 177–191, 2014.