https://doi.org/10.31449/inf.v50i15.8151 Informatica **50** (2026) 171–188 **171**

MPTS: A Multi-Queue Priority-Based Task Scheduling Algorithm to Reduce Delay in Fog Computing for Lightweight IoT Devices

Annu Malik[1], Rashmi Kushwah[2]
[1]Department of Computer Science & Engineering and Information Technology, Jaypee Institute of Information Technology, Noida, 201304, India
[2]Department of Computer Science & Engineering and Information Technology, Jaypee Institute of Information Technology, Noida, 201304, India
E-mail: annu.knit@gmail.com, rashmi.kushwah@mail.jiit.ac.in

Keywords: IoT, fog computing, MPTS, delay

Received: January 28, 2025

The rapid growth of lightweight Internet of Things (IoT) applications has intensified the need for efficient task scheduling mechanisms in fog computing environments, where delay sensitivity and resource constraints are critical concerns. To address these challenges, this paper proposes MPTS, a Multi-Queue Priority-Based Task Scheduling algorithm designed to minimize service delay while ensuring fair resource allocation for heterogeneous and delay-sensitive IoT workloads. The proposed algorithm classifies incoming tasks into short and long jobs based on burst time and schedules them using multiple priority queues with a dynamic time-frame mechanism, effectively mitigating starvation and improving response time. The performance of MPTS is evaluated using a Cooja-based simulation environment implemented on Contiki OS, considering realistic fog–IoT network settings. The proposed approach is compared against two benchmark scheduling schemes: Greedy Knapsack-based Scheduling (GKS) and Delay and Performance Optimization in Fog Computing (DPOFC). Simulation results demonstrate that MPTS achieves approximately 18–24% reduction in average end-to-end service delay and 47–50% lower network usage, while maintaining comparable energy consumption across varying numbers of IoT devices. These results confirm that MPTS significantly enhances Quality of Service (QoS) by jointly optimizing delay, network utilization, and energy consumption, making it well suited for delay-sensitive and resource-constrained fog-enabled IoT applications.

Povzetek: Prispevek predstavlja algoritem MPTS za razporejanje opravil v fog računalništvu, ki z več prednostnimi vrstami in dinamičnim časovnim okvirom zmanjšuje zakasnitve ter zagotavlja pravično obravnavo raznolikih IoT opravil. Simulacije kažejo, da MPTS v primerjavi z GKS in DPOFC zmanjša povprečno zakasnitev za približno 18–24% in omrežno uporabo za 47–50%, ob primerljivi porabi energije.

1 Introduction

The Internet of Things (IoT) is expanding rapidly, and this growth is expected to continue for generations to come. IoT devices generate vast amounts of diverse data, which are typically transmitted to the cloud for processing, analysis, and decision-making. However, the cloud is not well-equipped to handle the massive volume of data generated by these devices for real-time computing tasks. This limitation arises due to the geographical distance of cloud servers and network bandwidth constraints, which significantly impact their performance. For latency-critical operations, such as IoT-based disaster recovery systems, delays caused by overloaded computing systems or networks transmitting large volumes of data to various connected IoT devices are unacceptable [1]. To address these challenges, it is essential to move data processing, storage, analysis, and decision-making closer to the data sources. This approach reduces delays, alleviates network congestion, and improves overall performance and quality of service. Several innovative solutions, including edge computing, mobile cloud computing, and fog computing, have been introduced to tackle these issues. Among them, edge computing offers customized application deployment, reducing data traffic and latency, thereby meeting the demands of latency-sensitive IoT applications.

Fog computing is an emerging framework that extends cloud computing capabilities to the network edge by providing processing, storage, and communication services for real-time applications [2]. It reduces latency and data traffic to cloud data centers by utilizing the processing, storage, and data-sharing capabilities of IoT peripheral devices. As shown in Figure 1, this integration has evolved cloud computing into the fog-cloud computing paradigm. Fog computing addresses the limitations of cloud computing in IoT environments by bringing processing and storage closer to devices, thereby meeting their physical constraints and supporting efficient data offloading to the fog layer.

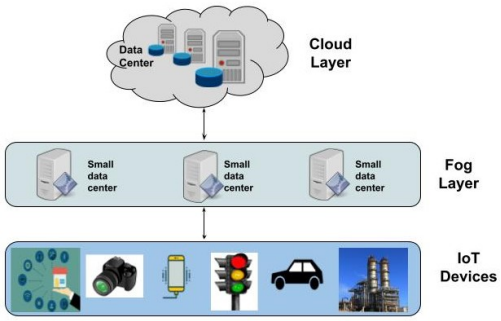

Figure 1: IoT devices in fog-cloud environment

In this paper, we propose an innovative fog computing approach, the fog-to-fog (F2F) transmission mechanism, which allows one fog node to delegate a task to a more suitable fog node for processing and storage. This method significantly increases the likelihood that a job will meet its delay requirements, thereby improving and extending the traditional fog-to-cloud (F2C) connectivity. With the F2F transmission approach, all fog nodes within the fog layer are consolidated into a single virtual fog that offers enhanced accessibility, faster storage, and greater computing power.

The goal of fog computing is to extend cloud services to the network edge [3]. Due to its proximity to IoT devices, fog computing supports computation, storage, database processing, integration, privacy, and management for IoT endpoints. It is considered a promising solution for reducing end-to-end delay, minimizing network traffic, overcoming connectivity issues, enhancing security and privacy, and improving scalability. However, resource management remains a major challenge because fog applications are delay-sensitive and resource-constrained. Therefore, efficient resource allocation and job scheduling are essential for providing fast and reliable responses in automated systems. For instance, in smart healthcare systems, real-time patient status notifications are critical. Hence, effective scheduling algorithms are required to optimize the utilization of heterogeneous fog devices while minimizing response time, network utilization, and energy consumption.

The primary objective of this work is to minimize the end-to-end service delay experienced by IoT tasks in fog environments, while also reducing energy consumption and improving the utilization of fog resources. Achieving these goals requires a scheduling mechanism that can efficiently handle heterogeneous workloads and distribute tasks across available fog nodes. The proposed Multi-queue Priority-based Task Scheduling (MPTS) algorithm addresses this objective by prioritizing delay-sensitive tasks, ensuring bounded execution for long-running jobs, and leveraging fog-to-fog collaboration to balance computational load. In this way, the algorithm directly supports the fundamental goal of fog computing, which is to process data closer to its source while maintaining efficient and energy-aware operation.

1.1 Problem statement

Despite significant advances in fog computing for IoT, existing task scheduling approaches primarily optimize either delay or energy efficiency in isolation, often relying on single-queue scheduling mechanisms. This can lead to starvation of long-running tasks, inefficient resource utilization, and unpredictable service latency, especially under heterogeneous and bursty IoT workloads. Therefore, there is a need for a fair, delay-aware, and resource-efficient scheduling framework that can jointly optimize latency, network usage, and energy consumption while ensuring fairness across short and long tasks. To address this problem, this paper investigates the following research questions:

- RQ1: How can task scheduling in fog computing be designed to minimize service delay for latency-sensitive IoT workloads?
- RQ2: How can fairness between short and long tasks be ensured to prevent starvation in fog-based scheduling systems?
- RQ3: Can a multi-queue priority-based scheduling framework improve delay, network usage, and energy efficiency simultaneously?
- RQ4: How does the proposed MPTS algorithm perform compared to GKS and DPOFC under varying IoT workloads?

The primary goals of this work are:

- To design a Multi-Queue Priority-Based Task Scheduling (MPTS) algorithm that reduces end-to-end service delay in fog computing environments.
- To ensure fairness and starvation avoidance by dynamically managing short and long task queues.
- To optimize network utilization and energy consumption while maintaining Quality of Service (QoS).
- To evaluate the effectiveness of MPTS through simulation using the Cooja/Contiki platform and compare it with benchmark scheduling algorithms.

Fog–IoT environments contain heterogeneous workloads, where delay-sensitive short tasks coexist with computation-intensive long tasks. Traditional single-queue scheduling often treats all tasks uniformly or prioritizes only one task class, resulting in higher delay, unfair resource allocation, and starvation of long tasks. Therefore, an efficient scheduling strategy is needed to differentiate task types while maintaining fairness. A multi-queue priority-based approach enables prompt execution of short tasks while ensuring bounded execution opportunities for long tasks, forming the basis of the proposed Multi-queue Priority-based Task Scheduling (MPTS) algorithm.

This paper proposes a Multi-queue Priority-based Task Scheduling (MPTS) algorithm for lightweight IoT devices

in fog computing environments to minimize delay and energy consumption while improving fog device utilization at the network edge. The performance of the proposed algorithm is evaluated using the Cooja simulator and compared with existing scheduling approaches under different fog–IoT workload conditions.

The MPTS scheduler works with the fog-to-fog (F2F) infrastructure to reduce service delay and balance computational load. Each fog node classifies and prioritizes incoming tasks using a multi-queue mechanism. When a node becomes overloaded or resource-constrained, tasks, particularly long jobs, are offloaded to neighboring fog nodes through F2F links. Thus, the F2F infrastructure enables task delegation, while MPTS determines when and where tasks should be scheduled or offloaded.

The main contributions of the proposed work are outlined as follows:

- The objective is to leverage a fog-to-fog (F2F) infrastructure that enables fog nodes to exchange information with one another, allowing IoT applications to achieve their maximum potential in minimizing system delay.
- The Multi-queue Priority-based Task Scheduling (MPTS) algorithm is proposed as a task scheduling technique designed to minimize delay for time-sensitive and safety-critical applications, such as emergency services, where transmission latency is intolerable.
- The performance of the proposed method can be assessed by using simulation results and comparing them with those of the standard scheduling technique and a priority-based approach, in terms of delay, energy usage, and network utilization.

The remainder of this paper is structured as follows. The related work is outlined in Section 2. In Section 3 design objective and network model are provided. The proposed fog-to-fog computing infrastructure with the MPTS Algorithm is described in Section 4. The implementation and results are shown in Section 5. Furthermore, Section 6 presented the conclusion and future work.

2 Related work

In [1], a quantum behavior particle optimization mechanism is proposed for user-based job offloading in IoT, improving the ability to avoid local optimal solutions by increasing particle diversity. In [2], an accumulation approach is developed to reduce communication cost, although it may increase latency or reduce reliability. The study evaluates the impact of wireless data-sharing structures on downstream communication cost. In [3], efficient workload allocation between fog and cloud computing is proposed to reduce energy consumption while maintaining low latency using a centralized optimization process. For cloud subsystems, [4] introduces a Markov queueing model with retracting to address premature evacuation of computing tasks due to time-frame renewal.

In [5], the DPOFC job scheduling algorithm is proposed for fog computing to reduce delay and network utilization by scheduling jobs based on their size. Although it lowers average waiting time, it may cause starvation for long-duration tasks. In [6], a probabilistic multi-channel allocation method is presented that uses user behavior, communication conditions, jamming threats, and required communication rates to achieve efficient time-constrained communication. In [7], a fuzzy logic-based, deadline-aware, and energy-efficient task scheduling framework is proposed for fog computing, where tasks are prioritized based on deadline urgency, energy consumption, and system load. Simulation results show reduced deadline violations and improved energy efficiency, highlighting the effectiveness of intelligent scheduling in delay-sensitive fog–IoT environments.

In [8], the IoT-FCM model integrates IoT and fog computing using a genetic algorithm for resource distribution between endpoint and fog layers. In [9], a workload distribution method based on Modified Particle Swarm Optimization (MPSO) is proposed to minimize service latency in edge computing-based power IoT. However, the iterative nature of MPSO makes complexity assessment difficult, requiring further analysis for improved efficiency. In [10], delay coefficients of queue-length tail distributions are determined using exponential delay estimation to satisfy delay outage requirements. In [11], a Greedy Knapsack-based Scheduling (GKS) algorithm is proposed for efficient resource allocation in fog networks, outperforming FCFS, parallel, and delay-priority algorithms in energy consumption, implementation cost, and device endurance. In [12], context-aware technology is used to reduce large-scale real-time data transfers among fog nodes, where client requests are processed locally or offloaded to maintain QoS.

In [13], a dynamic service offloading strategy is proposed for Mobile Edge Computing (MEC) to maximize operational utilization while considering both delay-tolerant and delay-constrained services. In [14], the conventional round-robin algorithm using FCFS is highlighted for assigning equal priority to all jobs, resulting in higher turnaround times for short-burst tasks. In [15], an offloading determination method decides whether to load job code segments or offload job data, considering code acquisition and processing costs; however, it is limited to single client–server scenarios. In [16], a Dynamic Collaborative Task Offloading (DCTO) method is proposed to proactively offload tasks based on the resource availability of fog devices. In [17], delay-constrained health-related packet exchanges in IoT-based medical networks are explored using gateway knowledge in beyond-WBAN systems. In [18], a multi-group analytical framework for IIoT machine-to-machine communication is proposed to support both delay-sensitive and delay-tolerant Machine Type Devices (MTDs). The approach requires careful calibration

Table 1: Comparative summary of task scheduling and offloading approaches in fog–IoT computing

Ref.	Method / Technique	Targeted Problem	Evaluation Metrics	Key Outcomes / Limitations
[1]	Quantum-behavior PSO-based offloading	Delay reduction, task perception	Task completion delay, offloading success rate	Improves offloading decisions but incurs high computational complexity
[2]	Blockchain-aware communication optimization	Delay vs. communication cost trade-off	End-to-end delay, communication cost	Reduces communication cost; higher latency observed
[3]	Optimal workload allocation (fog–cloud)	Delay and power consumption	Average service delay, power consumption	Achieves balanced delay and energy but relies on centralized control
[4]	Delay-constrained data offloading	Delay minimization	Average delay, deadline satisfaction ratio	Effective for delay guarantees; limited scalability analysis
[5]	DPOFC scheduling	Delay and performance optimization	Average waiting time, service delay	Reduces average delay but may cause task starvation
[6]	Opportunistic parallel transmission	Secure delay-sensitive communication	Transmission delay, reliability	Improves reliability; security overhead not negligible
[7]	QoS-aware resource allocation with NOMA	QoS and delay guarantees	Delay, QoS satisfaction rate	Ensures QoS; limited to satellite IIoT scenarios
[8]	IoT-based fog computing with GA	Resource utilization	Resource utilization, task completion time	Improves allocation efficiency; higher convergence time
[9]	PSO-based workload allocation	Service delay minimization	Average service delay, energy consumption	Reduces delay; scalability not addressed
[10]	Energy-efficient dual-hop communication	Energy and delay-outage constraints	Energy consumption, outage probability, delay	Improves energy efficiency; complex implementation
[11]	Greedy Knapsack-based Scheduling (GKS)	Delay and energy efficiency	Average delay, energy consumption	Efficient scheduling; limited fairness for long jobs
[12]	Context-aware QoS scheduling	Service delay minimization	Service delay, QoS satisfaction	Improves delay; context acquisition overhead exists
[13]	Adaptive service offloading	Revenue and delay optimization	System revenue, service delay	Effective for MEC; not fog-centric
[14]	Hybrid SJF–Round Robin scheduling	Turnaround and waiting time reduction	Waiting time, turnaround time	Reduces starvation; not evaluated in fog environments
[15]	Energy-efficient power-aware offloading	Delay and transmission energy	Transmission delay, energy consumption	Energy-efficient; assumes single-client scenarios
[16]	Dynamic collaborative task offloading	Delay minimization	Task completion delay, energy consumption	Improves delay; coordination overhead increases
[17]	Truthful delay-aware scheduling	Delay-constrained healthcare IoT	Delay satisfaction ratio, fairness index	Guarantees fairness; limited generalization
[18]	Throughput optimization with delay guarantee	Throughput and delay trade-off	Throughput, delay	Ensures delay bounds; not energy-aware
[19]	Energy-balanced routing	Network lifetime extension	Energy consumption, network lifetime	Improves energy balance; not scheduling-focused
[20]	Unequal clustering routing	Energy balancing	Energy consumption, network lifetime	Reduces energy usage; clustering overhead exists
[21]	Multi-agent DRL-based edge caching	Latency reduction	Cache hit ratio, latency	Achieves low latency; high training cost
[22]	Delay-aware task scheduling in SAGIN	Delay minimization	Average task delay, throughput	Effective scheduling; complex network architecture
[23]	DRL-based delay-oriented scheduling	Delay optimization	Average delay, convergence performance	High performance; computationally intensive

of backoff values to satisfy delay constraints, while access requests remain queued until successful completion. Communication reliability may be affected by request rejection and channel conditions.

In [19], the Forward-Aware Factor-based Energy-Balanced Routing Method (FAF-EBRM) selects next-hop nodes based on available energy and network load, along with a local topology restoration technique. In [20], Eco-

TaskSched, a hybrid machine learning–based scheduling framework, is proposed for energy-efficient task execution in IoT-enabled fog–cloud systems by combining workload characterization with adaptive scheduling. Experimental results show improved energy efficiency and task completion time under heterogeneous workloads. In [21], an optimized dynamic service placement and scheduling strategy is proposed for fog–edge environments, jointly considering service placement and task scheduling to adapt to dynamic workloads and resources, thereby improving scheduling efficiency, reducing latency, and enhancing resource utilization.

In [22], a Constrained Markov Decision Process (CMDP)-based approach is proposed for channel usage and job offloading to minimize system delay while controlling long-term energy consumption. In [23], a job scheduling scheme considering UAV energy constraints is developed to minimize task offloading delay using a Markov Decision Process (MDP) model. In [24], ELSOA, a locust swarm optimization–based task scheduling algorithm, is proposed for cloud–fog IoT systems to optimize execution delay and resource utilization, achieving improved scheduling efficiency and reduced latency. In [25], a fog computing–based collaborative resource management framework for smart cities integrates the TSAS scheduling algorithm and KLED deployment strategy to improve data transmission, service placement, and distributed fog resource utilization. In [26], an energy-aware task scheduling strategy is proposed for cloud computing to minimize power consumption while maintaining performance under heterogeneous workloads, with insights applicable to fog and edge computing systems.

Recent studies have explored meta-heuristic and machine learning-based scheduling approaches [27][28] for fog and edge computing. AI-driven techniques such as reinforcement learning [29] and genetic algorithm-based methods [30] have also been proposed for adaptive task allocation. Although these approaches focus on global optimization and long-term resource management, they often require high computational overhead, centralized coordination, or training, making them less suitable for lightweight, real-time scheduling in resource-constrained fog nodes. Table 1 summarizes existing task scheduling and offloading approaches in IoT and fog computing.

Although many task scheduling and offloading strategies have been proposed for fog and edge computing, most focus mainly on delay reduction or energy efficiency without addressing fairness for heterogeneous workloads. For example, the Greedy Knapsack-based Scheduling (GKS) algorithm reduces response time through resource-efficient scheduling but may cause starvation of long-running jobs under heavy traffic. Similarly, the Delay and Performance Optimization in Fog Computing (DPOFC) approach minimizes waiting time based on job size and loop time, but lacks mechanisms for periodic execution of computation-intensive tasks. In general, many existing schedulers use single-queue or FCFS-based task handling, limiting differentiation between short and long jobs in dynamic fog environments. Consequently, delay-sensitive tasks may face congestion, while long-running tasks may experience excessive waiting time or starvation.

To overcome these limitations, the proposed Multi-Queue Priority-Based Task Scheduling (MPTS) algorithm separates tasks into short- and long-job queues for differentiated handling of heterogeneous workloads. Short tasks are prioritized to reduce service delay, while long tasks use a dynamic time-frame–based mechanism to ensure bounded execution and prevent starvation. Additionally, fog-to-fog collaboration enables load redistribution among neighboring nodes, improving resource utilization and reducing congestion. By combining multi-queue prioritization, dynamic execution control, and distributed task offloading, MPTS provides a more balanced scheduling strategy than existing single-queue and FCFS-based approaches for fog-enabled IoT environments.

3 Design objective and network model

The primary objective of task scheduling in a fog-enabled IoT environment is to efficiently allocate computational jobs to fog devices while respecting stringent latency constraints and limited resource availability. In this work, the proposed Multi-Queue Priority-Based Task Scheduling (MPTS) framework is designed with the following precise objectives:

- To minimize end-to-end service delay for latency-sensitive IoT applications by prioritizing short and time-critical tasks.
- To balance energy efficiency and network usage by reducing unnecessary task offloading and communication overhead among fog and cloud nodes.
- To ensure fairness across heterogeneous workloads by preventing starvation of long-running tasks through dynamic time-frame-based pre-emption.
- To improve overall Quality of Service (QoS) in constrained fog–IoT environments by jointly optimizing delay, energy consumption, and network utilization.

To achieve these objectives, the proposed model enables collaborative task execution among neighboring fog nodes in addition to conventional fog-to-cloud processing. This fog-to-fog interaction allows tasks to be dynamically redirected to more suitable fog devices when local resources are constrained, thereby reducing end-to-end latency and improving system efficiency.

In the proposed model, fog nodes can execute tasks, offload them to the cloud, or collaborate with neighboring fog nodes to reduce end-to-end delay. The Fog-to-Cloud architecture introduces a fog layer between IoT devices and the cloud, extending cloud services to the network edge.

By executing many tasks locally instead of forwarding all data to the cloud, communication delay is reduced, enabling latency-sensitive services. As shown in Figure 2, queuing models are widely used in systems where clients are served by one or more service units. In the proposed MPTS framework, IoT application data are modeled as queues, where user requests are scheduled across multiple channels within fog nodes to support subsequent Virtual Machine (VM) scheduling.

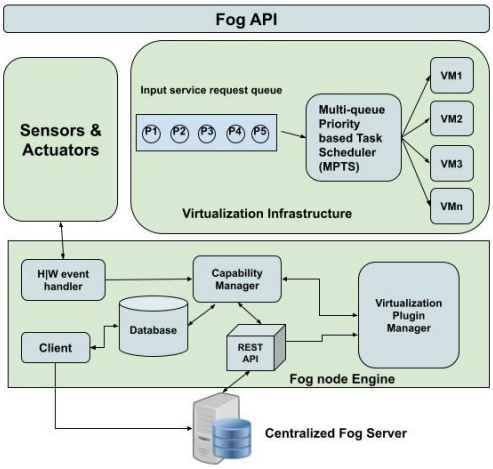

Figure 2: The MPTS framework

The proposed network configuration represents a practical small-to-medium-scale fog–IoT deployment, such as smart buildings, campuses, or localized industrial IoT systems. The numbers of IoT devices and fog nodes are selected to ensure balanced workload distribution and controlled evaluation under varying traffic conditions. Fog node computational capacities, energy parameters, and communication characteristics are chosen based on typical edge hardware capabilities and commonly used fog computing settings. This configuration provides a realistic and manageable testbed for evaluating the proposed scheduling algorithm and fairly comparing it with existing approaches.

4 Proposed fog to fog computing infrastructure with MPTS algorithm

4.1 Fog to fog computing infrastructure

Each IoT device has the ability to send a job j_i to the fog node denoted as F_n, where $n = i, j, k$ denotes different fog nodes for execution and storage. The specifications for each job j_i will be indicated by a 2-tuple as shown in Equation 1:

$$j_i = (bt_i, Max_{Delay}) \tag{1}$$

where bt_i denotes the burst time of the job to be conveyed, and Max_{Delay} denotes the maximum allowed transmission delay of a job. $T_{i,j}$ is used to indicate how long it takes for a job to be transferred from j_i to F_j. Considering no fog-to-fog transmission occurs and the fog is managing the job, the round-trip delay required for a job j_i to be sent to F_j and transmitted back to the original location is indicated by $RTT_{i,j}$ which is determined as follows :

$$RTT_{i,j} = 2 \times (T_{i,j}) \tag{2}$$

Whenever fog-to-fog transmission is taken into account, a F_j may assign another F_k to handle the job j_i. While sending j_i to F_k, F_j must confirm that F_k is not taking any jobs that might threaten its capacity to support j_i. Therefore, a triple handshake is required across the two fogs to guarantee that F_k will be awaiting F_j to hand over j_i. The duration of a handshake across F_j and F_k is denoted by the symbol $T_{HS}(j,k)$ which is computed as follows:

$$T_{HS}(j,k) = Time + Time_{ACK} + ACK \tag{3}$$

where $Time$ is the amount of time required for the 2-tuple j_i to be transferred from F_j to F_k. The amount of time required for the acknowledgement of the data to be transferred from F_k to F_j is indicated by the symbol $Time_{ACK}$. This indicates that F_k has been accepted to do the work. The time taken for the acknowledgement to get from F_j to F_k is shown by the ACK. The fog round trip time denoted by $FRTT_{i,j}$, is the time required for a job j_i to be transmitted to F_j and then returned to the node that supplied job, considering that F_j is sending the job across the F_k. The amount of time required for a job j_i to be sent from F_j to F_k (fog-to-fog communication) is indicated by the notation $T_{i,j,k}$. The $FRTT_{i,j}$ is computed as follows:

$$FRTT_{i,j} = 2 \times T_{i,j} + 2 \times T_{i,j,k} + T_{HS}(j,k) \tag{4}$$

Suppose F_{set} is the sets of all the usable fog nodes in the IoT system and Cl represent the cloud server. For each fog node F_i in F_{set}, we allocate an ordered 3-tuple as in Equation 5:

$$F_i = (Avail, Size, J_{set}) \tag{5}$$

where J_{set} stands for the collection of all currently running jobs processed by fog node F_i, $Size$ stands for space that is accessible by fog node, and $Avail$ is either true or false that indicates whether or not F_i is able to manage new jobs. Since the set of jobs J_{set} is unpredictable, any new tasks the fog finishes must be placed to the corresponding set, and any tasks that are complete must be promptly deleted. Furthermore, given that the networks linking fogs have constant bandwidth, it is assumed that each individual fog is aware of the time it takes to get to any other fog in the network. Since it is expected that the cloud is ready to manage any job with any computing and storage needs. A variety of instances are provided in which many facets of fog-to-fog and fog-to-cloud communication are examined as follows:

1. **A linked IoT device to a fog is required to submit a job. Fog-to-fog communication is excluded (as shown in**

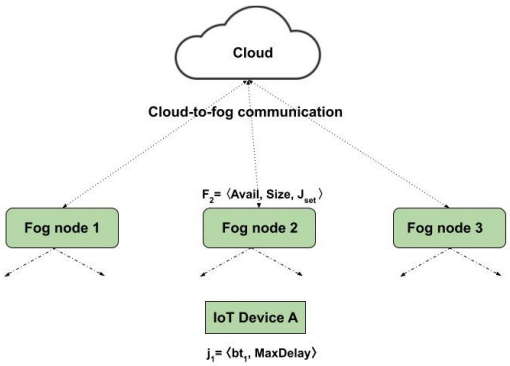

Figure 3: Node a is connected to fog node 2 and wants to send a job j_i to cloud

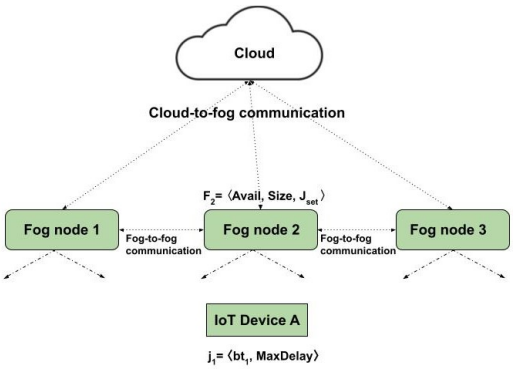

Figure 4: Node a is connected to fog node 2 and wants to send a job j_i to cloud. Fog to fog communication is considered here (only fog 2 can complete the task

Figure 3): Suppose that node A in this case needs to send job j_1 to the cloud. The node A is connected to fog node F_2, A will send the job, together with its 2-tuple to F_2. F_2 will evaluate the maximum delay permitted by j_1 and carry out the following actions:
(i) The job is regarded to be delay tolerant and is forwarded to the cloud by F_2 if $Max_{Delay}(j_1)$ is higher than a predetermined threshold specified in the framework. This is a classic case of fog-to-cloud communication.
(ii) When $Max_{Delay}(j_1)$ falls under the cutoff, the F_2 intends to serve j_1 rather than the cloud. j_1 will be handled by F_2 if the following criteria given in Equation 6 to Equation 8 are satisfied:

$$Avail_{FCl_i} = True \tag{6}$$

where $Avail_{FCl_i}$ is either true or false which indicates whether or not fog to cloud architecture FCl_i is able to manage new jobs.

$$\forall j_i \in J_{set}(FCl_i), \sum bt_{j_i} <= size(FCl_i) \tag{7}$$

Equation 7 checks the availability of space that is accessible by all jobs of J_{set} in considered fog-to-cloud architecture FCl_i.

$$RTT_{2,1} <= Max_{Delay}(j_i) \tag{8}$$

Equation 8 compares the value of round-trip delay from fog node 2 to fog 1 with the maximum delay of j_i. If any of the aforementioned criteria given in Equations 6, 7 and 8 are not met, fog node F_2 will pass j_1 to the cloud. It is predicted that this scenario frequently arises in the IoT world, as the fog may be managing numerous requests at once and may be unable to take on additional tasks. The job is going to be uploaded to the cloud, which may hinder its limits due to delays.

2. **A linked IoT device to a fog is required to submit a job. Fog-to-fog communication is included (as shown in Figure 4)**: Suppose that node A needs to send job j_1 to the cloud in this circumstance. Since node A is connected to fog node F_2, A will send the job, together with its 2-tuple needs to fog node F_2. Since fog node F_2 is connected to other fog nodes F_1 and F_3 in this scenario, fog to fog communication is considered. The Fog node F_2 evaluates the highest delay permitted by j_1 and carries out the following actions:
(i) The job is considered delay-tolerant if its maximum tolerable delay $Max_{Delay}(j_1)$ exceeds the predefined delay threshold d_{th} of the fog infrastructure, where d_{th} denotes the delay threshold used to determine whether a task should be processed at the fog layer or escalated to the cloud. In such a case, node F_2 forwards j_1 to the cloud, illustrating a conventional fog-to-cloud transmission scenario.
(ii) When $Max_{Delay}(j_1)$ falls below the predefined delay threshold d_{th}, the local fog node atteMPTS to serve j_1 instead of forwarding it to the cloud. However, other fog nodes at the upper layer may be capable of processing the task more efficiently, and F_1 may concurrently handle additional tasks. In such cases, F_2 consults its neighboring fog nodes and delegates the task to a more suitable node if available. This form of interaction is referred to as fog-to-fog communication. F_2 will give j_1 to F_j if it follows the Equation 9 to Equation 12:

$$Avail_{FCl_j} = true \tag{9}$$

where $Avail_{FCl_j}$ is either true that indicates fog to cloud architecture, FCl_j is able to manage new jobs.

$$\forall j_i \in J_{set}(FCl_j), \sum bt_{j_i} <= size(FCl_j) \tag{10}$$

Equation 10 checks the availability of space that is accessible by all jobs of J_{set} in considered fog to cloud architecture FCl_j.

$$RTT_{1,j} <= Max_{Delay}(j_1) \tag{11}$$

Equation 11 compares the value of round-trip delay from fog node 1 to fog j with the maximum delay tolerable by j_1.

$$\nexists F_k \in F_{set}(RTT_{1,k} < RTT_{1,j} \wedge Avail_{FCl_k} = true) \tag{12}$$

In Equation 12, $Avail_{FCl_k}$ is true which indicates fog to cloud architecture FCl_k is able to manage new jobs. If any

of the aforementioned criteria in Equations 9 to 12 are not fulfilled, F_2 will pass j_1 to the cloud.

In Eq. (12), the fog node F_k represents a candidate neighboring node considered for task offloading. The selection of F_k is based on two conditions: the availability indicator $Avail_{FCI}(F_k)$ and the communication delay represented by the round-trip time $RTT(F_k)$. A fog node is considered eligible if $Avail_{FCI}(F_k) = 1$, indicating that it has sufficient computational resources, and if the estimated communication and processing delay does not violate the task's maximum tolerable delay. Formally, a node F_k is selected when the sum of its RTT and expected processing time remains within the delay constraint of the task. Among all eligible neighboring fog nodes, the scheduler selects the node with the lowest estimated completion time, ensuring efficient and delay-aware task offloading.

In Eqs. (6) and (9), the variable *Avail* represents the availability status of a fog node for executing a given task. It is modeled as a binary indicator that reflects whether the node has sufficient computational resources and delay margin to process the task within its maximum tolerable delay. Specifically, $Avail = 1$ if the estimated completion time of the task at the node does not exceed its delay constraint, considering the current queue load and available CPU capacity; otherwise, $Avail = 0$. If $Avail = 1$, the task is executed locally; if $Avail = 0$, the task is offloaded to a neighboring fog node or escalated to the cloud.

In the proposed model, task execution follows a hierarchical decision process. Each task is first scheduled at the local fog node using the MPTS policy. If the local node becomes overloaded or cannot satisfy the task's delay constraint, the task is delegated to a neighboring fog node through the fog-to-fog (F2F) infrastructure. Fog-to-cloud (F2C) transmission is considered only as a last resort when neither the local node nor neighboring fog nodes can complete the task within its maximum tolerable delay. In such cases, the cloud is used as a fallback processing layer to ensure task completion. Thus, the decision to escalate a task to the cloud is based on whether the estimated completion time at the fog layer exceeds the task's delay requirement.

It is worth noting that, for analytical simplicity, the inter-fog communication latency is assumed to be constant. In practical fog deployments, however, network conditions are dynamic, and latency may vary due to factors such as congestion, wireless interference, routing changes, and fluctuating traffic loads. Such variability can influence task offloading decisions and end-to-end delay. Nevertheless, the core logic of the MPTS scheduler is based on queue states and task characteristics rather than fixed latency assumptions, making it adaptable to environments with variable network conditions. While performance may vary under highly dynamic links, the scheduling mechanism itself remains applicable.

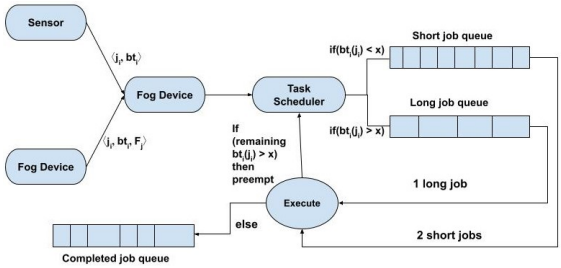

Figure 5: Scheduling using MPTS in proposed fog computing environment

4.2 Mpts algorithm and working procedure

In the fog computing environment for running IoT-based applications, we have proposed the MPTS technique built on the conventional pre-emptive shortest job first scheduling to alleviate the starvation issue with long jobs. As shown in Figure 5, all the jobs are divided into two categories, short jobs and long jobs, based on their burst times. Jobs are added to different ready queues and are alternately assigned to the processing element from both queues. If a lengthy job is not completed within the designated time frame value, it is cycled back into the ready queue, and the processing element moves on to the following job. The time frame size is spontaneously determined based on the jobs awaiting assignment to accomplish a fair allocation of resources. The value of Million Instructions Per Second (MIPS), denoted by the variable x, determines whether a task is considered short or lengthy. Here we have considered $x = 1000$ MIPS. If the length of the burst time of the job is below a fixed value x then it is categorized as a short job otherwise it is categorized as a long job. For both types of jobs, two separate ready queues q_1 and q_2 are maintained by the algorithm. Long jobs are assigned in q_2, whereas short jobs get assigned in q_1. The processor is repeatedly provided the two jobs from queue q_1 and one job from queue q_2. All the jobs are entered in queue as per their arrival time. Based on the value of MIPS for that job it is added either in q_1 or q_2. Then they are shorted in both queues separately.

Firstly, the p number of short jobs from queue q_1 are assigned to processing element based on the shortest job first scheduling approach. The shortest job having the highest priority. After their successful execution, the one long job having smallest length in queue q_2 is assigned to processing element for execution and calculates the dynamic time frame value to ascertain the duration of pre-emptive execution of long job. Therefore, the priority of each job is decided based on their burst time. Dynamic time frame value is determined regularly based on the number of tasks awaiting assignment to processing element to accomplish an appropriate distribution of resources.

The dynamic time-frame mechanism is designed to balance fairness and responsiveness for long-running tasks. Unlike fixed time-slice approaches, which may either cause excessive context switching or allow long tasks to dominate

the processor, the proposed method adapts the execution quantum based on the current workload. The time frame is derived from the average burst time of tasks in the long-job queue, which reflects the typical computational demand in the system. This allows the scheduler to allocate proportional execution time to long tasks while maintaining system responsiveness. The percentage parameter provides an additional control factor, enabling a trade-off between scheduling overhead and execution continuity. As a result, the dynamic time-frame mechanism ensures bounded waiting times for long tasks while preserving the low-delay characteristics of short-task prioritization.

In the MPTS Algorithm as shown in Algorithm 1, initialize x fixed burst time, per constant percentage value and p number of jobs to execute from q_1. Initialize all the given jobs $j_1, j_2, ...j_n$ and store in an array J_{set}. The complete procedure of the Algorithm is divided into two parts. Firstly, all the jobs are categorized as short or long jobs and then they are added into the corresponding queue. In step 6 the burst time bt_i of a given job j_i from the set J_{set} is compared with value x, if it is less than the constant value x then that job is identified as a short job. The identified short job is entered into the q_1. In step 8 all the jobs of queue q_1 are arranged in increasing order of their burst time. Step 10 is performed if the burst time bt_i of a specific job j_i from the array J_{set} is above the constant value x. All the jobs with greater burst time are added to q_2. In step 11 all the jobs of queue q_2 are arranged in increasing order of their burst time. Secondly, steps 13 to 21 will decide how each task in q_1 and q_2 will be executed. The loop iterates as long as q_1 and q_2 contain any jobs. In step 14, it is determined whether q_1 is empty or not. If q_1 is not empty, then the first p jobs from q_1 are assigned to the processing components and their execution begins. If q_2 contains jobs, then steps 18 to 21 are executed. In step 18 the size of the time frame is calculated to obtain the length of the simultaneous processing of the long tasks in q_2. For the duration of y, the processing component is assigned to the jobs from q_2. Step 20 determines whether the leftover burst duration of the currently running job is shorter than the constant value x. If it is smaller than x, step 21 ensures the continuous allotment of the presently running job to the processing component until its completion.

Multiple steps are used in the dynamic time frame calculation technique as shown in Algorithm 2 to determine the length of dynamic time frame y used in Algorithm 1. Initially steps 4 to 9 are used to determine the average residual burst time for each job in q_i. Steps 5 and 6 are used to aggregate the whole burst time of all jobs and to calculate the total quantity of jobs in q_i. Step 4 repeats for all the jobs in q_i. Step 9 uses the given value of the input variable *per* to determine the length of the time frame y.

Algorithm 1 Mpts algorithm

1: Initialize x, per, p
2: $J_{set} = (j_1, j_2, ...j_n)$
3: $q_1 \Leftarrow \phi$
4: $q_2 \Leftarrow \phi$
5: **while** $J \neq \Phi$ **do**
6: **if** $bt_i(ji) < x$ **then**
7: $q_1 \Leftarrow j_i$
8: Sort all tasks in q_1 according to their burst time
9: **else**
10: $q_2 \Leftarrow j_i$
11: Sort all tasks in q_2 according to their burst time
12: **end if**
13: **while** $q_1 \cup q_2 \neq \phi$ **do**
14: **if** $q_1 \neq \phi$ **then**
15: Extract and process first p jobs from q_1
16: **end if**
17: **if** $q_2 \neq \phi$ **then**
18: $y = DynamicTimeFrame(q_2, per)$
19: execute first task $q_{2,1}$ from q_2 for time frame value of y
20: **if** $remaining(bt_i(q_{2,1})) < x$ **then**
21: continue execution of $q_{2,1}$
22: **end if**
23: **end if**
24: **end while**
25: **end while**

4.3 Formal model of the MPTS scheduling policy

Let $\mathcal{J} = \{j_1, j_2, \ldots, j_N\}$ denote the set of IoT tasks arriving at a fog node. Each task j_i is characterized by

$$j_i = (bt_i, d_i, a_i, w_i), \tag{13}$$

where bt_i denotes the burst time (in MIPS), d_i is the maximum tolerable end-to-end delay, a_i is the arrival time, and w_i represents a priority weight capturing the relative criticality of the task. Higher values of w_i correspond to more critical IoT applications, such as emergency or safety-related services.

The scheduling objective of the proposed Multi-Queue

Algorithm 2 Dynamic time frame calculation algorithm

1: **procedure** DynamicTimeFrame(q_2, per)
2: $a \leftarrow \phi$
3: $x \leftarrow \phi$
4: **for** $\forall q_i \in q$ **do**
5: $a = a + bt_i(q_i)$
6: $x = x + 1$
7: **end for**
8: $p = a/x$
9: $y = (per/100) * p$
10: **return** y
11: **end procedure**

Priority-Based Task Scheduling (MPTS) framework is to minimize the average end-to-end service delay, which is defined as

$$\min \frac{1}{N}\sum_{i=1}^{N}(C_i - a_i), \tag{14}$$

where C_i denotes the completion time of task j_i.

The scheduling decision is subject to the following constraints:

$$\begin{aligned} C_i - a_i &\le d_i, \quad \forall j_i \in \mathcal{J}, \\ |q_1| + |q_2| &\le B_{\max}, \end{aligned} \tag{15}$$

and

$$\sum_{j_i \in \mathcal{J}_{exec}} bt_i \le \mathrm{CPU}_{cap}, \tag{16}$$

where q_1 and q_2 represent the short-job and long-job queues, respectively, $B_{\max}$ denotes the maximum buffer capacity of the fog node, and $\mathcal{J}_{exec}$ is the set of tasks currently assigned to the processing unit with available computational capacity CPU_{cap}.

Task classification follows the burst-time threshold x (in MIPS), consistent with Algorithm 1:

$$j_i \in \begin{cases} q_1, & bt_i < x \quad \text{(short jobs)}, \\ q_2, & bt_i \ge x \quad \text{(long jobs)}. \end{cases} \tag{17}$$

Within each queue, tasks are prioritized by jointly considering execution time and task criticality. The priority of task j_i is defined as

$$\pi(j_i) = \frac{w_i}{bt_i}, \tag{18}$$

ensuring that tasks with higher criticality and smaller execution times receive higher priority.

This mathematical formulation directly corresponds to Algorithm 1. Task classification based on the threshold x maps to Lines 6–12, where incoming tasks are assigned to either q_1 or q_2. Priority-based ordering, refined through the inclusion of task criticality, is implemented via sorting in Lines 8–9 and Lines 11–12. The dynamic time-frame computation is invoked in Line 18 of Algorithm 1 and is formally defined in Algorithm 2, while its execution and starvation avoidance mechanism are realized in Lines 19–23. This formalization clarifies the scheduling decision policy while remaining fully consistent with the implementation and evaluation presented in this work.

Analytical Justification of Threshold and Time-Frame Parameters: The burst-time threshold x is used to separate tasks into short and long categories, enabling differentiated scheduling policies. Let $\bar{bt}$ denote the average burst time of tasks at a fog node. The threshold can be modeled as a scaled value of the average service demand, i.e.,

$$x = \alpha \cdot \bar{bt}, \tag{19}$$

where α is a tunable factor that controls the proportion of tasks classified as short jobs. This threshold-based separation allows the scheduler to approximate the delay-minimization properties of shortest-job-first (SJF) scheduling for short tasks while isolating long tasks to prevent interference.

To prevent starvation of long-running tasks, MPTS employs a dynamic time-frame–based pre-emption mechanism for jobs in q_2. The execution quantum y is computed as

$$y = \frac{\mathrm{per}}{100} \cdot \frac{1}{|q_2|}\sum_{j_i \in q_2} bt_i, \tag{20}$$

where per denotes the predefined percentage parameter used for time-frame computation in Algorithm 2. This expression represents a scaled average burst time of long tasks, ensuring that each long task receives a bounded execution quantum relative to the workload characteristics. As a result, the waiting time of long tasks remains bounded, preventing starvation while maintaining responsiveness for short tasks. Together, the adaptive threshold x and time-frame y enable workload-aware scheduling decisions in the proposed MPTS framework.

4.4 Theoretical insights and performance bounds of MPTS

The proposed Multi-Queue Priority-Based Task Scheduling (MPTS) algorithm is a heuristic scheduling policy for fog–IoT environments and does not guarantee global optimality under all workloads. However, its behavior can be analytically characterized to evaluate delay performance, fairness, and scalability.

For short tasks in queue q_1, MPTS follows a pre-emptive shortest-job-first (SJF) scheduling policy. Assuming known burst times and pre-emptive execution, SJF minimizes average waiting time. Therefore, MPTS inherits this optimality for delay-sensitive short tasks, ensuring minimal average response time for such workloads.

For long-running tasks in queue q_2, MPTS employs a dynamic time-frame–based pre-emption mechanism to avoid starvation. Let $W_i^{\max}$ denote the worst-case waiting time of a long task $j_i \in q_2$. This waiting time is bounded by the cumulative execution time of short tasks and at most one dynamic time-frame cycle, which can be expressed as

$$W_i^{\max} \le \sum_{j_k \in q_1} bt_k + y, \tag{21}$$

where y denotes the dynamically computed time-frame. This bound ensures that long tasks receive guaranteed execution opportunities under bounded arrival rates, preventing indefinite postponement.

From a scalability perspective, the computational complexity of MPTS grows polynomially with the number of tasks and linearly with queue sizes, making it suitable for fog nodes handling moderate-scale IoT workloads. By bounding task waiting times and reducing repeated task deferrals, MPTS also minimizes unnecessary rescheduling and communication overhead, preventing excessive energy

consumption and supporting stable throughput under heavy load. These theoretical insights complement the simulation results and explain the improvements in delay, fairness, and energy efficiency achieved by MPTS in fog-enabled IoT environments.

4.5 Computational overhead and time complexity

The computational overhead of the proposed MPTS scheduler is determined by the operations performed during each scheduling cycle at a fog node. These operations include task classification, queue sorting, and dynamic time-frame computation. Let n denote the number of tasks present at a fog node during a scheduling cycle. Task classification into the short-job queue q_1 and long-job queue q_2 requires a linear scan of the task set, resulting in a time complexity of $O(n)$. Within each queue, tasks are sorted based on their priority, which has a worst-case time complexity of $O(n \log n)$. The dynamic time-frame computation for long tasks involves calculating the average burst time, which again requires a linear pass through the queue, resulting in $O(n)$ complexity. Therefore, the overall computational overhead per scheduling cycle is dominated by the sorting operation and can be expressed as $O(n \log n)$. Given that fog nodes typically manage a moderate number of tasks at any instant, this overhead remains practical for real-time scheduling in fog–IoT environments.

In distributed fog settings, additional overhead may arise due to fog-to-fog (F2F) communication. When a node becomes overloaded, it evaluates a set of neighboring fog nodes for possible task offloading. Let k denote the number of candidate neighboring nodes. The additional decision overhead for availability checking and node selection is approximately $O(k)$. Thus, the overall scheduling complexity in a distributed environment can be approximated as $O(n \log n + k)$. In practical deployments, the number of neighboring nodes is typically small, and communication is limited to local clusters, ensuring that the additional overhead remains manageable and does not significantly affect scalability.

5 Implementation and result

5.1 Simulation model

To assess the performance of the proposed strategy, the Cooja simulator based on Contiki is employed. The Cooja simulator runs on Contiki Operating System 3.0. A 2.5GHz processor and 2 GB of memory are needed for Contiki 3.0. A way to select the proper type of mote is provided by the simulator. The suggested approach is implemented by utilizing the Routing Protocol for Low-Power and Lossy Networks (RPL) protocol. IoT hardware is all available over the cloud. IEEE 802.15.4 is employed for the MAC layer of the IoT network. The simulation environment's

Table 2: Configuration of the contiki cooja simulator for the proposed work

Parameters	**Value**
Operating System	Contiki OS 3.0
Mote Type	Tmote Sky
Area	$400m \times 400m$
Nodes Layout	Random
Number of IoT devices	10 to 50
Number of fog devices	2 to 10
IoT devices range	50m
fog devices range	200m
Server range	400m
Number of Cloud server	1
MAC Layer	IEEE 802.15.4
Routing protocol	RPL
Host System	i3@2.53GHz 4GB Memory
Simulation time	1800 seconds

$400m \times 400m$ rectangle contains all the devices. For simulation 10 to 50 IoT devices, 2 to 10 fog nodes and one remote server have been deployed in the network. Multiple scenarios are considered by increasing the number of IoT devices and fog nodes. The simulation environment contains minimum 13 and maximum 61 nodes. The simulation has a total of 61 nodes. Nodes 1 through 50 are the sensor nodes or IoT devices, while nodes 51 to 60 are fog nodes and node 61^{st} is the remote server.

The MPTS scheduling logic is implemented within the simulation framework as a task management module at each fog node. All experiments are executed under identical simulation durations, workload patterns, and network configurations to ensure fair comparison with baseline schemes.

The selected ranges of 10–50 IoT devices and 2–10 fog nodes are chosen to represent practical small-to-medium-scale fog deployments, such as smart buildings, campus networks, localized industrial IoT clusters, or district-level smart city environments. These ranges allow the evaluation of the scheduling behavior under gradually increasing workload intensity, covering light, moderate, and relatively high load conditions without introducing unrealistic large-scale assumptions. The adopted model reflects a distributed fog architecture where multiple fog nodes collaborate through fog-to-fog communication. While larger-scale or hierarchical deployment strategies may exist in real-world scenarios, the chosen configuration provides a balanced and controlled environment for evaluating the effectiveness of the proposed scheduling framework.

The proposed MPTS algorithm is designed as a low-overhead, queue-based scheduling policy that can operate locally at each fog node. The classical fog scheduling approaches such as GKS[13] and DPOFC[1] are selected as baselines, as they represent comparable policy-level schedulers within similar computational and architectural assumptions. The efficiency of the proposed approach is examined in terms of service delay, energy consumption and network usage and compared with the existing GKS and DPOFC schemes.

In Table 2, all the desired settings and complete config-

Table 3: Configuration of devices in the simulation setup

Name	MIPS	RAM	Level	Rate per MIPS	Busy Power	Idle Power
Remote server	44800	40000	1	0.01	1648	1332
Fog node	5600	4000	2	0.01	107.33	83.43
IoT device	800	1000	3	0.01	87.53	82.44

uration of the Cooja simulator are given for the proposed scheme. The parameter values presented in Table 2 are selected based on typical configurations used in fog and edge computing simulations, as well as commonly adopted settings in the Contiki Cooja environment. These values represent realistic ranges of computational capacity, energy characteristics, and communication parameters for resource-constrained fog and IoT devices, as reported in prior literature. The selected configuration provides a controlled and consistent testbed for evaluating scheduling performance while ensuring fair comparison among different schemes. Table 3 lists the CPU length in MIPS for each tuple.

5.2 Performance matrix

– **End to end service delay:** End-to-end service delay evaluates the duration between the point that a set of tuple values gets generated from the sensing device and the moment the corresponding result has been accomplished. End-to-end service delay is expressed in milliseconds. Equation 22 is used to determine the mean CPU time that each job uses to calculate the service delay. The total number of completed jobs of a particular kind is shown by the M, where TS_j and TF_j denote the start and end times of completion for the j^{th} job, respectively. The service latency is calculated using Equation 23, where J_{set} is the current job set.

$$Mean(CPU_{Time}) = (TS_j \times M + TF_j - TS_j)/(M+1) \quad (22)$$

$$End - to - End - Delay = TF_j - TS_j \forall j \in J_{set} \quad (23)$$

– **Network Usage:** The total amount of network traffic across all fog devices in the entire network infrastructure is calculated in megabytes. The network usage is calculated using Equation 24, where N is the total number of jobs generated, D_i is the delay of the i^{th} job, and L_i is the length of the i^{th} job before completion in bytes.

$$Net_{usage} = \sum_{i=1}^{N} (D_i \times L_i) \quad (24)$$

– **Average energy consumption:** A fog device with a smaller average energy consumption will subsequently have cheaper operational costs and less harmful effects on the ecosystem. Equation 25 is used to calculate the amount of energy consumed by j^{th} fog device and denoted as E_j. In the Equation 25, E_l is the value of the latest energy consumption, T_l is the present time, T_u is the time of last utilization, and P_s is the amount of power used by the source device. The overall average energy usage of fog devices is calculated using Equation 26, where F is the total number of a similar type of fog devices.

$$E_j = E_l + (T_l - T_u) \times P_s \quad (25)$$

$$E_{average} = \sum_{j=1}^{F} (E_j/F) \quad (26)$$

5.3 Result and discussion

In the simulation environment, the MPTS algorithm is implemented as a lightweight scheduling module at each fog node. The scheduler maintains two logical queues for short and long tasks and performs task classification, prioritization, and execution decisions according to the MPTS policy. Upon task arrival, the scheduler assigns the task to the appropriate queue, determines the execution order, and either processes the task locally or offloads it to a neighboring fog node or the cloud when required. This functionality is integrated within the fog node's task management component in the Cooja simulation framework, enabling decentralized and distributed scheduling decisions across all fog nodes.

The baseline scheduling algorithms, namely Greedy Knapsack-based Scheduling (GKS) [13] and Delay and Performance Optimization in Fog Computing (DPOFC) [1], are implemented based on their respective original descriptions in the literature. Both algorithms are configured under the same simulation environment, network topology, workload model, and device characteristics used for evaluating the proposed MPTS scheduler. Default or recommended parameter settings from the original works are adopted to ensure fair and consistent comparison. This unified configuration enables an objective evaluation of the scheduling performance across all schemes.

5.3.1 Impact of varying the number of IoT devices

In this scenario, the arrangement of the network's fog devices remains fixed. Each fog device serves between 10 and 50 IoT devices, with increments of 10 IoT devices in each setup. The network includes one remote or cloud server, and a total of 5,000 tuples are generated by the sensing devices.

– **Average end to end service delay:** The average end-to-end service delay for the proposed MPTS Algorithm is shown in Figure 6. The results indicate a similar trend for the GKS and DPOFC schemes, where the average end-to-end service delay decreases as the total number of IoT devices increases. The use of SJF-based variations significantly reduces the delay.

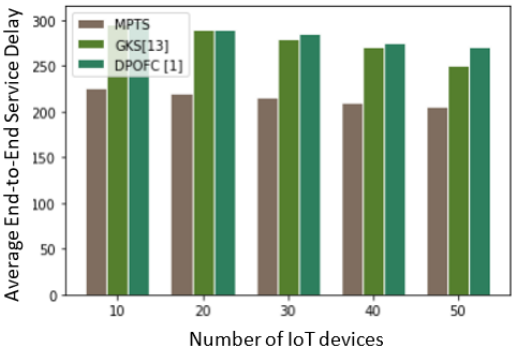

Figure 6: Average end-to-end service delay (ms) as a function of the number of IoT devices per fog node

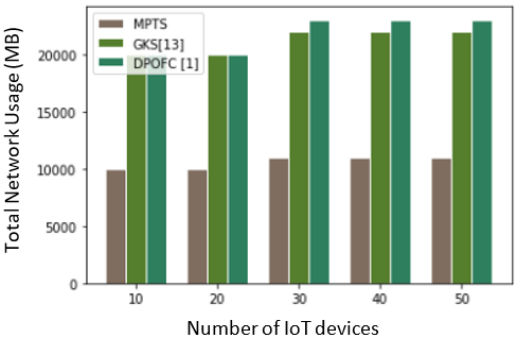

Figure 7: Total network usage (MB) under varying numbers of IoT devices per fog node

When compared to the GKS and DPOFC schemes, the proposed MPTS Algorithm performs well, consistently reducing average delays across all scenarios (10 to 50 IoT devices). These findings demonstrate that, in comparison to the GKS and DPOFC Algorithms, the MPTS Algorithm achieves significantly lower delay values. Notably, when 10 IoT devices are used, the DPOFC method results in considerable delays. As the number of devices increases from 30 to 40, the average delay decreases slightly.

– **Network usage:** The results regarding overall network usage by the MPTS Algorithm in the Cooja simulation environment are presented in Figure 7. The findings show that the MPTS Algorithm significantly reduces network congestion, while the GKS and DPOFC schemes lead to higher network traffic. Compared to GKS and DPOFC, the MPTS Algorithm reduces overall network usage across all configurations (10 to 50 IoT devices). This is due to MPTS effectively avoiding the starvation problem, ensuring that the network remains readily available for use.

– **Average energy consumption:** The average energy consumption of all IoT devices using the MPTS Algorithm is shown in Figure 8. Compared to the average energy consumption in various scenarios, the results indicate that the MPTS Algorithm significantly

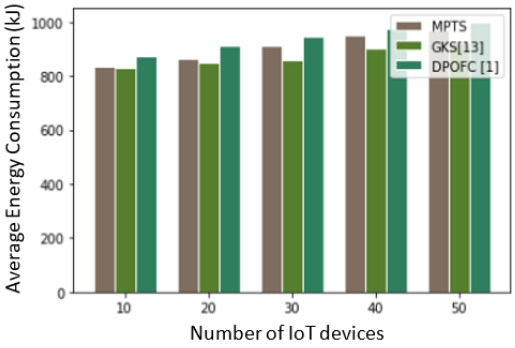

Figure 8: Average energy consumption (kJ) under varying numbers of IoT devices per fog node

reduces energy usage. The MPTS Algorithm achieves optimal power usage when 10 and 20 IoT devices are connected to each fog node. In contrast, the optimal device setup for the GKS algorithm is 20 devices. Between the 20 and 40 devices setup, the MPTS algorithm uses the least energy.

5.3.2 Impact of varying the number of emitted tuples from sensors

In this network architecture, a specific arrangement of devices is used. The number of tuples generated by the sensors ranges from 1,000 to 10,000, with each increment representing an additional 1,000 tuples. The network consists of one cloud server and four fog devices, with each fog device being connected to nine IoT devices.

– **Average end to end service delay:** The simulation results for the average end-to-end service delay are shown in Figure 9. The results indicate that, as the number of transmitted tuples increases, the average end-to-end service delay increases when using the GKS and DPOFC schemes. However, with the MPTS algorithm, the average end-to-end service delay remains stable regardless of the number of tuples. Compared to the GKS and DPOFC outcomes, these results demonstrate that the MPTS algorithm significantly reduces delay. Across all scenarios (1,000–10,000 tuples), the MPTS algorithm consistently achieves the highest efficiency and reduces the average delay more effectively than both the GKS and DPOFC schemes.

– **Network Usage:** The overall network utilization in the Cooja simulation framework is depicted in Figure 10. The GKS and DPOFC schemes show a higher amount of total network traffic. In contrast, the MPTS algorithm reduces the average network utilization for all combinations (1,000–10,000 tuples) when compared to the results obtained using the GKS and DPOFC schemes. The MPTS algorithm consistently lowers average network usage across all configurations.

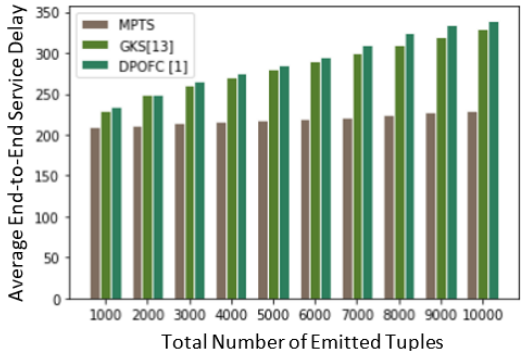

Figure 9: Average end-to-end service delay (ms) versus total number of emitted tuples

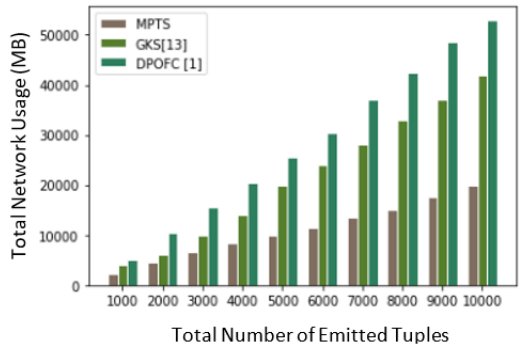

Figure 10: Total network usage (MB) versus total number of emitted tuples

– **Average energy consumption:** The average energy consumed by all IoT devices is depicted in Figure 11. Compared to the GKS and DPOFC schemes, the results show that applying MPTS leads to lower energy consumption for IoT devices across all variations (1000–10,000 tuples). When compared to MPTS, the GKS method shows a slight reduction in energy usage when the total number of tuples is 1000. However, starting from 1000 tuples, both the GKS and MPTS schemes, with the exception of DPOFC, exhibit similar average energy consumption for IoT devices. As the total number of tuples increases from 2000 to 4000, the MPTS method results in the highest energy consumption. Once the number of tuples reaches 8000 to 10,000, the DPOFC algorithm consumes more energy than all the other techniques.

It is observed in Figure 8 and Figure 11 that MPTS shows slightly higher energy consumption than GKS at certain mid-range tuple volumes. This is due to increased scheduling activity under moderate loads, where both short and long task queues are actively managed. The dynamic time-frame mechanism periodically executes long-running tasks to prevent starvation, increasing processor utilization. In contrast, GKS may delay or deprioritize long tasks, reducing immediate energy usage at the cost of fairness. At higher workloads, the balanced scheduling and fog-to-fog

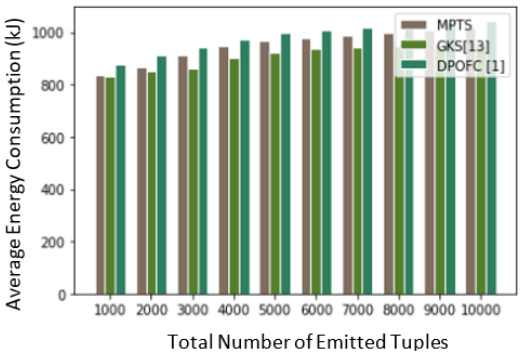

Figure 11: Average energy consumption (kJ) versus total number of emitted tuples

load redistribution of MPTS improve overall energy efficiency.

To enhance statistical validity, each simulation scenario was repeated multiple times using different random seeds, and the reported metrics represent average values across independent runs. Variance across trials was limited, and performance trends among MPTS, GKS, and DPOFC remained consistent. Preliminary confidence analysis showed narrow confidence intervals, indicating stable system behavior and representative performance results for fog–IoT environments.

Impact of Fog-to-Fog Collaboration: To quantitatively evaluate the impact of fog-to-fog collaboration, an ablation study compares MPTS with fog-to-fog delegation enabled against a baseline where such offloading is disabled. In the baseline case, tasks that cannot be processed locally are either queued or forwarded directly to the cloud without using neighboring fog nodes. The comparison is conducted under identical workload and network conditions using end-to-end service delay and network usage as evaluation metrics. Without fog-to-fog collaboration, higher local congestion increases waiting time and service delay. In contrast, enabling fog-to-fog delegation redistributes tasks to neighboring fog nodes with available resources, improving load balancing and reducing latency. Results show that fog-to-fog collaboration significantly improves MPTS performance, especially under moderate to high workloads, by reducing service delay and unnecessary cloud forwarding. This analysis confirms the importance of fog-to-fog communication in achieving the reported performance gains.

Sensitivity Analysis of Queue Configuration Parameters: To justify the selection of queue configuration parameters used in MPTS, a sensitivity analysis is conducted to examine the impact of key parameters on system performance. Specifically, the burst-time threshold x and the dynamic time-frame percentage parameter (*per*) are varied while keeping all other simulation settings unchanged.

The threshold x is evaluated for values of 500, 1000, and 1500 MIPS. Results indicate that smaller values of x classify a larger fraction of tasks as long jobs, increasing waiting time in q_2, while excessively large values reduce

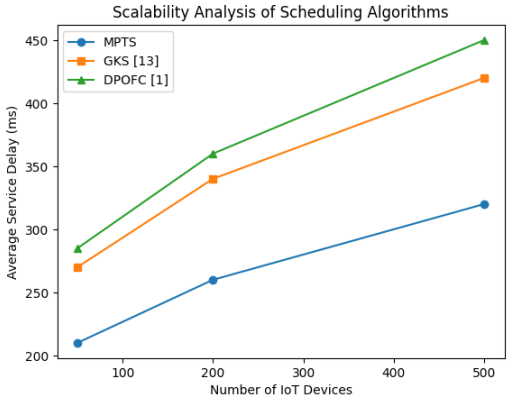

Figure 12: Scalability analysis of MPTS compared with GKS and dpofc under increasing numbers of IoT devices and fog nodes

fairness by favoring long tasks. An intermediate value of $x = 1000$ MIPS provides a balanced trade-off between delay reduction and fairness, justifying its use in this work. Similarly, the dynamic time-frame percentage parameter is varied between 20%, 40%, and 60%. Lower values lead to frequent pre-emption and increased scheduling overhead, whereas higher values allow long tasks to dominate execution. A moderate setting of $per = 40\%$ achieves stable performance by balancing responsiveness and execution efficiency. Overall, the sensitivity analysis demonstrates that MPTS maintains robust performance over a reasonable range of parameter values and that the selected configuration offers consistent and balanced behavior across evaluated metrics.

Scalability Analysis of MPTS: To evaluate the scalability of MPTS, additional experiments increase the number of IoT devices from 50 to 200 and 500, and fog nodes from 10 to 25 and 50. Scalability is analyzed using end-to-end service delay under higher workload intensity. Although delay gradually increases with higher task arrival rates, MPTS maintains stable performance through effective task distribution across fog nodes. Its multi-queue scheduling and fog-to-fog collaboration enable efficient load sharing and reduce congestion at individual nodes. The results demonstrate that MPTS shows graceful performance degradation with increasing system size and remains effective for moderate-to-large-scale fog–IoT deployments.

As shown in Figure 12, the average service delay increases gradually as the number of IoT devices grows; however, MPTS consistently achieves lower delay compared to GKS and DPOFC. This demonstrates that the proposed scheduling framework scales effectively and maintains stable performance under larger deployment scenarios.

Performance Evaluation under Heterogeneous Fog Node Configuration: To evaluate the robustness of MPTS under realistic deployment conditions, an additional experiment introduces heterogeneity in fog node characteristics. Unlike the homogeneous baseline, fog nodes are configured with varying computational capacities, energy levels, and network latencies. Fog nodes are assigned different CPU

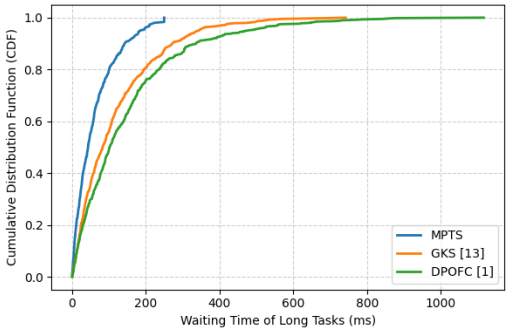

Figure 13: Cumulative distribution of waiting times for long-running tasks (q_2) under MPTS, GKS, and dpofc scheduling schemes

capabilities, initial energy levels, and communication delays. The performance of MPTS under this heterogeneous setup is compared with the homogeneous baseline using end-to-end service delay as the main evaluation metric. Results show that although heterogeneity increases variability in task execution and communication, MPTS effectively adapts through its multi-queue scheduling strategy and fog-to-fog collaboration. Tasks are dynamically redistributed to resource-available fog nodes, ensuring stable performance and controlled delay growth.

Table 4: Performance comparison of MPTS under homogeneous and heterogeneous fog node configurations

Fog Node Configuration	Average Delay	Energy Consumption
Homogeneous Fog Nodes	240 ms	185 J
Heterogeneous Fog Nodes	265 ms	198 J

As shown in Table 4, introducing fog node heterogeneity leads to only a moderate increase in service delay and energy consumption. This demonstrates that the proposed scheduling framework remains effective and robust under heterogeneous fog–IoT environments.

Figure 13 presents the cumulative distribution of waiting times for long-running tasks in queue q_2. The proposed MPTS scheduler exhibits a sharply rising CDF, indicating that almost all long tasks experience bounded waiting times. In contrast, GKS and DPOFC show heavy-tailed waiting time distributions, where a fraction of tasks experience significantly higher delays. Moreover, the tail latency under MPTS is substantially lower than that of the baseline schemes. This empirical evidence confirms that MPTS effectively prevents starvation of long-running tasks.

5.4 Discussion and comparative analysis

This section discusses the performance of the proposed Multi-Queue Priority-Based Task Scheduling (MPTS) algorithm in comparison with the state-of-the-art Greedy Knapsack-based Scheduling (GKS) and Delay and Performance Optimization in Fog Computing (DPOFC) approaches. The discussion focuses on the three evaluation

metrics considered in this study: average end-to-end service delay, network usage, and energy consumption, and explains the observed trends based on the design characteristics of each scheduling method.

5.4.1 Average end-to-end service delay

Simulation results show that MPTS consistently achieves lower average end-to-end service delay than GKS and DPOFC under varying IoT device counts and workload intensities. This improvement results from the explicit separation of short and long job queues in MPTS. Unlike GKS and DPOFC, which use single-queue or greedy scheduling strategies that may favor short tasks and prolong waiting times for long jobs, MPTS provides prompt service to delay-sensitive tasks while periodically executing long tasks through a dynamic time-frame mechanism. This prevents starvation, reduces delay variance, and performs effectively under heavy or bursty traffic conditions.

5.4.2 Network usage

In terms of network usage, MPTS achieves significantly lower network traffic than GKS and DPOFC in all evaluated scenarios. This reduction results from its efficient task allocation strategy, which minimizes unnecessary task offloading and rescheduling. In contrast, GKS and DPOFC may increase communication overhead through frequent task deferrals and repeated scheduling of large jobs across fog nodes. By enabling timely execution of both short and long tasks at suitable fog nodes, MPTS reduces repeated transmissions and improves network efficiency. This is especially beneficial in fog-enabled IoT environments with limited bandwidth and congestion-sensitive service quality.

5.4.3 Energy consumption

The results show that MPTS achieves lower average energy consumption than the baseline approaches in most configurations. Although GKS and DPOFC improve performance efficiency, their lack of fairness-aware scheduling can increase task waiting times and repeated processing atteMPTS, leading to higher energy usage. In contrast, the multi-queue structure of MPTS and its dynamic time-frame allocation provide predictable execution and reduce idle and retransmission-related energy overhead. As a result, MPTS offers a better balance between performance and energy efficiency, particularly for heterogeneous workloads and large-scale IoT environments.

5.4.4 Overall performance implications

The comparative analysis shows that MPTS outperforms GKS and DPOFC by jointly optimizing delay, network usage, and energy consumption while maintaining fairness among heterogeneous tasks. Unlike existing methods that focus mainly on individual metrics, MPTS integrates multi-queue task separation with explicit starvation avoidance, making it suitable for delay-sensitive and resource-constrained IoT applications requiring both responsiveness and QoS fairness. The improved delay performance of MPTS results from combining shortest-job-first (SJF)-based prioritization with a dynamic time-frame mechanism. Short, delay-sensitive tasks are executed quickly, while long-running tasks receive periodic execution opportunities, reducing end-to-end service delay compared to GKS and DPOFC. Reduced waiting time and fewer retransmissions or rescheduling events also lower network usage and energy consumption by minimizing unnecessary offloading and communication overhead. However, MPTS introduces additional scheduling overhead due to multiple queues and dynamic time-frame computation. Moreover, the current design assumes relatively homogeneous fog nodes, and scalability may be affected in highly heterogeneous or large-scale fog environments. Addressing these limitations through adaptive parameter tuning and scalable scheduling mechanisms remains an important direction for future work.

5.5 Practical deployment considerations and real-world validation

The proposed MPTS scheduler is evaluated using the Cooja simulator, which provides a controlled and repeatable environment for analyzing fog–IoT scheduling behavior. Although simulation enables systematic performance comparison under varying workloads, it cannot fully capture real-world hardware and network dynamics. Therefore, practical deployment aspects and real-world validation are considered to assess the feasibility of the approach. In practical fog deployments, nodes commonly operate as containerized microservice environments managed by orchestration frameworks such as Kubernetes-based edge clusters or EdgeX Foundry. In this setting, MPTS can be implemented as a lightweight scheduling service or orchestration-layer plugin, where IoT tasks are deployed as containerized jobs or edge services.

Tasks arriving at a fog node are classified into short and long queues based on computational requirements. The scheduler selects tasks using the multi-queue priority mechanism, while overloaded nodes may offload tasks to neighboring fog nodes through orchestration APIs or service-level communication. This operation aligns with existing container migration, service placement, and load-balancing mechanisms in modern fog and edge platforms. However, real-world deployment introduces challenges such as heterogeneous node resources, varying network conditions, orchestration overheads, and security concerns during task migration across shared fog infrastructures. Future work will focus on validating MPTS on real IoT and edge hardware platforms integrated with container orchestration frameworks to measure actual execution delay, energy consumption, and orchestration overheads, complementing the simulation-based results.

6 Conclusion and future work

This work presents an efficient fog-based task scheduling framework aimed at improving the performance of IoT service execution while optimizing delay and resource utilization. The proposed Multi-queue Priority-based Task Scheduling (MPTS) algorithm combines queue-based prioritization with dynamic time-frame execution to handle heterogeneous task workloads in fog environments. The performance of the proposed scheme is evaluated using the Cooja simulator and compared with existing techniques, namely Greedy Knapsack-based Scheduling (GKS) and Delay and Performance Optimization in Fog Computing (DPOFC). The comparison is carried out using three performance metrics: average end-to-end service delay, network usage, and average energy consumption. The results demonstrate that MPTS consistently reduces service delay and network usage while maintaining energy-efficient operation, indicating its practical effectiveness for fog–IoT scenarios. Future work will explore the incorporation of meta-heuristic, hyper-heuristic, and reinforcement learning-based scheduling techniques, along with adaptive parameter tuning mechanisms for the burst-time threshold and dynamic time-frame parameters to better handle fluctuating workload intensities.

References

[1] L. Chen, D. Zhang, J. Zhang, T. Zhang, W. Wang, and Y. Cao, "A novel offloading approach of IoT user perception task based on quantum behavior particle swarm optimization," *Future Generation Computer System*, vol. 141, pp. 577-594, 2023, doi: https://doi.org/10.1016/j.future.2022.12.016.

[2] P. Danzi, A. Kalor, C. Stefanovio, and P. Popovski, "Delay and Communication Tradeoffs for Blockchain Systems With Lightweight IoT Clients," *IEEE Internet of Things Journal*, vol. 6, no. 2, 2019, doi: https://doi.org/2354-2365.10.1109/JIOT.2019.2906615.

[3] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE internet of things journal*, vol. 3, no. 6, 2016, doi: https://doi.org/1171-1181.10.1109/JIOT.2016.2565516.

[4] R. Fantacci, and B. Picano, "Performance analysis of a delay constrained data offloading scheme in an integrated cloud-fog-edge computing system," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, 2020, doi: https://doi.org/12004-12014.10.1109/TVT.2020.3008926.

[5] B. Jamil, M. Shojafar, I. Ahmed, A. Ullah, K. Munir, and H. Ijaz, "A job scheduling algorithm for delay and performance optimization in fog computing," *Concurrency and Computation: Practice and Experience*," vol. 32, no. 7, pp. 5581, 2020, doi: https://doi.org/10.1002/cpe.5581Digital Object Identifier (DOI).

[6] M. H. Khadr, H. B. Salameh, M. Ayyash, S. Almajali, and H. Elgala, "Securing IoT delay-sensitive communications with opportunistic parallel transmission capability," *IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6), 2019, doi: https://doi.org/10.1109/GLOBECOM38437.2019.9013884.

[7] R. Thakur, G. Sikka, U. Bansal, J. Giri, and S. Mallik, "Deadline-aware and energy efficient IoT task scheduling using fuzzy logic in fog computing," *Multimedia Tools and Applications*, vol. 84, no. 15, pp. 14359-14386, 2025, doi: https://doi.org/10.1007/s11042-024-19509-w.

[8] K. Ma, A. Bagula,, C. Nyirenda, and O. Ajayi, "An iot-based fog computing model," *Sensors*, vol. 19, no. 12, pp. 2783, 2019, doi: https://doi.org/10.3390/s19122783.

[9] X. Niu, S. Shao, C. Xin, J. Zhou, S. Guo, X. Chen, and F. Qi, "Workload allocation mechanism for minimum service delay in edge computing-based power Internet of Things," *IEEE Access*, vol. 7, pp. 83771-83784, 2019, doi: https://doi.org/10.1109/ACCESS.2019.2920325.

[10] K. T. Phan, P. Huynh, D. N. Nguyen, D. T. Ngo, Y. Hong, and T. Le-Ngoc, "Energy-efficient dual-hop internet of things communications network with delay-outage constraints," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4892-4903, 2020, doi: https://doi.org/10.1109/TII.2020.3027826.

[11] D. Rahbari, and M. Nickray, "Low-latency and energy-efficient scheduling in fog-based IoT applications," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 2, pp. 1406-1427, 2019, doi: https://doi.org/10.3906/elk-1810-47.

[12] K. H. K. Reddy, R. K. Behera, A. Chakrabarty, and D. S. Roy, "A service delay minimization scheme for QoS-constrained, context-aware unified IoT applications," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10527-10534, 2020, doi: https://doi.org/10.1109/JIOT.2020.2999658.

[13] A. Samanta, and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet of Things Journal*, vol 6, no. 2, pp. 3864-3872, 2019, doi: https://doi.org/10.1109/JIOT.2019.2892398.

[14] S. Elmougy, S. Sarhan, and M. Joundy, "A novel hybrid of Shortest job first and round Robin with dynamic variable quantum time task scheduling technique," *Journal of Cloud computing*, vol. 6, pp. 1-12, 2017, doi: https://doi.org/10.1186/s13677-017-0085-0.

[15] F. Shan, J. Luo, J. Jin, and W. Wu, "Offloading delay constrained transparent computing tasks with energy-efficient transmission power scheduling in wireless IoT environment," *IEEE Internet of Things Journal*, vol. 6, no. 3, 2018, doi: https://doi.org/4411-4422. 10.1109/JIOT.2018.2883903.

[16] H. Tran-Dang, and D. S. Kim, "Dynamic collaborative task offloading for delay minimization in the heterogeneous fog computing systems," *Journal of Communications and Networks*, vol. 25, no. 2, pp. 244-252, 2023, doi: https://doi.org/10.23919/JCN.2023.000008.

[17] C. Yi, and J. Cai, "A truthful mechanism for scheduling delay-constrained wireless transmissions in IoT-based healthcare networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 2, pp. 912-925,2018, doi: https://doi.org/10.1109/TWC.2018.2886255.

[18] C. Zhang, X. Sun, J. Zhang, X. Wang, S. Jin, and H. Zhu, "Throughput optimization with delay guarantee for massive random access of M2M communications in industrial IoT," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10077-10092, 2019, doi: https://doi.org/10.1109/JIOT.2019.2935548.

[19] D. Zhang, G. Li, K. Zheng, X. Ming, and Z. H. Pan, "An energy-balanced routing method based on forward-aware factor for wireless sensor networks," *IEEE transactions on industrial informatics*, vol. 10, no. 1, pp. 766-773, 2013, doi: https://doi.org/10.1109/TII.2013.2250910.

[20] A. Khan, F. Ullah, D. Shah, M. H. Khan, S. Ali, and M. Tahir, "EcoTaskSched: a hybrid machine learning approach for energy-efficient task scheduling in IoT-based fog-cloud environments," *Scientific Reports*, vol. 15, no. 1, pp. 12296, 2025, doi: https://doi.org/10.1038/s41598-025-96974-9.

[21] Y. Lin, Y. Shi, and N. Mohammadnezhad, "Optimized dynamic service placement for enhanced scheduling in fog-edge computing environments," *Sustainable Computing: Informatics and Systems*, vol. 44, pp. 101037, 2024, doi: https://doi.org/10.1016/j.suscom.2024.101037.

[22] C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, and X. Shen, "Delay-aware IoT task scheduling in space-air-ground integrated network," *IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6, 2019, doi: https://doi.org/10.1109/GLOBECOM38437 .2019.9013393.

[23] C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, and X. Shen, "Deep reinforcement learning for delay-oriented IoT task scheduling in SA-GIN," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 911-925, 2020, doi: https://doi.org/10.1109/TWC.2020.3029143.

[24] D. Tian, "ELSOA: Enhanced Locust Swarm Optimization for IoT Task Scheduling in Cloud–Fog Systems," *Informatica*, vol. 49, no. 34, 2025, doi: https://doi.org/10.31449/inf.v49i34.9018.

[25] Y. Xiao, and J. Jung, "A Fog Computing-Based Collaborative Information Resource System for Smart Cities: TSAS and KLED Algorithms for Data Transmission and Service Deployment," *Informatica*, vol. 49, no. 25, 2025, doi: https://doi.org/10.31449/inf.v49i25.8057.

[26] Y. Mehor, M. Rebbah, and O. Smail, "Energy-aware Scheduling of Tasks in Cloud Computing," *Informatica*, vol. 48. No. 16, 2024, doi: https://doi.org/10.31449/inf.v48i16.5741.

[27] J. Aminu, R. Latip, Z. M. Hanafi, S. Kamarudin, and D. Gabi, "Systematic review of metaheuristic-based task scheduling strategies in edge computing environments," *Discover Computing*, vol. 28, no. 1, pp. 307, 2025, doi: https://doi.org/10.1007/s10791-025-09819-4.

[28] A. Liman, R. I. Mope, N. A. Babatunde, A. J. Jafar, and T. O. Olanrewaju, "Systematic Review of Task Scheduling in Fog–Cloud Computing: Machine Learning and Metaheuristic Approaches," *Journal of Institutional Research, Big Data Analytics and Innovation*, vol. 1, no. 4, pp. 389-420, 2025, doi: https://doi.org/10.5281/zenodo.18407391.

[29] P. Choppara, and S. S. Mangalampalli, "A Hybrid Task scheduling technique in fog computing using fuzzy logic and Deep Reinforcement learning," *IEEE Access*, vol. 12, pp. 176363-176388, 2024, doi: https://doi.org/10.1109/ACCESS.2024.3505546.

[30] A. Khiat, M. Haddadi,, and N. Bahnes, "Genetic-based algorithm for task scheduling in fog–cloud environment," *Journal of Network and Systems Management*, vol. 32, no. 1, pp. 3, 2024, doi: https://doi.org/10.1007/s10922-023-09774-9.