Resource Optimized Task Offloading in Delay Sensitive Edge Networks

Nasif Fahmid Prangon, Abdalaziz Sawwan, and Jie Wu
Center for Networked Computing, Temple University, USA
Emails: {nasifprangon,sawwan, jiewu}@temple.edu

***Abstract*—Task offloading is a popular approach in distributed systems to optimize resource usage by splitting the workload among devices in the network. This technique is effective in reducing the workload on individual devices, and helping achieve more efficient task execution by reducing execution time and conserving energy among devices. In this paper, we propose a novel method to evaluate task partitioning using a unique metric, effective processing rate (EPR) for edge networks with homogenous devices. This unique approach focuses on simplifying task offloading based on the communication delay, task processing rate, and the energy available in each device. It also discusses the impact of extending offloading beyond one-hop neighbors to two-hop neighbors and single-source versus multi-source offloading. We also discuss the willingness to help factor among one-hop and two-hop neighbors and its significance in task offloading performance. An effective strategy to meet these challenges is introduced and validated through theoretical analysis and extensive simulations.**

***Index Terms*—Delay, Edge Networks, Energy Efficiency, Multi-Access Edge Computing, Resource Optimization, Task offloading.**

I. Introduction

Mobile phones, sensors, and communication technologies have become an integral part of modern life in the last decades, thus marking the beginning of the digital era. These devices operate in diverse environments and inherently produce massive amounts of data that are delay-sensitive and need to be processed rapidly. Conventionally, these computations are performed on the cloud servers. However, transmission delay in data and consequently high energy consumption due to the distance between the source of data and the processing unit pose problems [1]. Edge computing addresses this by bringing computational resources closer to users [2].

Multi-access edge computing (MEC) is an extension of edge computing that utilizes operator resources for efficient data processing through wireless transmission [3]. MEC lowers latency and increases energy efficiency by decentralizing the processing. However, decentralized remote task offloading may still lead to inefficiency, especially when the applications require real-time processing.

This research was supported in part by NSF grant SaTC 2310298 for the first author and NSF grant CNS 2107014 for the second author.

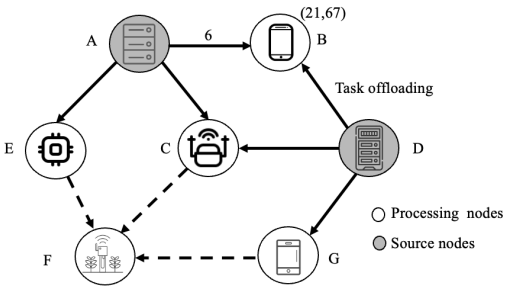


Fig. 1: Task offloading from source to multiple processing nodes, where (r, e) = (21,67) and $d = 6$.

Edge devices suffer from a host of constraints in the form of computation power and battery life, while neighboring resources may remain underutilized. Besides that, edge resources alone might not be enough to meet strict latency requirements. Based on this, we develop a task offloading mechanism between neighboring devices to allow for load sharing and enhance both energy and execution efficiency. Figure 1 shows such an offloading scenario from the source to other neighbors. The contribution of this work is a new offloading strategy, which tries to achieve optimal task processing time in an edge network. The novelty is essentially the dynamic distribution across multi-hop neighbors in order to reduce the processing time and consider energy constraints.

This is a network-based approach in homogenous resource-constrained devices that can be exemplified by applications such as remote environmental monitoring, in which wirelessly connected sensors offload tasks related to data processing to edge servers. Examples include fire-threat assessment and the detection of wildlife movements. The latency sensitivity of these tasks demands a distributed solution wherein the offloading of tasks to multiple edge or cloud servers executes the task with greater speed.

Task offloading involves distributing work among any k-hop neighbors based on the EPR, including the node's delay, its processing capability. These will be divided based on the availability of energy in the neighboring nodes. It is quite simple in the case of one-hop neighbors, but in the case of two-hop neighbors, there are a

number of paths with possibilities of energy constraints. Distribution will need to be effective in case any node is incapable of handling its own fraction of the task in order to maintain the performance of the system. We also investigate how a node's willingness to help dictates the task offloading among one-hop and two-hop neighbors.

II. Background and Related Works

Task offloading in edge computing has recently attracted much attention since many applications call for rapid data processing [4]. Traditional brute-force methods are impracticable to implement in large-scale networks since the complexity is exponential [5]. Therefore, recent research has focused on algorithms that reduce the search space to improve the efficiency, such as branch and bound, which, though improving computation efficiency, still scale poorly when the network size increases [6]. This motivated the investigation of certain heuristic approaches, such as a genetic algorithm, which balances solution optimality and computational burden via an iterative refinement of offloading strategies following the principles of natural evolution [7]. Despite their benefits, these methods generally require parameter fine-tuning and may not always reach global optimality. This calls for faster and simpler approaches.

Task processing has commonly been modeled deterministically, assuming that tasks start processing instantly when they arrive at the edge server [7], [8]. Works such as [8], [9] have developed adaptive offloading strategies that can achieve maximum revenue with ensured service quality by dynamically adjusting decisions according to network conditions and requirements of services. Other works, like [7], have developed a joint optimization of offloading tasks and resource allocation by framing it as a mixed-integer nonlinear programming problem to minimize energy consumption [10]. These methods, although effective, remain NP-hard and hence very computationally expensive

Recent works have targeted efficiency improvement in this direction. For example, [11] has presented an online approach toward jointly optimum network selection and job offloading in multi-dimensional resource-constrained MEC networks, which is near-optimal. Another work [12] modeled the problem of offloading as a generalized allocation model with constraints about hop counts and wireless communication ranges.

Our proposed approach is simpler and faster, enabling quick decisions based on current network states and available resources, avoiding the processing overhead typical of more complex optimization methods. While adaptive approaches like [13] adjust to network conditions and service types, our method puts higher emphasis on speed and simplicity, making it ideal for cases where decision-making needs to be quick, at the possible cost of ignoring fine optimizations brought by more advanced methods in particular instances.

TABLE I: Variables and Parameters

Symbol	Description
N	Set of nodes in the network
G	Directed graph representing the network, $G = (V, E)$
SN	Set of source nodes
PN	Set of processing nodes
T	Total tasks from source in a single-source network
T_s	Total tasks from source s in a multi-source network
r_i	Processing rate of node i
e_i	Energy capacity of node i
A	Task allocation vector
P	Total effective processing power
r_{ij}	Effective processing rate for connection (i, j)
d_{ij}	Total communication delay for connection (i, j)
S	Shortest paths
t_{busy}	Busy time vector for nodes

III. System Model and Problem Formulation

A. System Model

We model the edge network as a directed, connected graph; every node may act either as a source node (SN), in grey, or a processing node (PN), in white as illustrated in Figure 1. Each node presents a communication delay d with its neighborhood, modeled by the edges connecting the nodes. The processing capacity of every node is characterized by its task processing rate r and available energy e. The network is composed of nodes in sets$N$, facilitating task processing by establishing direct and two-hop connections. These connections belong to set S, allowing for the offloading of tasks to neighbors or via intermediate nodes.

B. Problem Formulation

The objective is to develop a task allocation strategy that can minimize the completion time of all tasks given the energy capacity for each node. A summary of the variables and parameters utilized in the system model is provided in Table I. Our approach is formulated as an optimization problem under the following criteria discussed below.

Assuming that the source node has enough energy e_s to process the total task amount T which is a value, all by itself in case its neighbors do not have sufficient energy to process the allocated task. Here energy capacity e_i for any node i is directly calculated by the amount of task the node can handle T_i. For a given total amount of T tasks, we would like to compute a task allocation vector, A, mapping the number of tasks to each node considering processing rate, energy limitations, and communication delays in a way that maximizes the network processing efficiency. The task distribution is done by calculating the EPR for each connection between SN

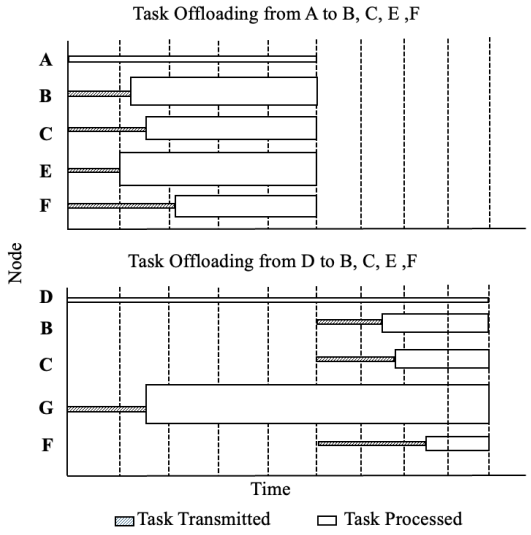


Fig. 2: Multi-source task offloading with shared processing nodes.

and PN. The EPR r_{ij} for connection (i, j) must account for communication delays, the nodes' processing rate, and is given by the equation:

$$r_{ij} = \frac{1}{d_{ij} + \frac{1}{r_j} + t_{\text{busy}_i}}$$

where d_{ij} signifies the cumulative delay over the connection. For multi-hop connections, d_{ij} is the sum of delays over the shortest path. r_j is the processing rate of the processing node j and t_{busy_i} is the time the node remains occupied for multi-source task offloading.

The aim is to minimize the maximum completion time across the network, defined by the optimization problem:

$$\min \max_{(i,j)\in N} \left(\frac{A_{ij}}{r_{ij}} + d_{ij} A_{ij} \right)$$

where $A_i j$ is the allocated task to PN j from SN i.

The allocation strategy is subject to the following constraints:

1) The sum of tasks allocated must not surpass the initial task amount T:

$$\sum_{(i,j)\in N} A_{ij} = T$$

2) The source node must have enough energy to process all tasks by itself:

$$T \leq e_s$$

IV. Algorithm Overview

A. Single-Source Resource Optimized Task Allocation (SSROTA)

The SSROTA algorithm will divide the tasks optimally among the connected nodes in the network such that all nodes finish their respective sub-tasks almost at the same time, minimizing the total execution time. The top part of Figure 2 where node A is the SN divides the total task into four parts based on the EPR of each PN. The white boxes are defined by the task-processing capability of each node. The wider the box, the greater the processing capability. Dashed boxes are used to depict how tasks are propagated from A to the destination nodes B, C, E, and F. All nodes start processing their tasks after receiving the entire allotted portion.

Algorithm 1 Single Source Resource Optimized Task Allocation

Require: N, T, $\{r_i\}$, $\{e_i\}$, $\{d_i\}$, $G = (V, E)$
Ensure: A

```
Initialize P ← 0
Initialize shortest paths S ← Dijkstra's algorithm applied to G
for all connections (i, j) ∈ E do
    Calculate total delay d_ij as the sum of delays over the shortest path from i to j using S
    r_ij ← 1 / (d_ij + 1/r_j)
    Update P ← P + r_ij
while there is remaining task T > 0 do
    for all nodes j ∈ V reachable from source and e_j > 0 do
        T_alloc ← min( (r_ij × T) / P', e_j )
        Update T ← T − T_alloc
        Update e_j ← e_j − T_alloc
        if e_j = 0 then
            Mark node j as inactive and remove from V
    Recalculate P for remaining active nodes by reiterating steps 3 to 6
return Task allocation vector A
```

The algorithm initializes the distribution priority of the nodes, referred to as P, to zero. It calculates an effective processing rate for each connection, considering a direct connection delay between nodes in the case of one-hop connections. For multi-hop connections, the delay is considered along the shortest path starting from the processing node to the source node.

Once the initial effective rate and energy capacities are calculated, it conveys all the tasks proportionally based on the practical connection rate over total priority P. It updates the residual energy at every connection, considering the task allocated there. If a node becomes out of energy, then it marks the node as inactive and recalculates P. Thus, the process keeps repeating until all the tasks are allocated or active connections are left. This way, the SSROTA algorithm ensures that all the nodes complete their tasks at the same time.

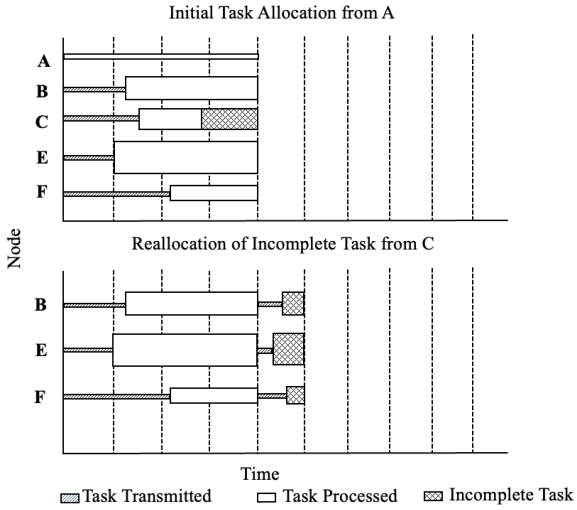


Fig. 3: Task redistribution when node runs out of energy.

B. Multi-Source Resource Optimized Task Allocation (MSROTA)

The MSROTA algorithm efficiently distributes tasks from multiple sources across a network's nodes, ensuring all nodes complete processing simultaneously, reducing overall execution time. Each node i has resources r_i, energy e_i, and delay d_i from the source. Task execution on a PN starts after receiving the full task allocation.

The multi-source algorithm updates EPR in real time based on node utilization, adapting to changes in availability. Once a PN is occupied, it remains locked for other sources until free, and its busy time is included in the delay calculation for subsequent tasks. After all PNs receive their tasks, processing begins, maximizing resource use and minimizing completion time.

For instance, in Figure 2, source D offloads tasks to nodes B, C, and F, shared with source A. These nodes remain locked until completing tasks from A, while node G handles a larger portion from D if it has enough energy. If a node depletes its energy, remaining tasks are redistributed among active nodes, as shown in Figure 3. Initially, node C runs out of energy. The incomplete task is reallocated to nodes B, E, and F based on the recalculated EPR. Depleted nodes are marked inactive, and the process continues until all tasks are allocated or no active nodes remain.

C. Algorithm Extension for Willingness to Help

As an extension to the algorithms, we discuss the impact of willingness to help among neighbors. As we traverse deeper into the network, despite choosing the shortest path, the delay increases between SN and PN. Moreover, the chances of nodes sharing multiple parents or sources increases the delay in multi-hop networks

Algorithm 2 Multi-Source Resource Optimized Task Allocation

Require: N, T_s, $\{r_i\}$, $\{e_i\}$, $\{d_i\}$, $G = (V, E)$
Ensure: A

```
Initialize P ← 0
Initialize shortest paths S ← Dijkstra's algorithm applied to G
Initialize busy time vector t_busy ← 0 for all nodes i ∈ V
for all connections (i, j) ∈ E do
    Calculate total delay d_ij as the sum of delays over the shortest path from i to j using S
    r_ij ← 1 / (d_ij + 1/r_j)
    Update P ← P + r_ij
for all sources s ∈ S do
    while there is remaining task T_s > 0 do
        for all nodes j ∈ V reachable from source s and e_j > 0 do
            Calculate effective processing rate r_ij ← 1 / (d_ij + 1/r_j + T_busy_j)
            T_alloc ← min( (r_ij × T_s) / P', e_j )
            Update T_s ← T_s − T_alloc
            Update e_j ← e_j − T_alloc
            Update T_busy_j ← T_busy_j + T_alloc
            if e_j = 0 then
                Mark node j as inactive and remove from V
        Recalculate P for remaining active nodes by reiterating steps 4 to 7
return Task allocation vector A
```

[14]. All these factors make a node less likely to be effective in task processing, which we call willingness to help. This is a function of the effective processing rate that can be defined as:

$$r_{ij} = \lambda / \left(d_{ij} + 1/r_j\right)$$

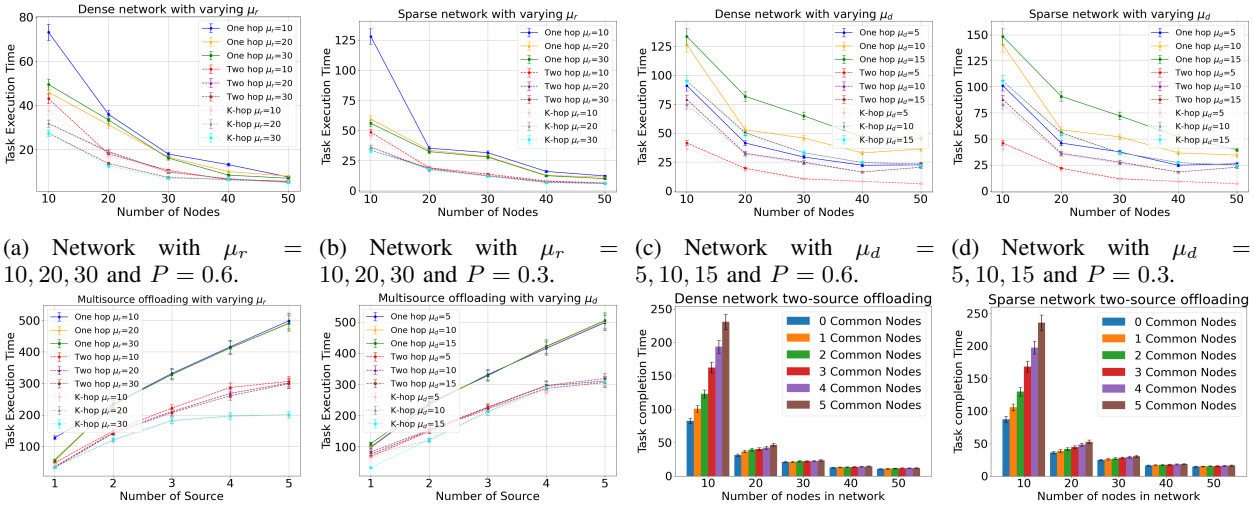
where λ is a scaling factor between 0 and 1 that represents a node's willingness to assist in task completion.

D. Algorithm Analysis

Theorem 1. *The algorithm converges to optimal task allocation for any number of hops.*

Proof: Suppose, for contradiction, that the algorithm does not yield the optimal task allocation. This would imply the existence of another allocation A' with either lower total delay or higher resource utilization than the allocation A produced by the algorithm.

Algorithm 1 starts by initializing the shortest paths S using Dijkstra's algorithm, ensuring the shortest paths from the source to all nodes in graph G. For each



(a) Network with $\mu_r = 10, 20, 30$ and $P = 0.6$. (b) Network with $\mu_r = 10, 20, 30$ and $P = 0.3$. (c) Network with $\mu_d = 5, 10, 15$ and $P = 0.6$. (d) Network with $\mu_d = 5, 10, 15$ and $P = 0.3$.

(e) Network with $\mu_r = 10, 20, 30$. (f) Network with $\mu_d = 5, 10, 15$. (g) Dense network with $P = 0.6$. (h) Sparse network with $P = 0.3$.

Fig. 4: Task offloading performance with a fixed task size $TA = 50$, varying node count, mean delay μ_d, and mean computation power μ_r (a)–(d). Multi-source task offloading with $TA = 50$, $P = 0.5$, and varying μ_r, μ_d (e) and (f). Two-source offloading vs. common nodes and increasing network size (g) and (h).

edge (i, j), it computes the total delay d_{ij} for multi-hop connections. The EPR is then calculated and the total effective resource P is updated. Tasks are allocated to reachable nodes based on:

$$T_{alloc} = \min\left(T \times \frac{r_{ij}}{P}, e_j\right)$$

The allocation is optimal as it minimizes delay and balances resource utilization. Any alternative allocation A' would increase delay or reduce resource efficiency, contradicting the assumption. Therefore, it provides the optimal task allocation for any number of hops.

This can be visualized as dividing water (tasks) among containers (nodes) via pipes, where each container has different capacities (energy) and pipes have different flow rates (delay and resource contribution). The algorithm finds the best path to each container, ensuring efficient flow without spillage (delay). If another method of distributing the water resulted in lower delay or better resource use, it would mean our algorithm missed a more efficient option. However, since our approach continuously chooses the most efficient paths, no better allocation exists, proving it is optimal.

E. Complexity Analysis

For SSROTA, Dijkstra's algorithm initializes the shortest paths with $O(N^2)$ for dense graphs and $O(N \log N)$ for sparse graphs. Calculating delays and processing rates takes $O(E)$, and the task allocation loop adds $O(N^3)$ for dense graphs and $O(N^2)$ for sparse graphs, resulting in overall complexities of $O(N^3)$ (dense) and $O(N^2)$ (sparse).

For MSROTA, the initialization and calculation steps are similar, with an additional outer loop over sources S. This gives overall complexities of $O(S \times N^3)$ for dense graphs and $O(S \times N^2)$ for sparse graphs.

V. Experimental Simulation and Evaluation

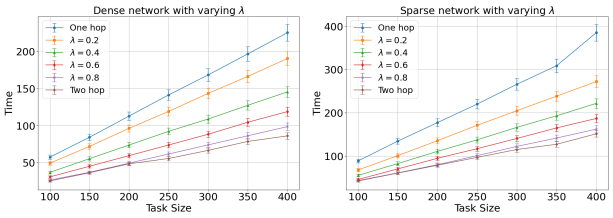
We generated random graphs with 10 to 50 nodes in both dense and sparse configurations using NetworkX. Each node was sampled from uniform distributions: 10-30 for *r*, 1-100 for *e*, and 5-15 for *d*. A 95% confidence interval was calculated for each property to ensure variability.

Edges were generated using the Erdős-Rényi model, with edge probability varied from 0 to 1 to simulate both dense and sparse graphs. For each sample, the total number of edges and their confidence interval were calculated to measure connectivity. The graph's structure was probabilistically determined, and only one-hop and two-hop nodes were considered for task offloading.

A. Single-Source Offloading Analysis

Figure 4a shows task execution times in a dense graph ($T = 50$, density 0.6, $\mu_d = 5$, $\sigma_d = 1$, $\sigma_r = 5$). Execution times decrease as node count increases (10–50) across resource rates ($\mu_r = 10, 20, 30$) for one-hop, two-hop, and k-hop types, demonstrating scalability. For $\mu_r = 10$, one-hop time drops from 73.16 to 7.58 as nodes increase from 10 to 50.

Figure 4b depicts similar trends in a sparse graph (density 0.3), confirming the algorithm's adaptability in reducing execution time across different node counts and resource rates.



(a) Dense network with $P = 0.6$ (b) Sparse network with $P = 0.3$

Fig. 5: Task offloading for one hop, two hop and two hop with $\lambda = 0.2, 0.4, 0.6, 0.8$.

Figure 4c compares execution times for varying delays μ_d, showing that increasing node count reduces times across all delay levels, with multi-hop strategies outperforming one-hop, especially in dense graphs.

Figure 4d reflects consistent results in sparse graphs, reinforcing the algorithm's effectiveness in minimizing execution time for both dense and sparse configurations.

B. Multi-Source Offloading Analysis

Figure 4e and 4f show task execution times for multi-source offloading ($T = 50$, $P = 0.5$) with varying mean delays μ_d and resource rates μ_r. Increasing sources and delays $\mu_d = 5, 10, 15$ lead to higher execution times, especially for one-hop tasks, while k-hop tasks perform better. Similar trends can be seen with resource rates $\mu_r = 10, 20, 30$, where k-hop tasks minimize execution times as resources improve.

Figure 4g compares task completion in a dense network with two-source offloading across varying node counts (10-50) and 0-5 common nodes. Fewer common nodes lead to lower completion times, but as network size grows, execution times decrease due to better load distribution. Fig. 4h shows similar trends in sparse networks, where higher common nodes increase completion times, but larger networks improve load handling.

C. Willingness to Help Analysis

In resource-optimized task allocation, λ is crucial, especially as tasks move further from the source. Figure 5a and 5b show that in dense networks, increasing λ from 0.0 to 1.0 reduces execution times, with two-hop offloading outperforming one-hop. Sparse networks show a similar trend, though with higher overall times due to fewer nodes. Higher λ improves load balancing and execution times, particularly in dense networks with more neighbors.

VI. Conclusion

In this paper, we presented a resource-aware task offloading technique for edge networks using the EPR metric to simplify task partitioning that considers communication delay, processing rate, and node energy. We extended the offloading from one-hop to two-hop neighbors and compared these with the k-hop scenario. Both single-source and multi-source offloading scenarios were analyzed. Simulation results prove that our algorithm reduces task execution time by optimizing task offloading, making it efficient and scalable for dense and sparse networks. The results point to the fact that nodes that lie farther away from the source may not be effective since the delay overhead caused by it is so huge. This method introduces the practicality of two-hop offloading. In general, this research provides a practical and efficient solution for the maximum use of resources and minimum execution time of tasks through optimal offloading.

References

[1] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, and M. Guizani, "Multi-access edge computing: A survey," *IEEE Access*, vol. 8, pp. 197 017–197 046, 2020.

[4] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[5] R. Lin, Z. Zhou, S. Luo, Y. Xiao, X. Wang, S. Wang, and M. Zukerman, "Distributed optimization for computation offloading in edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 8179–8194, 2020.

[6] P. Zhou, K. Shen, N. Kumar, Y. Zhang, M. M. Hassan, and K. Hwang, "Communication-efficient offloading for mobile-edge computing in 5g heterogeneous networks," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 237–10 247, 2020.

[7] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[8] 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.

[9] Y. Chiang, C.-H. Hsu, G.-H. Chen, and H.-Y. Wei, "Deep q-learning-based dynamic network slicing and task offloading in edge network," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 369–384, 2022.

[10] X. Li, L. Zhao, K. Yu, M. Aloqaily, and Y. Jararweh, "A cooperative resource allocation model for iot applications in mobile edge computing," *Computer Communications*, vol. 173, pp. 183–191, 2021.

[11] X. Qi, H. Xu, Z. Ma, and S. Chen, "Joint network selection and task offloading in mobile edge computing," in *Proceedings of the IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 475–482.

[12] C. Chen, Y. Zeng, H. Li, Y. Liu, and S. Wan, "A multihop task offloading decision model in mec-enabled internet of vehicles," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3215–3230, 2022.

[13] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[14] J.-C. Kuo, W. Liao, and T.-C. Hou, "Impact of node density on throughput and delay scaling in multi-hop wireless networks," *IEEE Transactions on Wireless Communications*, vol. 8, no. 10, pp. 5103–5111, 2009.