Resource Optimized Task Offloading in Delay Sensitive Edge Networks

Nasif Fahmid Prangon, Abdalaziz Sawwan, and Jie Wu

Dept. of Computer and Information Sciences Temple University, USA

ICNC 2025

Outline

1. Introduction
2. Problem Statement
3. Background and Motivation
4. Innovative Approach
5. Example Cases
6. Experimental Results
7. Conclusions and Future Work

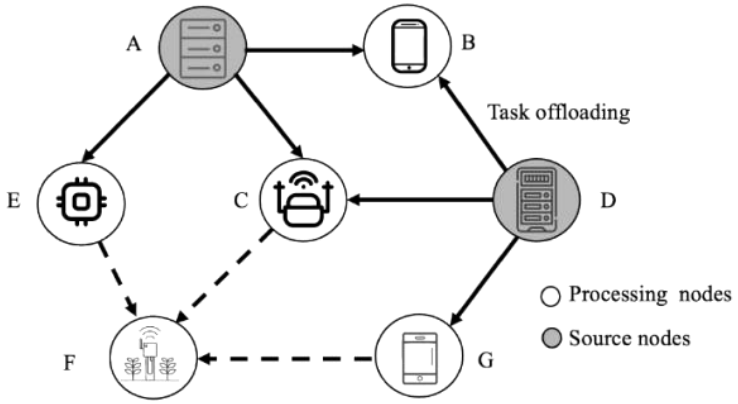
Introduction

- Edge computing brings computation closer to the source of data to reduce latency.
- Task offloading improves resource utilization and reduces execution time.

Challenges: Communication delays, energy constraints, and scalability issues.

Contribution of the Paper:

- A novel Effective Processing Rate (EPR) metric for task allocation.
- Evaluation of multi-hop task offloading performances.



Problem Statement

- Inefficiencies in task offloading due to communication delays, energy constraints, and processing limitations in edge networks.

- Need: A scalable, resource-optimized task offloading mechanism for delay-sensitive applications in edge networks.

- **Objective**

- Minimize task completion time while ensuring all nodes finish at the same time.

$$\text{minimize} \quad \max_{i \in N} \left(\frac{T_i}{P_i} + d_{S,i} \right)$$

Where, T_i is the number of tasks allocated to node i, P_i is the processing rate of node i, and $d_{S,i}$ is the shortest path delay from the source node to node i.

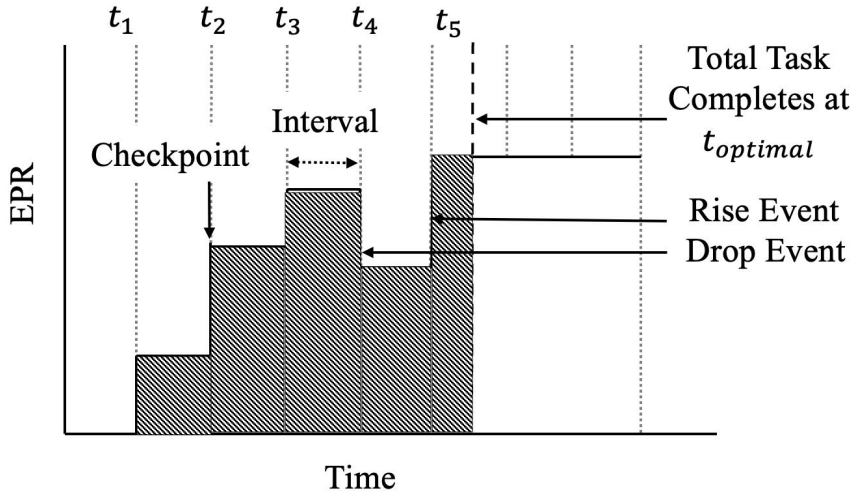
Background & Motivation

- Edge Computing
  - Edge computing minimizes latency by processing data closer to the data source rather than relying solely on centralized cloud servers.
  - Edge devices often have resource constraints like computation power and energy.
- Challenges in Existing Approaches
  - Traditional methods (e.g., brute force) are computationally expensive.
  - Adaptive methods require fine-tuning and often fail to achieve global optimization.
  - State-of-the-art approaches, such as heuristic methods like genetic algorithms and mixed-integer programming, face limitations due to high complexity, scalability issues, and the need for extensive fine-tuning.

Innovative Approach Checkpoint-Based Task Allocation

A **dynamic scheduling mechanism** that partitions execution time into **intervals**, ensuring efficient task distribution.

- **Precomputed Checkpoints**: Defined points where system state updates, based on task arrivals and completions.
- **Rise Events**: When a node becomes active and starts processing tasks at $t_{rise,}$.
- **Drop Events**: When a node stops processing tasks at t_{drop}.
- **Interval-Based Execution**: Tasks are allocated dynamically within each interval $[t_j + t_{j+1}]$, adapting to active nodes.



Innovative Approach (Cont.)

- **Goal**: Identify key time points (**checkpoints**) when nodes **start** and **stop** processing.

- **Steps:**
  - Compute shortest path delays.
  - Determine **rise time** and **drop time** for each node.
  - Sort these times to create a **global event timeline**.

- **Outcome**: A structured **sequence of events** for dynamic task allocation.

Algorithm 1 Precomputing Checkpoints

Require: Directed graph representing the network $G = (N, E)$, delays d_{ij}, lifetimes L_i

Ensure: Sorted list of checkpoints $\{t_1, t_2, \ldots, t_{2|N|}\}$

1: Run Dijkstra's algorithm to compute shortest path delays $d_{S,i}$ for all active nodes
2: **Compute:**
3: **for** each node $i \in N$ **do**
4: $\quad t_{\text{rise},i} \leftarrow d_{S,i}$
5: $\quad t_{\text{drop},i} \leftarrow t_{\text{rise},i} + L_i$
6: Sort the list of events $\{t_{\text{rise},i}, t_{\text{drop},i}\}$ in ascending order to form $\{t_1, t_2, \ldots, t_{2|N|}\}$
7: **return** Sorted list of checkpoints $\{t_1, t_2, \ldots, t_{2|N|}\}$

Innovative Approach (Cont.)

- **Goal**: Dynamically allocate tasks so that all nodes finish at the same time.
- **Steps:**
  - Start with **precomputed checkpoints** and initialize parameters.
  - Iterate through each **time interval** between checkpoints.
  - Update **processing rate** based on active nodes.
  - Compute **task completion** in the interval.
  - Update the **remaining tasks** and move to the next interval.

- **Outcome**: Tasks are distributed **efficiently** while ensuring **synchronized completion**.

Algorithm 2 Task Distribution with Checkpoints

Require: Precomputed checkpoints $\{t_1, t_2, \ldots, t_k\}$, total tasks T_{total}

Ensure: Tasks distributed so all nodes finish at the same time

1: Initialize $j \leftarrow 1$, $T_{\text{remaining}} \leftarrow T_{\text{total}}$, $\mathcal{R}_{\text{eff}} \leftarrow 0$

2: **while** $T_{\text{remaining}} > 0$ **do**

3: Update $\mathcal{R}_{\text{eff}}$ for all available nodes

4: Compute required processing time for that interval:

$$\Delta t = \min\left(t_{j+1} - t_j, \frac{T_{\text{remaining}}}{\mathcal{R}_{\text{eff}}}\right)$$

5: Compute tasks completed:

$$T[t_j, t_j + \Delta t] = \mathcal{R}_{\text{eff}} \times \Delta t$$

6: Update remaining tasks:

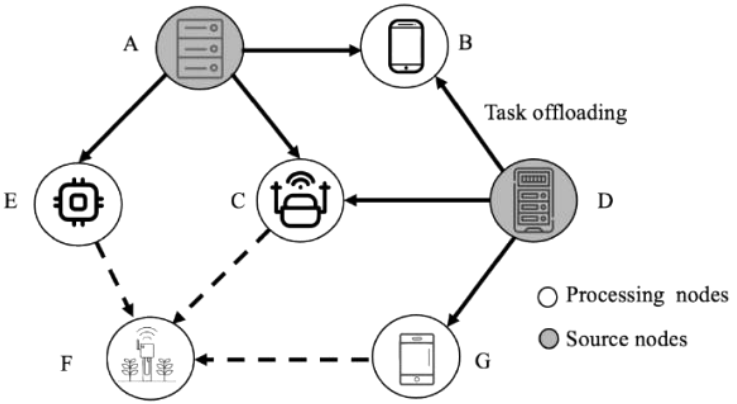
$$T_{\text{remaining}} \leftarrow T_{\text{remaining}} - T[t_j, t_j + \Delta t]$$

7: Move to the next interval: $j \leftarrow j + 1$

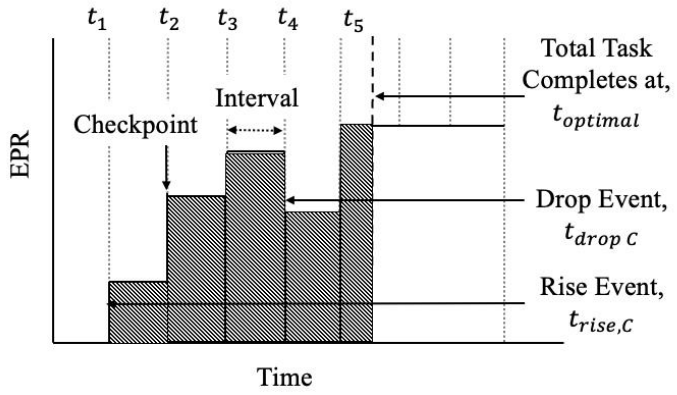
8: **return** Total completion time

Example (Single-source)

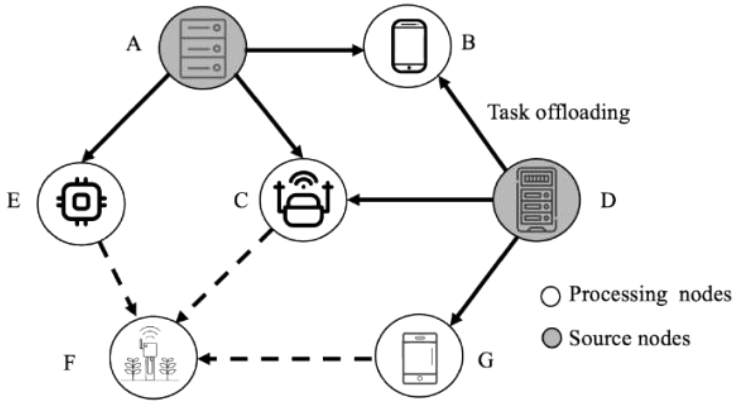
- Node **A** identifies **B, C, and E** as one-hop and **F** as a two-hop neighbor.
- **Checkpoints** are precomputed based on shortest path delays and node lifetimes.



- *C* starts processing at $t_{rise,C}$ and stops at $t_{drop,C}$ completing e.g. 30 task units.
- Tasks are processed dynamically in intervals $[t_j + t_{j+1}]$, with EPR updated as nodes join or leave.
- If *C* dies while processing the task, it is removed, and its remaining tasks are completed locally.



Example (Multi-source)



- Source A offloads tasks to nodes B, C, E, F, while source D offloads to B, C, F, G.
- Shared nodes prioritize tasks based on arrival time with busy time dynamically managed.
- Unique node G absorbs additional tasks to reduce delays and balance the load.
- Task transmission (shaded bars) and processing (unshaded bars) are sequentially managed across nodes.

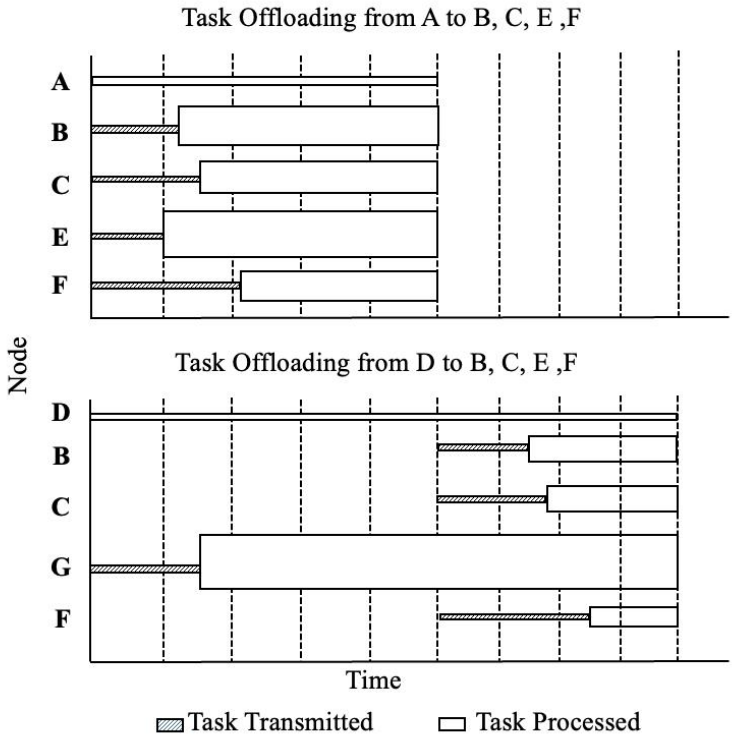


Fig: Multi-source task offloading with shared processing nodes.

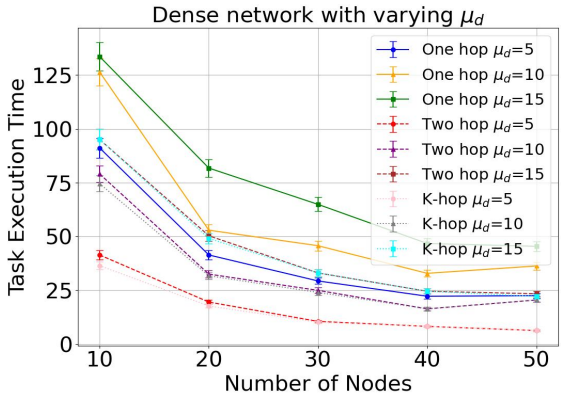
Proof Of Optimality

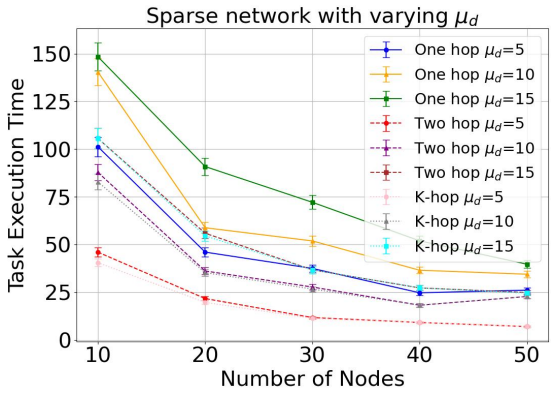
- Suppose an alternative allocation A' achieves a lower completion time or better efficiency than our algorithm's allocation A^* .
- Our algorithm ensures that all **terminally-active nodes** (nodes that remain active until the last processing interval) **finish at the same time**, while making maximum use of available nodes.
- If an alternative allocation A' assigns more tasks to a terminally-active node n_{higher}, its completion time is at least:

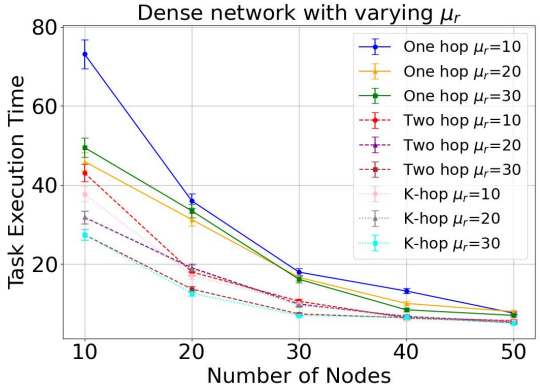
$$t_{\text{higher}} = d_{S,n_{\text{higher}}} + \frac{A'}{P_{n_{\text{higher}}}}$$

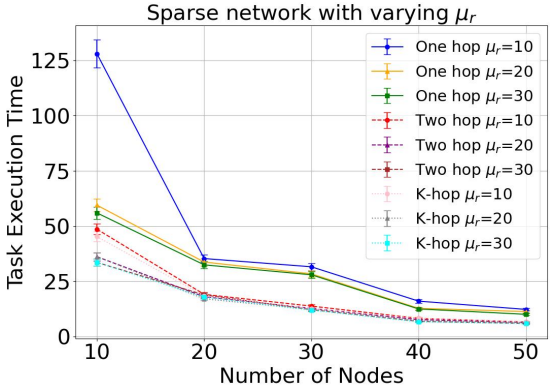
- Since our algorithm ensures all terminally-active nodes complete at t^*_{final} with $A^* < A'$, n_{higher} would take **longer** under A'.

Experimental Results (Multi-hop Offloading)



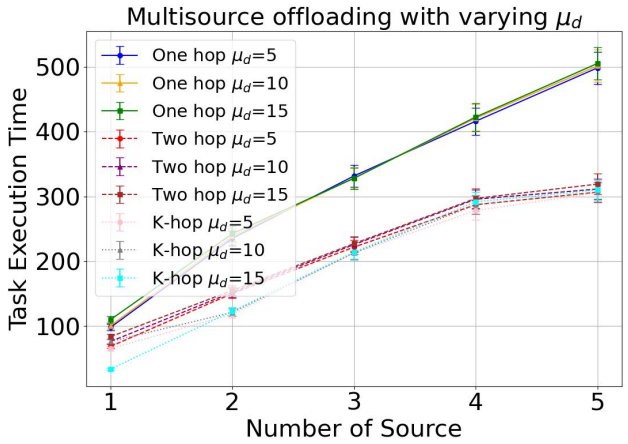


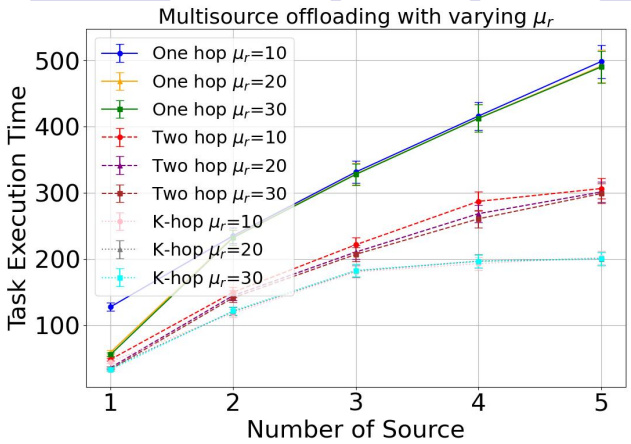




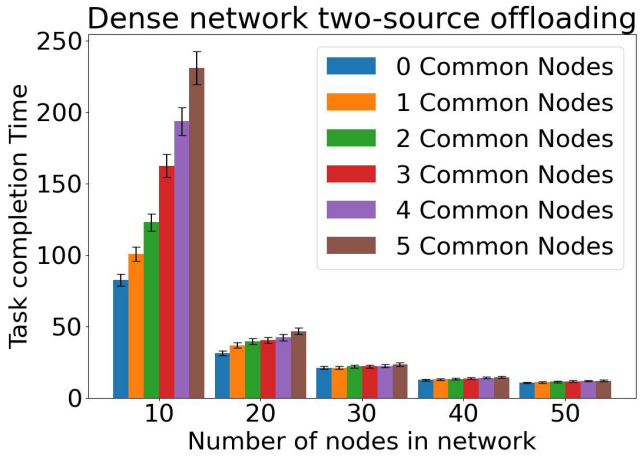
Task offloading performance with a fixed task size TA = 50, varying node count, mean delay μ_d, and mean computation power μ_r

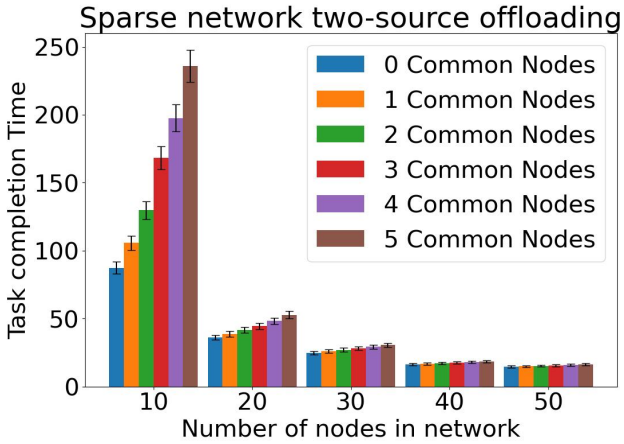
Experimental Results





Multi-source task offloading with Task size = 50, P = 0.5, and varying μr , μd.





Two-source offloading vs. common nodes and increasing network size.

Algorithm Extension for Willingness to Help

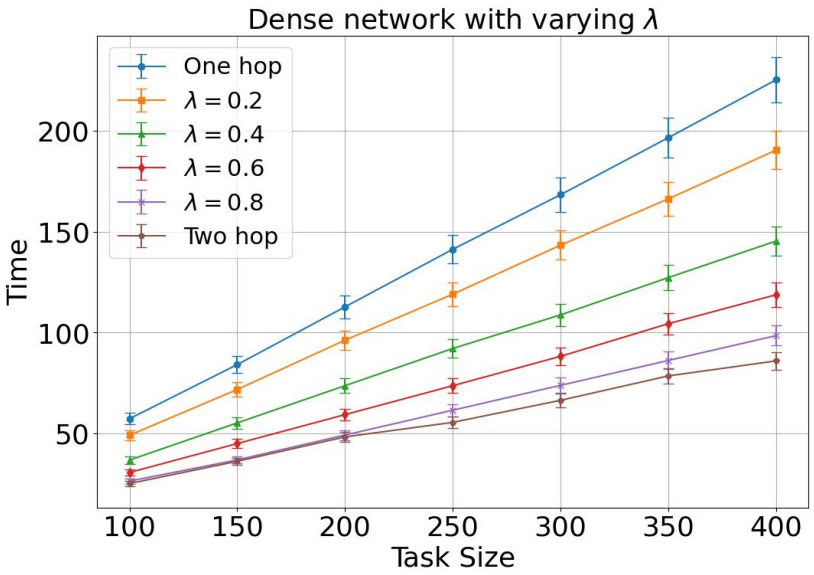
- In multi-hop networks, **willingness to help** reflects a node's likelihood to assist in task processing.
- The EPR metric is adjusted to incorporate a **scaling factor** (λ) representing willingness:

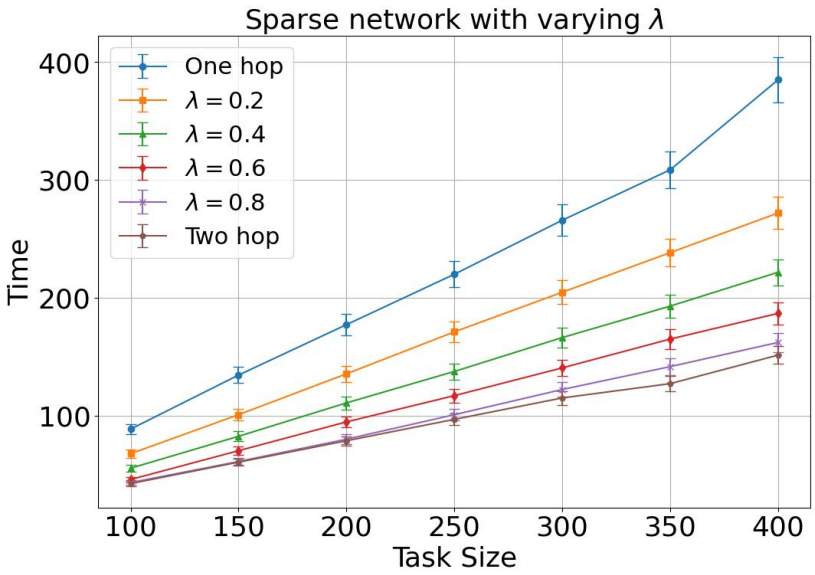
$$R_{\text{eff}} = \sum_{i \in \text{Active Nodes}} \lambda_i \cdot P_i$$

- **Impact of Willingness**:
  - Higher λ values improve task allocation efficiency and reduce overall execution time.
  - Lower λ values represent less effective nodes in processing due to higher delays or shared dependencies.

- **Significance**:
  - Incorporating willingness ensures realistic modeling of node behavior in dynamic edge networks.
  - It highlights the trade-offs between node utilization and task completion time.

Experimental Results





Task offloading for one hop, two hop and two hop with λ = 0.2, 0.4, 0.6, 0.8 .

Conclusion

- Proposed SSROTA is optimal for task offloading in multi-hop single source edge networks.
- Introduced the concept of willingness to help to improve task allocation by dynamically adapting to network conditions.
- Suitable for dynamic and heterogeneous networks, enhancing scalability and adaptability.
- Potential to support real-world applications like IoT and edge-cloud integration.

- **Future Directions**:
  - Extend algorithms to heterogeneous environments with dynamic node mobility and fluctuating resources.
  - Explore machine learning integration for predictive task allocation and network optimization.
  - Investigate further refinements for energy-aware multi-tier networks.

Thank you!
Q & A

tuq24449@temple.edu