Integrate Energy Balancing System into IoT Network Real-time Task Balancing

1st Jianxi Wang
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
j2828wan@uwaterloo.ca

2nd Han Zhang
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
h974zhan@uwaterloo.ca

3rd Yansong Shu
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
p2shu@uwaterloo.ca

***Abstract*—Traditional techniques use cloud computing for data processing, but it could cause significant delays. Recent research suggests an Ant Colony Optimization (ACO) approach for real-time task assignment [5], which may result in uneven battery depletion. This paper proposed an improved energy-balancing system for real-time task-balancing (EBS-RTTB) to overcome the challenges of the trade-off between computational efficiency and energy consumption. This suggested approach integrates state-of-charge (SoC) evaluation into the task-offloading process. It achieves energy balancing by considering energy status in the pheromone update mechanism and node selection process, and prevents any node from rapid energy exhaustion. The simulation experiment compares basic ACO techniques and versions with energy awareness, and demonstrates that this improved strategy prolongs the runtime of the network while maintaining real-time efficiency with just a small increase in response time. This finding highlights the feasibility of integrating energy status into IoT task assignments, paving the way for more robust and sustainable network designs in future developments.**

***Index Terms*—Internet of Things (IoT), Real-Time Task Distribution, Energy Balancing, State-of-Charge (SoC), Ant Colony Optimization (ACO), Pheromone-Based Task Assignment**

I. Introduction

Over the past decades, research on the IoT and distributed computing has produced significant insights for improving real-time task assignment, which indicates the crucial command for the decentralized and efficient task management system. Traditional IoT devices often process data through cloud servers, which may cause serious latency for real-time applications. For instance, in a smart traffic system, cloud computing could result in delays in accident detection, which may reduce the response time for emergencies. Furthermore, since many IoT nodes have battery capacity limitations, overladed certain nodes may eventually exhaust their energy and reduce the overall lifespan of the IoT network.

To overcome the latency issue caused by cloud computing, researchers have utilized edge computing as an alternative. Recent research proposes a real-time task assignment method for IoT networks based on the Ant Colony Optimization (ACO) approach [5]. This approach represents each node as a collaborative ant that leaves "pheromone" trails, which indicates the computational capacity for executing tasks, and allows decentralized work offloading across different IoT devices. By minimizing the usage of cloud infrastructure, the execution latency and reliability of IoT systems are improved.

However, this approach mainly focuses on balancing computing demands and reducing latency without concern about balancing the energy consumption among different IoT nodes, which results in faster energy depletion in a certain device and a shorter lifespan for the entire network. When the energy of even one node is depleted early on, it could cause a series of negative consequences, including loss of coverage when the node is responsible for some crucial sensing tasks, reducing the overall processing capacity for the network, and potentially increasing the latency of other nodes as they have to take extra tasks to compensate the offline device. Furthermore, replacing or recharging a depleted device may need a considerable cost, especially for large-scale IoT systems. As a result, there is a growing demand for ACO-based systems to guarantee the network's efficiency and durability. Integrating state-of-charge (SoC) measurements into the task assignment process will help the system distribute tasks properly, avoiding fast energy depletion of any node, and ultimately prolonging the network's lifetime.

Building on previous work, this study presents an improved task assignment and energy-balancing system to integrate the SoC evaluation into a pheromone-based task offloading process, which enables node selection decisions that take consideration of both processing capabilities and remaining battery percentages. The proposed method is denoted as EBS-RTTB - Energy Balancing System for Real-Time Task Balancing. This study explores the effect of introducing energy evaluation into the ACO-inspired method.

The main objectives of this study are:

1) Integrates an energy balancing mechanism into the task assignment process, which prevents the overload of low battery nodes and increases the lifespan of the IoT system.
2) Maintained the efficiency of the real-time task-distributing system, which allows efficient offloading among different nodes while still meeting the latency requirements of the network.
3) Added state-of-charge (SoC) measurements into the decision-making process to prevent the energy depletion of a single node and protect the system.

4) Conducted simulation experiments to validate the feasibility of this study and demonstrate the improvement of the system's durability and runtime while retaining efficiency.

The rest of this paper is structured as follows: Section II provides background on Ant Colony Optimization and real-time task management, emphasizing essential methods for IoT energy balancing. Section III explains the design and simulation implementation of EBS-RTTB explaining how we integrates SoC measurement into the pheromone model. In Section IV, experimentation is used to analyze the outcomes of this study, while Section V concludes the paper.

II. Related Works

A. Ant Colony Optimization

In WSN, each IoT node must collaborate in an indirect and decentralized manner to achieve complex collaborative tasks, such as real-time task distribution. Such indirect collaboration for an optimized global objective is named "stigmergy". One of the "stigmergy" implementations, named Ant Colony Optimization (ACO), was proposed by Dorigo [1] and is inspired by the way natural ants collaborate to find an optimal path to multiple food sources.

ACO treats each individual as an artificial ant which can emit a pheromone into the environment to attract other individuals to follow along, therefore achieving a collaborative action without involving direct communication. Each pheromone will have different attractive strengths based on environmental factors; for example, when finding the optimal path to the food source, each ant will emit pheromone along their path to the food, and the path with the most ants will naturally accumulate more pheromones and become more attractive.

To apply ACO mathematically, each solution of a problem has its own pheromone which is computed as a numerical value, and the solution with larger pheromone value is the more possible one to be selected as the final solution. The way to compute pheromone in ACO depends on the problem definitions. Several ACO-inspired IoT researches, including [2]–[4], has been proved to be effective in optimizing task offloading, load balancing, and network resource utilization.

B. Pheromone-Inspired Real-Time Task Balancing Algorithm

Real-time task balancing can be considered as an optimization problem, where each solution is a decision from the task manager that identifies which node each execution task should be assigned to for execution so that the total latency is minimized. With this problem definition, Wilson [2] applied the pheromone computation into the problem and proposed Pheromone-Inspired Real-Time Task Balancing (pi-RTTB) algorithm.

The architecture of pi-RTTB can be visualized in Fig. 1. As shown in the architecture, pi-RTTB has a MESH topology network, where each node n_i is connected to the task manager which can be any node from n_1 to n_i, meaning any node can be the task manager. The task manager is an IoT node specialized for receiving tasks, denoted as the set $\mathcal{F}$ from nodes and assign tasks to the most suitable node for execution. The entire process involves three steps - node initialization, task assignment handshake, and pheromone update.

1) Node Initialization: Node initialization involves computing the initial pheromone of each node connected to the task manager, denoted as P_0. The formula for P_0 is shown in (1):

$$P_0 = \frac{K_\alpha \times PC}{IC \times CPI \times CT} \tag{1}$$

where PC is the number of processing cores, IC is the total number of instructions executed when running the benchmark programs, CPI is the average clock cycles for executing a single instruction, and CT is the elapsed time per clock cycle. The value for $IC \times CPI \times CT$ is extracted by running several benchmark programs and therefore we can simplify P_0 based on the total execution time of the benchmark programs, as shown in (2):

$$P_0 = \frac{K_\alpha \times PC}{T_{benchmark}} \tag{2}$$

where $T_{benchmark}$ is the total execution time of the benchmark programs in seconds. K_α is computed to capture other factors that also affect the computational efficiency of the node, as shown in (3):

$$K_\alpha = \frac{[N\sum(xy) - \sum x \sum y]}{[N\sum(x^2) - (\sum x)^2]} \tag{3}$$

where N is the number of benchmark programs, x is the expected threshold completion time for each program, and y is the actual completion time for each program.

2) Task Assignment Handshake: After nodes' pheromones are initialized, the task manager will start assigning tasks to nodes. This process involves a 3-way handshake (highlighted as red, blue, and green in Fig. 1):

- **TTRC**: the task manager will broadcast the task as a function package named Turnaround Time Resolution Call (TTRC). Each TTRC package, similar to Function-as-a-Service (FaaS) model structure [6], includes the body of the function, its return type, size of inputs, and Turnaround Time Constraint (TTC) as a threshold.
- **RTRC_ACK**: when a node receives a TTRC package, it will compute its Estimated Turnaround Time (ETT) based on the approximated instruction counts to execute the task based on the function body, and if $ETT < TTC$, the node will respond to the task manager with an RTRC_ACK package including its MAC and IP address, its ETT value, and its pheromone value.
- **Task Assignment**: after the task manager receives RTRC_ACK from all nodes with $ETT < TTC$, it will select the node to execute the task using Roulette Wheel [7], and the probability of selecting node i for the task j is computed in (4):

$$p_{select,i,j} = \frac{(P_{i,t(j)})^\alpha (ETT_{i,j})^{-\beta}}{\sum_{i=1}^{N} (P_{i,t(j)})^\alpha (ETT_{i,j})^{-\beta}} \tag{4}$$

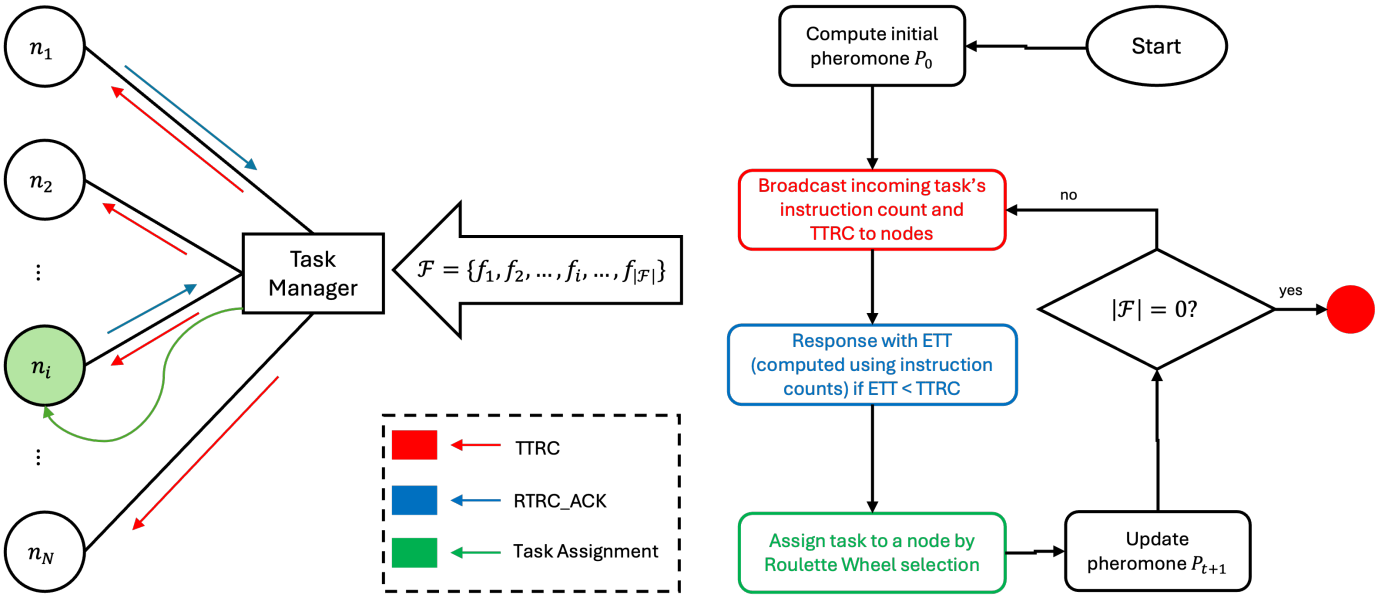

Fig. 1. Architecture and Flow Diagram for pi-RTTB. EBS-RTTB also inherit this structure with modifications in pheromone computation and Roulette Wheel selection.

where $P_{i,t(j)}$ is the pheromone value of node i at time step t when task j is received, $ETT_{i,j}$ is node i's ETT value on task j, and α and β are parameters deciding the importance balancing between pheromone and ETT. With Roulette Wheel selection, all nodes are possible to be selected so that starvation is avoided.

3) Pheromone Update: After the node executes a task and sends the output back to the task manager, it records the actual execution time and updates its pheromone value as shown in (5):

$$P_{t+1} = \frac{P_0}{\max(R_a - R_e, 0) + 1} \tag{5}$$

where R_a is the actual execution time and $R_e = ETT$. The intuition is that if the actual execution time is longer than the expected execution time, then a penalization should be added to the node based on the amount of delay by scaling P_0 down, and if $R_a < R_e$ then no penalization will be applied.

C. Energy Efficient Computing for IoT

Several researches have been made to integrate efficient energy usage in the IoT network to extend the network lifespan. For example, Jannu et al. [8] proposed an energy-efficient cluster head selection strategy that minimizes routing energy consumption by considering residual energy, intra-cluster distance, and distance to the base station in their quantum-informed ant colony optimization algorithm. Similarly, Ruixin et al. [9] proposed an energy-efficient virtual machine scheduling algorithm that balances IoT task loads by predicting request arrivals and adaptively optimizing energy usage while maintaining service quality.

To design an energy-efficient solution, some researchers use State of Charge (SoC) which gives an approximation of the battery percentage of an IoT node. There are many ways to estimate SoC, and one common estimation strategy used is Columb Counting [10], as shown in (6):

$$SoC(t) = SoC(t_0) - \int_{t_0}^{t} \frac{\eta(\tau) I(\tau)}{C} \, \mathrm{d}\tau \tag{6}$$

where $\eta(\tau)$ is the charging efficiency $I(\tau)$ is the current at time t in hours, and C is the battery capacity. Assuming η and I are constants and $t_0 = 0$, we can further simplify SoC into a linear model, as shown in (7):

$$SoC(t) = \max(SoC_0 - At, 0) \tag{7}$$

where SoC_0 is the initial SoC and $A = \frac{\eta I}{C}$ is the constant coefficient.

Recent research by Jeong [11] has developed a lightweight, ultra-low power circuit for SoC measurement in miniature IoT batteries. This breakthrough enables the design of energy-aware systems that can rely on SoC measurements without significant concern about the additional energy overhead traditionally associated with SoC estimation.

III. Methodology

A. EBS-RTTB Design

The design of EBS-RTTB is an extension of pi-RTTB [5] with consideration of energy efficiency. EBS-RTTB modified two components - pheromone computation and Roulette Wheel selection.

1) Pheromone Computation for EBS-RTTB: To integrate energy efficiency into pi-RTTB, EBS-RTTB adds SoC to the pheromone computation. The modified initial pheromone, denoted as P_0', is shown in (8):

$$P_0' = P_0 \times SoC_0 \tag{8}$$

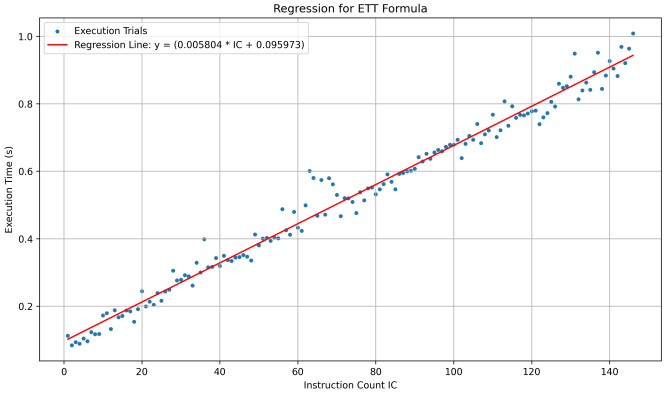

Fig. 2. Regression visualization for $R_e = ETT$.

where P_0 is computed in (2) and SoC_0 is state-of-charge in percentage extracted from the power circuit introduced by Jeong [11] when the node is initialized. For simulation purpose, $SoC(t+1)$ is computed using (7). The updated pheromone after executing a task is modified as shown in (9):

$$P'_{t+1} = \frac{P'_0}{\omega_1 \max(R_a - R_e, 0) + \omega_2(1 - SoC(t+1)) + 1} \tag{9}$$

where ω_1 and ω_2 are tunable weights deciding the importance of $R_a - R_e$ and SoC in pheromone computation and $SoC(t+1)$ is the SoC value in percentage at time $t+1$. In our experiments, we set $\omega_1 = 1$ and $\omega_2 = 50$ for EBS-RTTB since $R_a - R_e$ can be 50 times larger than $1 - SoC(t+1)$ in some cases. Compared to (5), which is the pheromone update formula for pi-RTTB, the only difference is the extra term $1 - SoC(t+1)$ in the denominator, and therefore if $\omega_1 = 1$, $\omega_2 = 0$, and $SoC_0 = 1$, then (9) is equivalent to (5). The intuition is that in addition to execution time, if the node's SoC is low, its pheromone will decrease, which makes it less attractive for accepting future incoming tasks.

Since the original pi-RTTB paper [5] did not mention the formula for computing R_e which is ETT, we proposed a linear regression model for simulation. We consider a dummy 500×500 matrix multiplication function, denoted as f_{dummy}, as a single instruction to execute, and we collect the execution time of f_{dummy} with different instruction count IC on a single core. The formula for computing R_e and ETT is shown in (10):

$$R_e = ETT = \frac{T_{avg,f_{dummy}} \times IC + e}{PC} \tag{10}$$

where $T_{avg,f_{dummy}}$ is the average execution time for f_{dummy} in seconds, e is the regression error term, IC is the instruction count, and PC is the number of CPU cores. The numerator part $T_{avg,f_{dummy}} \times IC + e$ is the regression model. We collected execution time by increasing IC until the execution time reach 1 second. After regression test, we found that $T_{avg,f_{dummy}} = 0.005804$ and $e = 0.095973$, and the regression visualization is shown in Fig. 2.

2) Roulette Wheel Selection for EBS-RTTB: To enhance the energy balance for task assignment, EBS-RTTB adds SoC to the computation of the selection probability in Roulette Wheel, as shown in (11):

$$p'_{select,i,j} = \frac{(P_{i,t(j)})^{\alpha}(ETT_{i,j})^{-\beta}(SoC_{i,t(j)})^{\gamma}}{\sum_{i=1}^{N}(P_{i,t(j)})^{\alpha}(ETT_{i,j})^{-\beta}(SoC_{i,t(j)})^{\gamma}} \tag{11}$$

the difference between $p_{select,i,j}$ in (4) and $p'_{select,i,j}$ in (11) is the extra $(SoC_{i,t(j)})^{\gamma}$ term, and therefore if $\gamma = 0$ then it is equivalent to pi-RTTB selection in (4). The intuition is that besides faster runtime $ETT_{i,j}^{-1}$ and larger pheromone $P_{i,t(j)}$, the selection probability should also consider the battery percentage SoC of the node when task j arrives, and the node with low SoC should decrease its possibility of being selected even it has a great computational efficiency. This way, the task manager will consider energy balancing alongside with efficient task execution. In simulation, we set $\gamma = 1$ for EBS-RTTB experiment.

B. Simulation Implementation

The simulation is implemented using Python with version $>= 3.11$, and VSCode is used as the editor to visualize animation. This section will introduce the implementation details and experimental setups.

1) Node Initialization: Each IoT node contains the following information:

- **Name**: the name of the node for identification purpose.
- **PC**: number of processing cores.
- **P0**: the initial pheromone P'_0 of node computed using (8).
- **Battery**: a simulated linear battery using (7), including the value of SoC_0, SoC, and A; in simulation, $SoC_0 = 1$ for all nodes to simulate a fully charged network.
- **P_record**: all historical pheromone values of the node.
- **SoC_record**: all historical SoC values of the node.

Since $T_{benchmark}$ and K_{α} is needed to compute P'_0 which requires several benchmarking programs, we consider 5 benchmarking programs for each to execute:

- **b1**: get all primes from 0 to 9999.
- **b2**: 500×500 matrix dot product.
- **b3**: sort a list with length 10^6.
- **b4**: Monte Carlo simulation for π estimation [12].
- **b5**: get the 20th Fibonacci number recursively.

The expected threshold completion time for the 5 benchmarking programs, denoted as $X = [x_1, x_2, x_3, x_4, x_5]$, is [0.004, 0.008, 0.35, 0.02, 0.0006] in seconds, which is the pre-executed average time. K_{α} is then computed using (3) so that P'_0 can be computed. When a node is started in simulation, a unique port will be assigned to it and the node will act as a server waiting to accept requests from the task manager. The information for all nodes is stored in a public database for task manager to access and setup animations. In our experimental setup, we initialized 5 nodes with configurations shown in Table I.

2) Task Assignment: The task manager is simulated as a client that send requests to nodes. There are 4 message types a task manager can send in our simulation:

TABLE I
NODE CONFIGURATIONS FOR THE EXPERIMENT

Name	PC (Cores)	A (battery drain speed)
iPhone 13 Pro	6	0.469
Galaxy S21	8	0.45
Realme GT	8	0.444
Rock Pi	6	0.48
Echo Dot	4	0.278

- **TTRC**: the TTRC package when broadcasting a task assignment request. Instead of sending the full task body, TTRC in our simulation will only send the instruction count IC.
- **TA**: task assignment message that will be sent after Roulette Wheel selection. This message will only be sent to the node being selected in a round. Same as TTRC simulation, TA will only send IC to the node and let the node execute f_{dummy} IC times.
- **X**: this is a dummy message that will do nothing besides equalizing the length of P_record and SoC_record for each node. This message will be sent to nodes not being selected in a round.
- **TERMINATE**: this will terminate the entire network and will be sent when any node's SoC becomes zero to simulate the death of the entire network.

Note that message **X** and **TERMINATE** is designed specifically for simulation and is not needed when applying to the real IoT devices. The instruction count IC will be generated randomly in a queue, ranged from 10 to 100. To make a fair comparison between EBS-RTTB and other algorithms, we make this random queue identical among experiments by seeding it to 42.

3) Node Response: The response and actions a simulated node will do after receiving specific requests from the task manager includes the following:

- **TTRC Received**: the node will respond with an RTRC_ACK, including its current pheromone value $P_{i,t(j)}$, $ETT_{i,j}$ computed using (10), and $SoC_{i,t(j)}$.
- **TA Received**: the node will execute f_{dummy} for IC times, updates its pheromone using (9) and SoC using (12), and send to the task manager the actual execution time R_a.
- **X Received**: the node will do nothing besides adding an extra length for P_record and SoC_record.
- **TERMINATE Received**: the node will terminate.

Considering the fact that using (7) with configuration of A in Table I to update SoC can make the simulation extremely slow, we add an accelerator in the computation of SoC update so that the battery drainage can go faster. The SoC update with accelerator is shown in (12):

$$SoC(t) = \max(SoC_0 - A_{accelerated}t, 0) \tag{12}$$

where $A_{accelerated} = 100A$.

The final setup of the network can be visualized in Fig. 3, where each IoT node with its name is colored red with opacity

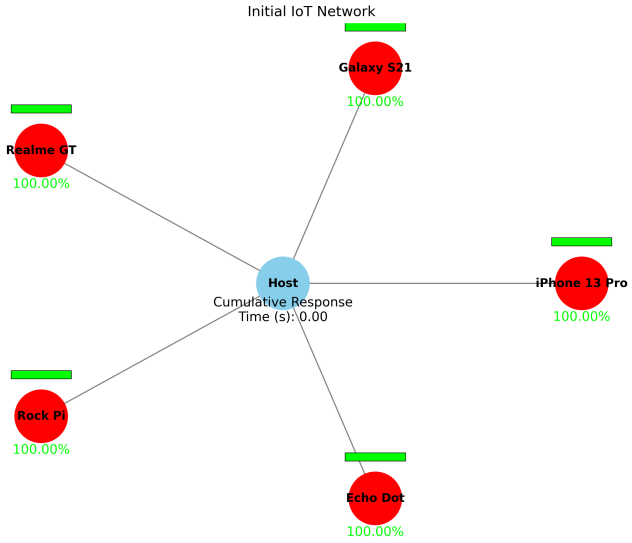

Fig. 3. Network topology of the experimental setup.

based on its pheromone value, and their battery status is shown on the battery bar with percentage. The host node colored blue is connected to all other nodes and will record the cumulative response time for all tasks.

IV. EXPERIMENTAL RESULT

To evaluate the effectiveness of the proposed energy-aware task distribution mechanism, a series of simulations are carried out with different configurations.

- **Baseline:** No energy consideration ($\omega_2 = 0$, $\gamma = 0$), which is Pi-RTTB.
- **Energy Balance in Pheromones:** Energy weight is applied in the pheromone update algorithm, but without considering the balance logic in device selection ($\omega_2 = 50$, $\gamma = 0$). In this case, the more healthy the battery status of the device, the higher the pheromone value will be.
- **Energy-aware scheduling:** Both energy weight and balancing mechanism enabled ($\omega_2 = 50$, $\gamma = 1$).

A. Battery Consumption and Network Lifetime

Figure 4 tracks the status of the node whose battery drains first in each network, as the number of tasks increases. The blue curve (Pi-RTTB; $\omega_2 = 0$, $\gamma = 0$) shows the shortest life with the first node drained after approximately 2500 tasks. Introducing energy weighting to the pheromone update (orange curve) slightly delays the death of the system to around 3600 tasks. However, with fully energy-aware scheduling (green curve), the network can operate up to about 4700 tasks, representing an improvement of 88% in lifespan compared to Pi-RTTB.

Pi-RTTB and EBS-FTTB were used to process 30 units of tasks, respectively. Fig. 5 and 6 illustrate the power status of the devices in these two networks. After the same workload, the EBS-RTTB network has a more balanced and healthy battery status.

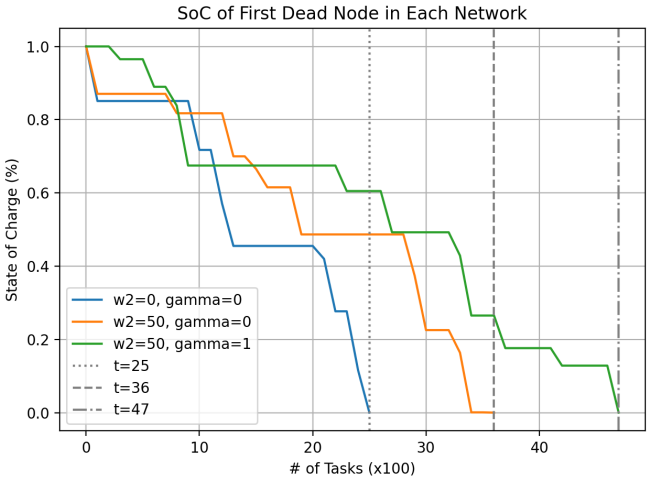

Fig. 4. SoC of first dead node under different configurations.

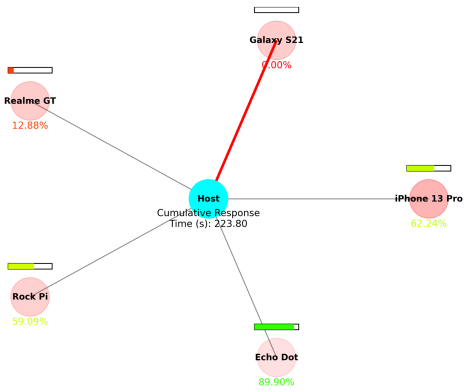

Fig. 5. Power status of the Pi-RTTB network. One device has been offline due to power exhaustion, and another device has only 12.88% remaining power.

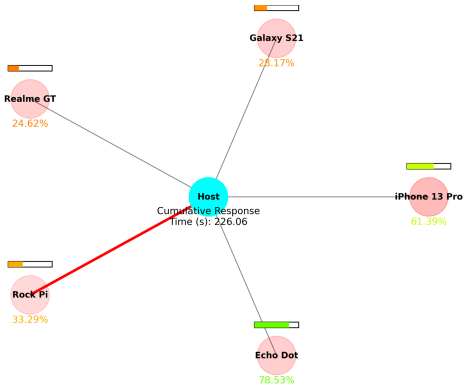

Fig. 6. Power status of the EBS-RTTB network. The network can still continue to work with the lowest power at 24.62%.

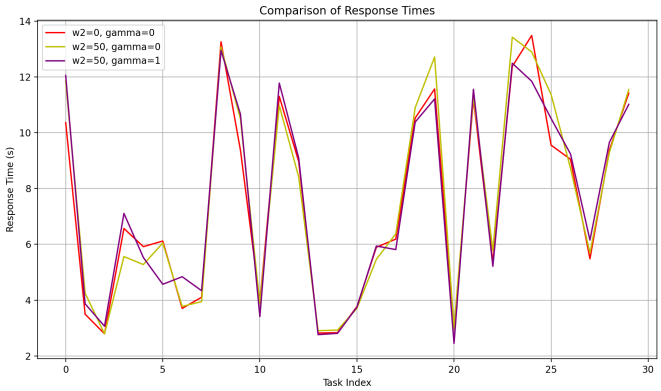

Fig. 7. Response time of 30 tasks among the three algorithms.

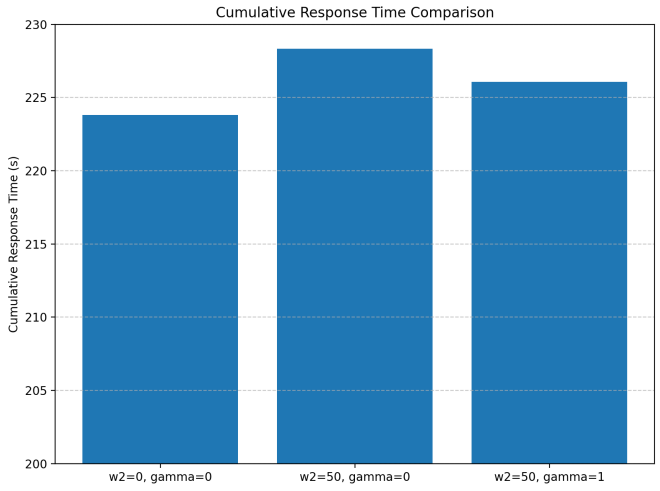

Fig. 8. Cumulative response time among the three algorithms after 30 tasks.

B. Response Time Performance

In theory, after counting the battery status in the algorithm, the effect of the computing speed on the assignment strategy is reduced, leading to a longer response time for tasks. To evaluate the impact of the energy balance strategy on task computation efficiency, the response time for each task was collected when different allocation strategies were adopted, as shown in Fig. 7. Although the algorithms represented by the yellow and purple lines take battery status into account with varying levels, they do not significantly affect the response time of each task.

The cumulative response time of the network system for all tasks better shows the overall impact, as shown in Fig. 8. When the battery state is taken into account, the system response time increases slightly, which is consistent with theory. The system's computation efficiency is sacrificed. However, it is acceptable because the impact is not significant.

V. Conclusion

This study proposed an enhanced energy-aware task-balancing approach, which integrates state-of-charge (SoC)

into the ACO-based pheromone-inspired real-time task-balancing system [5]. The simulation experiments demonstrate the significant improvement of the lifespan and runtime under this suggested approach by preventing the energy exhaustion of a single node in the IoT system while not significantly influencing the response time and still retaining acceptable efficiency. These results indicate that adding the energy measurement strategy to task assignments will greatly enhance the durability and robustness of the IoT network.

However, as the current study is simply based on simulation, physical implementations could be conducted to demonstrate the practical applicability of this proposed model. Future studies could consider the following areas: First, the energy cost of computing SoC, possibly utilizing the ultra-low power SoC measuring chip presented by Jeong et al.[11]. Second, the present simulation did not concern the propagation delay, which may influence the accuracy of the response time in a practical situation. In addition, replacing the dummy tasks with the real-world tasks in the simulation provides a more complete evaluation of the IoT system. Furthermore, as the current model only supports the MESH topology, future works could adapt this model to a more comprehensive multi-hop design. Finally, it is also critical to develop a more robust method for Estimated Turnaround Time (ETT) calculation to improve the overall accuracy of the task assignment decision.

Overall, this simulation experiment demonstrates a promising result, and the recommendations highlight some crucial areas which could help future studies verify the feasibility and adaptability of this proposed approach in real-world IoT deployment.

REFERENCES

[1] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. dissertation, Politecnico di Milano, Milan, Italy, 1992.

[2] M. J. Alam, R. Chugh, S. Azad, and M. R. Hossain, "Ant colony optimization-based solution to optimize load balancing and throughput for 5G and beyond heterogeneous networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2024, no. 44, pp. 1–23, Mar. 2024, doi: 10.1186/s13638-024-02376-2.

[3] S. K. Sharma and X. Wang, "Towards massive machine type communications in ultra-dense cellular IoT networks: Current issues and machine learning-assisted solutions," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 426–471, Firstquarter 2020, doi: 10.1109/COMST.2019.2949411.

[4] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, Oct. 2018, doi: 10.1016/j.future.2018.04.057.

[5] M. Wilson, H. Nunoo-Mensah, K. O. Boateng, and F. A. Acheampong, "A Real-Time Task Balancing Strategy for IoT Networks Using Ant Colony Optimization," Jun. 23, 2024, doi: https://doi.org/10.21203/rs.3.rs-4625359/v1.

[6] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," *J. Syst. Res.*, vol. 1, pp. 1–8, 2018.

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[8] S. Jannu, S. Dara, C. Thuppari, A. Vidyarthi, D. Ghosh, P. Tiwari, and G. Muhammad, "Energy efficient quantum-informed ant colony optimization algorithms for industrial Internet of Things," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 3, pp. 1077–1086, Mar. 2024.

[9] B. Ruixin and F. Liu, "Task arrival based energy efficient optimization in smart-IoT data center," *Mathematical Biosciences and Engineering*, vol. 18, no. 3, pp. 2713–2732, 2021.

[10] "Coulomb Counting - an overview," ScienceDirect Topics. [Online]. Available: https://www.sciencedirect.com/topics/engineering/coulomb-counting

[11] D. Jeong, K. Y. Lee, S. Bang, D. Sylvester, and D. Blaauw, "A 42 nJ/Conversion On-Demand SoC Indication Circuit for Small IoT Batteries," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 1–3, 2015.

[12] D. Saha and S. Roy, "Estimation of Pi Using Monte Carlo Simulation," in *Proceedings of the 2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, Coimbatore, India, 2016, pp. 1-4. doi: 10.1109/ICACA.2016.7887937.