MQTT-EES: Optimizing Energy Efficiency by Aggregating Sensing Tasks on IoT Devices

Nico Bokhari, Zhengquan Li, Zheng Song
Dept. of Computer and Information Science, University of Michigan-Dearborn, MI, USA
{nebokha, zqli, zhesong}@umich.edu

2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall) | 979-8-3315-1778-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTC2024-Fall63153.2024.10757779

***Abstract*—MQTT is a widely utilized protocol in the IoT domain, specifically designed to minimize energy consumption in battery-powered, energy-intensive IoT devices. With the proliferation of smart home devices, there is a notable increase in co-located IoT devices capable of publishing to the same topic, as well as an increase in subscribers accessing diverse data from these devices. However, the architecture of current MQTT brokers do not effectively optimize task scheduling among multiple potential publishers. Our observations suggest that although aggregating sensing tasks on the same IoT device does not significantly impact the total sensing energy consumption, it substantially reduces communication energy costs by minimizing the long-tail energy expenses associated with wireless communications. In this paper, we introduce MQTT-EES (MQTT *E*nergy *E*fficient *S*cheduling), which further optimizes the energy efficiency of MQTT by allocating sensing tasks on IoT devices with the goals of 1) minimizing long-tail communication energy consumption; and 2) prolonging the overall lifespan of the IoT system. We formulate the energy consumption challenge as an NP-hard problem and propose a greedy algorithm to tackle it. Our simulations show that MQTT-EES reduces average energy consumption by up to 12% and extends the overall lifespan of the system 2.78 times compared to standard MQTT implementations.**

I. Introduction

The advent of the Internet of Things (IoT) has revolutionized the way we interact with the world as the integration of network, computational, and intelligent technologies into everyday objects has transformed various domains including smart homes, healthcare, and industrial automation. Among the myriad of protocols facilitating this transformation, MQTT (Message Queuing Telemetry Transport) stands out due to its efficient, lightweight design tailored for energy-constrained environments typical of IoT devices. These devices run an MQTT Client software to perform sensing tasks, which involve the capture and transfer of their environments' data to other applications connected on a shared MQTT network. Figure 1 exhibits an example MQTT network, which shows the MQTT Broker, an intermediary software entity that uses a topic-based publish-subscribe architecture to facilitate MQTT communication amongst IoT devices(publishers) and subscribing applications(subscribers).

The increase in smart–home devices has led to a surge in MQTT networks in which IoT devices operate (and sense) the same physical environment, thus they perform the same sensing task, thereby transferring duplicate sets of data. Simultaneously, there is a growing number of applications that subscribe to smart-home devices' data, yet only require one set of data. This scenario presents unique challenges,

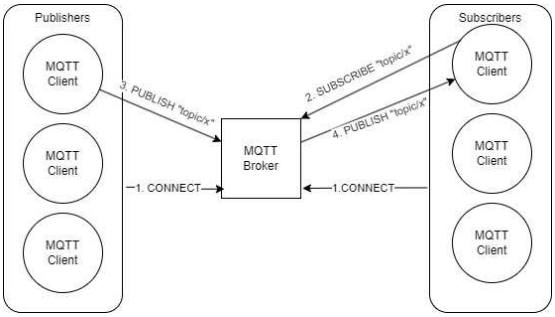

Fig. 1: MQTT-Based System

including higher network traffic, which results in subscribers experiencing higher latency. Additionally, many IoT devices waste energy sensing and transferring the replicated data, as well as any re-transmissions triggered by the higher network latency. Unfortunately, current MQTT Broker implementations lack mechanisms to distribute multiple sensing tasks amongst co-located publishers such that the minimum amount of data is transferred to satisfy all subscribers' latency requirements. As a result, these IoT systems experience inefficient energy use and reduced system longevity.

We observed IoT systems with IoT devices that transfer data via wireless communication, which inherently results in a problem of tail energy consumption. Tail energy consumption refers to the unnecessary energy consumed by a publisher after completing a data transfer, as the publisher remains in a high-power state for a certain tail time. Considering the unnecessary energy consumption from uncoordinated sensing tasks, the tail energy effect, as well as the limited battery power available to IoT devices, IoT systems with co-located publishers risk system shutdown.

Thus, this paper introduces MQTT-EES, a novel sensing task orchestration system to optimize the energy efficiency of MQTT. Since each device has various sensing capabilities, and each subscriber has different latency requirements to satisfy, there exists a finite number of task aggregations with realistic energy limitations of IoT systems, so MQTT-EES approaches this NP-hard problem with a greedy algorithm. Specifically, tasks are allocated to publishers which optimize the number of data transfers performed during tail windows. In doing so, each publisher effectively utilizes their high energy state, thereby minimizing their energy consumption. In this capacity, more sensing tasks may be aggregated to all publishers, resulting in the extension of the IoT system's overall lifespan. The main

contributions of our paper are as below:

- We propose MQTT-EES, a novel task orchestration system to improve the energy efficiency of MQTT by aggregating tasks on IoT devices. By formulating energy efficient task aggregation as an NP-hard optimization problem and presenting a greedy algorithm to solve it, MQTT-EES minimizes long-tail communication energy consumption, and prolonging the overall lifespan of the MQTT system.
- We evaluated MQTT-EES under simulation-based experiments that incorporates different system parameters to test diverse IoT conditions, including the number of publishers, subscribers, topics, as well as the size of the tail window. Our comprehensive evaluation results that MQTT-EES significantly extends the overall lifespan of the system 2.78 times compared to standard MQTT implementations and reduces the average energy consumption by up to 12%.

The rest of the paper is as follows. Section II provides an overview of the MQTT protocol and its previous challenges. Section III provides our system modeling to describe IoT systems with co-located devices. Section IV overviews the optimization problem and our solution's greedy approach. Section V showcases MQTT-EES's results and its experiment configurations. Section VI draws conclusions from MQTT-EES's results and a summary of its contributions.

II. Background and Related Works

A. MQTT System

Many MQTT-based systems implement energy-saving technologies given the protocol's low CPU usage and lightweight code footprint [1]. Unfortunately, MQTT's re-transmission mechanisms have been shown to increase energy consumption under high traffic scenarios [2]. Furthermore, the protocol, over wireless communication, is susceptible to unnecessary tail energy consumption, especially during transmissions of small packet sizes [3] [4] [5]. There exist studies that attempt to decrease tail energy consumption through an additional end-to-end communication packet attribute [6], or an additional payload of delay-tolerant data within session heartbeat transmissions [7]; however, they both require incompatible modifications to the latest MQTT specification [8]. MQTT's many-to-many communication enables multiple publishers to transmit data over a single topic for multiple subscribers to receive; this paradigm's flexibility often leads to heavy network traffic if not scheduled or balanced heedlessly. While MQTT-EES takes issue aggregating sensing tasks across heterogeneous IoT devices, there exists many solutions which aggregate the queuing or processing of sensed data through schedulers or load-balanced network architectures to stabilize the network's traffic and latency.

B. MQTT System Optimization

Cloud-computing principles have emerged out of many IoT network management solutions as clustered/bridged MQTT Brokers and other supporting fog gateways may be created, deployed, and destroyed dynamically. For example, [9] proposes a framework that deploys additional MQTT Brokers and load balances MQTT Client connections such that the cloud CPU utilization stabilizes at a user-defined percentage. Another example is [10], which dynamically spawns cloud MQTT Brokers within close proximity of edge IoT device nodes to maintain latency requirements of connected MQTT Clients. While [11] is another cloud-based load-balanced IoT solution, its ensures the QoS of mission-critical health data in addition to maintaining the energy efficiency of a 4-tier cloud-fog architecture. Cloud-orchestrated IoT solutions may optimize the scalability of MQTT-based systems, but the aforementioned solutions do not model the energy efficiency of individual battery-powered IoT devices. Many IoT schedulers focus on organizing data queues to best meet high priority data QoS. [12] is novel for presenting a multi-Broker IoT scheduler which adapts individual Broker's input and output network traffic to maintain stable network traffic and the delivery of critical message types. The system's energy consumption model is contingent upon IoT devices which exhibit the same sensing, processing, and transmission ability; thus, IoT devices with heterogeneous sensing capabilities are not considered. [13] assumes heterogeneous IoT devices while orchestrating priority queues to ensure the delivery of highly prioritized data, but does not model the individual devices' energy consumption during execution. Various solutions harness scheduling and cloud-computing principles to optimize MQTT-based systems, but many neglect to model IoT devices' constrained energy resource, and an approach to optimize heterogeneous device utilization through tail energy reduction and task distribution.

III. System Model

We consider the MQTT-EES comprised of a set of publishers $P = \{p_1, p_2, \ldots, p_m\}$ and a set of sensing topics $T = \{t_1, t_2, \ldots, t_n\}$. Each publisher p_j is characterized by its energy capacity C_j, which indicates the total amount of energy available for task executions over the observation period T_{obs}. Additionally, each publisher has a specific set of sensing capabilities denoted by $S_j = \{s_{j1}, s_{j2}, \ldots, s_{jn}\}$, where $s_{ji} = 1$ if publisher j is capable of sensing data for topic i and 0 otherwise. The topics are defined by their sensing frequencies f_i assigned by subscribers, representing the intervals at which the sensor data need to be read and sent to the broker within the observation period. Let $F = \{f_1, f_2, ..., f_n\}$ represents the set of frequencies for all topics.

As the publisher devices usually send data to the broker via wireless communication (e.g., WiFi and Bluetooth), which inherently results in a problem of tail energy consumption. Tail energy consumption here refers to the energy consumed by a publisher after completing a data transfer. Even when the transfer is finished, the publisher remains in a high-power state for a certain tail time, consuming unnecessary energy.

In this case, if multiple sensing tasks happen at the same time on a publisher, it is intuitive to consider it as a single energy-consuming event (we assume the time of sensing the data and sending data to the broker for each topic is constant

and same) as it will not trigger separate tail energy windows. The energy consumption is related to the effective number of task executions. Effective executions refer to the count of task executions within the observation period T_{obs} that are considered distinct in terms of energy consumption. Due to the tail energy effect, multiple tasks executed at the same moment are consolidated and counted as only one effective execution. The number of effective executions may be less than the total number of task executions. Therefore, if MQTT-EES can assign the topics to publishers to reduce the number of effective executions within the observation period T_{obs}, the more energy can be saved.

IV. Problem Formulation and Solving

A. Problem Formulation

We aim to minimize the total energy consumption across all publishers by intelligently assigning each sensing topic to a single publisher. The assignment considers each publisher's energy capacity, sensing capabilities, and the tail energy consumption of executing sensing tasks, with the goal of minimizing the number of effective executions and thereby optimizing the overall energy index.

Let x_{ij} be a binary decision variable where $x_{ij} = 1$ if topic t_i is assigned to publisher p_j, and $x_{ij} = 0$ otherwise. We define a vector $\mathbf{x}_j$ for each publisher j, encapsulating the assignment of all topics to that publisher:

$$\mathbf{x}_j = [x_{1j}, x_{2j}, \ldots, x_{nj}]$$

The output of the optimization model is $\mathcal{X} = \{\mathbf{x}_j, \forall j \leq m\}$. The objective is to minimize the total energy index, which is the sum of the individual energy indices of all publishers; each index represents the ratio of the total energy cost of a given publisher (including the sensing and communication cost) to the given publisher's remaining.

$$\begin{aligned}
\mathcal{X} &= \arg\min \sum_{j=1}^{m} \frac{E_j}{C_j} \\
&= \arg\min \sum_{j=1}^{m} \frac{\textbf{Sensing}(\mathbf{x}_j, T, F) + \textbf{Commn}(\mathbf{x}_j, T, F)}{C_j} \\
s.t. &: \sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1, \ldots, n\} \\
s.t. &: x_{ij} \leq s_{ji}, \quad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \\
s.t. &: \sum_{i=1}^{n} x_{ij} \cdot e_i \leq C_j, \quad \forall j \in \{1, \ldots, m\}
\end{aligned} \tag{1}$$

where $\textbf{Sensing}(\mathbf{x}_j, T, F)$ calculate the total sensing energy consumption for publisher j based on its assigned topics (through $\mathbf{x}_j$). $\textbf{Commn}(\mathbf{x}_j, T, F)$ calculates the total communication energy consumption for publisher j based on its assigned topics (through $\mathbf{x}_j$). The observation period is noted as T, and the set of all topics' frequencies is noted as $F = \{f_1, f_2, \ldots, f_n\}$.

For the Constraints, we explain them one by one as follows:

- **Assignment Constraint**: Each topic must be assigned to only one publisher who is capable of sensing it: $\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1, \ldots, n\}$
- **Sensing Capability Constraint**: A topic can only be assigned to a publisher if the publisher has the capability to sense it: $x_{ij} \leq s_{ji}, \quad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}$
- **Energy Capacity Constraint**: The total energy used by a publisher for the effective executions should not exceed its energy capacity: $\sum_{i=1}^{n} x_{ij} \cdot e_{pe} \leq C_j, \quad \forall j \in \{1, \ldots, m\}$. Here, e_i is the estimated energy consumption for a single execution of topic i, factoring in the energy saved by batching executions due to the tail energy effect.

B. Problem Solving

We start with introducing a brute-force approach to ease the understanding how MQTT-EES works to reduce the energy consumption. Then we move to our greedy approach to solve the problem more efficiently considering the constrained resources on IoT devices.

1) Brute-Force Approach: The brute-force approach simply explores all possible assignments of tasks (or sensing topics) to publishers. This method evaluates every possible combination of task-to-publisher assignments to identify the one that yields the lowest total energy index, which is indicative of the most energy-efficient distribution of tasks across the network of publishers. By assessing each potential assignment, the brute-force method ensures that no possible solution is overlooked, thereby guaranteeing the optimal assignment that minimizes energy consumption given the constraints of publisher capacities and task requirements. For each possible assignment, we calculate its resulted energy index for each publisher. This requires calculating effective executions of a given topic assignment. The following example illustrate the calculation of effective executions.

Consider an example involving a publisher assigned three sensing tasks over a one-minute observation period. Suppose Task 1 has a frequency of 10 seconds, Task 2 has a frequency of 20 seconds, and Task 3 has a frequency of 30 seconds. Given the assumption of a zero tail duration, tasks executed simultaneously are aggregated into a single energy-consuming event. The execution times within the one-minute period are as follows:

- Task 1 executes at 0, 10, 20, 30, 40, and 50 seconds.
- Task 2 executes at 0, 20, and 40 seconds.
- Task 3 executes at 0 and 30 seconds.

Here, describe how this example uses a tail window of 0, in which tasks with the exact same time points will be executed concurrently. In our system this tail window is held constant. We merge the timing of all task executions into a list $\{0, 0, 0, 10, 20, 20, 30, 30, 40, 40, 50\}$. Then we observe overlaps in execution times across these tasks. At the 0-second point, all three tasks execute simultaneously, counted as one effective execution. Similarly, overlaps at 20 and 30 seconds, where Task 1 and Task 2, and Task 1 and Task 3 overlap,

respectively, are each considered a single effective execution. Thus, after remove the overlaps, the timing of executive task executions is: $\{0, 10, 20, 30, 40, 50\}$.

Despite the total of 11 (6+3+2) scheduled task executions across the three topics, the effective executions are fewer due to the overlaps. Specifically, there are effectively 6 distinct execution times when considering the energy efficiency achieved through task overlaps at 0, 20, and 30 seconds. Thus, the number of effective executions is reduced to 6, illustrating how strategic scheduling batches tasks thereby significantly enhance energy efficiency by minimizing the total number of high-power states induced by sensing task executions.

2) Greedy Approach: The brute-force method exhaustively searches through all possible assignments whose size grows exponentially with the number of topics and publishers are present. This inherent complexity means our problem is classified as NP-hard and hard to obtain the optimal solution within polynomial-time. Thus, we purpose a greedy method, which incrementally builds a solution by making locally optimal choices at each step. This approach focuses on assigning each task to the publisher that will incur the smallest increase in energy consumption, given the current state of task assignments and each publisher's remaining energy capacity.

At the core of the greedy strategy is a calculation that estimates the energy a publisher will use if a new task were assigned to it, considering the task's frequency and the observation period. By iterating over the tasks and dynamically selecting the best publisher for each based on current energy usage and capacity constraints, the greedy approach avoids the explosion of possibilities faced by brute-force methods. The result is a faster, more practical solution that, while not guaranteeing the absolute optimal distribution of tasks, provides a near-optimal arrangement that significantly reduces overall energy consumption and ensures a more balanced utilization of publisher resources. This method is particularly advantageous in real-time systems or those with a large number of tasks and publishers, where computational efficiency and scalability are critical. The following is the pseudo code of the greedy approach:

The algorithm begins by initializing an empty mapping for *assignments* to keep track of the tasks assigned to each publisher. Besides, it sets the current energy usage (E_{current}) for each publisher to zero, preparing for subsequent energy consumption calculations. Then the algorithm iterates through each task that needs to be assigned. For each task, it evaluates which publisher would be the best candidate based on energy efficiency index. After determining E_{increase} for a task with a particular publisher, the algorithm computes a ratio (E_{ratio}) of the new total energy usage to the publisher's total energy capacity. This ratio helps in assessing the energy efficiency of assigning the task to the publisher. The publisher that results in the lowest E_{ratio} for the task is selected as p_{best}. This choice indicates an optimal balance between the task's energy requirements and the publisher's energy capacity. The selected task is then assigned to p_{best}, the publisher's current energy usage is updated, and the assignment is recorded. This procedure is iteratively applied to all tasks, ensuring each is allocated in a manner that optimally utilizes the available energy resources. Upon completing the assignments for all tasks, the algorithm returns the final *assignments* mapping, detailing how tasks have been distributed across publishers.

Algorithm 1 Greedy Approach

```
Initialize assignments as an empty mapping of publishers to tasks
Initialize E_current for each publisher as zero
for each task in tasks do
    p_best ← null
    E_min ← ∞
    for each p in P do
        E_increase ← EnergyIncrease(p, task.freq, T)
        E_new ← E_current[p] + E_increase
        E_ratio ← E_new / p.allEnergyCapacity
        if E_new ≤ p.allEnergyCapacity and E_ratio < E_min then
            p_best ← p
            E_min ← E_ratio
        end if
    end for
    if p_best ≠ null then
        assignments[p_best].append(task)
        E_current[p_best] ← E_current[p_best] + E_increase
    end if
end for
return assignments
function EnergyIncrease(p, freq, T)
    frequencies ← the frequencies of all tasks assigned to p.
    y_1 ← TotalExe(frequencies, T)
    y_2 ← EffectiveExe(frequencies, T)
    e_1 ← 10          ▷ Assume the energy of sensing one data is a constant
    e_2 ← 5           ▷ Assume the energy of transferring one data is a constant
    E_increase = e_1 * y_1 + e_2 * y_2
    return E_increase
end function
```

For the function *EnergyIncrease*, it estimate the energy consumption increase from a new task assignment by: 1) calculating the energy consumption of executing all the sensing tasks by calculating the total executions; 2) calculating the communication energy by first calculating the effective executions (y) from the aggregated frequencies of assigned tasks, reducing energy consumption by batching tasks (by the way of listing all the timing of executing tasks in a observation period, like we did in the brute-force approach). We assumes a constant energy consumption for sensing the data and transferring the data, simplifying the energy increase estimation (E_{increase}).

V. Evaluation

Our simulation measured two system performance metrics, *system energy consumption* and *system lifespan*. Simulated MQTT environments are varied in the number of subscribers, topics, and publishers, as well as the length of publishers' tail window. The default values of these components are listed in Table I.

Parameters	Default Values
# Publishers	8
# Subscribers	8
# Topics	8
Tail Window (ms)	250

TABLE I: Default Settings for System Parameters

The simulation also declares other non-varied constants for measuring both energy consumption and system lifespan, including:

- Range of topic frequencies F $(100 - 5000\text{ms})$
- Length of Observation Period T (1 hour = $3,600,000$ ms)
- Sensing Energy Cost per Execution (0.0005%)
- Communication Energy Cost per Execution (0.005%)

Additionally, the sensing capabilities and subscriptions for each topic are randomly generated at the beginning of each round, and shared among all tested scheduling algorithms.

A. System Energy Consumption

To verify the effectiveness of our scheduling , we compared MQTT-EES with one baseline: *Random Scheduling* in which the MQTT broker randomly assign the sensing tasks to the publishers that share the corresponding sensing capabilities. The simulation for each scheduling approach runs a 30-round experiment per varied parameter (number of publishers, subscribers, topics; size of tail window) to simulate diverse, real-world IoT environments. The details of how these parameters are varied are summarised in Table II. In the experiments, when one parameter is varied, all other parameters are held at the default values described in Table I. Specifically, during each round, the number of publishers, subscribers, and topics vary from 3 to 15 through random generation. The tail window is also varied given that IoT devices may adopt different network interfaces that consume different tail energy consumptions.

Fig. 2 to Fig. 5 compares the average energy consumption (i.e., the overall energy consumption divided by the number of system publishers) of MQTT-EES and *Random Scheduling* under different system configurations. To present the average energy reduction in using MQTT-EES over *Random Scheduling*, Fig. 6 plots the distribution of average energy reduction as a result of subtracting MQTT-EES's average energy consumption from *Random Scheduling* over each system parameter.

Parameter p	**Variance Range of Parameter** p
Publishers	$3 \leq p \leq 15$
Subscribers	$3 \leq p \leq 15$
Topics	$3 \leq p \leq 15$
Tail Window (ms)	$p \in \{100, 250, 500, 1000\}$

TABLE II: Value Range for Varied System Parameters

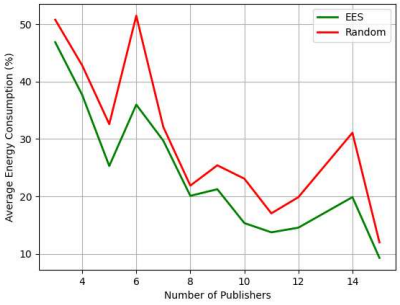

Fig. 2: Average Energy Consumption(%) vs # of Publishers

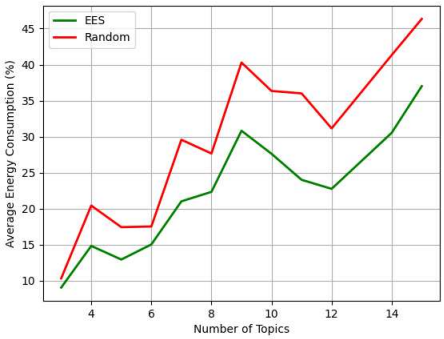

Fig. 3: Average Energy Consumption(%) vs # of Topics

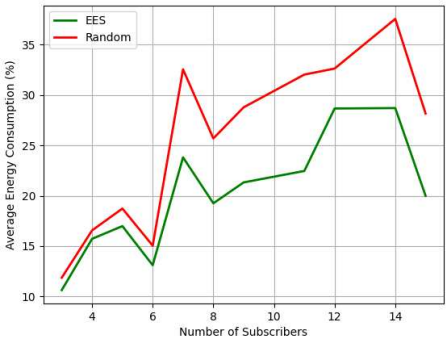

Fig. 4: Average Energy Consumption(%) vs # of Subscribers

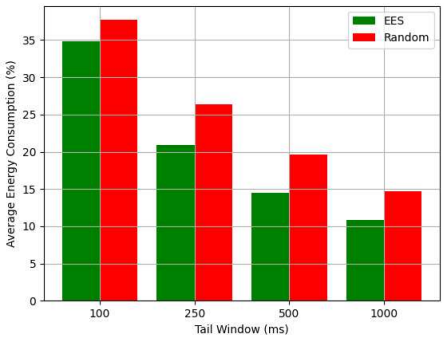

Fig. 5: Average Energy Consumption(%) vs. Tail Window

As the number of publishers increase in Fig. 2, the average energy consumption decreases for both MQTT-EES and *Random Scheduling* since both approaches distribute sensing tasks among all capable publishers. Since the number of sensing tasks is set to a default constant, adding more publishers further divides the sensing tasks, thereby decreasing energy consumption per publisher.

As the number of topics varies, the number of sensing tasks varies, which is evident in Fig. 3. As the number of topics increase, both scheduling approaches allocate more sensing tasks to a default number of publishers. In this capacity, each device in the simulated IoT system is allocated more sensing tasks, thereby increasing the number the executions, resulting in an increase in per-device energy consumption.

Fig. 4 shows that the average energy consumption increases as the number of subscribers increases. We attribute this to a simulation specific phenomenon in which the number of subscriptions on a topic increases as the number of subscribers increases. While the individual topic frequencies in F remain the

minimum latency requirements collected from the subscribers upon subscription, additional subscriptions result in a higher likelihood that the minimum latency requirement approaches the minimum value in the range of topic frequencies.

In Fig. 5, a wider tail window (i.e 1000ms) considers more sensing tasks as running concurrently, thereby decreasing the number of executions which results in decreased energy-consumption. In contrast, a smaller tail window (i.e 100 ms) requires sensing tasks to occur relatively close together in time which restricts the number of sensing tasks that may run concurrently. As less sensing tasks run concurrently, the number of executions increase, resulting in an increase in average energy consumption.

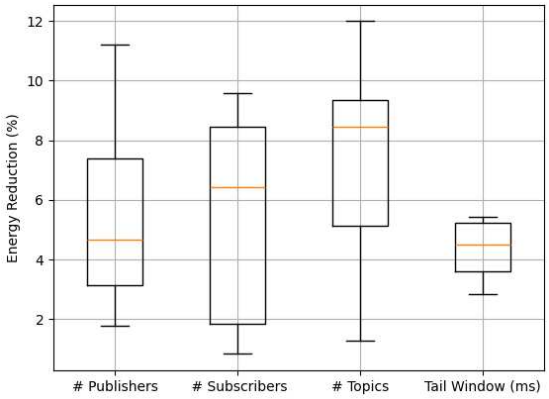

Fig. 6: Average Energy Reduction(%) between *MQTT-EES* and *Random Scheduling* across all experiments

In Fig. 6, we visualize the distribution of energy reductions as a result of subtracting MQTT-EES's average energy consumption from *Random Scheduling*. In summary, MQTT-EES outperformed *Random Scheduling* in all parameters configurations, with an reduction of 12% in terms of the average energy consumption.

B. System Lifespan

The system lifespan was evaluated under default parameters with three approaches. As illustrated in Fig. 7 , MQTT-EES lasts 3.41 hours until 1 publisher's simulated battery is reduced to 0. When compared to the lifespans of standard MQTT and *Random Scheduling*, MQTT-EES increases an IoT system's lifespan by 278% and 142% respectively. The improvement is confirmed by our approach's assertion that scheduled tasks are aggregated onto a single publisher given it consumes the least energy upon task execution. MQTT-EES differs from the standard MQTT implementation as it assigns a task to all publishers that are capable of doing so.

VI. Conclusion

In this paper, we purposed MQTT-EES, designed to optimize energy efficiency in IoT environments. Our approach can reduce average energy consumption by 12% and extends system lifespan by 2.78 times compared to standard MQTT implementations. These improvements are achieved through strategic task allocation and a greedy algorithm that tackles the NP-hard problem of energy optimization among co-located

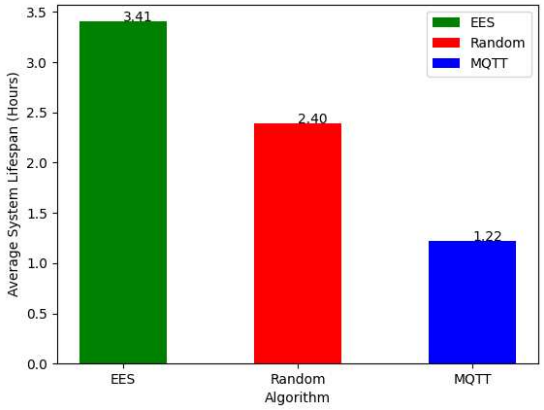

Fig. 7: Average System Lifespan(Hr) vs 3 Algorithms

IoT devices. These impressive results not only demonstrate substantial energy savings and enhanced system longevity but also suggest broader implications for sustainability in IoT.

VII. Acknowledgement

This research is supported by NSF through the grant #2104337.

References

[1] S. Jiwangkura, P. Sophatsathit, and A. Chandrachai, "Mqtt of iot classification in energy saving," in *proceeding of 11th International Conference on Data Mining, Computers, Communication and Industrial Applications (DMCCIA-2017), Kuala Lumpur*, 2017.

[2] J. Toldinas, B. Lozinskis, E. Baranauskas, and A. Dobrovolskis, "Mqtt quality of service versus energy consumption," in *2019 23rd international conference electronics*. IEEE, 2019, pp. 1–4.

[3] D.-H. Mun, M. Le Dinh, and Y.-W. Kwon, "An assessment of internet of things protocols for resource-constrained applications," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 555–560.

[4] Y. Li, Y. Wang, and T. Lan, "Mobile ad prefetching and energy optimization via tail energy accounting," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2117–2128, 2018.

[5] N. Ferraz Junior, A. A. Silva, A. E. Guelfi, and S. T. Kofuji, "Performance evaluation of publish-subscribe systems in iot using energy-efficient and context-aware secure messages," *Journal of Cloud Computing*, vol. 11, no. 1, p. 6, 2022.

[6] Y. Im and M. Lim, "E-mqtt: End-to-end synchronous and asynchronous communication mechanisms in mqtt protocol," *Applied Sciences*, vol. 13, no. 22, p. 12419, 2023.

[7] T. Zhang, X. Zhang, F. Liu, H. Leng, Q. Yu, and G. Liang, "etrain: Making wasted energy useful by utilizing heartbeats for mobile data transmissions," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 113–122.

[8] O. Standard, "Mqtt version 5.0," *Retrieved June*, vol. 22, p. 2020, 2019.

[9] L. M. Pham, N.-T.-T. Le, and X.-T. Nguyen, "Multi-level just-enough elasticity for mqtt brokers of internet of things applications," *Cluster Computing*, vol. 25, no. 6, pp. 3961–3976, 2022.

[10] T. Rausch, S. Nastic, and S. Dustdar, "Emma: Distributed qos-aware mqtt middleware for edge computing applications," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 191–197.

[11] N. Singh and A. K. Das, "Energy-efficient fuzzy data offloading for iomt," *Computer Networks*, vol. 213, p. 109127, 2022.

[12] S. Abdullah and K. Yang, "An energy-efficient message scheduling algorithm in internet of things environment," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2013, pp. 311–316.

[13] F. L. de Caldas Filho, R. L. Rocha, C. J. Abbas, L. M. E. Martins, E. D. Canedo, and R. T. de Sousa, "Qos scheduling algorithm for a fog iot gateway," in *2019 Workshop on Communication Networks and Power Systems (WCNPS)*. IEEE, 2019, pp. 1–6.