Resource Inference for Sustainable and Responsive Task Offloading in Challenged Edge Networks

Alessio Sacco, *Student Member, IEEE,* Flavio Esposito, *Member, IEEE,* and Guido Marchetto, *Senior Member, IEEE,*

***Abstract*—In many edge computing applications, Unmanned Aerial Vehicles (UAVs) are required to be coordinated to perform several tasks. Each task is usually modeled as a process that a UAV runs, and could include hovering an area to find survivors after a natural disaster or sense and preprocess an image in cooperation with the edge cloud. Optimally and rapidly (re)assigning tasks to such IoT agents as the network conditions fluctuate and the battery of these agents quickly drains is a challenging problem. Existing solutions designed to proactively offload tasks are either energy unaware or they require solving computationally intensive task, and hence are less portable on constrained IoT devices. In this paper, we propose RITMO, a distributed and adaptive task offloading algorithm that aims to solve these challenges. RITMO exploits a simple yet effective regressor to dynamically predict the length of future UAV task queues. Such prediction is then used to anticipate the node overloading and avoid agents that are likely to exhaust their battery or their computational resources. Our results demonstrate how RITMO helps reduce the overall latency perceived by the application and the energy consumed by the nodes, outperforming recent solutions.**

***Index Terms*—edge computing, task offloading, UAV, regression prediction**

I. Introduction

Distributed applications running on Internet of Things (IoT) devices that require to perform a mission independently are opening many applications, sometimes improving lives, sometimes even saving them [1]–[3]. Typical examples are Unmanned Aerial Vehicles (UAV) networks, e.g., drones or fixed wings planes, equipped with AR/VR interfaces [4] cameras, sensors, or civilian tablets and smartphones [5], [6]. Such systems have been employed, for instance, for precision agriculture [7], in disaster response and environmental monitoring [8], [9], or to provide connectivity to ground stations [10]. Autonomous and semi-autonomous drones have also been helping humans and other machines accomplish many tasks, spanning from industrial inspection to survey operations, from rescue management systems to military or first responder support. In these applications, UAV networks can be used to collect a massive quantity of data that needs to be offloaded at the network edge for heavy audio/video processing, where resources to execute Machine Learning (ML) algorithms are readily available.

The challenge of keeping an acceptable quality of service with stringent delay constraints for these network grows with the drone-based and IoT-based applications, especially in challenged networked scenarios [1], [11], [12]. In these circumstances, edge network managers and application programmers need to overcome a few challenges, e.g., unstable network conditions and high peaks of loss rate. To react against the dynamic nature of connectivity and with UAV hardware and battery failures, the research community has proposed several techniques aimed at changing how tasks are re-assigned to agents at runtime. These solutions are centralized [13], [14], and distributed [15], [16], and share the use case of multiple agents that coordinate to accomplishing a mission, i.e., a set of logically ordered tasks, solving the problem differently. Some of them focus on the resilient mission planning problem [16], a problem formulation similar to a task offloading among a fleet of UAV agents; others focus on the awareness of agents' health to replan [15]; others yet enable agents to autonomously tackle complex, large-scale missions, in the presence of actuator failures [14]. While these solutions have a sound design and good performance, none of them is able to look at the past and learn from prior errors, in order to anticipate job demands and network fluctuations by orchestrating the task assignment through a resource usage prediction.

In this paper, we propose RITMO (Resource Inference for Task MigratiOn), a solution that proactively redistributes job loads among multiple processes running within distributed agents (nodes). Differently from our preliminary version presented in [17], the migration process aims to jointly minimize the energy use and the task completion time. In particular, each agent predicts the future queue length and accordingly migrates jobs (i.e., drone tasks) to agents, guaranteeing performance. To determine the agent's future load, our system uses a predictor based on time-series forecasting, specifically the Autoregressive Integrated Moving Average (ARIMA) algorithm [18]. Unlike other machine learning-based methods, the features exploited by ARIMA are restricted to just one value in time-series forecasting. We have experimented that exploiting just one value in time-series forecasting fits well with our constrained environments, such as the drone swarms cooperating to finalize a mission. In fact, compared to other regressors, this technique does not require a large amount of memory to store past values. Such information can then be used to adapt the agent's load to a policy profile that achieves our goal of minimizing task completion time and energy consumption. Our results show how RITMO provides better performance with respect to the benchmark algorithms, even for many nodes and with high failure rates.

The rest of the paper is structured as follows. In Sec-

Alessio Sacco and Guido Marchetto are with DAUIN, Politecnico di Torino, 10129 Turin, Italy (e-mail: alessio_sacco@polito.it, guido.marchetto@polito.it).

Flavio Esposito is with the Department of Computer Science, Saint Louis University, St. Louis, MO 63103 USA (e-mail: flavio.esposito@slu.edu).

tion II, we present the most related solutions to RITMO, and in Section III, we describe some applications that can benefit from the usage of RITMO. Section IV introduces the RITMO's model and formalizes the problem definition. The algorithm utilized to solve such a problem is then described in Section V, while Section VI outlines the main components of our solution. Then, Section VII shows the performance of RITMO and the advantages over similar solutions. Finally, Section VIII concludes our paper.

II. Related Work

Providing a persistent and adaptive service resilient to failure is a crucial problem for any IoT network in general and robotic or drone networks in particular. As such, it is not surprising that there are several proposed solutions to tackle this problem. In this section, we cite a few representative (centralized and distributed) solutions that clarify our contributions.

Recently, decentralized approaches have been proposed to improve the adaptability and the persistence of distributed IoT systems [13]–[15], [19]–[21]. For example, the authors in [20] address the problem of task allocation and scheduling. The problem of task allocation in robotics is similar to our considered task offloading, except that the host running the task is not limited to UAVs, but can also be a server located on an edge cloud.

Moreover, similarly to [20], our solution can distribute workload efficiently among agents, but our predictive system exploits a time series prediction approach to optimize the system load. Nevertheless, our solution can manage both centralized and distributed management architectures, given that it is agnostic to the agent architecture. Inspired by [22], we utilize a network queuing model to estimate tasks that are waiting for the execution aboard the agent; however, we differ for both the considered problem and the prediction model. We study the problem of reassigning tasks among IoT devices and edge cloud processes that cooperate for a mission. We model the failure and the overloading of agents with a regressor algorithm [18].

Mission planning for IoT systems. For effective mission planning solutions, the literature reports two main methodologies: reactive and proactive [23]. In reactive planning, tasks are reassigned upon the occurrence of particular events, such as critical outages or measurement indexes triggering alerts. Unfortunately, since these approaches do not anticipate failures, they can potentially lead to catastrophic mission results. For example, it may cause too frequent changes in high-dynamic scenarios, thus reducing performances and increasing the computational burden.

Conversely, proactive planning, often operating at fixed time instants, evaluates the impact of disturbances from the environment on the mission's expected outcome. By migrating new assignments to uncontested agents, these methods can mitigate possible performance degradation before its occurrence. Thus, for very uncertain and dynamic environments, proactive approaches are the preferred choice [23]–[25] since they allow to trade between performances and too frequent reassignments.

Prediction for UAV systems. One example of solutions that proactively distribute tasks among the agents is APRON [26], which proposed using Jackson's network model to estimate the number of functions in the system for a replanning algorithm. Similarly, [25] presents a solution that monitors the execution of tasks in real-time and reassigns them to maintain a desired performance metric for the whole network. Our approach employs a predictive control approach to limit offloading decisions and find (on-line) the best task assignment. We share with these solutions the idea of proactively migrating tasks, but we differ in the model, the algorithm, and the architecture presented to solve the replanning problem.

A similar solution is presented in [15], proposing HAP. This concurrent learning adaptive control architecture establishes feedback between the high-level planning based on Markov Decision Processes (MDP) and the vehicle-level adaptive control algorithm. Using this feedback, HAP can anticipate the failures and proactively reassess vehicle capabilities after any failures for an efficient replanning schema that accounts for changing capabilities. Our load prediction model is different. While HAP estimates vehicle capabilities using the adaptive controller's vehicle health model, RITMO explicitly predicts future loads on an agent to adapt the system's overall load.

III. Motivating Applications and Use Cases

Despite becoming popular with military and emergency response, UAVs have been widely adopted also for civilian applications. The applicability ranges from agriculture and surveying to video making and real estate. This section describes two examples of motivating applications where our solution is incredibly effective, both of which share the underlying challenged edge networks.

A. Disaster Response

An efficient architecture for edge offloading is crucial for critical applications, such as real-time video conferencing with the incident commander to recognize faces of disaster victims [27], or the detection of children in an attempt to reunite them with their families [28]. Similarly, virtual beacons can be principally used to track their location.

Encouraged by the decrease of costs related to UAV technology, also the humanitarian community started piloting the use of UAV systems in humanitarian crises several years ago [29]. The setup for disaster response, e.g., after a hurricane or a severe flood, can be helped by utilizing UAVs, such as disaster mapping and information gathering, community capacity building, logistics, and even transportation of goods. From an operation perspective, the main goal is to image and map the affected areas in the shortest time possible from the mobilization request to take the immediate response and provide assistance to civilians.

Two phases occur simultaneously: UAV flight operations and data processing. The former includes an initial configuration on the UAV and the subsequent taking-off, flight, and landing. During the flight, the main goal is to properly acquire the required data and send them to a processing unit nearby. During the data processing phase, the data is mined

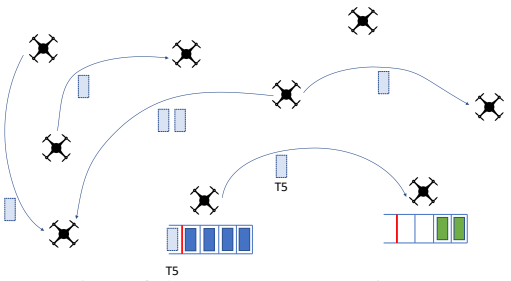


Fig. 1: Overview of the system: agents, i.e., UAVs, are modeled as queues containing tasks that need to be run. The system migrates tasks from (currently or likely to be) overloaded or failing agents to agents with increased availability.

by the application. These operations typically occur after preprocessing tasks, e.g., data resampling or image selection (to limit the processing only to the minimum set of images required to cover the affected areas).

To this end, edge computing can propel several applications by enabling data processing in real-time. Sending the imagery depicting the situation to the close edge cloud has been studied in the literature [1], [26], [30], providing good results. A solution like RITMO can be of tremendous help to these applications.

B. IoT for Precision Agriculture

Plant phenotyping refers to quantitative estimation of the plant characteristics, including physiological, ontogenetical, morphological, and biochemical properties, e.g., shape, canopy structure, leaf size, and color [7], [31], [32]. High-throughput phenotyping is a rapidly growing area of research that considers hundreds of genotypes to facilitate genetic studies and accelerate the breeding of advanced crop varieties to ensure food, feed, fiber, and energy security. In recent years, rapid advances in UAVs have boosted the use of near-earth aerial imaging in various fields, providing low-cost data acquisition at high spatial-, spectral-, and temporal resolutions. Consequently, today UAVs have become essential platforms for cost-effective and high-throughput phenotyping [33], [34].

Thermal remote sensing cameras mounted on versatile and affordable UAVs have been increasingly used in precision agriculture, especially for detecting water stress and irrigation scheduling [7], [35], [36]. Significant progress has been made in UAV-based plant phenotyping and plant stress detection. One of the main challenges in this massive use of UAVs for agriculture, however, is the appropriate management of the swarm that can potentially optimize their tasks' accomplishments. RITMO has the potential to facilitate the utilization of UAV-based technologies by overcoming these limitations.

IV. Model

This section discusses the task migration problem amongst the UAVs and formulates a mathematical model to solve this problem. The system we envision is shown in Fig. 1.

A. System Model

We consider a network of N agents with the aim of complete a set of M tasks. Let $\mathcal{I} \triangleq \{1, \ldots, N\}$ and $\mathcal{J} \triangleq \{1, \ldots, M\}$ be the sets of node indexes and tasks, respectively. A task $j \in J$ indicates an atomic action executed by an agent, such as monitor a location, visit a target, or measure a quantity. To capture real-world scenarios, like the ones mentioned in Section III, we deal with heterogeneous agents (e.g., multicopter and fixed-wing vehicles with different sensors) that can execute only certain tasks.

Let us denote the amount of tasks of the i-th node as q_i within $[0, q_i^{max}]$, $q_i^{max} \in \mathbb{R}^+$. We also assume that the agent can execute only one task at a time, each one characterized by certain execution order. To avoid burdening the notation, task deadlines have not been considered.

We assume that at certain instants with a period r, a reassignment may take place. In other words, tasks can be moved among the agents to increase the performance of the network according to certain criteria. Furthermore, we assume that all new tasks created between two consecutive reassignments are put in the destination queue waiting for their actual allocation. For the sake of brevity, with the term "task offlaoding" or "task migration" we denote both the reallocation of existing tasks and the assignment of new ones. Given a generic device $i \in \mathcal{I}$ and a generic task $j \in \mathcal{J}$, we introduce the variable $x_{i,j} \in \{0, 1\}$ which is equal to 1 if the task j is assigned to the node i, and it is equal to 0 otherwise.

Let the matrix $\Theta \in \mathbb{B}^{N \times M}$, where each element $\Theta_{i,j}$ models the ability of the agent i to perform the task j. Clearly, each node is limited by its resources that is equipped with, i.e., CPU, memory, and bandwidth available on the node. Besides, this metric considers the heterogeneity of the agents as well as possible degradation due to malfunctions or hardware outages. If $\Theta_{i,j} = 0$ the agent i is unable to execute the task j, whereas if $\Theta_{i,j} = 1$ it can be executed at the best of the agent capabilities.

Task completion time. We consider that each task j has a processing time of $t_{i,j}^{proc}$ seconds, also depending on the node i. The total time of execution for the task j that traversed agents whose set of indexes is $\mathcal{P} \triangleq \{1, \ldots, P\} \subset \mathcal{I}$ is hence defined as:

$$\mathcal{N}_j = t_{P,j}^{proc} + \sum_{k=1}^{P} t_{k,j}^{w} + \sum_{k=2}^{P-1} t_{k-1,k,j}^{m}, \tag{1}$$

where $t_{k,j}^{w}$ denotes the waiting time for task j on the node k and $t_{k-1,k,j}^{m}$ refers to the migration time when the task leaves the node $k-1$ reaching node k.

If the task j is migrated, the data associated to task, δ_j, is sent to the destination node. Hence, we can characterize the migration time as:

$$t_{k-1,k,j}^{m} = \min_{l} \left\{ \sum_{t=i}^{i+l-1} R_t \geq \delta_j \right\}, \tag{2}$$

where i denotes the current time slot, $k-1$ is the source node, k is the destination node, and R_t is the data rate at time t. This quantity is clearly affected by the channel model, described in what follows (Section IV-B).

Task energy consumption. Besides the time to complete the task, we also contemplate the energy consumed during its execution. For each task j we define the energy required for its processing $E_{i,j}^{proc}$, also depending on the node i, as:

$$E_{i,j}^{proc} = P(i) \times t_{i,j}^{proc}, \tag{3}$$

where $P(i)$ is the computation power of the mobile device i. Specifically, each agent operates in a constant CPU speed s_i and a computation power $P(i)$ that is assumed to be a convex function of CPU speed s_i [37], where $i = 1, \ldots, N$.

However, the energy consumption model should also reflect the migration process. For this reason, we define the energy consumed for the transmission as:

$$E_{k-1,k,j}^{tran} = P(k-1,j) \times t_{k-1,k,j}^{m}, \tag{4}$$

where m_k was the migration time when the task leaves the node $k-1$, and $P(k-1,j)$ is the transmission power of node $k-1$ offloading task j. The total energy consumption for the task j that traversed agents whose index is in the set $\mathcal{P}$ is hence defined as:

$$\mathcal{E}_j = E_{i,j}^{proc} + \sum_{k=2}^{P} E_{k-1,k,j}^{tran}. \tag{5}$$

This formula reflects our assumption that the energy spent for the task to wait before its execution is negligible. We also simplify the formulation by not considering the power of receiving data on mobile device, since it is constant and often smaller compared to the transmitting power, as demonstrated by previous work [38].

B. Channel Model

Our algorithm uses the channel conditions to make replanning decision. In this section we show the details of the model that we used for the wireless channel between two nodes. In particular, we model the wireless channel as a random process of g_t under a time-slot scheme, where g_t denotes the channel gain at time t. The channel gain is a complex number whose magnitude is the attenuation of the signal and angle is the phase shift of the signal at a given time instant. We consider three alternative models, specifically: *(i) block-fading channel*, where the channel states $\{g_t\}$ do not change over the execution of the application [39]. Hence, the data rate of the channel R_t is constant over time; *(ii) IID stochastic channel*, where the random variables $\{g_t\}$ are independent and identically distributed (IID). This evolution affects the data rate R_t which is also IID over time; *(iii) markovian stochastic channel*, where the evolution of $\{g_t\}$ is a Markovian random process with a discrete state space. We also make the assumption that the transmission power on the node is fixed. As such, the data rate R_t is fully determined by the channel state of g_t.

In the case of the Markovian stochastic channel, we adopt the Gilbert–Elliott (GE) channel model [40], [41], where the channel conditions are classified into two states: "good" and "bad", denoted as G and B respectively. The two states correspond to a two-level quantization of the channel gain, i.e., when the measured channel gain is above some value, the channel is labeled as good, otherwise bad. Accordingly, we define the channel gain in the good state to be g_G, and in the bad state to be g_B. Therefore, the data rate, R_t can take two values, R_G and R_B, for the good and bad channel state, respectively,

$$R_t = \begin{cases} R_G & \text{if } g_t = g_G, \\ R_B & \text{if } g_t = g_B, \end{cases} \tag{6}$$

and the transition matrix of the channel state is:

$$\mathbb{P} = \begin{pmatrix} p_{GG} & p_{GB} \\ p_{BG} & p_{BB} \end{pmatrix} \tag{7}$$

These assumptions regarding the channel model affect the available data rate R_t, that in turn impacts the task migration time among two nodes. Also, the channel conditions are monitored by the agent in order to detect and avoid paths with low available bandwidth, as mentioned in Section V-C.

C. Problem Formulation

In the light of the aforementioned characterization, we are ready to expose the problem that RITMO aims to solve. More specifically, we formulate the problem as Integer Linear Program (ILP) for a system composed of N agents, each capable to handle a sequence of no more than q_i^{max} tasks, and M tasks that have to be reassigned between them. The objective is to minimize the completion time and the energy consumption for all the tasks in the system by controlling the decision variables $x_{i,j} \in \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$, in the program described as:

$$\min_{x} \quad \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j} x_{i,j} \tag{8}$$

$$\text{s.t.} \quad \sum_{j=1}^{M} x_{i,j} \leq q_i^{max} \quad \forall i \in \mathcal{I} \tag{9}$$

$$\sum_{i=1}^{N} x_{i,j} \leq 1 \quad \forall j \in \mathcal{J} \tag{10}$$

$$x_{i,j} \leq \Theta_{i,j} \quad \forall (i,j) \in \mathcal{I} \times \mathcal{J} \tag{11}$$

where the *cost value* $c_{i,j} \geq 0$ is a generic, nonnegative function of the assignment, to be minimized.

We can observe that (9) and the subsequent constraints limit the usage of resources to be at most the maximum available resources. Namely, the maximum number of tasks assignable to a node is q_i^{max}; a task can be assigned to no more than one agent; a task can be assigned only to agents that can execute it. Given these conditions, the cost function (8) expresses the desiderata of the migration process. The offloading procedure is executed to find a more efficient mapping between agents and tasks. A "good" offloading strategy should reduce as much as possible both energy consumption and the task completion time, which can potentially cause performance degradation. Therefore, the coefficients constituting the cost values account for these two aspects, but the way these values are accounted for depends on the utilized strategy, that is, the function that generates such costs $c_{i,j}$. For example, a cost function could consider the queue utilization q_i as a principal indicator. Alternatively, a more elaborate metric could quantify the

effectiveness of the offloading decision taken by the agent i by capturing both time to execute a task and energy consumed by the agent to execute a task, Part of our contribution is comparing the impact of different cost functions (or offloading policies) that generate the cost coefficients $c_{i,j}$ (Section V-B).

It may be noted that when the problem in (8) is a centralized optimization problem, it can be solved using well-known ILP algorithms such as the Branch and Bound technique [42]. In the rest of the paper we discuss our proposed decentralized strategy, based on local communications among agents in the network, to provide a solution to the offloading problem for heterogeneous agents (8)-(11). The distributed offloading problem can be summarized as follows.

Problem IV.1. *For each time interval r, each agent, denoted with an index i, has to decide if each task currently queued needs to be executed or migrated to another agent. When the agent decides to offload the task, it needs to select also the best destination agent according to its pre-configured offloading policy.*

We solve this problem with the RITMO offloading algorithm, detailed in the next section.

V. The RITMO Algorithm

Intending to solve the previous problem, we design an algorithm responsible for making online migration decisions. Such a decision determines whether and when the migration starts and, in this case, where the task should migrate.

A. Predicting the IoT device's load

Our presented migration mechanism leverages traditional regression algorithms to predict future values using history and its evolution in the past.The history used, hence, is composed of past values associated with the timestamp: the presence of such a tuple $< timestamp, value >$ leads to the name time series. Among the possible methods in this class of regressor, we select ARIMA [18] for its ability to account for trend and noise in collections of data. Hence, we formulate the task of load prediction, i.e., queue's length prediction, as a regression problem, where a real value number (future load) is predicted on the basis of many single input features (past load values).

Formally, a standard notation for this method is ARIMA(p, d, q), where the parameters account for seasonality, trend, and noise in datasets. In particular, *(i)* p captures the number of lag observations included in the model and is often denoted as auto-regressive component; *(ii)* d captures the integrated part of the model, i.e., the number of times that the raw observations are differenced, also referred to as the degree of differencing; *(iii)* q captures the moving average part of the model and refers to the extent of the moving average window, also denoted as the order of moving average. The ARIMA overall model is given by the following equation:

$$\left(1 - \sum_{i=1}^{p} \alpha_i L^i\right)(1 - L)^d y_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right)\epsilon_t, \quad (12)$$

where L is the lag operator, i.e., the number of past samples considered during the prediction; α_i are the parameters of the autoregressive part of the model; the θ_i are the parameters of the moving average while ϵ_t are error terms. Such error terms ϵ_t are commonly assumed to be independent and identically distributed (IID) variables sampled from a normal distribution with zero mean, which is what we did.

At each epoch t, the monitoring agent gathers information regarding the queue length. Such a number is thus inserted in chronological order and comprises the historical dataset used to build the model and perform the prediction. The data collection frequency is undoubtedly a key metric and largely depends on the processing time and task arrival rate. It is also affected by the prediction that occurs every r seconds. We set this time interval different to t to decouple the two actions, i.e., data collection and data prediction. In such a way, the granularity of collected data can be denser, and the model can leverage a larger history.

In light of these considerations, in the experiments, we set t to be half of the processing time to collect fresh data but not overload the node with the duty of collecting metrics. Whilst, we set r to be the processing time. However, these two values can be relaxed if prediction and task migration can occur less frequently.

At each prediction time r, the ARIMA's model, trained on the data collected over time, produces the one-step-ahead forecasting. In case such a predicted length exceeds a defined threshold z or the node runs out of available resources, the migration process begins.

B. Selecting the next IoT device

The task migration mechanism requires selecting a destination node, that can be chosen according to different policies. Such a policy can be used to minimize the cost of our problem (8), by varying the definition of cost value. Herein we enumerate some criteria for node selection, where each migration policy represents a different profile. The profile refers to the desirable load on each node, considering the available CPU, memory, and bandwidth resources of the node. Thus, prior to selecting the destination node, the system computes the time the hosting node would keep the job queued before execution. By considering the current and estimated node load, the system attempt to avoid nodes with a highly loaded queue that can hinder fast execution. In this regard, our solution is provided by default with a small yet representative set of migration policies:

(i) Load Balancing: one of the easiest schema, characterized by an equal distribution of tasks among all the available nodes. Specifically, the migration manager selects as destination the node with less enqueued tasks, and in case of more idle nodes, the destination is chosen uniformly within this subset. Formally, the index of destination m is given by:

$$m = \min_i \{q_1(t), \ldots, q_N(t)\}, \quad (13)$$

where $q_i(t)$ is the queue length of node i at time t.

(ii) Harmonic: it refers to a well-known randomized algorithm often employed to solve the k-server problem [43]. Such a problem consists in efficiently moving k servers over the nodes of a graph G in response to a set of requests,

where each request is a sequence of k-points. The harmonic policy aims at minimizing the total distance covered by the servers to reach the requested points. Despite the fact that our considered problem differs from the k-server problem, our strategy still uses a version of the Harmonic algorithm to select the destination. Formally, the probability of selecting the node m as destination is given by:

$$p_m = \frac{q_m(t+1)^{-1}}{\sum_i q_i(t+1)^{-1}}, \quad (14)$$

where $q_m(t+1)$ is the predicted queue's length of node m at time $t+1$.

(iii) Cost Minimization: during the execution, a profile of the available nodes is shaped, which takes into account the computation resources. Hence, the destination is chosen according to the cost of migrating, which depends on the average service time and energy consumption: a node with a lower estimated cost has a higher probability of being selected. We denote the cost of migrating task j from node i to node m as $c^{mig}_{i,j,m}$. The cost is computed as the sum of estimated completion time and estimated energy consumption. In turn, the cost differs in the case of a migration decision or a a local execution. In the former case, the service time is the sum of: *(i)* transmission time of task j migrating from i to m, $t^m_{i,j,m}$; *(ii)* waiting time on the node m, $t^w_{m,j}$; *(iii)* processing time for task j on the node m, $t^{proc}_{m,j}$. The energy consumption is also affected by the migration, as the sum of: *(i)* energy consumed for transmission of task m from i to j, $E^{tran}_{i,j,m}$, and *(ii)* energy required for processing of task m on the destination node m, $E^{proc}_{m,j}$. The two components are then weighted using α and β, and the cost of migration is formally:

$$c^{mig}_{i,j,m} = \alpha(E^{tran}_{i,j,m} + E^{proc}_{i,j}) + \beta(t^m_{i,j,m} + t^w_{m,j} + t^{proc}_{m,j}), \quad (15)$$

where α and β are two weights measuring the importance of completion time compared to the energy consumption.

Notably, the waiting time on the destination node $t^w_{m,j}$ is estimated using our regressor algorithm. Hence, with this policy, the prediction is not only used to establish when to migrate, but also to estimate the waiting time on the possible destination nodes. The migration time, instead, is estimated by looking at the channel metric collected by the node. Specifically, using the throughput of the wireless channel, the node can easily approximate the value of $t^m_{i,j,m}$.

In the latter case of local execution, the cost has still the two components. The service time is the sum of: *(i)* waiting time of task j on the origin node i, $t^w_{i,j}$; *(ii)* processing time on the node i, $t^{proc}_{i,j}$. The energy consumption only depends on the energy required for processing of task j on the source node i, $E^{proc}_{i,j}$. The two components are then weighted using α and β, and the cost of local execution is formally:

$$c^{loc}_{i,j} = \alpha E^{proc}_{i,j} + \beta(t^w_{i,j} + t^{proc}_{i,j}). \quad (16)$$

In this policy, the node compares the two costs and decides the most profitable one, and applies the selected strategy.

(iv) Closest Node: the system manager assigns the migrating task to the agent closest to the source node. The distance between source and destination has indeed a significant impact on the migration time and the energy spent for such transmission.

(v) Random: the migration destination node is randomly selected between all the available nodes. Despite being extremely easy, this strategy may result in good performance due to the small overhead introduced by the process of destination selection.

Using our provided APIs, the user can specify the preferred option according to the specific use case. Moreover, more strategies can be included in our solution whose logic can be easily adjusted, for example, setting a policy that considers more strict application requirements. It is also possible to assign priority to both nodes and tasks in order to avoid overloaded queues for highly important jobs. However, we demonstrate that this subset is sufficient for many network conditions, and we provide useful insights for efficiently set this policy in Section VII.

C. Channel monitoring

Channel and link estimation is a critical part of almost every sensor network protocol. In this paper, we let agents communicate directly with each other, with each node capable of monitoring the channel between itself and the rest of the fleet. Knowing the packet reception rate of candidate neighbors lets a protocol take the most energy-efficient decision, e.g., next routing hop. To this end, one basic indicator is the receive signal strength indicator (RSSI), which is the strength of a received radio signal [44]. Such a measure is implemented and widely-used in 802.11 standards. At larger distances, the signal gets weaker, and the wireless data rates get slower, leading to lower overall data throughput. Although it is recommended not to use this metric for distance measurements in localization algorithms [45], it is a promising indicator of how well a particular radio can hear the remotely connected client [46].

RSSI is, thus, a promising indicator when its value is above the sensitivity threshold, i.e., -87 dBm. Below this value, the packet reception rate (PRR) drastically downgrades due to variations in local phenomena such as noise. In this case, it may be difficult to sustain a link reliably or achieve high throughputs, especially in the presence of external interference. Conversely, above the sensitivity threshold, the PRR exceeds the 85%, denoting a strong signal. Despite the generality of these considerations and threshold value [46], [47], our architecture allows customizing this parameter as explained in Section VI.

D. Overall Procedure

The final goal of the algorithm underpinning RITMO is to minimize the completion time and the energy usage of any task. To accomplish this, our solution proactively migrates tasks between nodes. This migration is intended to release resources of overloaded agents and exploit the spare resources of other available nodes, to speed-up the computation. Two main questions about migration decisions need to be addressed by such a strategy: *(i) when* and *(ii) where*. *(i)* The first aspect is about time and refers to when performing the migration. We start a migration either when the predicted queue length outstrips a threshold or when the available resources on the

node are insufficient to complete the task. In these circumstances, the tasks in the queue are migrated to another node of the system, whose capacity can fulfill the demand and can complete them in a shorter time. *(ii)* The second aspect is about the location and refers to selecting a proper destination node to satisfy the system requirements. However, the decision about the destination often privileges a key metric at the price of degrading other quantities. Our system allows the user to choose from the options described in Section V-B according to the business logic. It can be, for example, that the key metric is the speed in deciding, the average usage of resources, or the average task completion time. This multitude of options originates diverse policies for the controller logic that we implemented in the system.

Algorithm 1 Prediction-based migration decision on any node

```
Let t be the epoch, and r the prediction period
Let z be the queue size threshold
for every epoch t do
    Monitor the queue and node state
    if notAvailableResources then
        dst ← get_dst(node, t)
        migrate remaining tasks in the queue to dst
    if r has elapsed since last prediction then
        q_{t+1} ← future predicted queue size on the node
        if q_{t+1} > z and channelIsGood(node) then
            dst ← getDst(node, t)
            migrate remaining tasks in the queue to dst
            close;
```

In algorithm 1 we summarize the main steps of our procedure. Every epoch t, the module running on each node obtains the statistics, saves them, and then provides them quantities to the predictor model. Such a model uses these quantities for the prediction that, occurring every period r, estimates the queue size at time $t+1$. The regressor computes the future size q_{t+1} and afterward compares this value to the threshold z, set as a quantity denoting when many tasks are enqueued. If q_{t+1} exceeds z, surplus tasks present in the queue at time t are moved to another node. The function $getDst(node, t)$ returns the destination node according to the selected policy, for example, one of the profiles described in Section V-B. Moreover, the migration can be triggered by the absence of available CPU or memory resources on board of the node i. In this case, the $notAvailableResources$ function returns true, initiating a new migration.

The $channelIsGood(node)$ function returns a boolean value according to the RSSI state, indicating the strength of the signal. As described in Section V-C, when an RSSI lower than a threshold denotes a signal too weak, subject to external interference. In these circumstances, it is recommended not to migrate tasks, and the function returns false. The desired RSSI is for values exceeding the threshold, commonly set to -87 dBm. However, we also consider a safety margin, and our function returns true for values above a safe threshold of -70 dBm.

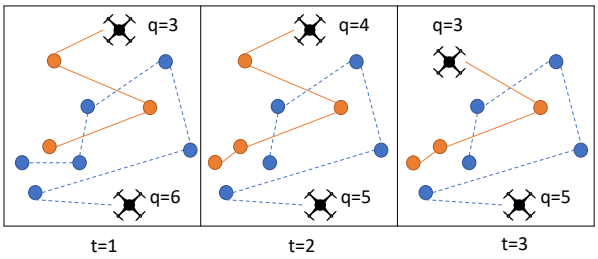


Fig. 2: System composed of 2 nodes has to execute 9 tasks. At time $t = 1$ tasks are assigned; then, at subsequent time instant $t = 2$, one task migrates and is reassigned as a consequence of the queue length prediction; finally, orange drone executes one tasks and its queue is thus reduced.

E. An Illustrative Example

To better understand the proposed procedure, we illustrate a simple example showing the task migration in a drone fleet. Considering a fleet of $N = 2$ unmanned aerial vehicles, 9 tasks are still pending and waiting for being executed ($M = 9$). We summarize the scenario in Fig. 2, where the tasks are different zones to be monitored. The nodes can vary for their characteristics, e.g., multicopter or fixed-wing, but in the following, we assume they have the same set of skills and tasks can be assigned indiscriminately to any node.

At the initial timestamp ($t = 1$), the orange drone has 3 tasks in this queue, while the blue one has 6 tasks left. As a result of the future load prediction, it appears that it is convenient that the blue drone migrates one task to the other agent. Although there is only one possibility in this example, the destination device should be selected on the basis of some policies, as mentioned previously (Section VII-E). After this migration ($t = 2$), however, we observe that the orange drone has one more task pending, and the blue one has reduced its queue length. Finally ($t = 3$), the orange drone accomplishes one task and moves to the next spot. The system evolves so forth, completing and migrating remaining tasks.

This simple example demonstrates the impact of task migration, which helps share the load among the agents of the system. This prevents the uncontrolled growth of the queue of any agent, moving tasks to under-congested nodes.

VI. RITMO Architecture

The solution presented has two main objectives: managing a fleet of IoT devices and efficaciously distributing the load among them. To this end, we design an architecture aiming to enable policy-based destination decisions. The system consists of multiple modules that can be replaced on-demand and in a short time to accommodate the requirements based on the peculiarities of the use case. In the following, we summarize the components of our system, e.g., the APIs, and the agent services offered.

We depict in Fig. 3 our management architecture, which enables the monitoring of network connectivity and the tasks reassignment, via estimation of the load on nodes and customizable controller logic. Our management layer, indeed, sits between the *Operating System* (at the bottom), e.g., Robotic

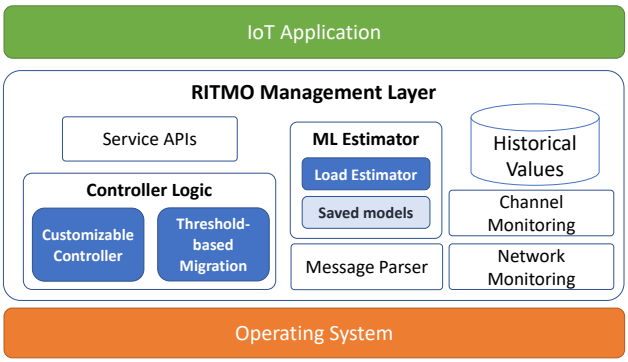


Fig. 3: RITMO's architecture: the management layer is located between the operating system and the IoT application. Some of the provided features are monitoring network connectivity, load estimation, and tasks migration, along with the customizable controller logic.

Operating System (ROS) [48], and the *IoT application* (at the top). The IoT application running on top can adapt to diverse business logics and environment settings by exploiting the provided API to customize the logic of such controllers, as well as to adapt the tasks migration logic to a centralized or distributed fashion. Depending on the application requirements, the architecture can implement the policy that best fits the context. An example of these applications is the set-up of disaster response for live audio/video analytic, as mentioned in Section III. During a disaster, since the network is very unstable, a policy as closest node or random can result extremely effective.

Service APIs enable the customization of two of the main components in RITMO: (i) the controller logic to fit multiple challenged scenarios, (ii) the logic of the task offloading algorithm, either in a centralized or distributed fashion. By interacting with this module, the same program can tailor different contexts, adapting to different requirements and network conditions. Another relevant component is represented by the *Historical Values*, committed to maintaining the past network states and the partially replicated database. These values constitute the past used by ARIMA for the prediction of future states. Still, other saved information is also historical dynamic states $i.e.$, depending on the network, configuration, and connectivity condition. Responsible for filling this database is *Network Monitoring*, that runs a watchdog process to monitor the other agents state and interact with them. A similar process is at the basis of *Channel Monitoring*, used to collect information about the channel between the node and the others. To the rescue of understanding the messages received as heartbeat comes the *Message Parser* module. Our object model is defined through Google Protocol Buffers [49], in charge of delimiting, serializing, and deserializing the messages.

The core of the adaptive migration mechanisms resided in the *Controller Logic* component, that can module the task migration rate of the network of IoT devices, e.g., drones. As explained in our algorithm (see Section V-D), we employ a *Threshold-based Migration*, that impose a migration every time that the predicted number of tasks on the agent exceeds the value of a threshold z. However, the architecture is modular and pluggable and can be extended with other user-defined controllers to replace the threshold-based logic.

The prediction of future network utilization is performed by the *ML estimator*, consisting of two sub-modules. The first key feature is the*Load Estimator*, as it estimates the future load exploiting the current and historical values. Such a prediction attempts to estimate the relationships between the features, i.e., system state, and a dependent variable, system load. To further reduce the training time, the prediction can take advantage of *Saved models*, obtained via offline training, so that the agent is not involved in the learning process. In this case, however, the advantage of a reduced overhead comes at the cost of models that can not adapt to conditions never experienced.

VII. Evaluation Results

In this section, we first analyze the accuracy of the ARIMA regressor compared to other time-series methods. Then, we describe how the channel model impacts the quality of RITMO and how decisions about destination nodes affect system performance. Finally, we compare RITMO against state-of-the-art solutions to determine the benefits of our solution, also considering the context of a surveillance application.

A. Experimental Setup

To evaluate the performance of the proposed task offloading strategy, we developed a C++ event-driven simulator, where a networked fleet of drones tries to accomplish a mission, corresponding to a set of geo-locations to reach. In this context, all drones cooperate to complete assigned tasks in the shortest possible time while reducing energy consumption. In case of disaster response, for example, each drone has indeed to explore an area with a camera and microphone looking for signals indicating survivors, as described in Section III. We set the default parameters for the evaluation to mimic the conditions that occur in this scenario. Therefore, we use the default threshold-based mechanism, i.e., the migration of drone's tasks is triggered when the queue length exceeds a threshold. Besides, if not otherwise specified, in our experimental campaign, we set the default configuration to considering 50 drones in the fleet, 1 m for the average distance between nodes and no failures (percentage 0%); the destination node is selected according to either the load balancing or cost minimization schema, as explained in what follows. Reported results are obtained after 35 trials, and the graph's bars refer to a confidence interval of 90%. Concerning the scenario, the arrival rate of new tasks was generated with a Poisson process of 0.02 Hz. The capacities of agents were assumed equal for all $a_i \in A$. Moreover, the processing time depends on the task to be executed and the hosting agent, but for simplicity in the following, we consider a fixed quantity, and we often refer to it as t_{proc}. In all cases, we assume that q_i^{max} is always large enough to assign all the tasks.

Table I summarizes the configuration parameters utilized during the following evaluation, where the default values are reported in bold. These default values are required to

uniform the comparison among different parameters under consideration, as in any experimental campaign. However, in the following, we also consider the impact and the reason behind the choice of some of them.

TABLE I: Parameters setting for the experimental campaign.

Parameter	Values
Number of nodes	10, **50**, 100, 150
Nodes' average distance [m]	**1**, 2, 3, 5, 10
Node failure [%]	**0**, 10, 50, 90
Transmission power, $P(k,m)$, [W]	0.1
Computation power, $P(i)$, [W]	0.5
Processing time, [s]	5, **10**, 20, 30, 50
Number of trials	**35**
Confidence interval [%]	**90**

B. Performance Indexes

In this section, we describe the performance indexes used to evaluate our and compared task offloading approaches. The key indicator we use is the average cost per task, Z, computed as follows:

$$Z \triangleq \frac{1}{M} \sum_{j=1}^{M} (\alpha \, \mathcal{N}_j + \beta \, \mathcal{E}_j). \tag{17}$$

In the following, we set $\alpha = 0.05$ and $\beta = 20$ in order to make the completion time comparable to the energy consumption.

To quantify the experimental error of the prediction, we utilize the absolute relative error given by the formula:

$$err = \frac{|y - \hat{y}|}{y}, \tag{18}$$

where y defines the actual value and $\hat{y}$ the predicted one.

C. Prediction Analysis & Accuracy

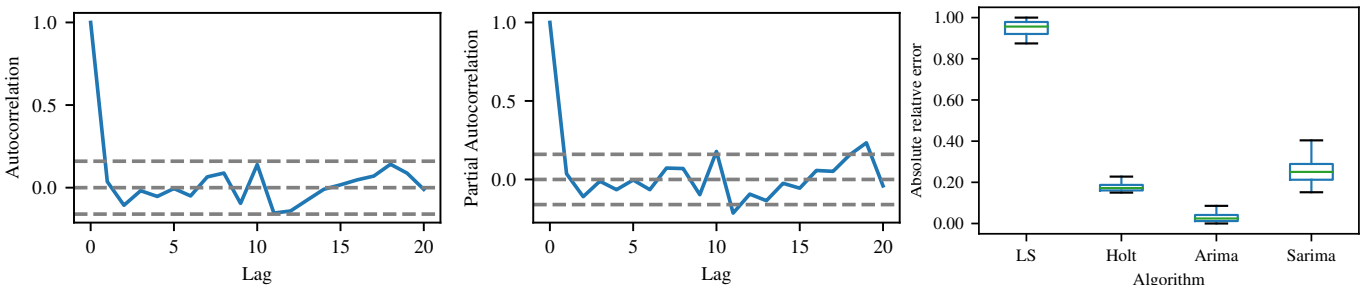
To opportunely choose the ARIMA's parameters, we carry out an initial study of the prediction performance. In particular, the ARIMA algorithm is influenced by three parameters: p, d and q, as mentioned in Section V-A. Two of these parameters for the algorithm configuration are derived from the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots (Fig. 4a and Fig. 4b): ACF is used to determine q while PACF for p. ACF is a common method to establish how well the present value of the series is related to its past values. On the other hand, PACF measures the correlation between the time series with a lagged, i.e., past, version of itself, but after eliminating the already found. The p and q values can be inferred from the figure as follows: *p* is the x-value at which the function of the PACF graph crosses the upper confidence interval (dotted line) for the first time [50]. Similarly, *q* is the x-value where the function of the ACF chart crosses the upper confidence interval for the first time. Experiments and results shown in Fig. 4 refer to our collected dataset comprised of more than $40,000$ historical samples, then split into training (80%), validation (10%), and test (10%) set. While the prediction error is computed on the test only, the parameters investigation is conducted on the validation set. From the graphs, it is possible to identify that $p = q = 1$. We further investigate empirically the optimal value of d using the cross-validation, and we found that $d = 1$ provides the best performance.

Moreover, we evaluate the accuracy of the ARIMA method in comparison to other time-series algorithms. A good predictor should at least outperform a very trivial algorithm in which the next value is the exact replica of the Last Sample (LS). Given the simplicity of the approach, it is not considered a statistical method; still it is a recommended baseline to establish the quality of the regressor algorithm. Besides LS, we study other two time-series alternatives: *(i) Holt-Winters (Holt)*, a basic model that captures three submodels (also known as influences) to fit a time series, i.e., an average value, a slope (or trend) over time and a cyclical repeating pattern (seasonality); *(ii) SARIMA*, following the same definition of the analogous ARIMA, it can also include seasonal components of the time series. This makes SARIMA able to deal with seasonal effects. Fig. 4c displays the error of the mentioned algorithms encountered during prediction, using the formula (18). The results show that the ARIMA outperforms the other solutions: the dynamic of the conditions makes inefficient a simple method like LS, and also Holt is unable to consider this data evolution. Besides, as the collected metric does not exhibit seasonality, SARIMA appears to be vain. We can hence conclude that ARIMA is the preferred choice to fit this context.

We then evaluate the time required to train the dataset, and we report it in Fig. 5. As mentioned, our choice is to offline train the model and then use it to online predict future values. As can be noticed, Holt-Winters is the longest to converge to a stable model, while the others demand less time to train. Furthermore, ARIMA is faster than SARIMA, given the reduced number of parameters to configure. However, the time needed for training for this class of algorithms is definitely shorter than the training time of other deep learning or reinforcement learning solutions [21].

D. Channel Model Impact

We then compare the impact of the diverse channel models over RITMO. Our solution executes all the tasks of the application under the three models: a simple block-fading channel with a constant data rate $R = 60$ kb/s, IID whose expected data rate is $\mathbb{E}(R) = 60$ kb/s, the Markovian channel characterized by $p_{GG} = 0.995$ and $p_{BB} = 0.96$, with $R_G = 100$ kb/s and $R_B = 10$ kb/s. For each channel model, we report in Fig. 6 the cost defined in (17) evaluated for the four migration policies, namely: *(i) load balancing*: when an agent's queue exceeds a set threshold, the drone with less waiting tasks is selected. In the case of two nodes with the same number of enqueued tasks, the system selects the closest agent. If still more nodes have the same properties, the destination is chosen randomly among them; *(ii) random*: the destination is a randomly selected; *(iii) closest*: when an agent's queue overcomes the threshold, the system reassigns its tasks to the closest agent. If two agents are at the same distance from the task, we insert the task in the queue of the drone with fewer tasks in its queue; ties are split at random if two queues have the same number of tasks; *(iv) cost minimization*: for each task we select the node



(a) (b) (c)

Fig. 4: Analysis of the regressor method. (a) Autocorrelation function (ACF) and (b) Partial autocorrelation function (PACF) for collected data, used for tuning the predictor's parameters p and q. (c) Error in queue length prediction for different time series algorithms, where ARIMA provides the higher accuracy.

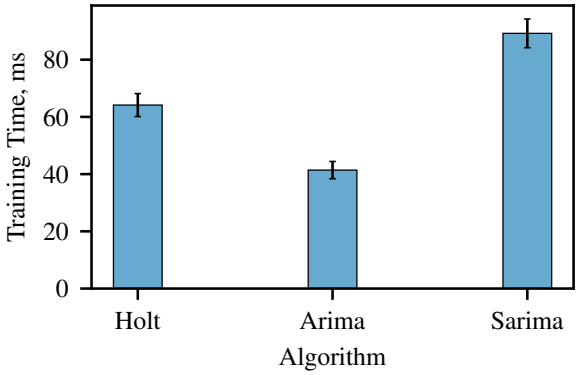


Fig. 5: Training time comparison for the diverse regressor algorithms. ARIMA is the fastest to converge.

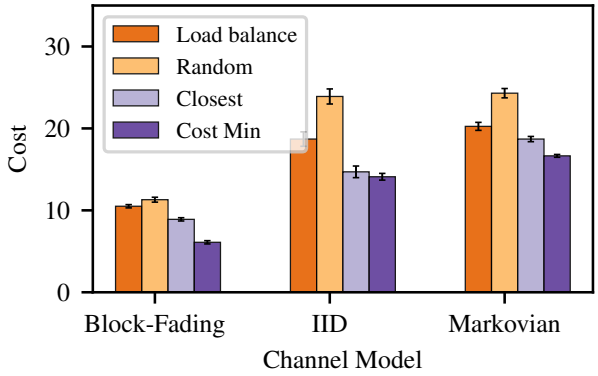


Fig. 6: Effects of diverse channel models. The four migration policies are evaluated for the *(i)* block-fading, *(ii)* independent and identically distributed (IID), and *(iii)* markovian.

which minimizes the migration cost of (15), if such a cost is lower than (16). Otherwise, the task is kept on the source node. Nevertheless, the purpose of this experiment is not to compare policies, that will be conducted in Section VII-E, but rather to analyze how the channel models affect the system performance.

First, it is fundamental to remark that the block-fading is conveniently used as a baseline approach. However, it poorly approximates real channel conditions by using optimistic assumptions, leading to lower costs than other models. We can then observe how the IID and Markovian channel models produce comparable results. For this reason, in the following, we could utilize one of these two options indiscriminately. However, we set as default Markovian due to the reduced variance in the obtained results and its more realistic model.

E. Consequence of Diverse Migration Policies

In this section, we compare the effects of diverse strategies for the destination node selection in a scenario with stable conditions and no failing agents (Fig. 7). Specifically, we evaluate (a) the task completion time and (b) the energy consumption for increasing nodes number, (c) completion time, and (d) energy consumption for an increasing percentage of node failures. In these circumstances, a few observations are: *(1)* Cost minimization policy is particularly effective when $N \leq 100$. In fact, for a partially limited number of agents, the state of channels and other nodes can be monitored, and this information ameliorates the decision. However, when this number rises, the overhead in accounting these metrics rises as well, leading to intractable model. This increases both the task completion time and the energy consumption, and is reflected in the cost metric which for $N = 150$ is the highest among the other policies. *(2)* Closest agent and random policies, even if very trivial, cut off costs for large drone swarms. From the graphs can be observed that they shorten task completion time and can also diminish the average energy needed to perform a task. When the options for migration magnify, it may be convenient to decide quickly, as in closest and random, even if less accurate. In fact, in these circumstances, minimizing the expected cost incurs in high complexity that is often unnecessary. *(3)* For large fleet, e.g., $N = 150$, load balancing tasks among agents provides the lowest costs. Similarly to the second observation, we observe that when the fleet size increases, simple decisions are preferred. This profile can indeed balance the effect of lower complexity and decisions based on the current load, and emerges as preferable choice.

According to these considerations, in what follows, we set the *cost minimization* policy when the number of drones is lower than 100, and *load balance* for a higher number of nodes.

F. RITMO Performance

Furthermore, we compare our solution against two of the most related methods that are found in the literature: HAP [15]

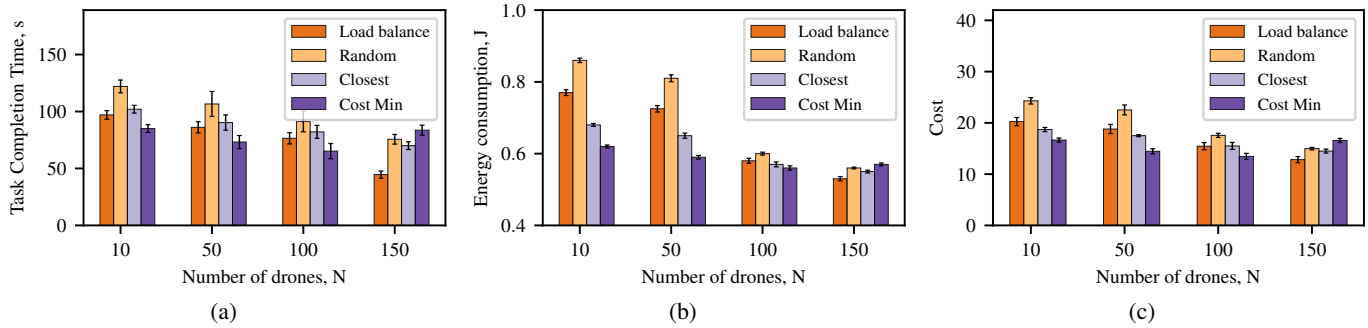


Fig. 7: Destination selection policies comparison. (a) Task completion time, (b) energy consumption, and (c) system cost of a fleet of drones using APRON with different migration policies: *(i)* load balancing, *(ii)* random, *(iii)* closest, *(iv)* cost minimization task migration.

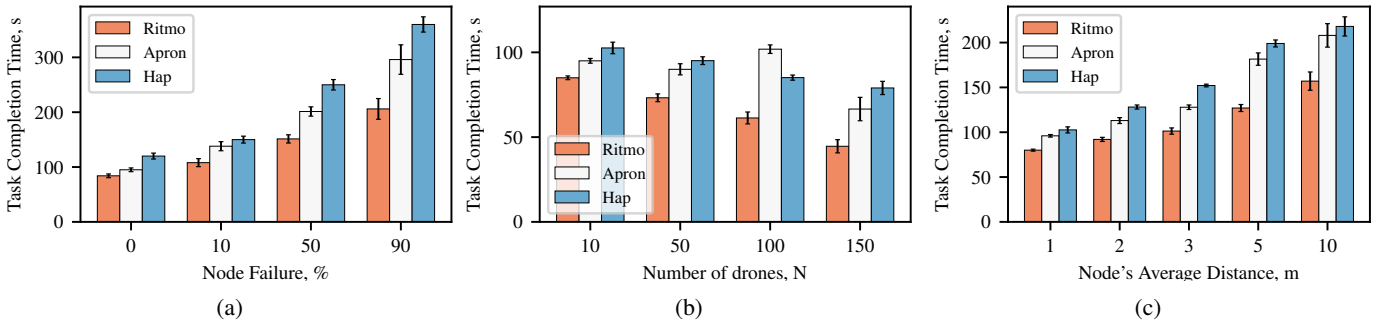


Fig. 8: System performance evaluation in terms of time to complete tasks. (a) Comparison of different solutions at varying percentage of node failures. (b) Completion time of different algorithms for an increasing number of nodes. (c) Effect of the average distance between nodes on the task completion time.

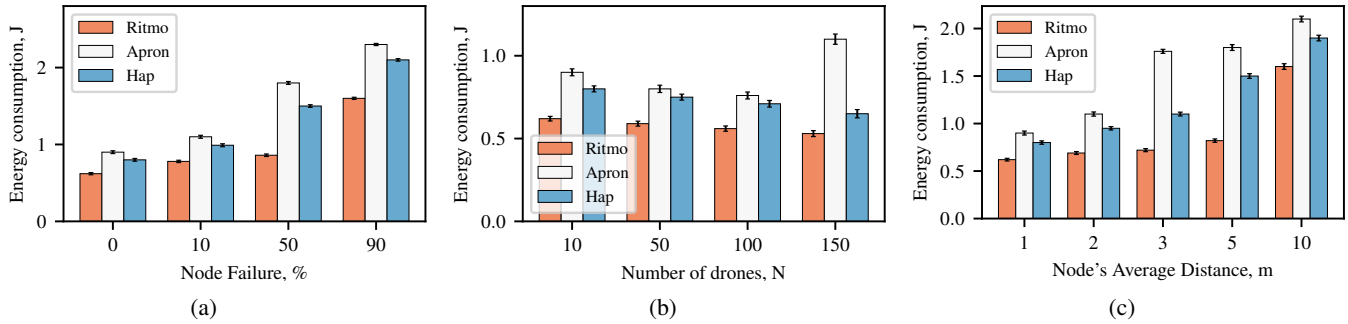
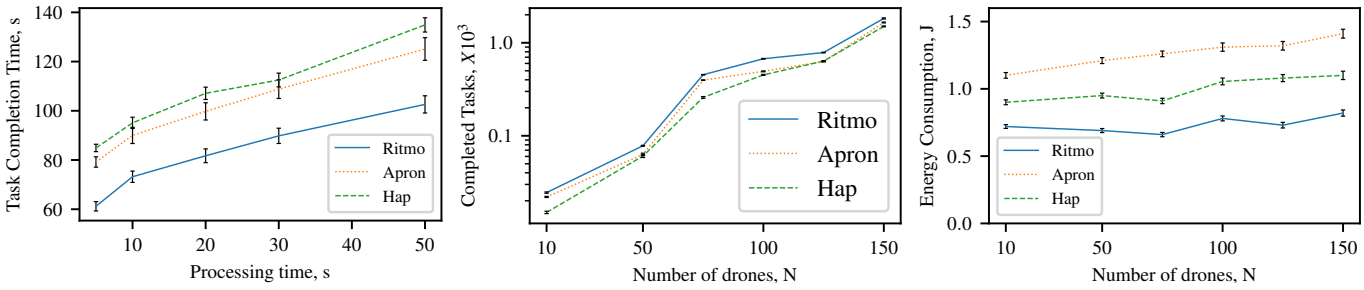


Fig. 9: System performance evaluation in terms of energy consumption, at varying (a) percentage of node failures. (b) number of agents in the system. (c) the average distance between two agents.

and APRON [26]. The former can establish close feedback between the high-level planning based on a Markov Decision Process (MDP) and the execution level. Using this feedback, it then anticipates failures at the planning level. The latter approach, instead, exploits Jackson's network model to control operations of a network of IoT devices while the network states evolve. Although this work is presented with several options to select the destination, in the following, we employ the closest node policy since it has been shown that this setting provides better results [26].

In Fig. 8a we show the average execution time of tasks for the three algorithms when the percentage of failures varies. It can be observed how RITMO achieves the shortest completion time with respect to analogous solutions, considering its ability to manage a large number of failures by reassigning the uncompleted tasks pro-actively and re-actively. We also compare the completion time for an increasing number of nodes in the system in Fig. 8b. As the fleet grows up, RITMO can exploit all the available resources of agents without overloading them, diminishing tasks' completion time. This ability makes RITMO outperforming the other solutions, as can be seen in the graph. In particular, the benefits brought by our solution comes even higher when the number of nodes increases. In such a scenario, indeed, APRON is not always able to take advantage of the more available resources offered by more drones in the system, while RITMO can execute a more profitable migration. Moreover, we consider the impact of the average distance among two nodes over the system performance (Fig. 8c). Clearly, as the distance increases, the task completion time rises as well. Our findings, however, show how RITMO provides better performance even when it may be challenging to handle the agent locations, which indicates the efficiency of our proposed system.

Another important aspect is the sustainability of the so-



(a) (b) (c)

Fig. 10: (a) Impact of different values of processing time on the average time spent to complete a task. (b) - (c) For the use case of a disaster response, RITMO is able to provide the highest number of completed tasks in the interval and lowest energy consumption.

lution, and, consequently, we evaluate the energy consumed during the mission for the different algorithms (Fig. 9). We replicate the same conditions of Fig. 8, but we report now the average energy consumption (the mean between nodes). We can immediately observe how RITMO consistently lowers energy consumption compared to other benchmark solutions. In particular, RITMO provides stronger results when the challenged conditions are exacerbated, e.g., remarkable percentage of failures (Fig. 9a) and large fleet of drones (Fig. 9b). Remarkably, for $N = 150$ RITMO halves the consumption with respect to APRON solution.

During our experimental campaign, we also evaluated the impact of various task processing times (Fig. 10a). We can observe how this time only partially affects the completion time, which, instead, largely depends on the time spent waiting for the execution. The migration is meant to reduce this time. Therefore, even for heavier tasks, the solution moves jobs among the agents to reduce this waiting time.

G. Benefits to the On Top Application

Lastly, we consider the performance when RITMO is employed for a monitoring surveillance use case, as the one presented in [1], which entails a disaster response setup. The scenario includes an edge cloud where information is sent to the close computation, i.e., video record, and a corresponding task is received, i.e., the location to explore with the camera. Thus, we evaluate the specific metrics for this application, and we quantify the number of completed tasks and the average energy spent during the execution of the application in Fig. 10b and Fig. 10c, respectively. The experiments refer to 5-minutes of execution, and a task takes 5 seconds. Comparing RITMO against the other approaches, we can notice how our solution can increase the number of completed tasks by the system. At the same time, RITMO reduces the average energy spent by each agent.

VIII. Conclusion

This paper presents RITMO, a solution managing a fleet of robotic agents, e.g., drones, to increase the resilience in task replanning and migration problems. In the presence of challenged edge networks, indeed, diminishing the time taken to complete assigned tasks while not overloading agents is particularly challenging. To this end, RITMO models the network of nodes as a network of queues and predicts the number of future tasks in the agent's queue. This information is thus used to determine the IoT device's future utilization and proactively redistributes tasks among the fleet. Our results show that RITMO is notably effective as a policy programmability mechanism for networks of UAVs. In particular, our solution jointly shortens the task completion time and energy consumption with respect to other benchmark solutions.

IX. Acknowledgement

This work has been partially supported by NSF awards CNS-1647084, CNS-1836906, and CNS-1908574. We are also indebted to Donato Di Paola for motivating us to investigate this problem and for his contributions to our earlier research on the topic.

References

[1] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "An architecture for adaptive task planning in support of iot-based machine learning applications for disaster scenarios," *Computer Communications*, vol. 160, pp. 769 – 778, 2020.

[2] D. Chemodanov, F. Esposito, A. Sukhov, P. Calyam, H. Trinh, and Z. Oraibi, "Agra: Ai-augmented geographic routing approach for iot-based incident-supporting applications," *Future Generation Computer Systems*, vol. 92, pp. 1051–1065, 2019.

[3] R. Charney, T. Rebmann, F. Esposito, K. Schmid, and S. Chung, "Separated after a disaster: trust and privacy issues in sharing children's personal information," *Disaster medicine and public health preparedness*, p. 1–8, 2018.

[4] S. C. Rajashekar, S. Gururajan, F. Esposito, and D. Ferry, "Reconfigurable swarms and multi-user, cooperative uas flights through a virtual reality interface," in *AIAA Scitech 2020 Forum*, 2020, p. 0737.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb 2018.

[7] V. Sagan *et al.*, "Uav-based high resolution thermal imaging for vegetation monitoring, and plant phenotyping using ici 8640 p, flir vue pro r 640, and thermomap cameras," *Remote Sensing*, vol. 11, no. 3, p. 330, 2019.

[8] A. V. Ventrella, F. Esposito, and L. A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 80–87.

[9] W. Muhammad, F. Esposito, S. C. Rajashekar, and S. Gururajan, "Vocal intent programmability for uas in disaster scenarios," in *AIAA Scitech 2020 Forum*, 2020, p. 0736.

[10] N. H. Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, 2016.

[11] J. Franz, T. Nagasuri, A. Wartman, A. Ventrella, and F. Esposito, "Reunifying families after a disaster via serverless computing and raspberry pis (demo)," in *IEEE International Symposium on Local and Metropolitan Area Networks*, Washington, DC, June 2018.

[12] A. Sacco, F. Esposito, and G. Marchetto, "A federated learning approach to routing in challenged sdn-enabled edge networks," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 150–154.

[13] J.-S. Marier, C. A. Rabbath, and N. Léchevin, "Health-Aware Coverage Control With Application to a Team of Small UAVs," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 5, pp. 1719 – 1730, September 2013.

[14] N. Kemal Ure *et al.*, "Decentralized learning-based planning for multiagent missions in the presence of actuator failures," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2013.

[15] N. K. Ure, G. Chowdhary, J. P. How, M. A. Vavrina, and J. Vian, "Health aware planning under uncertainty for uav missions with heterogeneous teams," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 3312–3319.

[16] Choi, Han-Lim *et al*, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. on Robotics*, Aug 2009.

[17] A. Sacco, F. Esposito, and G. Marchetto, "Resource inference for task migration in challenged edge networks with ritmo," in *Proceedings of the 9th IEEE International Conference on Cloud Networking (CloudNet '20)*. IEEE, 2020, pp. 1–7.

[18] G. E. Box, G. M. Jenkins, and G. C. Reinsel, *Time series analysis: forecasting and control*. John Wiley & Sons, 2011, vol. 734.

[19] S. S. Ponda, H.-L. Choi, and J. P. How, "Predictive planning for heterogeneous human-robot teams," in *InfoTech*, 2010.

[20] C. J. Shannon, L. B. Johnson, K. F. Jackson, and J. P. How, "Adaptive mission planning for coupled human-robot teams," in *American Control Conference (ACC), 2016*. IEEE, 2016, pp. 6164–6169.

[21] A. Sacco, F. Esposito, and G. Marchetto, "Rope: An architecture for adaptive data-driven routing prediction at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 986–999, 2020.

[22] A. Duminuco *et al.*, "Proactive replication in distributed storage systems using machine availability estimation," in *CoNEXT*, 2007.

[23] B. Bethke, J. How, and J. Vian, "Multi-uav persistent surveillance with communication constraints and health mangement," in *AIAA Guidance, Navigation, and Control Conference*, 2009, p. 5654.

[24] J. Redding, Z. Dydek, J. P. How, M. A. Vavrina, and J. Vian, "Proactive planning for persistent missions using composite model-reference adaptive control and approximate dynamic programming," in *American Control Conference*, 2011.

[25] D. Di Paola, M. Gaggero, A. Petitti, and L. Caviglione, "Optimal control of time instants for task replanning in robotic networks," in *2016 American Control Conference (ACC)*. IEEE, 2016, pp. 1993–1998.

[26] A. V. Ventrella, F. Esposito, A. Sacco, M. Flocco, G. Marchetto, and S. Gururajan, "APRON: an Architecture for Adaptive Task Planning of Internet of Things in Challenged Edge Networks," in *Proceedings of the 8th IEEE International Conference on Cloud Networking (CloudNet)*. IEEE, 2019, pp. 1–6.

[27] H. Trinh *et al.*, "Energy-aware mobile edge computing for low-latency visual data processing," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2017, pp. 128–133.

[28] S. Chung, C. Mario Christoudias, T. Darrell, S. I. Ziniel, and L. A. Kalish, "A novel image-based tool to reunite children with their families after disasters," *Academic emergency medicine*, vol. 19, no. 11, pp. 1227–1234, 2012.

[29] P. Boccardo, F. Chiabrando, F. Dutto, F. G. Tonolo, and A. Lingua, "Uav deployment exercise for mapping purposes: Evaluation of emergency response applications," *Sensors*, vol. 15, no. 7, pp. 15 717–15 737, 2015.

[30] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "Sustainable task offloading in uav networks via multi-agent reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 5003–5015, 2021.

[31] A. Walter, F. Liebisch, and A. Hund, "Plant phenotyping: from bean weighing to image analysis," *Plant methods*, vol. 11, no. 1, pp. 1–11, 2015.

[32] V. Sadras, G. Rebetzke, and G. Edmeades, "The phenotype and the components of phenotypic variance of crop traits," *Field Crops Research*, vol. 154, pp. 255–259, 2013.

[33] S. Madec, F. Baret, B. De Solan, S. Thomas, D. Dutartre, S. Jezequel, M. Hemmerlé, G. Colombeau, and A. Comar, "High-throughput phenotyping of plant height: comparing unmanned aerial vehicles and ground lidar estimates," *Frontiers in plant science*, vol. 8, p. 2002, 2017.

[34] G. Yang, J. Liu, C. Zhao, Z. Li, Y. Huang, H. Yu, B. Xu, X. Yang, D. Zhu, X. Zhang *et al.*, "Unmanned aerial vehicle remote sensing for field-based crop phenotyping: current status and perspectives," *Frontiers in plant science*, vol. 8, p. 1111, 2017.

[35] D. J. Mulla, "Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps," *Biosystems engineering*, vol. 114, no. 4, pp. 358–371, 2013.

[36] C. Z. Espinoza, L. R. Khot, S. Sankaran, and P. W. Jacoby, "High resolution multispectral and thermal remote sensing-based water stress assessment in subsurface irrigated grapevines," *Remote Sensing*, vol. 9, no. 9, p. 961, 2017.

[37] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2005, pp. 303–314.

[38] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing." *HotCloud*, vol. 10, no. 4-4, p. 19, 2010.

[39] E. Biglieri, J. Proakis, and S. Shamai, "Fading channels: Information-theoretic and communications aspects," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2619–2692, 1998.

[40] M. Zafer and E. Modiano, "Minimum energy transmission over a wireless fading channel with packet deadlines," in *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 1148–1155.

[41] L. A. Johnston and V. Krishnamurthy, "Opportunistic file transfer over a fading channel: A pomdp search theory formulation with optimal threshold policies," *IEEE Transactions on Wireless Communications*, vol. 5, no. 2, pp. 394–405, 2006.

[42] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[43] Y. Bartal and E. Grove, "The harmonic k-server algorithm is competitive," *Journal of the ACM (JACM)*, vol. 47, no. 1, pp. 1–15, 2000.

[44] R.-H. Wu, Y.-H. Lee, H.-W. Tseng, Y.-G. Jan, and M.-H. Chuang, "Study of characteristics of rssi signal," in *2008 IEEE International Conference on Industrial Technology*. IEEE, 2008, pp. 1–3.

[45] A. T. Parameswaran, M. I. Husain, S. Upadhyaya *et al.*, "Is rssi a reliable parameter in sensor localization algorithms: An experimental study," in *Field failure data analysis workshop (F2DA09)*, vol. 5. IEEE, 2009.

[46] K. Srinivasan and P. Levis, "Rssi is under appreciated," in *Proceedings of the third workshop on embedded networked sensors (EmNets)*, vol. 2006. Cambridge, MA, USA., 2006, pp. 239–243.

[47] K. Benkic, M. Malajner, P. Planinsic, and Z. Cucej, "Using rssi value for distance estimation in wireless sensor networks based on zigbee," in *2008 15th International Conference on Systems, Signals and Image Processing*. IEEE, 2008, pp. 303–306.

[48] Robotic Operating System. http://www.ros.org/.

[49] Google Protocol Buffers. http://code.google.com/apis/protocolbuffers.

[50] J. H. F. Flores, P. M. Engel, and R. C. Pinto, "Autocorrelation and partial autocorrelation functions to improve neural networks models on univariate time series forecasting," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, June 2012, pp. 1–8.

Alessio Sacco received the M.Sc. degree in Computer Engineering from the Politecnico di Torino, where he is currently pursuing the Ph.D. degree in Computer Engineering. His research interests include architecture and protocols for network management; implementation and design of cloud computing applications; algorithms and protocols for service-based architecture, such as Software Defined Networks (SDN), used in conjunction with Machine Learning algorithms.

Flavio Esposito is an Assistant Professor with the Department of Computer Science at Saint Louis University (SLU). He also has an affiliation with the Parks College of Engineering at SLU. He received an M.Sc. degree in Telecommunication Engineering from the University of Florence, Italy, and a Ph.D. in computer science from Boston University in 2013. Flavio worked in the industry for a few years, and his main research interests include network management, network virtualization, and distributed systems. Flavio is the recipient of several awards, including four National Science Foundation awards and two best paper awards, one at IEEE NetSoft 2017 and one at IEEE NFV-SDN 2019.

Guido Marchetto (M'06-SM'21) received the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2008, where he is currently an Associate Professor with the Department of Control and Computer Engineering. In 2009, he visited the Department of Computer Science at Boston University. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures. He is Senior Member of the IEEE and he serves as an Associate Editor of the IEEE Transactions on Vehicular Technology.