Computer Communications 160 (2020) 0–9

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

An architecture for adaptive task planning in support of IoT-based machine learning applications for disaster scenarios

Alessio Sacco [a,*], Matteo Flocco [b], Flavio Esposito [b], Guido Marchetto [a]

[a] Department of Control and Computer Engineering, Politecnico di Torino, Italy
[b] Department of Computer Science, Saint Louis University, USA

ARTICLE INFO

Keywords:
Network of queues
Machine Learning

ABSTRACT

The proliferation of the Internet of Things (IoT) in conjunction with edge computing has recently opened up several possibilities for several new applications. Typical examples are Unmanned Aerial Vehicles (UAV) that are deployed for rapid disaster response, photogrammetry, surveillance, and environmental monitoring. To support the flourishing development of Machine Learning assisted applications across all these networked applications, a common challenge is the provision of a persistent service, i.e., a service capable of consistently maintaining a high level of performance, facing possible failures. To address these service resilient challenges, we propose APRON, an edge solution for distributed and adaptive task planning management in a network of IoT devices, e.g., drones. Exploiting Jackson's network model, our architecture applies a novel planning strategy to better support control and monitoring operations while the states of the network evolve.

To demonstrate the functionalities of our architecture, we also implemented a deep-learning based audio-recognition application using the APRON NorthBound interface, to detect human voices in challenged networks. The application's logic uses Transfer Learning to improve the audio classification accuracy and the runtime of the UAV-based rescue operations.

1. Introduction

Recent years have witnessed the proliferation of mobile computing and Internet-of-Things (IoT), where billions of mobile and IoT devices are connected to the Internet, generating large datasets to be consumed by several (distributed) applications. A subset of these applications requires IoT devices to be separately programmed to perform a mission independently. Typical examples of such scenarios are heterogeneous networks composed by Unmanned Aerial Vehicles (UAVs), e.g., drones, and other IoT sensors, that together connect a plethora of sensors, including hyperspectral cameras, microphones, or civilian tablets and smartphones [1,2]. These systems have been employed in the past with success to support first responders in man-made or natural disaster scenarios [3–8]. The role of drones in the IoT in general, and in disaster response in particular, could become even more prominent in the future as they have the potential to enable, improve, and optimize novel and existing rescue operations and services. More broadly, autonomous and semi-autonomous drones will undoubtedly continue to help humans also in other tasks, spanning from industrial inspection to survey operations to military operation support.

A network of drones can be used to collect large quantities of data, that can then be uploaded at the edge of the network for heavy audio/video processing, where resources to execute Machine Learning (ML) algorithms are available [3,4].

In the conditions imposed by challenged networks such as those present after a natural disaster scenario, keeping such IoT devices well-functioning could be a challenge [3,9]. Although delay and disruption tolerant protocols and architectures exist [10], the problem of maintaining an acceptable quality of service with stringent delays for these networks depends not only on the quality of the connectivity, but also on the dynamic nature of the tasks that the drones are required to accomplish.

Both centralized [11,12] and distributed [13,14] approaches that allow an edge network of IoT devices, drones, or robots in general, to provide a persistent and adaptive service already exist. Some of them focus on the resilient mission planning problem [14], others on agents' health-aware solutions [13]. Others yet [12] concentrate on the problem of enabling multi-agent teams to autonomously tackle complex, large-scale missions, over long time periods in the presence of actuator failures.

These solutions have sound design, and they address different failure models under specific applications, but a unique solution that ensures a resilient drone mission execution, under all possible failure

* Corresponding author.
E-mail addresses: alessio_sacco@polito.it (A. Sacco), matteo.flocco@slu.edu (M. Flocco), flavio.esposito@slu.edu (F. Esposito), guido.marchetto@polito.it (G. Marchetto).

https://doi.org/10.1016/j.comcom.2020.07.011
Received 1 December 2019; Received in revised form 6 July 2020; Accepted 7 July 2020
Available online 11 July 2020

models and applications probably cannot exist. To this end, we propose an Architecture for the Programmability of RObotic Networks (APRON), extending our preliminary results presented in [15] with a more in-depth evaluation and a practical example of AI application for UAVs. The architecture enables the programmability of different mechanisms involved in the mission execution problem of UAVs or other edge-based distributed agents. APRON is a software layer that sits between the (robotic) operating system (*e.g.*, ROS) [16]) and any IoT software application. The APRON architecture contains classical network management mechanisms, such as network monitoring, repair, and control operations *e.g.*, neighbor discovery, as well as mechanisms specific to the resilient mission execution problem, *e.g.*, adaptive control, and neighbor failure estimation. Finally, it provides a NorthBound interface for application programmers.

Our contribution in this paper is two-fold. First, we detail our middleware architecture for IoT device management and propose an optimization algorithm for task re-scheduling, in case of an IoT device experiencing a failure. Second, we introduce a disaster response application as an AI use case application that uses APRON's underlying resilient network services. Our application is designed to detect sounds generated by humans, for example, those that are victims of a natural or man-made disaster to be rescued, or survivors to locate under an avalanche, where video alone may be insufficient. In particular, our edge audio processing application uses Deep Neural Networks (DNNs) techniques to classify the audio sent from the drone fleet and helps locate human sounds. The speed is crucial for rescue operations; hence, we explored and exploited the properties of *Transfer Learning* to reduce the training time and increase the classification accuracy (human/non-human sound).

We also used a Mission Allocation Simulator [17], to test the scalability of our approach, and deployed our solutions over a prototype powered by virtual network testbed to evaluate the practicality of APRON.

Throughout this paper, in Section 2 we describe some applications where APRON can be used. Although inspired by the disaster response use case, our approach has indeed broader applicability. We present the problem in Section 3, then we describe in detail the *load and failure estimator* component (Section 4), that leverages a Jackson's network model to support monitoring and control operations while the states of the network evolve. The estimator computes a close form of the average number of tasks in a mission, whether they are queued or in execution. Such an estimator can then be manipulated by application programmers to determine the utilization of each drone and the mean queuing time (both waiting and execution time) for each task. Such information can then be used (in conjunction with our APRON API) to design controllers that adapt to specific applications. We overview the components of our APRON architecture in Section 5, focusing on its *Controller* component and how programmability is achieved.

The audio processing application is presented in Section 6, providing details about our enhanced DNNs model. We present our experimental results in Section 7, while the presentation of the literature is in Section 8 and Section 9 concludes our paper.

2. Motivating applications

Although we mainly focus on the disaster response scenario, we argue this solution can be utilized in many other use cases, which are affected by comparable problems. In the following, we briefly describe two applications whose requirements can be satisfied by using the presented approach.

Disaster Response. Unmanned Aerial System (UAS, i.e., drone) swarms for disaster area search and rescue are important examples of intelligent physical systems that could benefit from APRON: they can autonomously support high-level semantic interface capabilities in uncertain work environments with a minimum of operator supervision. UAS search swarms are ideal for quickly locating survivors and identifying emergent threats following earthquakes or other natural (or man-made) disasters that render structures damaged and potentially unsafe to enter. Search swarm techniques are also broadly applicable to other surveillance applications such as structural inspection, agricultural surveillance, and post-disaster family reunification via facial recognition, to name a few. The need for skilled operators limits how current generation UAS platforms can be further integrated alongside first responders, while training and budgetary constraints prevent existing lifesaving organizations from adopting this technology broadly. Autonomous search swarms will integrate with and be usable by organizations of any size and capability. For such a system to be effective, a variety of technical challenges must be addressed; UAS platforms are dramatically resource-constrained with flight times of tens of minutes, limited on-board computational capacity, power and weight limitations for sensing, and often unreliable radio links to ground operator stations. Deploying a truly intuitive tasking mechanism for such a system requires: (1) interpreting high-level semantic commands such as hand gestures or natural language, and the autonomy to execute such orders with little to no operator intervention; (2) continuous communication of the swarms' present capability to execute orders, depending on the availability of resources such as power and current computational capacity; and (3) automatic management and maximization of those resources subject to mission objectives.

Intelligent Transportation Systems. The application of Information and Communication Technologies (ICT) to the transport sector made the classic transportation systems evolving towards an Intelligent Transportation System (ITS). Sensing, analysis, control, and communications technologies are applied to transportation in order to improve efficiency, sustainability, safety, mobility, environment impact, and comfort. Examples of ICT where APRON could be helpful are Advanced Driving Assistance Systems technologies that provide collision avoidance and driver aids, such as night vision, driver alertness, and adaptive cruise control or even fully autonomous driving systems. The reason why APRON can be useful lays in the characteristic of these systems: (*i*) high number of devices and produced data; (*ii*) wireless communications among vehicles, and between vehicles and the roadside infrastructure are inherently unstable and exacerbated by the high mobility of the (IoT) agents; (*iii*) collaboration among agents to exploit the sensory data; (*iv*) real-time, event-triggered, asynchronous and periodic generated traffic; (v) reliability and safety; (vi) cost-effective and user-friendly. In case of low-latency and location-aware services, the on-board processing of the data implies a significant processing power that is not suitable for resource-constraint devices; the processing on a centralized and far cloud implies problems due to bandwidth and latency; while, a more viable option is the delegation of the computation to the edge system, because the collection and processing of the data closer to its source reduce latencies and traffic loads to the cloud.

3. Problem definition: Mobile task offloading

One example of a harsh environment where frequent connectivity losses occur due to infrastructure problems is natural or man-made disaster scenarios. We define in this section the general problem of tasks offloading from a fleet of nodes or agents, e.g., drones, to a close edge computing server. While more specific scenarios affected by the same issues are mentioned in Section 2, and Section 6 describes the application that we developed to provide the required heavy computation for the audio processing.

3.1. Problem definition

The task offloading solutions enable the execution of complex jobs in nearby surrogate machines, often called cloudlets, instead of running them on a mobile node [18]. Typically the cloudlets are near the mobile

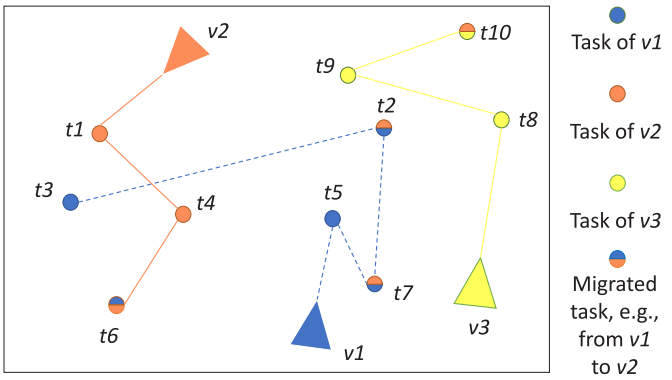

Fig. 1. Example of mission planning paths followed by three drones to complete their tasks. Migration occurs among nodes to balance the overall load. We use this figure with only a few drones to clarify the migration process and workflow, but we tested APRON's scalability in subsequent experiments.

devices and are reachable via a small edge network, to ensure low-latency connections. However, in critical scenarios, such as for disaster response, two problems need to be addressed: *(i)* a large number of failures occur because of hostile conditions, *(ii)* edge nodes are scarce and often overloaded. For these reasons, it is likely to observe unacceptable delays and significant losses, leading to an increase in average job completion time. We hence face the following challenge:

Problem 3.1. Given a set of devices offloading a set of computationally intensive jobs on an edge computing server, e.g., Ground Control System (GCS), we define the Task Offloading Problem as the edge network management problem minimizing the average completion time of an offloaded set of tasks by effectively orchestrating the load on the underlying infrastructure.

The programmable edge computing load orchestration entails two main processes: *(i)* enforcing a given load profile on edge agents, *(ii)* migrating the tasks whose expected running time is considerably high to another node that most likely completes it within a shorter time. Enforcing the load profile means that the solution aims to balance the load among the nodes of the infrastructure so that similar nodes have identical target loads. However, load balancing techniques are inappropriate when severe failures may occur, hence we design the migration process by using a self-adapting mechanism.

3.2. Task migration properties

To solve Problem 3.1, two questions need to be addressed: when is it opportune to offload (both from the mobile device to a GCS and among mobile nodes), and where should the task migrate to? The solution presented leverages a proactive, self-adjusting adaptation mechanism that migrates a given task when its hosting node reaches a threshold. Such a threshold can be customized due to the architecture of the solution (Section 5), but as default, it is set as the average number of jobs queued and running across the entire edge system.

When do we migrate a job to another edge agent? Each edge agent computes such a threshold independently, using Jackson's network model. This model can be exploited to estimate the average number of tasks on each edge computing node i, denoting such a quantity as $\hat{n} = E[n_i]$.

Where do we migrate the job? The destination node of the job migration schema is chosen by modeling the network as a network of queues. Task can migrate from the queue of the source node to the queue of the destination node. Let $p_{ii'}$ the probability of the migration between the source i and the destination i'. Mathematically, our strategy can be formalized as follows: let the system be composed of Q nodes (queues);

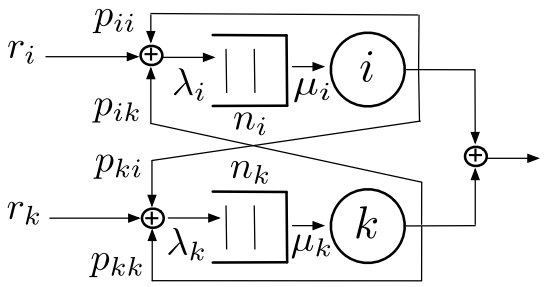

Fig. 2. Example of open Jackson queue with two agents (*e.g.* drones): tasks belonging to failing agents are reassigned.

the destination node $dest$ where the tasks are offloaded, is chosen by solving the following equation:

$$dest = \arg\max_{i'} p_{ii'} \tag{1}$$

The probability is computed to guarantee efficiency in the offloading procedure, after an evaluation phase about the migration delay overhead calculated according to the model presented in Section 4.

Example. Fig. 1 shows an example of the task schedule of three agents. Each drone receives the set of tasks and moves from one location to the next where it has to accomplish the task. For instance, agent v_2 migrates its original tasks t_2 and t_7 to the drone v_1 and t_10 to v_3 in order to balance the overall load.

4. Mission planning via network of queues

The IoT application usually affects the failure models, hence it is considered as impossible to find a failure model fitting all scenarios. For this reason, instead of predicting the failure of nodes or links, we leverage the service component to compute the probability of having a given number of tasks still to be completed. Thus, the goal is to obtain the average number of tasks in execution and in the queue. Such a piece of information can, in turn, be manipulated by the application to establish the instantaneous utilization of each agent and the mean queuing time for each task. Finally, this knowledge can be used to design a controller that adapts to the application.

We model the set of potentially failing nodes as a network of queues, according to Jackson network class [19]. The effect of an agent's fault is the migration of all its tasks to a different queue. In this context, the completion time of a task in a non-failing agent is the mere waiting time that the task spends in the queue. On the other hand, the completion time of a migrated task is the sum of all the waiting times across the visited agents, plus the time spent in the queue of the agent that executes it (holding time). In fact, when a node fails, all its tasks still to be executed are migrated and reassigned to a new node. Potentially, a task can be reassigned more than once to nodes if the agent's failure occurs before its execution.

We model each agent as a single queue storing all the tasks that will be eventually executed. Thus, the network is modeled with a network of Q queues, and there is a (directed) edge from queue i to queue q if a task "migrates" to agent q after agent i's failure (Fig. 2). The system is assumed to be an *open* task replanning process constituted by $\mathcal{Q} = \{1, 2, \ldots, Q\}$ agent's queues, a vector $\mathbf{n} = \{n_1, n_2, \ldots, n_Q\}$ that indicates the number of tasks belonging to each of the $\mathcal{Q}$ queues, and an operator $T_{ii'}$ on $\mathbf{n}$:

$$T_{ii'} = (\ldots, n_i - 1, \ldots, n_i' + 1, \ldots) \tag{2}$$

that removes one element from the agent's queue i and adds it to queue i, or exits the system. The term *open* denotes that tasks may enter or exit the system. We denote by λ_q the external arrival to queue q, if any. The vector $\mathbf{n}$ is assumed to be a Markov process with state space:

$$\mathcal{N} = \{\mathbf{n} : n_q \geq 0, q = 1, \ldots, Q\} \tag{3}$$

and transition rates given by:

$$q(\mathbf{n}, T_{ii'}(\mathbf{n})) = p_{ii'}, \tag{4}$$

where $p_{ii'}$ is the probability of a task to migrate from agent's queue i to queue i' after agent i's failure. The Markov process $\mathbf{n}$ is *irreducible* for $n > 0$, in other words, each task can potentially migrate from one agent's queue to any other queue, and *aperiodic*, that is, an agent can be only temporarily unavailable, and hence a task can return to a state i at any (irregular) time. Ultimately, we are able to show that, at the steady state, the distribution of the number of tasks in each queue, or being executed, obeys the "product form" distribution, *i.e.* , it can be written as the product of the probability functions depending on the single agent's queues:

Proposition 4.1. *For each agent q, the average arrival rate of a task in its queue is given by $\Lambda_q = \lambda_q + \sum_{k=1}^{Q} p_{kq}\Lambda_k$. In addition, if we denote with $p(n_1, n_2, \dots, n_Q)$ the steady state probability that there are n_q tasks in the qth agent's queue for $q = 1, 2, \dots, Q$, and if $\Lambda_k < \mu_q$ for $q = 1, 2, \dots, Q$, that is, we assumed that each agent can execute at most one task at the time, and that the arrival rate is smaller than the departure rate of task from the system, and so there is a steady state distribution, then such steady state probability is computed as:*

$$p(n_1, n_2, \dots, n_Q) = p_1(n_1)p_2(n_2)\cdots p_Q(n_Q), \tag{5}$$

where $p_q(n_q)$ is the steady state probability that there are n_q tasks in the qth agent's queue, if such queue is treated as a $M/M/1$ queuing system with an average arrival rate Λ_q, and average task execution time $\frac{1}{\mu_q}$ for each agent. Furthermore, each agent's queue q behaves as if it were an independent $M/M/1$ queuing system with average arrival rate Λ_q.

Proof. Proposition 4.1 is a straightforward corollary of the Jackson's theorem [19], in case of a single server per queue.

Based on Proposition 4.1 and Little's law [20], hence, we estimate the number of tasks in each queue at the steady state according to the formula: $\hat{n} = E[n_i] = \frac{\rho_i}{1-\rho_i}$, where ρ_i is the utilization factor of agent i's queue, defined by: $\rho_i = \frac{\Lambda_i}{\mu_i}$.

In accordance with this result, the ultimate goal of the controller algorithm is to balance the utilization of all queues. Consequently, when the utilization of queue i becomes too low with respect to the average of the system, the agent may request a task migration from other agents. When instead the queue utilization, or the estimated utilization, becomes too high, it means the system of agents is experiencing a high failure rate. In such a situation, the algorithm can proactively redistribute the load by increasing the replan frequency and migrating tasks to under-loaded nodes. Along with the proactive migration, when a failure occurs, the system can react by assigning the remaining tasks to another agent, thanks to a continuous monitoring function implemented by the architecture. Further details about APRON architecture are explained in the following section.

5. APRON architecture

In this section, we present our proposed management layer amongst the Operating System (at the bottom), e.g., Robotic Operating System (ROS) [16], and the IoT application (at the top). Fig. 3 shows the management architecture, whose mechanisms allow establishing and monitoring the network connectivity, to estimate the link and node failures, and to replan the mission via a customizable controller logic. Application atop can take advantage of the provided API to customize the logic of such controllers, adapting to different failure models, as well as to customize the mission planning logic, in a centralized or distributed fashion.

In the following, we summarize the network components of the solution, including the API, and the agent mission services offered, while the next section is about the mission replanning component.

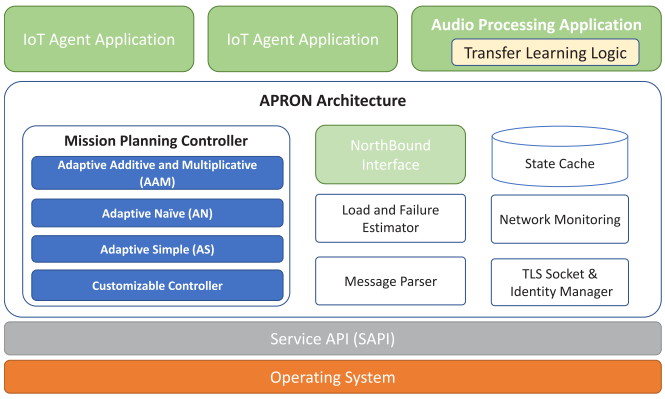

Fig. 3. APRON Architecture: a management layer between the IoT application and the operating system to establish and monitor network connectivity, to estimate failures and to adapt the task (re-)planning based on the customizable controller logic.

Network Monitoring. Inspired by the vast majority of networked systems, the connectivity management component runs a network discovery protocol, and a watchdog process running a heartbeat protocol to monitor alive connections. The architecture does not require an IP address and hence does not inherit the multihoming and mobility shortcomings of the TCP/IP architecture. As in recently proposed clean-slate Internet architectures [21,22], we bind the agent addresses to the application names, not to the network interfaces.

TLS Socket & Identity Manager. Since each agent may belong to multiple overlays, it is necessary to be authenticated ahead of the communication establishment. This component is responsible for the management of agent identities across multiple overlay networks and provides secure connectivity through the Transport Layer Security (TLS) protocol.

State Cache Manager. This component handles the partially replicated database while maintaining network states. The database entries are related either to static states *i.e.*, states that depend merely on the agent, or dynamic states *i.e.*, states that depend on the network, configuration, and connectivity condition. The state cache is also used as a log to store application states, for example, IoT device battery usage.

Service API. This service supports the customization of two main components: (*i*) the controller logic, that can fit multiple (failure) scenarios, (*ii*) the logic of the mission planning algorithm, either in a centralized or distributed fashion. In such a way, the same program suits different contexts, adapting to different requirements and network conditions.

Message Parser and Object Model. To define our object model, as well as to implement the logic of message delimiting, serializing and deserializing, we use Google Protocol Buffers [23], since it is more efficient than other text-based abstract syntax notation languages as JSON and XML.

IoT Application. Atop the APRON architecture, the application can run exploiting the services offered by the architecture. Examples of these applications are in Section 2. Among them, this paper presents an application of live audio analytic where is present the concept of Transfer Learning, as described in Section 6.

NorthBound Interface. The application can communicate with the APRON Architecture via the APIs that constitute the NorthBound Interface. All the offered services are indeed accessed through the REST APIs, a standard de-facto, which exposes the resources by using a uniform and predefined set of stateless operations.

Load and Failure Estimator. This component is the core of the architecture as it provides the information required to perform the task migration. By leveraging the methodology presented in Section 4, it

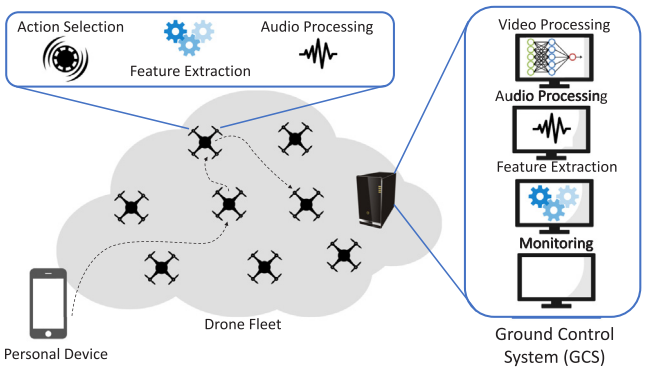

Fig. 4. Overview of the drone system realized exploiting the APRON underlying architecture. The computation is offloaded to the Ground Control System (GCS) that performs computationally intensive task.

computes the expected average number of tasks and compares the current state to estimate the failure rate. The load is redistributed proactively in order to prevent the failure, and, when the failure occurs, the tasks previously on the failed queue are migrated to another agent.

Mission Planning Controller. The framework supports a class of controllers to tailor the mission replanning rate $R(t)$ of the network of IoT devices, e.g., drones. However, our architecture is modular and pluggable, hence it can be extended with other user-defined controllers. By controller, we do *not* mean a Software-Defined Networking (SDN) controller, but feedback controller. We already implemented some controllers herein presented, whose rate depends on $\hat{n}(t)$ that denotes the estimated number of tasks currently into the network and discussed in Section 4.

Adaptive Naive (AN) controller. The replanning rate varies with the ratio between $\hat{n}(t)$ and $\acute{n}$ that represents the desired number of tasks in the system at the steady state, based to the following equation:

$$R(t+1) = \frac{\hat{n}(t)}{\acute{n}} R(t) \tag{6}$$

Adaptive Simple (AS) controller. The replanning rate varies with $\hat{n}(t)$ and n' according to the following equation:

$$R(t+1) = k(\hat{n}(t) - \acute{n}) \tag{7}$$

where k is the gain term.

Adaptive Additive and Multiplicative (AAM) controller. The replanning rate varies with $\tilde{n}$ and $\hat{m}$ that are number of completed tasks and estimated number of tasks missing the deadline at time t, respectively:

$$R(t+1) = \begin{cases} k\frac{\hat{m}(t)}{\tilde{n}(t)+1} & \hat{m}/(\tilde{n}+1) > 0 \\ -\alpha & \text{otherwise} \end{cases} \tag{8}$$

where α is a positive constant.

Customizable controller. Aside from the provided strategies that the architecture is equipped with, the replanning rate can follow other policies defined by the user, for example via the NorthBound Interface.

6. Drone-sourced live audio analytic

By exploiting the management layer aforementioned, it is possible to develop applications with resilient drone mission execution in challenged networks. We implemented a novel application in the context of disaster response scenario, where a distributed set of drones is managed in a straightforward way.

Specifically, humans control the swarm of drones to monitor the area after a disaster. The drones continuously record the audio of the environment sending it to the GCS. This machine processes the received audio, and if a human presence is detected, it sends the approximated human location to the drone. The task of the drone is hence reaching the specified position so that it is possible for the human to help survivors if necessary. When drones are used to predict and assess disaster [24] or supply emergency commodities [25] to survivors, operations must proceed as quickly and efficiently as possible. The heavy computation of feature extraction and audio processing, which is based on Neural Networks(NNs), is hence performed by a powerful machine, and the drone agent just records the audio and executes the specific commands.

In addition, the user may control a specific drone using speech instead of a computer or a physical drone controller. The human speaks directly to the personal device, where an application utilizes Natural Language Processing (NLP) and Natural Language Understanding (NLU) techniques to discover the intent behind users' words. NLU is instrumental in this process because it allows the user to speak conversationally to the program rather than memorize specific commands that are trivially passed on to the drone. These benefits are most observable when there are multiple kinds of drones that need to coordinate to solve a single mission. The personal device, *e.g.*, mobile phone, elaborates it and sends the proper commands to the selected drone. When the drone receives the instructions, it starts a new task to perform the requested operation.

Fig. 4 sketches the main components of the system, highlighting the functionalities of the elements. The whole fleet is managed by the operator who can easily instruct drones, fundamental in challenged networked environments, such as in response missions following a natural disaster.

These situations also require resilient mission systems to manage tasks in case of failure. However, this complex supervision is hidden from the human that can only focus on the execution of high-level jobs. On the other hand, the management layer ensures a resilient distributed system by monitoring and estimating the failures for each network component. When failures occur, tasks running or queued on the damaged node are reassigned to a new agent, and the user is notified of the update and the estimated effects.

This level of abstraction facilitates the rising of resilient IoT applications. In the next subsection, we describe the audio processing needed by a disaster response application; however, edge computing applications that can benefit from this processing are not limited to the presented application.

6.1. Human activity detector

As a use case, we implemented a binary classifier that predicts the human-nature of an audio file. We decided to adopt a transfer learning technique because it can speed up the time it takes to develop and train a model by reusing the knowledge of a complex model extensively trained on a comprehensive dataset. This helps speed up the model training process and accelerate results. Our use case does not strictly require online training, but it can be implemented in our architecture thanks to the benefit of transfer learning. We have also studied the work required to train a model to prove that online training is possible. We trained our model on the ESC-50 dataset [26], a collection of 2000 environment recordings. This dataset consists of five-seconds-long recordings organized into 50 semantical classes (with 40 examples per class) arranged into five major categories. Among these categories, the one labeled as "Human, non-speech sounds", represents the set of audio files that should be recognized by a drone when it is monitoring a disaster scene. Instead, the other classes refer to animals, natural and water sounds, domestic sounds, and urban noises.

We address the limited size of our dataset and the complexity of audio data by utilizing the concept of transfer learning. This technique aims to improve a learner from one domain by transferring knowledge from a related domain. We choose VGGish [27] as our pre-trained model, a Convolutional Neural Network trained on Audio Set [28]. Audio Set is a dataset of generic audio events released in 2017, comprising an ontology of 632 audio event categories and a collection

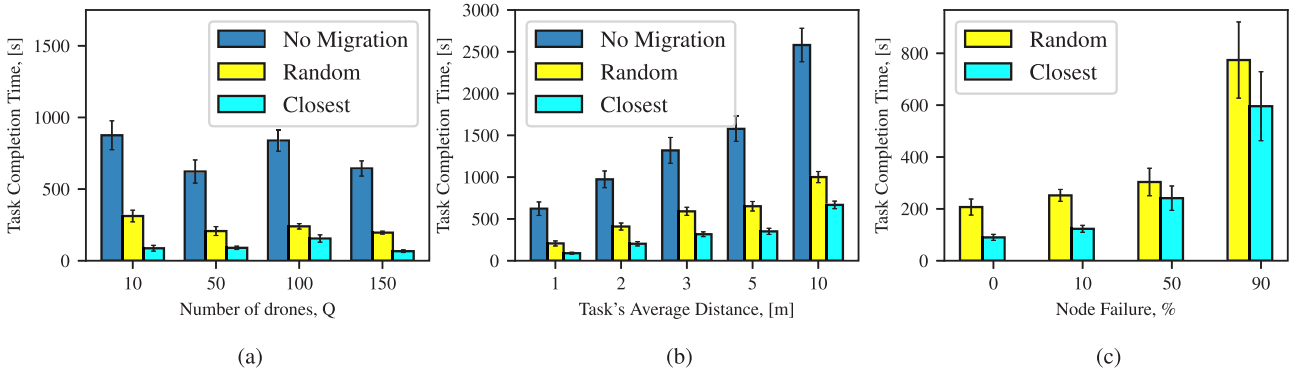

Fig. 5. Task completion time of a fleet of drones using APRON with different replanning policies: *(i)* no task migration, *(ii)* random task migration, *(iii)* closest task migration. The graphs represent the task completion time at different conditions: (a) number of drones, (b) task distance, (c) percentage of node failures.

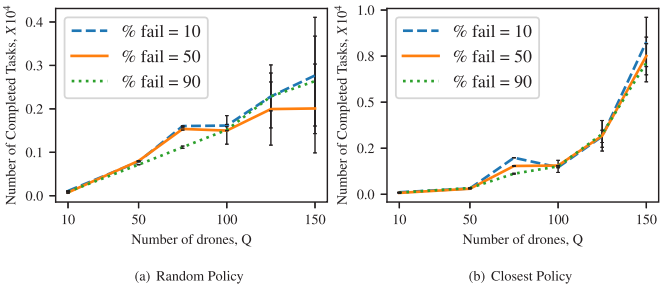

Fig. 6. The graphs depicts the endurance, i.e., number of completed tasks before the first failure, for (a) random policy, and (b) closest policy.

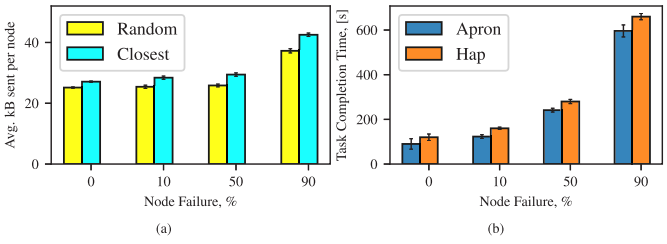

Fig. 7. (a) Bytes exchanged per node for increasing number of agents in the network. (b) Comparison with different architectures, in terms of time to complete tasks.

of 1,789,621 labeled ten-seconds-long excerpts from YouTube videos. VGGish is a variant of the VGG model, a model used for large-scale image classification. We chose this model because it is trained on a dataset that comprehends most of the classes that are present in the ESC-50 dataset. Then, the knowledge acquired by the VGGish model during its training is useful to generate an internal representation of the audio data that is employed by our final model. The changes to this model concern the input size, to make it suitable for audio features, and the final portion of the model: the last group of convolutional layers has been dropped, and a 128-wide fully connected layer acts as a compact embedding layer at the very end of the model.

We use VGGish as part of a bigger model: we append a set of convolutional layers on top of it, and we train them on the ESC-50 dataset.

7. Evaluation results

In this section, we evaluate the performance of our solution with the development of a C++ event-driven simulator able to run in every machine. Throughout this evaluation, we consider the use case of a networked fleet of drones deployed to accomplish a mission, constituted by a set of actions ordered by the GCS. Examples of these actions can be geo-locations to reach in order to explore the area via camera and microphone, sending the streaming that will be elaborated to locate survivors in disaster response. Each drone receives the instructions and tries to execute them; at the same time, it determines whether migration is necessary by computing a threshold. This value is estimated by using our Jackson network model and triggers the migration of tasks on board. To this end, all agents cooperate to complete the assigned jobs in the shortest possible time. We summarize in Table 1 the configuration parameters utilized during the following evaluation, where the default values are reported in bold.

We run experiments on a fleet of different sizes, namely consisting of 10, 50 100, or 150 drones. Tests also consider varying the average distance between two consecutive geo-location needed to be visited by a drone, which has been 1, 2, 3, 5, or 10 meters. Moreover, we evaluate the performance in case of three distinct task migration policies: *(i) no replanning (task migration)*: agents in the system do not cooperate, but each one tries to accomplish all and only the tasks in its own queue; *(ii) random task replanning*: when an agent's queue exceeds a set threshold, the next drone which will receive the tasks in excess is selected randomly; *(iii) closest task replanning*: when an agent's queue overcomes the threshold, the system reassigns its tasks to the closest node. In case there are two or more agents at the same distance from the task, the destination node is the one with fewer tasks in its queue; ties are split at random if two queues have the same number of tasks.

Table 1
Parameters setting.

Parameter	Values
Number of nodes	10, **50**, 100, 150
Nodes' Average Distance [m]	**1**, 2, 3, 5, 10
Node failure [%]	**0**, 10, 50, 90
Number of Trials	**30**
Confidence Interval [%]	**90**

The results demonstrate how this framework is an effective tool for the policy-based reallocation problem. A few observation can be deducted from Figs. 5–6–7 regarding the overall system performance: *(1) Task migration policies show shorter mission completion time.* As can be seen in Fig. 5a, the enforcement of (any) reallocation policy permits the agents to terminate their tasks in a shorter time. Both migration policies, i.e., random and closest policy, exploit all the available agents without overloading them. In particular, the advantage of the migration policies is higher when the number of nodes in the topology increases because tasks can be managed by more drones. *(2) The closest agent policy achieves lower completion time with respect to the random task replanning policy.* Fig. 5a–b–c exhibit that the closest migration policy provides better performance taking advantage of all agent geo-location, hence a more efficient mission plan. The advantages of the closest policy are even larger in case of failures, as demonstrated in Fig. 5c.

This confirms how the selection of the closest agent produces lower completion time than the migration of the task to a random one. Nonetheless, when the percentage of failures is high, the two policies exhibit similar results. Evidence for this is in the same Fig. 5c, since the confidence intervals of the two policies are slightly overlapped. *(3) The task completion time decreases with the agent travel distance.* As expected, by increasing the average distance among two consecutive tasks, the completion time increases as well, as shown in Fig. 5b. *(4) The completion time increases when the drone failure increases.* As evident from Fig. 5c, when the number of available drones decreases, fewer agents are available for completing the tasks, therefore the queue's size increases, leading to an increase of the time to complete the tasks. *(5) The number of failures does not affect the performance when the number of drones is reasonably low.* The evaluation of the number of tasks accomplished before the failure of the first task shows that the performance does not significantly change when the number of drones is lower than 100. In fact, the points of the random policy for a small Q (Fig. 6a) have the same order of the corresponding values obtained with the closest policy (Fig. 6b). *(6) The improved performance of closest policy involves a larger, yet reasonable, amount of messages exchanged.* Fig. 7a shows the messages exchanged in a centralized configuration, where the controller receives information from the agents about their status and location. For the closest task migration, more frequent packets are sent to the controller compared to the random policy, respectively every 2 s and every 4 s.

Furthermore, we compared APRON against HAP [13], a solution aiming to anticipate failures at the planning level by establishing close feedback between the high-level planning based on Markov Decision Processes (MDP) and the execution level learning capable adaptive controllers. This model is used for replanning to account for failures and degradation. We report in Fig. 7b the time to complete tasks for the two algorithms (closest policy for APRON) at varying the percentage of failures, when 50 drones are utilized. We can notice how APRON can shorten completion time w.r.t. HAP, due to its ability to control a large number of agents. On the other hand, HAP can tackle more complex environments, but at the cost of a more complex model to threat.

7.1. Audio detector accuracy

Part of our contribution is the development of an audio speech detector to detect the presence of possible survivors after a disaster. To evaluate the performance of each classifier, we plot the ROC curve, a standard tool used for visual comparison, which shows the trade-off between the true positive and the false positive rate. The area beneath the ROC curve measures the accuracy of the model. A model with perfect accuracy would have an area of 1.0, while a model closer to the diagonal is less accurate.

A key requirement for an audio surveillance system is the ability to detect events of interest, even in the presence of different background sounds at different energy levels. In order to address this problem, we have selected a dataset where environmental recordings are available in a unified format, 5-second-long clips, 44.1 kHz [26]. The ESC-50 dataset consists of 2000 labeled environmental recordings equally balanced between 50 classes (40 clips per class), grouped in 5 loosely defined major categories (10 classes per category): animal sounds, natural soundscapes, and water sounds, human (non-speech) sounds, interior/domestic sounds, exterior/urban noises. This dataset has been extensively studied in the literature [29–31], hence a comparison among different proposed classifiers is straightforward and one of the main advantages of such a dataset.

To reflect the real scenario, the classes are not balanced, rather the number of samples for the *Human sounds* class is less than the samples for the *Non-Human sounds*. Due to the underlying class distribution, we cannot simply compute the *accuracy* to check the correctness of results. Thus, we employ Receiver Operating Characteristics (ROC) curve and Area Under the Curve (AUC) metrics to measure the accuracy

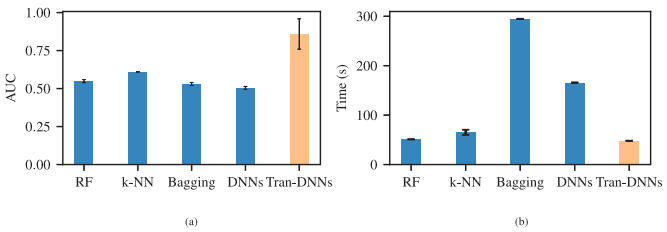

Fig. 8. The graphs shows the advantages of Transfer Learning usage in terms of (a) training time and (b) AUC of the classifier.

of algorithms. Specifically, AUC is the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative example. It tells how much the model is capable of distinguishing between classes, and the higher AUC, the better the model is at predicting 0s as 0s and 1s as 1s.

Over such a dataset, we compared several classifiers: Random Forest Classifier (RF), k-Nearest Neighbors (KNN), Bagging Classifier, Deep Neural Networks (DNNs). These classifiers are compared against a DNNs enriched with Transfer Learning (Tran-DNNs). Two models are combined for the classification problem, where the first model applied was already trained on a different dataset. VGGish [27] is our pre-trained model, which outputs an array of 128 values for each second of the file audio. These features are the input of our model, which can now receive less but more meaningful information about the original audio. In Fig. 8a we demonstrate how this approach can provide a higher AUC compared to other methods where the Transfer Learning is not applied. Although the pre-trained model was validated on different data, it is extremely useful in pre-processing the audio file and providing a set of features that simplify further processing. As shown in the graph, the transferred DNNs achieves an improved AUC w.r.t. the same DNNs without the usage of the prior model.

The number of hidden layers and the number of neurons of DNNs and Tran-DNNs is obtained via cross-validation. The optimal configuration is composed of 3-layers with 128-128-64 neurons, where the output is a binary representation where "0" means "Human sound", "1" otherwise. We can hence conclude that by using Transfer Learning, we can use a simple network for the classification problem.

The second benefit brought by the transfer learning technique is the training time reduction, as can be seen in Fig. 8b. Results refer to a training process performed on Ubuntu, Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz. The model receives as input a simpler version of features, with a reduced dimensionality but higher meaningfulness. The transferred model does not need to be re-trained, but produces as output an effective set of features, that are the input of the new classifier.

This demonstrates how Transfer Learning can be used to saving time or getting better performance, achieving the same performance of other methods in a shorter amount of time. The same model (DNNs), if applied in conjunction with a transferred model, provides higher accuracy and in a shorter amount of time. The training time of the standard model is indeed more than five-times the time for the model in the Transfer Learning case.

7.2. Noise reduction

The task of audio detection also entails a noise reduction processing, which is crucial for mitigating the noise originated by the drone rotors. To treat this aspect, we evaluate performance over a dataset where the background noise is prominent, similarly to the audio recorded by drones. The data are publicly available at [32]. Although these samples are not obtained directly from drones, this dataset is of great value, due to the similar context wherein the events are generally mixed with a complex background. Moreover, the availability of the source makes

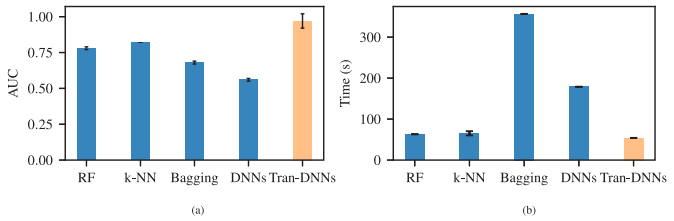

Fig. 9. The graphs show the (a) training time and (b) AUC of the classifier applied after a noise redaction processing that is needed to mitigate the effect of drone rotors noise. The figures point out advantages of Transfer Learning usage.

the reproducibility easier. The set comprises three classes, namely glass breaking, gun shots, and screams, for a total of 6000 events. Further, these events are available at 6 different values of signal-to-noise ratio (namely 5 dB, 10 dB, 15 dB, 20 dB, 25 dB and 30 dB).

Fig. 9 shows how the Transfer Learning approach is beneficial also when a pre-processing for reducing the effect of drone rotors noise is applied. After the filter for removing such audio noises is applied, the classifier is trained to correctly detect the scream class. In this case, the training time encompasses the filtering process too, leading to an increased time compared to Fig. 8b. However, the advantages brought by Transfer Learning are even more pronounced than in Fig. 8, where the effects of noise were neglected.

7.3. Application advantages

In addition to the estimation of the system performance and the comparison of different policies of APRON, we evaluate the tangible benefits for an edge computing application. We tested specifically the proposed application (Section 6), in cases where APRON is deployed and not. In this scenario, the drones are performing the task of reaching a geo-location as in the previous examples, and the audio recording task in the background. The adverse conditions of challenged networks impose to face tasks that are lost because the node that was hosting them failed. The system is able to reassign these tasks, but the delay perceived by the user drastically increases. For this reason, a layer such as APRON is effective in mitigating the effects upon a failure. Fig. 10 highlights the main 2 advantages of APRON: *(i)* efficient fault response, *(ii)* accurate failure estimation; the management layer offered by APRON allows a smaller number of lost tasks in different scenarios. The presence of APRON is evident especially in critical conditions, *i.e.*, high percentage of node failures, a high number of drones to control. On the other hand, the distance between nodes does not notably affect the number of completed tasks.

8. Related work

Delivering adaptive and resilient to failures services is crucial in almost every IoT network, especially for robotics and drones fleet. To tackle this problem, several solutions have been proposed due to the relevance of the problem and the many scenarios affected. We describe a few representative solutions to clarify our contributions to the resilient task planning problem, as well as equivalent audio processing applications.

IoT at the edge. The proliferation of IoT devices led to the generation of a massive amount of data. The processing of collected data and the decisions making could be performed onboard, but this approach inevitably drains the battery of the IoT device. On the other hand, central cloud servers are inefficient at handling all the collected data because of limited computing, communication, storage, high overall energy, and, most importantly, latency. To better address this problem, recent solutions have proposed the offloading of data processing at the edge of the network. The proximity to the IoT devices is the key enabler of several advantages such as low and predictable latency, reduction of bandwidth consumption, context awareness.

When an edge computing service is deployed, several challenges have to be faced, such as the implementation of an offloading strategy to efficiently distribute the workload in the system, handle the mobility (or communication disconnection) while reliable cooperation is guaranteed [33–35]. In this context, the drone itself may be considered as a fog node.

Resilient and Adaptive IoT Systems. Critical applications and the surrounding environment can be affected by problems, e.g., interference, medium access conflicts, multipath fading, shadowing, which can cause significant packet losses. More specifically, the disaster response scenario is tied to edge offloading and its effectiveness is crucial for some applications. We could mention the real-time video conferencing with the incident commander featuring face recognition of disaster victims [36], or the detection of children in an attempt to reunite them with their guardians [37], whereas virtual beacons can be mainly used to track their location. In this scenario, the reliability is essential for the effectiveness of the applications [38]. [39] and [13] focus on the prediction of failures, where the former applies a Bayesian-inference probabilistic for the computation of the estimate failure probability in case of monitored batteries. In the latter, the authors present close feedback between the high-level planning based on Markov Decision Processes (MDP) and the execution level learning-focused adaptive controllers. By exploiting this feedback, the framework anticipates the failures and reassesses vehicle capabilities after the failures. This proactive behavior allows an efficient replanning to account for changing capabilities. However, our solution does not predict the failure of links or agents, but it computes a close form of the average number of tasks in a mission, that can be used to adapt to the situation of the system.

Moreover, the adaptability and the persistence of distributed IoT systems can exploit decentralized approaches [11–13,40]. In [41] the authors faced the problem of task allocation and scheduling over a heterogeneous team of human operators and robotic agents. The human operator acts as the centralized component that interacts with unmanned agents. Operator, vehicle, and task are selected according to a multi-objective optimization function that depends on a reward assigned when the task is completed, the cost of the vehicle to perform the task, and the cost of the operator to supervise the task assignment. As in [41], our solution can also be used to distribute workload efficiently among agents, but our predictive system is based on a Jackson network approach. Our solution is indeed agnostic to the agent architecture and can manage both centralized and distributed management approaches.

Prediction with a network of queues. The network services have also been studied for shared (peer-to-peer) storage networks, with the aim of checking the robustness. For example, a theoretical control approach to modeling and predicting data availability through redundancy is proposed in [42]. New redundant fragments need to be introduced in order to ensure a given level of availability in the event of storage node failures.

Inspired by [42], we leverage a network queuing model to estimate objects (tasks) that will temporarily or permanently disappear from the agent's (peer to peer) network; however, our failure prediction model is different, as we model an agent failure and the reassignment of its task with a Jackson network of queues [19].

Machine Learning for Audio Detection. A key assumption for machine learning and data mining algorithm is that the training data must be in the same feature space and have the same distribution of the predicted data. However, in certain real-world scenarios, this assumption does not hold. Indeed, collecting the needed training data that provide the characteristics of the test data can be difficult and expensive. Thus, related data is usually added to the dataset to prevent this problem, but the difference in data affects the prediction of the learner. In such cases, transfer learning techniques would greatly improve the performance of the model avoiding many expensive data-labeling efforts [43].

Recent research has widely studied the effectiveness of transfer learning applied to image classification. Moreover, many pre-trained

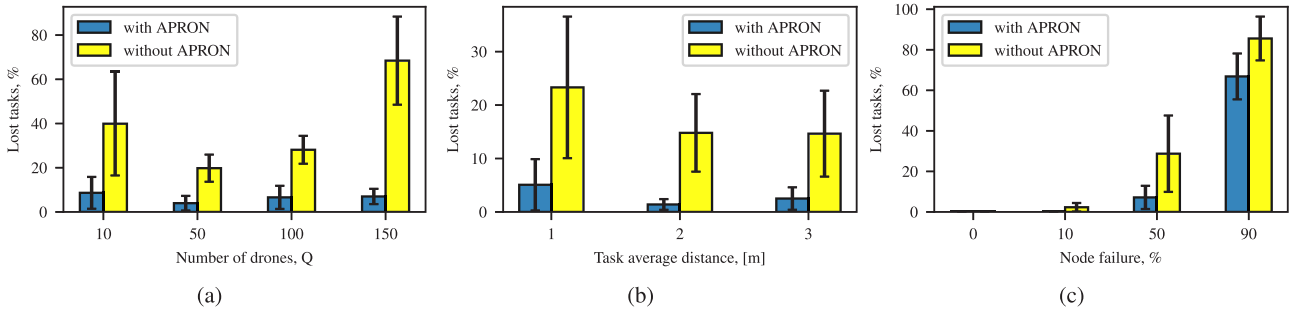

Fig. 10. Comparison of application performance using the closest task migration replanning policies. The graphs represent the percentage of lost tasks when 500 tasks are completed at different conditions: (a) number of drones, (b) task distance, (c) percentage of node failures.

models ready to be used for knowledge transfer are available. For example, Keras, one of the most popular deep learning library, provides model definitions and pre-trained weights for many popular architectures, such as VGG16 [44], ResNet50 [45], Xception [46], MobileNet [47], and more.

However, the success of transfer learning applied to the image domain has not been ported to the audio domain, due to the complexity of the latter. Audio signals contain many more features compared to static images. Thus, only a few complex pre-trained models are available to the research community, and mostly they are an adaption of popular models for image recognition, such as VGGish [27]. For this reason, knowledge transfer has not been extensively employed in this field, and most of the studies revolve on Music Information Retrieval [48–50] rather than audio classification. In fact, sound detection has its own uniqueness, which makes it hard to apply and port among different use cases. Nevertheless, some attempts in voice recognition fields provide very good results, as in [51] where the spherical k-means algorithm for feature learning is adopted for audio signals. Xu et al. presented an interesting work about the detection of semantic events in soccer video by applying a heuristic mapping. This is done by means of audio keywords, created from low-level audio features by using support vector machine learning.

9. Conclusion

This paper exposes a novel edge computing application which, leveraging Machine Learning algorithms, is able to detect the presence of humans in disaster scenarios. To speed up the computation and guarantee acceptable reliability of the application, we also deployed a management architecture whose goal is to re-plan tasks in the presence of challenged edge networks. Such a layer leverages Jackson's network queues model to estimate the number of tasks, queued or in execution. Thus, the application can determine the instantaneous utilization of each IoT device, and the mean queuing time (both waiting and execution time) for each task to be executed or offloaded to the edge of the network. Our results demonstrate how this management layer is an effective tool for policy programmability of the mission re-planning problem for any IoT device deployed in challenged networked environments. Furthermore, the time for the audio processing is reduced when the underlying service is running, since the application atop can exploit functionalities able to improve the overall performance of the system.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been partially supported by NSF under Award Numbers CNS1647084, CNS1836906, and CNS1908574.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.
[2] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2017) 450–465.
[3] D. Chemodanov, F. Esposito, A. Sukhov, P. Calyam, H. Trinh, Z. Oraibi, AGRA: AI-augmented geographic routing approach for IoT-based incident-supporting applications, Future Gener. Comput. Syst. 92 (2019) 1051–1065.
[4] A.V. Ventrella, F. Esposito, 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.
[5] N.H. Motlagh, T. Taleb, O. Arouk, Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives, IEEE Internet Things J. 3 (6) (2016) 899–922.
[6] K. Coleman, F. Esposito, R. Charney, Speeding up children reunification in disaster scenarios via serverless computing, in: Proceedings of the 2nd International Workshop on Serverless Computing, in: WoSC '17, 2017, p. 5.
[7] J. Franz, T. Nagasuri, A. Wartman, A.V. Ventrella, F. Esposito, Reunifying families after a disaster via serverless computing and raspberry pis, in: 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2018, pp. 131–132.
[8] R.L. Charney, T. Rebmann, F. Esposito, K. Schmid, S. Chung, Separated after a disaster: Trust and privacy issues in sharing children's personal information, in: Disaster medicine and public health preparedness, Cambridge University Press, 2019, pp. 1–8.
[9] J. Franz, T. Nagasuri, A. Wartman, A.V. Ventrella, F. Esposito, Reunifying families after a disaster via serverless computing and raspberry pis, in: 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), IEEE, 2018, pp. 131–132.
[10] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, H. Weiss, RFC 4838, delay-tolerant networking architecture, irtf Dtn Res. Group 2 (4) (2007) 6.
[11] J.-S. Marier, C.A. Rabbath, N. Léchevin, Health-aware coverage control with application to a team of small UAVs, IEEE Trans. Control Syst. Technol. 21 (5) (2013) 1719–1730.
[12] N.K. Ure, G. Chowdhary, Y.F. Chen, M. Cutler, J.P. How, J. Vian, Decentralized learning-based planning for multiagent missions in the presence of actuator failures, in: 2013 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2013, pp. 1125–1134.
[13] N.K. Ure, G. Chowdhary, J.P. How, M.A. Vavrina, J. Vian, Health aware planning under uncertainty for UAV missions with heterogeneous teams, in: 2013 European Control Conference (ECC), IEEE, 2013, pp. 3312–3319.
[14] H.-L. Choi, L. Brunet, J.P. How, Consensus-based decentralized auctions for robust task allocation, IEEE Trans. Robot. 25 (4) (2009) 912–926.
[15] A.V. Ventrella, F. Esposito, A. Sacco, M. Flocco, G. Marchetto, S. Gururajan, APRON: An Architecture for adaptive task planning of internet of things in challenged edge networks, in: 2019 IEEE 8th International Conference on Cloud Networking (CloudNet), IEEE, 2019, pp. 1–6.
[16] Robotic Operating System, http://www.ros.org/, online.
[17] Donato Di Paola, The Multi-Agent Robotic Simulator (MARS) https://github.com/donatodipaola/mars, online.
[18] M. Satyanarayanan, A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets, GetMobile: Mob. Comput. Commun. 18 (4) (2015) 19–23.
[19] J.R. Jackson, Networks of waiting lines, Oper. Res. 5 (4) (1957) 518–521.
[20] K.S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, vol. 13, Wiley Online Library, 1982.
[21] J. Day, I. Matta, K. Mattar, Networking is IPC: A guiding principle to a better internet, in: Proceedings of the 2008 ACM CoNEXT Conference, 2008, pp. 1–6.

[22] I. Seskar, K. Nagaraja, S. Nelson, D. Raychaudhuri, Mobilityfirst future internet architecture project, in: Proceedings of the 7th Asian Internet Engineering Conference, 2011, pp. 1–3.
[23] Google Protocol Buffers, http://code.google.com/apis/protocolbuffers. online.
[24] M. Erdelj, E. Natalizio, K.R. Chowdhury, I.F. Akyildiz, Help from the sky: Leveraging UAVs for disaster management, IEEE Pervasive Comput. 16 (1) (2017) 24–32.
[25] S. Chowdhury, A. Emelogu, M. Marufuzzaman, S.G. Nurre, L. Bian, Drones for disaster response and relief operations: A continuous approximation model, Int. J. Prod. Econ. 188 (2017) 167–184.
[26] K.J. Piczak, ESC: Dataset for environmental sound classification, in: Proceedings of the 23rd ACM International Conference on Multimedia, 2015, pp. 1015–1018.
[27] S. Hershey, S. Chaudhuri, D.P. Ellis, J.F. Gemmeke, A. Jansen, R.C. Moore, M. Plakal, D. Platt, R.A. Saurous, B. Seybold, et al., CNN Architectures for large-scale audio classification, in: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2017, pp. 131–135.
[28] J.F. Gemmeke, D.P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R.C. Moore, M. Plakal, M. Ritter, Audio set: An ontology and human-labeled dataset for audio events, in: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2017, pp. 776–780.
[29] A. Kumar, M. Khadkevich, C. Fügen, Knowledge transfer from weakly labeled audio using convolutional neural network for sound events and scenes, in: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2018, pp. 326–330.
[30] Y. Tokozume, Y. Ushiku, T. Harada, Learning from between-class examples for deep sound recognition, 2017, arXiv preprint arXiv:1711.10282.
[31] R.N. Tak, D.M. Agrawal, H.A. Patil, Novel phase encoded mel filterbank energies for environmental sound classification, in: International Conference on Pattern Recognition and Machine Intelligence, Springer, 2017, pp. 317–325.
[32] P. Foggia, N. Petkov, A. Saggese, N. Strisciuglio, M. Vento, Reliable detection of audio events in highly noisy environments, Pattern Recognit. Lett. 65 (2015) 22–28.
[33] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, J. Henkel, Computation offloading and resource allocation for low-power IoT edge devices, in: IEEE 3rd World Forum on Internet of Things, IEEE, 2016, pp. 7–12.
[34] M. Chiang, T. Zhang, Fog and IoT: An overview of research opportunities, IEEE Internet Things J. 3 (6) (2016) 854–864.
[35] C. Puliafito, E. Mingozzi, G. Anastasi, Fog computing for the internet of mobile things: Issues and challenges, in: IEEE International Conference on Smart Computing, IEEE, 2017, pp. 1–6.
[36] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, K. Palaniappan, 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.
[37] S. Chung, C. Mario Christoudias, T. Darrell, S.I. Ziniel, L.A. Kalish, A novel image-based tool to reunite children with their families after disasters, Acad. Emerg. Med. 19 (11) (2012) 1227–1234.
[38] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (2015) 2347–2376.
[39] J. Yu, State-of-health monitoring and prediction of lithium-ion battery using probabilistic indication and state-space model, IEEE Trans. Instrum. Meas. 64 (11) (2015) 2937–2949.
[40] S.S. Ponda, H.-L. Choi, J.P. How, Predictive planning for heterogeneous human-robot teams, in: AIAA Infotech@Aerospace Conference, 2010, pp. 3349.
[41] C.J. Shannon, L.B. Johnson, K.F. Jackson, J.P. How, Adaptive mission planning for coupled human-robot teams, in: American Control Conference (ACC), 2016, IEEE, 2016, pp. 6164–6169.
[42] A. Duminuco, E. Biersack, T. En-Najjary, Proactive replication in distributed storage systems using machine availability estimation, in: Proceedings of the 2007 ACM CoNEXT Conference, 2007, pp. 1–12.
[43] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2009) 1345–1359.
[44] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[45] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.
[46] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1251–1258.
[47] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv:1704.04861.
[48] P. Hamel, M.E.P. Davies, K. Yoshii, M. Goto, Transfer learning in MIR: Sharing learned latent representations for music audio classification and similarity, in: 14th International Conference on Music Information Retrieval (ISMIR '13), 2013, pp. 9–14.
[49] A. Van Den Oord, S. Dieleman, B. Schrauwen, Transfer learning by supervised pre-training for audio-based music classification, in: Conference of the International Society for Music Information Retrieval (ISMIR 2014), 2014, p. 6.
[50] K. Choi, G. Fazekas, M. Sandler, K. Cho, Transfer learning for music classification and regression tasks, 2017, arXiv preprint arXiv:1703.09179.
[51] J. Salamon, J.P. Bello, Unsupervised feature learning for urban sound classification, in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 171–175.