POLITECNICO DI TORINO
Repository ISTITUZIONALE

APRON: an Architecture for Adaptive Task Planning of Internet of Things in Challenged Edge Networks

Original
APRON: an Architecture for Adaptive Task Planning of Internet of Things in Challenged Edge Networks / Ventrella, A.V., Flavio, E., Sacco, A., Flocco, M., Marchetto, G., Srikanth, G.. - ELETTRONICO. - (2019), pp. 1-6. (2019 IEEE 8th International Conference on Cloud Networking (CloudNet) Coimbra, Portugal 4-6 November 2019) [10.1109/CloudNet47604.2019.9064091].

Availability:
This version is available at: 11583/2752092 since: 2020-08-26T11:40:31Z

Publisher:
IEEE

Published
DOI:10.1109/CloudNet47604.2019.9064091

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

08 October 2026

APRON: an Architecture for Adaptive Task Planning of Internet of Things in Challenged Edge Networks

Agnese V. Ventrella* Flavio Esposito* Alessio Sacco† Matteo Flocco*
Guido Marchetto† Srikanth Gururajan*
*Saint Louis University, USA †Politecnico di Torino, Italy

***Abstract*—Recently, the growth of Internet of Things (IoT) devices combined with edge computing opened many opportunities for several novel applications. Typical examples are Unmanned Aerial Vehicles (UAV) that are deployed for photogrammetry, surveillance, disaster rapid response and environmental monitoring. A common challenge across all these networked applications is the ability to provide a persistent service — a service able to continuously maintain a high level of performance — responding to events that may change the state of the network, $e.g.$, nodes or link failures. To cope with this challenge, in this paper we propose APRON, an edge cloud-assisted architecture for distributed and adaptive task planning management in a network of IoT devices, $e.g.$, drones. APRON uses a novel planning strategy that, leveraging a Jackson's network model, supports monitoring and control operations while the states of the (edge or cloud) network evolve. By using APRON, edge computing application programmers can design and implement a wide range of IoT task management policies leveraging different protection methodologies across several failure models.**

I. Introduction

Distributed applications for a network of drones or Internet of Things (IoT) devices that independently need to accomplish a mission —set of tasks— are opening many opportunities for new business models and applications. Typical examples of such applications are urban mobility-on-demand systems, networks of unmanned vehicles for rapid disaster response and environmental monitoring, and systems to provide connectivity to ground stations [1]. Autonomous and semi-autonomous drones will surely continue to help humans in accomplishing many tasks, spanning from industrial inspection to survey operations, from rescue management systems to military or first responder support. The role of drones and IoT has the potential to become even more prominent in the future as they enable, improve, and optimize novel and existing services.

While drone and IoT applications flourish, the challenges of keeping such devices well-functioning increase, especially under the challenging conditions imposed by a disaster scenario [2], [3]. Although delay and disruption tolerant protocols and architectures exist [4], 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 [5], [6] and distributed [7], [8] approaches that allow an edge network of IoT, drones or robots in general to provide a persistent and adaptive service already exist. Some of them focus on the resilient mission planning problem [8], others on agents' health-aware solutions [7]. Others yet [6] focus 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 models and applications probably cannot exist. To this end, we propose an Architecture for the Programmability of RObotic Networks (APRON). The architecture enables the programmability of different mechanisms involved in the mission execution problem of UAV or other edge cloud-based distributed agents. APRON is a software layer that sits between the (robotic) operating system ($e.g.$, ROS) [9]) and any IoT software application, and 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.

In this paper, we overview the components of our APRON architecture (Section II), we present in details its *Controller* component and its *failure estimator* component (Section III), 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 the tasks in a mission. This, in turn, can be manipulated by application programmers to determine the utilization of each agent. Such information can then be used (in conjunction with our APRON API) to design controllers that adapt to specific applications.

We built a Mission Allocation Simulator [10], to test the scalability of our approach, and a virtual network testbed to evaluate the practicality of APRON. We also created a possible use-case in which APRON can be exploited, a system of drones extremely useful 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 (Section IV).

Our initial results (Section V) focus on a few failure protection policies, but with our APRON fully developed prototype, programmers will be able to leverage our architecture and API to design failure protection strategies for a wide range of (distributed) drone applications. Then, we describe the details in a use case study in which we used our API to implement an agent application (Section IV).

II. APRON Architecture

In this section, we describe our proposed management architecture, depicted in Figure 1. APRON is a management layer that sits between the IoT application and the operating system, for example the Robotic Operating System (ROS) [9]. APRON mechanisms allow establishment and monitoring of network connectivity, estimations of failures, and enable adaptivity of the mission by a re-planner based on the customizable controller logic. Application programmers can leverage 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.

We summarize the common network management mechanisms, the API, and the agent mission planning components, while we give more attention to the mission replanning component in the next section.

Network Monitoring and Addressing. Similar to most networked systems, APRON has a connectivity management component that runs a network discovery protocol, and a watchdog process running a heartbeat protocol to monitor alive connections.

State Cache Manager. This component is a handler for the partially replicated database maintaining network states. The cache contains static states $i.e.$, states that depend solely on the agent and dynamic states $i.e.$, states that depend on the network and on other agents such as configurations and connectivity. The state cache may also store application states for logging statistics for example for IoT device battery usage.

Secure Identity Manager. Each agent may belong to multiple overlays and hence may require to be authenticated prior to a connection establishment. This APRON component manages agent identities across multiple overlay networks and provides secure connectivity with the Transport Layer Security (TLS) library.

Message Parser and Object Model. To define our APRON management object model, as well as to implement the logic of message delimiting, serializing and deserializing, we use Google Protocol Buffers [11], since it is more efficient than other text-based abstract syntax notation languages as JSON.

APRON API. We are implementing support for a set of API to customize three main components: (i) the logic of the mission planning algorithm, in a centralized or distributed fashion, (ii) the logic of the controllers, adapting to different (failure) scenarios, and (iii) the Planning Logic.

Mission Planning Controller. APRON supports a class of basic controllers to customize the mission replanning rate $R(t)$ of the network of IoT devices, e.g. drones. Our architecture

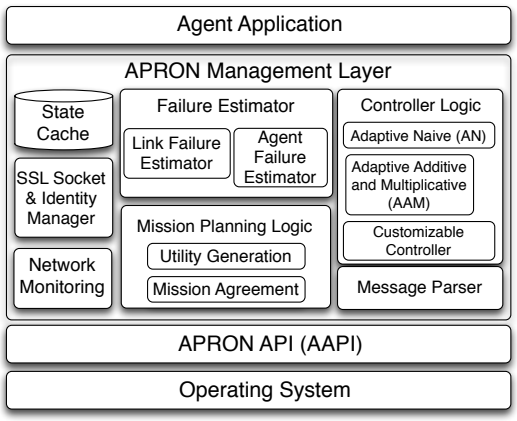


Fig. 1. APRON Architecture: a management layer between the IoT application and the operating system to establish and monitor network connectivity, to estimate failures and to adapt the task (re-)planning based on the customizable controller logic.

is however modular and could support other user-defined controllers. By controller, we mean a feedback controller, not a Software Defined Network (SDN) controller. Some controllers are herein proposed. Note that $\hat{n}(t)$ is the estimated number of tasks currently into the network and discussed in section III.

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, according to the following equation:

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

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

$$R(t+1) = k(\hat{n}(t) - \acute{n}) \tag{2}$$

where k is the gain term.

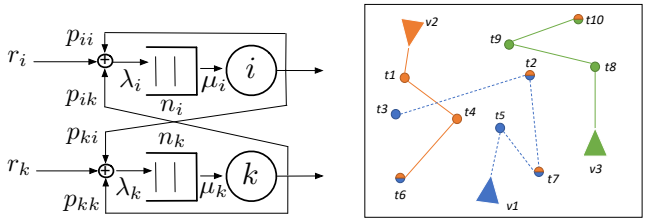
Adaptive Additive and Multiplicative (AAM) controller. The replanning rate varies with $\tilde{n}$ and $\hat{m}$ that are number of completed tasks and estimated number of tasks missing the deadline at time t, respectively:

$$R(t+1) = \begin{cases} k\frac{\hat{m}(t)}{\tilde{n}(t)+1} & \hat{m}/(\tilde{n}+1) > 0 \\ -\alpha & \text{otherwise} \end{cases} \tag{3}$$

where α is a positive constant.

III. Mission Planning via Network of Queues

Failure models are dependent on the IoT application; a failure model that would fit all scenarios probably cannot exist. Instead of predicting the failure of links or agents, we use this APRON component to compute the probability of having a given number of tasks still to be completed. Our goal is to obtain the average number of tasks, whether they are queued or in execution. This in turn can be manipulated by application programmers to determine the instantaneous utilization of each agent, and the mean throughput queuing time (both waiting and execution time) for each task. Such information can then be used, in conjunction with our APRON API, to design a controller that adapts to the application.



(a) (b)

Fig. 2. (a) Example of open Jackson queue with two agents (*e.g.* drones): tasks belonging to failing agents are reassigned. (b) Mission planning paths followed by three drones to complete their tasks.

We model the network of potentially failing agents with a Jackson's network of queues [12]. We model the effects of an agent failing with the migration of each of its tasks to a different queue. In case of drone agents, the waiting time of the first completed task by a non-failing agent is merely the traveling time of such an agent. We assume that the sequence of execution times each agent spends traveling and performing the task are independent identically distributed (*i.i.d.*) random variables. If a task is executed by an agent after n other tasks, its completion time depends on the waiting time that the task spends in the queue. When an agent fails, all its tasks still to be executed are reassigned. In general, a task can be reassigned multiple times to failing agents. A task assigned to a failing agent has therefore a completion time that depends on the time it has spent in all the queues of the failing agents, plus the time spent in the queue of the agent that executes it (holding time).

We model each agent with a single queue that stores all the tasks to be executed. We consider a network of Q queues, where the nodes are agent's 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 (Figure 2a.) We assume our system to be an *open* task replanning process consisting of $\mathcal{Q} = \{1, 2, \ldots, Q\}$ agent's queues, a vector $\mathbf{n} = \{n_1, n_2, \ldots, n_Q\}$ defining the number of tasks belonging to each of the $\mathcal{Q}$ queues, and an operator $T_{ii'}$ on $\mathbf{n}$:

$$T_{ii'} = (\ldots, n_i - 1, \ldots, n'_i + 1, \ldots) \tag{4}$$

that removes one element from the agent's queue i and adds it to queue i, or exits the system. The term *open* indicates that tasks can enter or exit the system. The vector $\mathbf{n}$ is assumed to be a Markov process with state space:

$$\mathcal{N} = \{\mathbf{n} : n_q \geq 0, q = 1, \ldots, Q\} \tag{5}$$

and transition rates given by:

$$q(\mathbf{n}, T_{ii'}(\mathbf{n})) = p_{ii'}, \tag{6}$$

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 $n > 0$, that is, it is possible for each task to be migrated 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. We were able to show that, at the steady state, the distribution of 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 function depending on the single agent's queues:

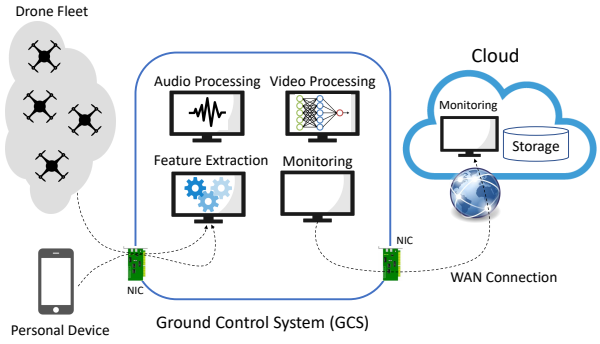


Fig. 3. Overview of the drone system realized exploiting the APRON underlying architecture. The computation is offloaded to the Ground Control System (GCS) that performs computationally intensive task.

Proposition III.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 $p(n_1, n_2, \ldots, n_Q)$ the steady state probability that there are n_q tasks in the q^{th} agent's queue for $q = 1, 2, \ldots, Q$, and if $\Lambda_k < \mu_q$ for $q = 1, 2, \ldots, Q$, 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:*

$$p(n_1, n_2, \ldots, n_Q) = p_1(n_1)p_2(n_2)\cdots p_Q(n_Q), \tag{7}$$

where $p_q(n_q)$ is the steady state probability that there are n_q tasks in the q^{th} 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 III.1 is a straightforward corollary of the Jackson's theorem [12], when there is a single server per queue. □

Using Proposition III.1 and Little's law [13], we can estimate the number of tasks in each queue at the steady state to be: $\hat{n} = E[n_i] = \frac{\rho_i}{1-\rho_i}$, where the utilization factor ρ_i of agent i's queue is defined by: $\rho_i = \frac{\lambda_i}{\mu_i}$.

This result means that when the utilization of each queue becomes too low (compared to the average in the system), an agent may request a task migration from other agents. When instead the (global or local) queue utilization becomes too high, or it is estimated to be too high, *i.e.* the system of agents is experiencing a high failure rate, a (distributed) replanning algorithm can proactively increase the replan frequency to redistribute the load.

IV. Agent Application

Since APRON exposes control of the operating system layer to edge computing application programmers through an API, we have designed and begun implementing an application that

leverages this architecture and API to give users a straightforward way to control a drone or a distributed set of drones. In this system, a human may control a drone or swarm of drones using speech instead of a physical drone controller or computer. The program utilizes Natural Language Processing (NLP) and Natural Language Understanding (NLU) techniques to discover the intent behind users' words. The user speaks directly to the personal device, *e.g.*, mobile phone, and the audio is sent to the Ground Control System (GCS) that elaborates it and sends the proper commands to the selected drone. The drone receives the commands and starts a new task for performing the desired operation.

This application is extremely useful because it reduces the technical barriers involved in piloting drones. Rather than learning how to use multiple physical controllers, humans can converse with a single application to manage the whole fleet in challenged networked environments that often necessitate intuitive and quickly-deployed IoT devices. One such example includes rapid response missions following a natural disaster. In cases where drones are used to predict and assess disaster [14] or supply emergency commodities [15] to survivors, it is essential that operations proceed as quickly and efficiently as possible. In these circumstances, the drones record the video and the audio of the environment continuously and send it to the GCS asking to process them and detect human presence and locate the position. The stream of data generated this way is collected by this node, where applications perform preliminary analytic before sending aggregated data to the Cloud.

These situations also require resilient mission systems to manage tasks in case of failure. Nonetheless, managing mission resiliency is not efficient for humans, especially in challenging environments. Users who control UAVs should not be concerned with managing or ensuring a resilient distributed system; rather, they may only want to focus on the ordering and completion of high-level tasks. The application leverages APRON's network monitoring and failure estimator component to address this problem. When failures occur, the program senses its immediate impacts and reassigns tasks effectively. Then, the conversational application can update the user of any changes to the mission and its estimated effects on other UAVs.

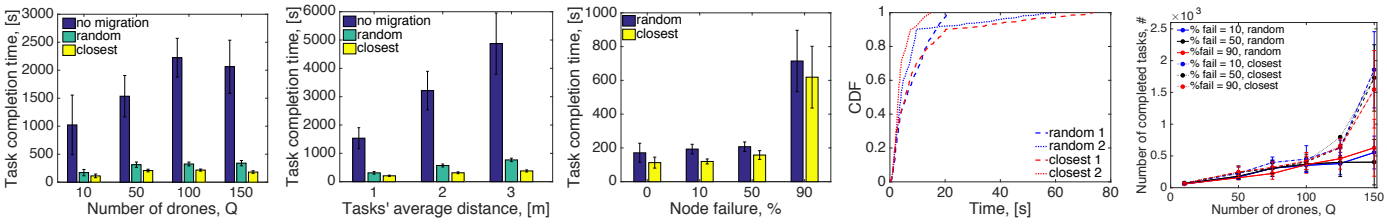
We claim that the level of abstraction provided by APRON enables programmers to rapidly prototype UAVs and resilient IoT applications like this one. However, edge computing applications that can benefit from APRON are not limited to the presented application. Examples of ICT where APRON could be helpful 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. These 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.

V. Evaluation Results

We evaluate the performance of the proposed mission planning with the development of a C++ event-driven simulator. As a use case, we considered a networked fleet of drones trying to accomplish a mission, represented by a set of geolocations to reach. These locations could be, for example, with the aim of exploring an area with a camera and microphone looking for signals indicating survivors in disaster response. Each drone estimates the threshold that triggers the migration of its tasks by using our Jackson network model. Therefore, all drones cooperate to complete the assigned tasks in the shortest possible time. Figure 2b shows the task schedule of three agents. A drone has to move from one location where it has to complete the task to the next. For example, agent v_1 originally had tasks t_5 and t_3 but it will complete also t_7 and t_2 migrated from the drone v_2.

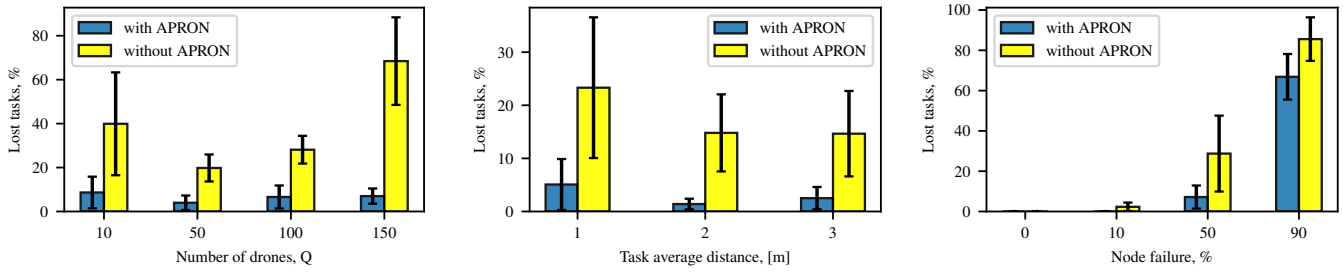
In our simulations we considered a fleet made of 10, 50 100, or 150 drones. The average distance between two tasks, that we considered being geolocation from a drone to visit has been 1, 2, or 3 meters. We investigated three different task reallocation policies: *(i) no replanning (task migration)*: agents (drones) do not cooperate but each one tries to accomplish only its own tasks; *(ii) random task replanning*: when an agent's queue exceeds a set threshold, it migrates its tasks to a randomly selected drone; *(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. We considered two scenarios: stable conditions with no failing agents, and random agents or link failures.

Our results show that APRON can be used as a tool to evaluate different task reallocation policies. Consider Figure 4. A few observations when simulating scenarios without failures are: *(1) Task migration policies show shorter mission completion time.* As shown in Figure 4a, the random policy and the closest policy allow agents to complete their tasks in a shorter time. Task migration policies take advantage of all the agents available without overloading them. *(2) The closest agent policy achieves lower completion time with respect to the random task replanning policy.* From Figure 4a-b, it is possible to note that the closest migration policy allows better exploitation of all agent geolocation hence achieving a more efficient mission plan. *(3) The task completion time decreases with the agent travel distance.* As expected, when the average travel distance between two consecutive tasks increases, the completion time also increases as shown in Figure 4b. In case of agents' random failures, the following results have been obtained. *(1) The closest agent policy achieves lower completion time than the random task replanning policy.* In case of 10 and 50% of failures, the closest policy shows better performance, confirming that the selection of the closest agent produces lower completion time than the migration of the task to a random one. This could be seen in Figure 4c.



(a) (b) (c) (d) (e)

Fig. 4. 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; while the last two graphs depicts the (d) cumulative distribution function in case of 90% of failure, and (e) Endurance: number of completed tasks before the first failure.



(a) (b) (c)

Fig. 5. 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.

But, in case of a high percentage of failures, the two policies show similar results. This could be seen also in the Cumulative Distribution Function (CDF) in Figure 4d. The results of the two simulations are plotted and they show similar performance for both policies. *(2) The completion time increases when the drone failure increases.* As shown in Figure 4c, when the number of available drones decreases, fewer agents are available for completing the tasks, therefore the queue's size increases, causing an increase of the time to complete the tasks. *(3) 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. Figure 4e shows the overlapping of the confidence intervals.

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 IV), in case 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 background. Figure 5 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.

VI. Related Work

The problem of providing a persistent and adaptive service resilient to failure is crucial for any type of IoT network in general, and robotic or drone networks in particular; so it is not surprising that there are several proposed solutions to tackle this problem. We cite only a few representative (centralized and distributed) solutions to clarify our contributions to the resilient task planning problem.

IoT at the edge. The proliferation of IoT devices led to the generation of a massive amount of data. Processing this data onboard inevitably drains the battery of the IoT device, while central cloud servers are inefficient at handling all the collected data mostly because of latency. To confront this problem, recent solutions proposed to offload the data processing at the edge of the network. Delivering edge computing services also presents challenges: a few examples are the implementation of effective offloading strategies to have an efficient workload distribution in the system, or a reliable cooperation scheme to handle mobility, such as frequent communication disconnection and low and predictable latency to save bandwidth [16]–[18]. In our scenario, the Cloud is mainly used to collect aggregated data, a delay-tolerant task.

Resilient IoT Systems. Harsh environments can be affected by problems, such as interference, medium access conflicts, multipath fading, shadowing, and so on, which can cause significant packet losses. In this scenario, and in particular in case of emergency response applications, reliability needs to be guarantee [19]. [20] and [8] focus on the prediction of failures, the first uses a Bayesian-inference probabilistic to estimate failure probability for the monitored batteries, the second establishes close feedback between the high-level planning based on Markov Decision Processes (MDP) and

the execution level learning-focused adaptive controllers. This feedback enables the proposed framework to plan by anticipating the failures and reassessing vehicle capabilities after the failures. Instead of predicting the failure of links or agents, our solution 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.

Adaptive and persistent network services. Given the distributed nature of IoT and robotic agents, decentralized approaches have also been proposed [5]–[7], [21]. In [22] the authors proposed a solution to 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 [22], our solution can also be used to distribute workload efficiently among agents, but our predictive system is based on a Jackson network approach. Moreover, APRON is agnostic to the agent architecture and it can be used to manage both centralized and distributed systems of any agent, not merely drones.

Prediction with network of queues. The robustness of network services has been studied also in distributed (peer-to-peer) storage networks. In [23] for example, a control theoretical approach to model and predict data availability by means of redundancy is proposed. To assure a given level of availability in case of storage node failures, new redundant fragments need to be introduced. As in [23], we also use 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 [12].

VII. Conclusion

In this paper we presented APRON, a management architecture whose design is driven by the attempt to increase the programmability of the resilient task replanning problem in presence of challenged edge networks. APRON leverages a Jackson's network queues model to estimate the number of tasks, queued or in execution. This information can be manipulated by application programmers to determine the instantaneous utilization of each IoT device, and the mean throughput queuing time (both waiting and execution time) for each task to be executed or offloaded to the edge cloud. Our simulations showed, with a case study of a network of UAVs, how APRON can be an effective tool for policy programmability of the mission replanning problem for any IoT device deployed in challenged networked environments. We also discussed an application using the APRON API to remotely control a fleet of UAVs by merely using a natural language processor.

Acknowledgment

This work has been partially supported by the NSF awards CNS-1647084, CNS-1836906 and CNS-1908574.

References

[1] 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.

[2] 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.

[3] 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*, 2017.

[4] V. Cerf, S. Burleigh, A. Hooke, L. Torgenson, R. Durst, K. Scott, K. Fall, and H. Weiss, *RFC 4838 - Delay Tolerant Networking Architecture*. IETF DNT Research Group, April 2007.

[5] 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.

[6] N. K. Ure, G. Chowdhary, Y. F. Chen, M. Cutler, J. P. How, and J. Vian, "Decentralized learning-based planning for multiagent missions in the presence of actuator failures," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2013, pp. 1125–1134.

[7] 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 *European Control Conference (ECC)*, 2013.

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

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

[10] Donato Di Paola, "The multi-agent robotic simulator (mars) https://github.com/donatodipaola/mars ," online, 6 2019.

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

[12] J. Jackson, "Networks of waiting lines," *Operations Research*, vol. 5 (4), pp. 518–521, August 1957.

[13] K. S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*. John Wiley and Sons Ltd., 2002.

[14] M. Erdelj, E. Natalizio, K. R. Chowdhury, and I. F. Akyildiz, "Help from the sky: Leveraging uavs for disaster management," *IEEE Pervasive Computing*, no. 1, pp. 24–32, 2017.

[15] S. Chowdhury, A. Emelogu, M. Marufuzzaman, S. G. Nurre, and L. Bian, "Drones for disaster response and relief operations: a continuous approximation model," *International Journal of Production Economics*, vol. 188, pp. 167–184, 2017.

[16] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, and 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.

[17] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, 2016.

[18] C. Puliafito, E. Mingozzi, and 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.

[19] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 2347–2376, 2015.

[20] J. Yu, "State-of-health monitoring and prediction of lithium-ion battery using probabilistic indication and state-space model," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 11, pp. 2937–2949, 2015.

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

[22] 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.

[23] A. Duminuco, E. Biersack, and T. En-Najjary, "Proactive replication in distributed storage systems using machine availability estimation," in *CoNEXT*, 2007.