Tailoring FANET-Based 6G Network Slices in Remote Areas for Low-Latency Applications

Christian Grasso
DIEEI – University of Catania
CNIT – Research Unit of Catania
Catania, Italy
christian.grasso@unict.it

Raoul Raftopoulos
DIEEI – University of Catania
CNIT – Research Unit of Catania
Catania, Italy
raoul.raftopoulos@phd.unict.it

Giovanni Schembra
DIEEI – University of Catania
CNIT – Research Unit of Catania
Catania, Italy
giovanni.schembra@unict.it

***Abstract*— In the near future, 6G networks will be able to provide its users with heterogeneous tailored end-to-end network services for edge AI-powered applications. However, in many 6G use case scenarios there may not be structured network infrastructures. One solution to meet this problem is to bring computing, networking and storage facilities on site by means of UAVs. The target of this paper is to define a management framework for Flying Ad-Hoc Networks (FANET) aimed at bringing edge computing to remote areas, guaranteeing low latency to ground devices with heterogeneous requirements in terms of mean latency. The proposed framework allows zero-touch management of different network slices realized by partitioning the Computing Elements (CE) installed on each UAV of the FANET and wireless links connecting them to each other. An Inter-Slice Orchestrator is in charge of deciding this resource partition, while an Intra-Slice Orchestrator manages horizontal offload among UAVs for each slice via Reinforcement Learning, in order to minimize the end-to-end latency.**

Keywords—6G, Network Slicing, UAVs, Deep Reinforcement Learning, Low-latency

I. INTRODUCTION

5G networks are characterized by programmability and modularity needed for supporting a multitude of heterogeneous use cases by means of different logical networks, called network slices. Each slice can be designed to satisfy the requirements of a specific use case, on top of a common network infrastructure.

The edge computing paradigm has quickly increased the need to readjust optimization models defined in cloud environment [1][2] to take into account the reduced amount of computing and storage resources that the edge offers.

The next generation of wireless communications, i.e. 6G networks, with the support of Artificial Intelligence (AI), ultra-reliability, and zero-touch network management, is expected to emerge in the near future [3]. Edge Intelligence [5][6], powered by AI techniques, is considered one of the key missing elements in 5G networks, and will represent a key enabler for future 6G networks. Powerful Deep Reinforcement Learning (DRL) techniques are expected to be at the core of decision-making processes to reach optimal performance.

In the past, UAVs organized as Flying Ad-hoc Network (FANET) have been used to enhance coverage, capacity and reliability of wireless cellular networks and to provide fog- and edge-computing facilities to remote areas. However, there are various challenges to be solved before flying platforms can effectively be used in wireless networks [7]. Among them, energy efficiency is considered one of the most critical aspects, since the battery lifetime of UAVs constitutes a strong limitation to their flight autonomy. Equipping the UAVs with inappropriate Computing Elements (CEs) could determine either an increase in energy consumption, and therefore a reduction of the flight autonomy, or unacceptable performance, making the FANET unable to meet the network slice requirements. Improving performance of a slice dedicated to time-critical vertical applications could be achieved by increasing the amount of CE computing power dedicated to that slice, but this in turn could worsen the performance of the other slices. It is therefore necessary to both optimize the inter-slice computational resource sharing and the intra-slice performance.

Therefore, in this paper, we propose a zero-touch management framework for a FANET aimed at bringing edge computing on site in remote areas not directly connected to the core network infrastructure. An Inter-Slice Orchestrator splits the computation power and the transmission bitrate of each UAV between the different slices, and an Intra-Slice Orchestrator manages the horizontal offload among UAVs for each network slice, to allow overloaded UAVs to forward the jobs to less loaded UAVs, hence reducing the latency and satisfying each slice requirements. The horizontal-offload decision problem is defined as a Markov Decision Process (MDP) to be solved via DRL.

The rest of the paper is structured as follows. Section II describes the reference system, including the proposed architecture. In Section III, we formulate the horizontal offloading problem as an MDP to be solved by each Intra-slice Orchestrator using different DRL algorithms. To this purpose, a system model is defined to formulate the offloading problem by means of a MDP. Section IV presents some numerical results to evaluate the behavior of the FANET management framework proposed in this paper. Finally, Section V concludes the paper.

II. SYSTEM DESCRIPTION AND ASSUMPTIONS

In this paper, we propose a framework to provide ground devices (GDs) deployed in a remote area with 6G network slicing. We assume that the geographic area where these GDs are deployed is not covered by any network infrastructure with edge computing facilities. To this purpose, in order to support heterogeneous vertical applications with strict latency requirements, a FANET is used to bring edge computing on site. GDs can therefore *vertically* offload jobs to the FANET. GDs can be grouped according to their characteristics in terms of job offloading rate and QoS requirements, the latter expressed in terms of mean latency. Therefore, the FANET is required to provide different network slices, one for each set of GDs.

Let N be the number of UAVs in the FANET, each equipped with a CE to provide GDs with edge computing services. The geographic area covered by the FANET is subdivided into zones, one for each UAV. The vertical

ISBN 978-3-903176-48-5© 2022 IFIP

offloading process for each slice is time variant according to the current state of activity of the GDs connected to that slice in each zone: the higher the activity, the higher the emission rate of jobs offloaded to the FANET.

Let M be the number of network slices the FANET has to provide. Therefore, each UAV dedicates a portion of its CE to each slice. Wireless links between UAVs are sliced in M portions as well, with the aim of providing isolated communication channels.

Moreover, in order to guarantee isolation when accessing the CE of each UAV, one queue is associated to each CE slice. This way, jobs belonging to different slices do not interfere with each other. Each of these queues, here referred to as *Processing Queue*, are managed with a First-In First-Out (FIFO) policy.

The time variability of the job arrival process coming from GDs can cause periods of computational overload and periods of underload of the UAVs serving these GDs. Consequently, the processing latency of jobs suffered in the FANET is not constant, but depends on the arrival rate and the computing capacity of the CE assigned to that slice. Long periods of low activity of GDs determine underload of the Processing Queues with a consequent decreasing of short-term average processing latency; on the contrary, long periods of high zone activities determine overload situations with a consequent increasing of short-term average processing latency.

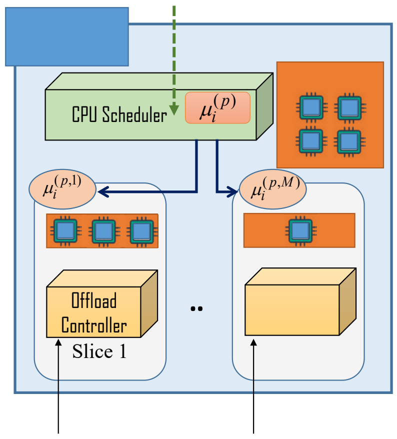
For this reason, in this paper we introduce *horizontal offload* among UAVs, to smooth the processing load of each UAV. The UAV that receives a job from ground is called the *Dwell UAV*, whereas the UAV where the job is processed is referred to as its *Processing UAV*. Jobs to be offloaded are transmitted on the wireless links according to the UAV-to-UAV link transmission rates. A queue called *Offloading Queue* is used to enqueue jobs when the transmission link is used to transmit previous jobs. Of course, the state of this queue depends on the channel transmission rate: the better the channel state, the higher the service rate of this queue.

The amount of jobs to be offloaded from one (overloaded) UAV to another (underload) one is decided according to the current state of the UAV queues. In order to achieve the target of isolation among slices, these decisions are taken for each slice independently.

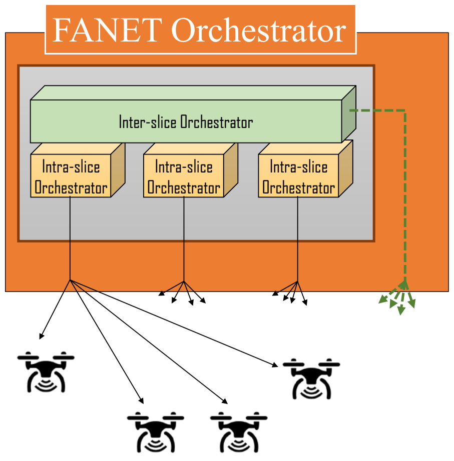
The FANET management architecture is shown in Fig. 1. More specifically, Fig. 1.a presents the functional architecture of each UAV. The *CPU Scheduler* is in charge of assigning different portions of the CPU processing rate to each slice. As shown in the figure, for the generic UAV i, we indicate the whole CPU processing rate of its CE as $\mu_i^{(p)}$, while $\mu_i^{(p,\sigma)}$ represents the portion of CPU processing rate assigned to the slice σ, for $\sigma \in [1, M]$. Of course, the following relationship links the above processing rates for each UAV i:

$$\sum_{\sigma=1}^{M} \mu_i^{(p,\sigma)} = \mu_i^{(p)} \qquad (1)$$

Deciding whether to process a job locally or offload it to another UAV is made according to an *offloading scheduling policy* that is applied by a local entity named *Offload Controller*. M Offload Controllers, i.e. one for each slice, are available in each UAV.



(a) Functional Architecture of the generic UAV i



(b) FANET Orchestrator Architecture

Figure 1: FANET Management Architecture

The *Offload Controller* of a given slice σ in a UAV covering a zone whose activity is in state a decides to offload jobs according to a probability $\rho_a^{(\sigma)}$. Dependence of the offloading probabilities on the zone activity makes the behavior of Offload Controllers adaptive to the time-variant activity behavior of the covered zones.

Designing the offloading scheduling policy is not trivial because horizontal offload causes additional latency due to the transmission from the Dwell UAV to the Processing UAV. Moreover, since horizontal offload causes a re-distribution of the load inside the FANET, this decision should be taken by each UAV with a coordination with the other UAVs.

This coordination is in charge of a centralized entity named *FANET Orchestrator*. As shown in Fig. 1.b, the FANET Orchestrator is hierarchically structured in two layers: Inter-slice Orchestrator and Intra-slice Orchestrators.

The *Inter-slice Orchestrator* is in charge of choosing the amount of CPU, $\mu_i^{(p,\sigma)}$, of the CE in the UAV i that has to be assigned to the slice σ. This decision is communicated to the CPU Scheduler of UAV i for all the slices, as shown in Fig. 1.a.

Each *Intra-slice Orchestrator*, on the other hand, is in charge of choosing the offloading probabilities to be used by the Offload Controllers of all the UAVs for the slice it manages.

For the sake of simplicity, we assume that all the UAVs in the FANET are in line-of-sight (LOS) with each other. Moreover, we neglect interference coming from any ground devices for both the altitude of the FANET and the remote places where it operates. Therefore, we can assume that all UAVs are directly connected to each other with a single-hop link having the same transmission rate, here indicated as r_L. Details regarding physical transmission from one UAV to another one are covered by many papers in the current literature (e.g. [8]) and are considered out of the scope of this paper.

Let $T_o^{(TX)}$ be the mean latency suffered by jobs on the UAV-to-UAV links. It is the ratio between the link transmission rate, r_L, and the mean job size, *J*. When the link from the *Dwell UAV* to the *Processing UAV* is busy transmitting a job, new jobs to be horizontally offloaded are enqueued in the *Offloading Queue*. When an offloaded job arrives to the Processing UAV, it is enqueued in its Processing Queue associated to the network slice of the flow of that job. Therefore, offloaded jobs suffer latency in the Processing Queue of the CE where they are processed, and also an additional latency in the Offloading Queue where they wait for transmission from their Dwell UAV to their Processing UAV. For the above reasons, choosing whether to offload a job or not is crucial for the overall system performance and requires optimization techniques to avoid introduction of unnecessary latency. This will be the objective of the next section.

III. Intra-slice Orchestration Problem for Horizontal Offload Decisions

Let us formulate the problem of choosing the horizontal offloading probabilities to be sent by the Intra-slice Orchestrator of each slice to the Offload Controllers. Fig. 2 shows the data-plane model of the generic UAV *i* for a given slice σ and the settings it receives by the FANET Orchestrator. More specifically, for the generic slice σ, let us define:

- $S^{(P,i,\sigma)}(t)$: the state of the Processing Queue of the UAV *i*, representing the current number of jobs in the Processing Queue, including the one being served in the CPU;
- $S^{(O,i,\sigma)}(t)$: the state of the Offloading Queue of the UAV *i*, representing the current number of jobs in the Offloading Queue, including the one being transmitted on the wireless TX interface;
- $S^{(Z,i,\sigma)}(t)$: the activity state of the zone the UAV *i* is covering.

The Intra-slice Orchestrator of the considered slice is in charge of choosing a set of offloading probabilities, one for each possible zone activity state. This decision will be taken periodically. Let us define the *decision epoch* as the time interval between two consecutive decisions regarding the set of offloading probabilities. During each epoch, each UAV will offload jobs coming from the GDs of a given slice σ

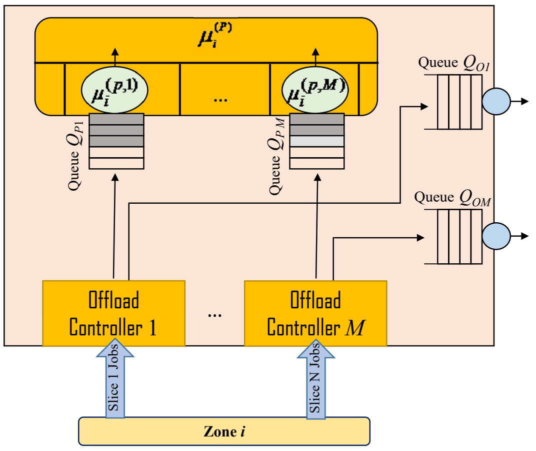


Figure 2: Data-Plane Model of the Generic UAV *i*

with probability $\rho_a^{(\sigma)}$, which depends on the activity state *a* of its zone. By so doing, the job arrival rate entering the FANET through the UAV *i* is split between the Processing Queue and the Offloading Queue.

In Section III.A we model the whole system and define the MDP.

A. Markov Decision Process

In this section, we describe the MDP that is used to support the decision-making process of the Intra-slice Orchestrator (here referred to as the agent of the decision problem) of the generic slice σ. Specifically, the agent learns the optimal set of offloading probabilities to be sent to the relevant Offload Controllers running in all the UAVs of the FANET for the slice σ, as shown in Fig. 1.

The behavior of the job vertical offloading process as seen from the FANET perspective is modeled as a bi-dimensional stochastic process $\boldsymbol{A}(t)$, whose generic element, $A^{(i,\sigma)}(t)$, represents the time-variant job arrival process to the UAV *i* in the slice σ. In order to capture both first- and second-order statistics of each of these processes, we model it as a job arrival process modulated by an underlying continuous-time Markov chain [9].

As depicted in Fig. 2, each UAV *i*, with $i \in [1, N]$, has two queues for each slice, that is, the Processing Queue, Q_P, and the Offloading Queue, Q_O. Let K_P and K_O be their sizes, that is, the maximum number of jobs that they are able to accommodate, including the one in the service facility. We describe the whole UAV state, consisting of the *M* Processing Queues and the *M* Offloading Queues, and the *M* activity states for each zone, as follows:

$$\boldsymbol{S}^{(\Sigma)}(t) = \left(\boldsymbol{S}^{(P)}(t), \boldsymbol{S}^{(O)}(t), \boldsymbol{S}^{(Z)}(t)\right) \tag{2}$$

where $\boldsymbol{S}^{(P)}(t)$, $\boldsymbol{S}^{(O)}(t)$ and $\boldsymbol{S}^{(Z)}(t)$ are three $N \times M$ matrices whose generic elements, $S^{(P,i,\sigma)}(t)$, $S^{(O,i,\sigma)}(t)$ and $S^{(Z,i,\sigma)}(t)$, for each $i \in [1, N]$ and for each $\sigma \in [1, M]$, have already been defined.

The *environment* of the MDP is the system described so far, as seen by the Intra-slice Orchestrator of the considered slice σ. It is constituted by the state of the Processing

Queues, Offloading Queues of all the UAVs in the FANET and the activity states of the zones covered by the FANET for the slice σ. The *state observations*, indicated as s_n, are the system state observed by the agent at the beginning of the epoch *n*. The *action space*, indicated as $P^{(o)}$, is the set of possible offloading probabilities $\rho_a^{(\sigma)}$ for each zone activity state *a* for the slice σ.

The *state transition* is considered stochastic because the next state does not depend on the actions decided by the agent only, but also on external factors that are not controlled by the agent, such as arrivals of new jobs and changes of the zone activity states.

Finally, the *reward* is defined to represent the objective of the proposed framework: maintaining the FANET processing latency, for each slice, low and equal for all the jobs as much as possible, independently of the activity state of the zones where they have been generated, and thus independent of their access point to the FANET. Naturally, the minimum latency achieved in a slice can be decreased by an external action of the Inter-slice Orchestrator, by increasing the amount of computing power of the CE assigned to σ, i.e. $\mu_i^{(P,\sigma)}$, but this occurs with the detriment of some other slices.

In order to introduce the reward function, we define the random process array $\underline{\delta}^{(\sigma)}(n)$ whose generic element, $\underline{\delta}^{(\sigma)}(n)$, for each $i \in [1, N]$, represents the mean value of the FANET latency suffered during the decision epoch *n* by jobs generated in the slice σ and processed by UAV *i*. As specified in Section II, for each job, this latency is the sum of the time spent in the Processing Queue of this UAV and the time spent in the Offloading Queue of its Dwell UAV. Of course, the latter is present only if the job is offloaded by another UAV.

In order to minimize the mean value of the latency $\underline{\delta}^{(\sigma)}(n)$, the reward is defined as a function of $\overline{L_F}^{(\sigma)}(n)$, representing the *average total latency* suffered in the FANET by the jobs vertically offloaded by the GDs of the generic slice σ:

$$r_n = -\overline{L_F}^{(\sigma)}(n) = -\frac{1}{N}\sum_{i=1}^{N} E\left\{\delta_i^{(\sigma)}(n)\right\} \tag{3}$$

The term $E\left\{\delta_i^{(\sigma)}(n)\right\}$ in (3) is calculated by the UAV *i* by measuring the probability distribution of the latencies suffered by all the jobs it processes, defined as the difference between the processing instant and the generation instant. If we indicate the latency probability distribution measured by the UAV *i* at the end of the epoch as $f_i^{(L_F,\sigma)}(l)$, the Intra-slice Orchestrator of the slice σ derives $E\left\{\delta_i^{(\sigma)}(n)\right\}$ as follows:

$$E\left\{\delta_i^{(\sigma)}(n)\right\} = \sum_{\forall l} l \cdot f_i^{(L_F,\sigma)}(l) \tag{4}$$

IV. Numerical Results

In this section, we evaluate the performance of the proposed framework by showing some numerical results. In Section IV.A, we give some details of the reference scenarios that we considere for performance evaluation. In Section IV.B, we show the numerical results and discuss how the computing element processing rate and the link transmission rate allocated to each slice influence the overall performance. Moreover, the impact of the number of UAVs making the FANET is analyzed.

A. Use Case Description

We consider different FANETs with different number of UAVs that cover an area where $M = 2$ different slices are required. The arrival rate of jobs vertically offloaded by GDs has been characterized by analyzing the measurements collected in a real testbed deployed at the University of Catania Campus, constituted by two sets of video cameras. The first set has been used to count vehicles crossing the main road intersections, while the other one has been used for motion detection of pedestrians at the entrance of some buildings.

We therefore identified, for each zone, and for each slice, two main activity states, hereinafter referred to as *low-activity* (L) and *high-activity* (H), i.e. the set of activities is $\Im^{(A)} = \{L, H\}$. The transition rate matrix and the job-arrival rate array, calculated as the solution of an inverse eigenvalue problem [10]-[11] from the traces of the first slice are:

$$\underline{Q}_1^{(A)} = \begin{bmatrix} -3.32\cdot 10^{-3} & 3.32\cdot 10^{-3} \\ 3.26\cdot 10^{-2} & -3.26\cdot 10^{-2} \end{bmatrix} \tag{5}$$

$$\underline{\Lambda}_1^{(A)} = [7.81/N, 13.84/N] \text{ kjob/s} \tag{6}$$

On the other hand, the transition rate matrix and the job-arrival rate array for the second slice are:

$$\underline{Q}_2^{(A)} = \begin{bmatrix} -2.28\cdot 10^{-3} & 2.28\cdot 10^{-3} \\ 2.04\cdot 10^{-2} & -2.04\cdot 10^{-2} \end{bmatrix} \tag{7}$$

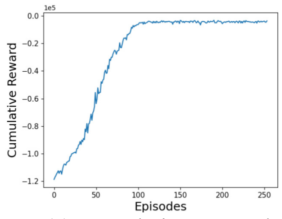
$$\underline{\Lambda}_2^{(A)} = [7.25/N, 12.4/N] \text{ kjob/s} \tag{8}$$

We assume that each slice has a *target average latency*, representing the maximum average latency that it can tolerate. More specifically, we consider $\overline{D}_{F,TGT}^{(1)} = 10$ ms for slice 1, and $\overline{D}_{F,TGT}^{(2)} = 5$ ms for slice 2. Let $K_P = K_O = 200$ jobs be the sizes of the Processing and the Offloading Queues of each UAV.

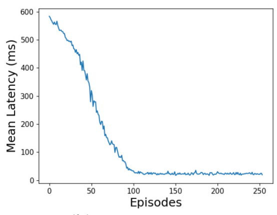
We compare our framework with two heuristic algorithms for performance comparison, i.e. Local Processing (LP), in which each job is always processed by the Dwell UAV, and Uniform Offloading (UO), in which each UAV has the same probability to either process the jobs locally or offload it to the least loaded UAV.

The experiments were performed via a simulator based on OpenAI Gym [13] and Pytorch [14]. We choose to use the Proximal Policy Optimization (PPO) [12] algorithm, one of the most widely used RL methods based on Actor-Critic methods. The main benefits that this algorithm provide are consistency, stability and little parameter tuning. In our setup, both actor and critic networks have two fully connected layers with 128 neurons each. We used the Adam optimizer with a learning rate of $3\cdot 10^{-4}$ and set the discount factor γ = 0.95. We trained each agent for $5\cdot 10^{4}$ steps on multiple environments with different seeds. This enabled us to speed up the convergence time.

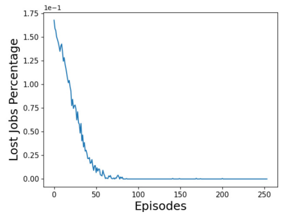
To evaluate the convergence of the proposed method, in



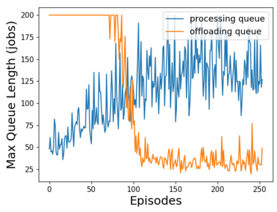
(a) Cumulative Reward



(b) Mean Latency



(c) Lost Packet Percentage



(d) Max Episode Queue Length

Figure 3: Measured Performance during Training

Fig. 3 we show the performance of the system with $N = 4$ UAVs as the training progresses, in terms of cumulative episode reward, mean latency, lost job percentage and maximum queues lengths. As we can see in Fig. 3.a, the reward converges in about 100 episodes. As the cumulative episode reward improves, the overall system performance improves as well, as shown by the decrease of both lost job percentage and latency in Figs. 3.b and 3.c, respectively. Fig. 3.d shows the maximum length of both processing and offloading queues during each episode. During the first few episodes, the offloading probabilities are too high compared to the transmission rate capabilities of each UAV, therefore the offloading queues saturate. Then, as the PPO agent learns, the offloading probability are intelligently adjusted and therefore the saturation of the offloading queues is avoided, which in turn means that less packets will have to be discarded.

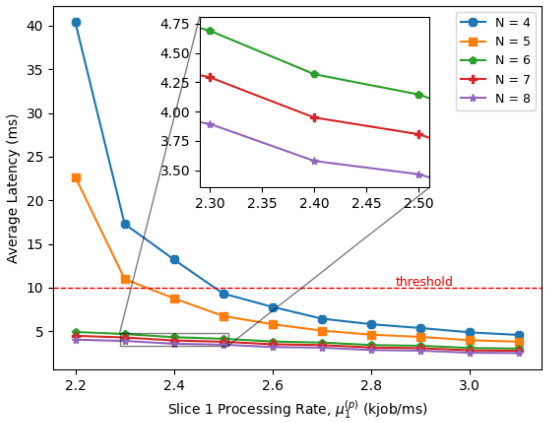
B. *Performance Evaluation*

In this section, through some simulation experiments, we evaluate the performance of the proposed FANET management framework, referred in the figures as Horizontal Offloading Management (HOM).

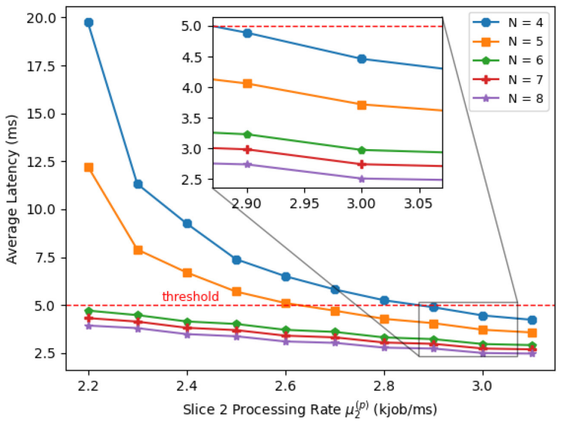
For performance evaluation, we first measured the performance the FANET provides to each slice, as the processing rate of the UAV CE allocated to that slice increases in the range [2.15, 3.15].

As expected, as the available processing rate for each slice increases, the overall performance improves, as shown in Fig. 4, where the average total per-slice latency is presented for different numbers of UAVs. The processing rate needed to each slice to reach the defined target performance (highlighted as a threshold line in the figures) can be obtained by looking at the minimum processing rate whose performance satisfies the requested one.

Fig. 5 shows the FANET performance as the offloading rate among UAVs increases. As the offloading rate increases, the latency in the offloading queues decreases, and therefore the Intra-Slice Orchestrator can easily leverage the processing capabilities of all the UAVs in the FANET. On the other hand, as the offloading rate gets worse, offloading



(a) Slice 1

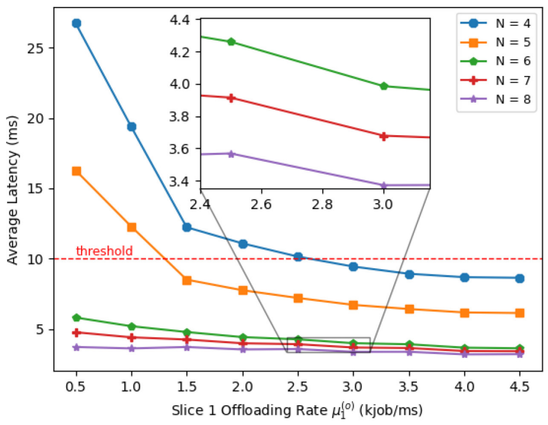


(b) Slice 2

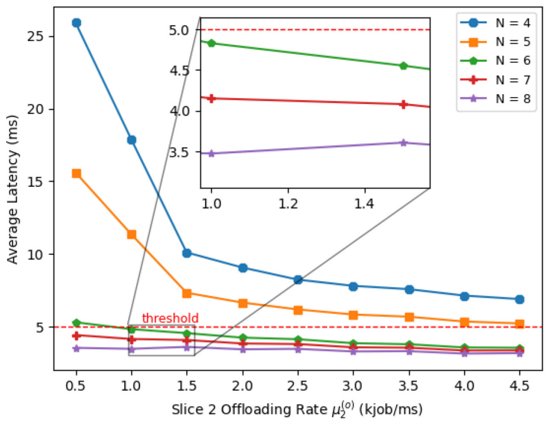
Figure 4: Average Latency Scenario 1 – Increasing Processing Rate

introduces too much latency, which in turns means that the Intra-Slice Orchestrator will be discouraged to frequently offload jobs. Moreover, you can see in Fig. 5.b that, by increasing the offloading rate, although the performance improves, it is still not sufficient to guarantee the latency requirements defined by the target average latency of each slice when the number of UAVs is less than 6. On the other hand, improving the processing rate, as already shown in Fig. 4, we are able to satisfy the requirements even in the case of FANETs with $N = 4$ and $N = 5$ UAVs.

However, the CPU computation power strongly affects the power consumption and, consequently, the maximum time of each UAV battery charge lifetime, and therefore the maximum duration of the FANET mission. For this reason, in Fig. 6 we show the FANET flight autonomy as a function of the CPU computation power of the UAV CE. Specifically, we show, for each FANET, the minimum processing rate required to satisfy the target average latency of both the slices. We assumed that the FANET is realized with quadcopter UAVs with an engine power consumption $\wp^{(EN)} = 66$ W, with Lithium batteries with capacity $\beta_C = 80$ Wh. The total power consumption of each UAV, $\wp^{(TOT)}$ is calculated as the sum of $\wp^{(EN)}$ and the power consumption of the CE, $\wp^{(CE)}$, that can be calculated as $\wp^{(CE)} = \varsigma \cdot \mu^{(P)}$, where ς is the average energy consumption required to process one job. From measurements on a real testbed, we have used $\varsigma =$



(a) Slice 1



(b) Slice 2

Figure 5: Average Latency Scenario 2 – Increasing Offloading Rate

59.1 mJ. The FANET flight autonomy depends on both $\wp^{(TOT)}$ and the battery capacity, β_C, as $\chi = \beta_C / \wp^{(TOT)}$. As expected, the FANET flight autonomy increases as the battery capacity increases and the CPU processing rate decreases. Also, the proposed framework, labeled in Fig. 6 as HOM, is able to let the FANET mission last much longer compared to the other heuristics by achieving lower latency with lower processing rate. This is due to the fact that the offloading probabilities are intelligently and dynamically changed when using the HOM policies, whereas they are statically assigned in the other policies. From a joint analysis of Figs. 4, 5 and 6, the system designer could decide CPU computation power, number of UAVs and link transmission rate that meet both each slice requirements and guarantee certain FANET flight durations.

V. Conclusions

In this paper, we have considered scenarios where a FANET is deployed in a remote area with insufficient network infrastructures to support specific delay-sensitive vertical applications. The requirements of each slice can be met by intelligently leveraging horizontal offload among UAVs and allocating the proper amount of computing element (CE) to each slice. The Inter-slice Orchestrator is an entity in charge of deciding the amount of CPU computation power to the different slices. The Intra-slice Orchestrator entity of each

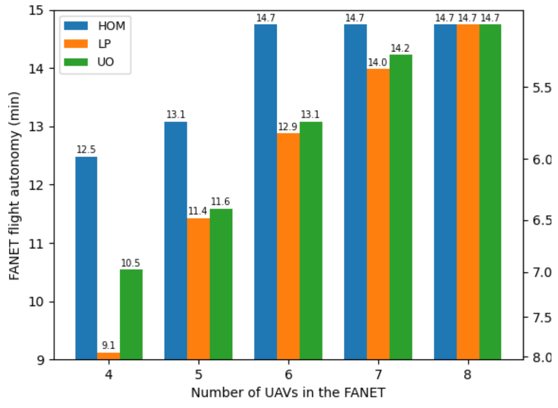


Figure 6: FANET flight autonomy

slice receives the current FANET state and calculates, via Deep Reinforcement Learning, the job offloading probabilities to be applied by each Offload Controller.

An extensive simulation campaign demonstrates the gain of the proposed framework compared to two heuristics chosen as a reference.

Acknowledgments

This work has been partially supported by MIUR, Italy under the PRIN “Liquid_Edge”, by the PIACERI project “OMNIA”, and the POR S6 project.

References

[1] G. Colajanni and P. Daniele, “A cloud computing network and an optimization algorithm for iaas providers,” in Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing, ser. ICC ’17. New York, NY, USA.

[2] A mathematical network model and a solution algorithm for iaas cloud computing,” in Networks and Spatial Economics. Springer, 2019.

[3] H. Yang, A. Alphones, Z. Xiong, D. Niyato, J. Zhao and K. Wu, “Artificial-Intelligence-Enabled intelligent 6G networks,” IEEE Network, vol. 34, no. 6, pp. 272-280,Nov./Dec. 2020.

[4] C. Qiu, X. Wang, H. Yao, Z. Xiong, F. R. Yu and V. C. M. Leung, "Bring Intelligence among Edges: A Blockchain-Assisted Edge Intelligence Approach," GLOBECOM 2020, Taipei, Taiwan, 2020.

[5] C. Qiu, X. Wang, H. Yao, Z. Xiong, F. R. Yu and V. C. M. Leung, "Bring Intelligence among Edges: A Blockchain-Assisted Edge Intelligence Approach," GLOBECOM 2020 - 2020 IEEE Global Communications Conference, Taipei, Taiwan, 2020, pp. 1-6.

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," in Proc. of the IEEE, vol. 107, no. 8, Aug. 2019.

[7] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," in IEEE Communications Magazine, vol. 58, no. 3, pp. 55-61, March 2020.

[8] T. Zeng, M. Mozaffari, O. Semiari, W. Saad, M. Bennis and M. Debbah, "Wireless Communications and Control for Swarms of Cellular-Connected UAVs," 2018 52nd Asilomar Conference on Signals, Systems, and Computers, 2018, pp. 719-723.

[9] M. A. Farahani and M. Guizani, “Markov modulated Poisson process model for hand-off calls in cellular systems,” 2000 IEEE Wireless Communications and Networking Conference, vol. 3, pp. 1113-1118, Chicago, IL, USA, 2000.

[10] Chia-Lin Hwang and S. Q. Li, “On the Convergence of Traffic Measurement and Queueing Analysis: A StatisticalMatch Queueing (SMAQ) Tool”, *IEEE/ACM Trans. on Netw.*, vol. 5, no. 1, Feb. 1997.

[11] A. Lombardo, G. Morabito and G. Schembra, "An accurate and treatable Markov model of MPEG-video traffic," Proceedings. *IEEE INFOCOM '98*, San Francisco, CA, USA, March 29 to April 2, 1998.

[12] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[13] "Gym: a toolkit for developing and comparing reinforcement learning algorithms," https://gym.openai.com/, last access November 09, 2021.

[14] "Pytorch: an open source machine learning framework," available at https://pytorch.org/, last access November 09.