Dynamic Distributed Computing for Infrastructure-Assisted Autonomous UAVs

Davide Callegaro[1], Sabur Baidya[2] and Marco Levorato[1]
[1]Donald Bren School of Information and Computer Sciences, UC Irvine, CA, US
[2]Department of Electrical and Computer Engineering, University of California, San Diego
e-mail: {dcallega, levorato}@uci.edu sbaidya@eng.ucsd.edu

***Abstract*—The analysis of information rich signals is at the core of autonomy. In airborne devices such as Unmanned Aerial Vehicles (UAV), the hardware limitations imposed by the weight constraints make the continuous execution of these algorithms challenging. Edge computing can mitigate such limitations and boost the system and mission performance of the UAVs. However, due to the UAVs motion characteristics and complex dynamics of urban environments, remote processing-control loops can quickly degrade. This paper presents Hydra, a framework for the dynamic selection of communication/computation resources in this challenging environment. A full – open-source – implementation of Hydra is discussed and tested via real-world experiments.**

***Index Terms*—Edge Computing, Urban Internet of Things, Unmanned Aerial Vehicles, Autonomous Systems.**

I. Introduction

The ability to observe and analyze the surrounding environment to inform decision making is the key to autonomy. In physical systems, state information is extracted by acquiring and processing endogenous and exogenous signals in real-time. Despite the important advances both in algorithms and embedded platforms of the recent years, the execution of sensing-processing-control pipelines in lightweight airborne platforms such as commercial Unmanned Aerial Vehicles (UAV) is a non-trivial problem. Rather intuitively, there exists an inherent tradeoff between three key metrics: accuracy, decision delay, and energy consumption. Improving accuracy of analysis often requires increasing complexity, which comes at the price of a larger execution time, and thus decision delay, or a larger weight and energy consumption.

Herein we focus on video analysis, and specifically on object detection, an important component of most advanced autonomous systems. Modern object detection algorithms take the form of Deep Neural Networks (DNN). The most performing DNNs are extremely complex, and their execution requires powerful computing platforms. Two key recent advancements make object detection possible in constrained mobile devices:

- The development of techniques such as distillation, pruning and quantization led to the construction of effective simplified DNN models with a significantly reduced complexity compared to the full models.
- The development of powerful embedded computers equipped with accelerators and GPUs.

This work was partially supported by the NSF under grant IIS-1724331 and DARPA under grant HR00111910001.

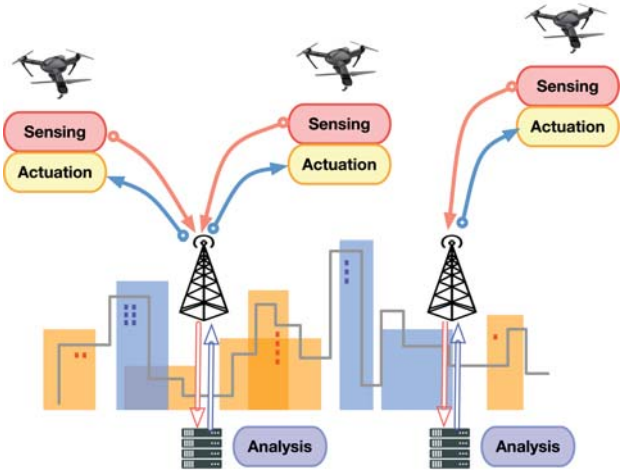

Figure 1: Edge computing scenario considered in this paper: UAVs offload analysis modules to ground edge servers. The architecture we developed enables the seamless distribution of modules across devices, as well as to re-route data analysis in real-time to improve performance.

Intuitively, despite the clever optimized design of simplified models, aggressive complexity reduction results in a perceivable degradation of accuracy. We remark that lower complexity also means a shorter execution time, that is, a smaller time between information acquisition and control – here referred to as *capture-to-control time* – a critical parameter for an effective control. The modern GPU-equipped embedded boards mentioned above allow fast execution of fairly complex DNN models. However, the use of GPUs to speed up execution significantly increases energy consumption, another crucial metric, especially when considering airborne systems.

Edge computing [1]–[3], a paradigm where compute-capable devices connected to the network edge take over the computation tasks of mobile devices, can mitigate the issue described above. The mobile devices, in the scenario at hand the UAVs, are alleviated from the computation burden of analysis, thus removing that component from the overall energy consumption, and possibly speeding up execution [4]. However, offloading computation tasks to an edge server introduces significant uncertainty [5], mainly due to the volatility of the wireless link connecting the UAV to the server. Intuitively, while the additional uncertainty is often not a prominent factor in many mobile applications, using edge computing

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

to support UAVs autonomous operations necessitates meeting hard constraints on the capture-to-control time on a continuous flow of tasks.

Most recent contributions on edge computing for UAVs focus on planning aspects, and mostly from an purely abstract perspective [4,6]–[8] or center their attention on UAV-assisted edge computing and cloudlets [9]–[13]. This paper seeks insights on edge computing *for* UAVs from a real-world deployment and real-world testing, especially to characterize and counteract temporal variations in capture-to-control time shaped by variations in the channel conditions, including gain and congestion level. Specifically, we make the following contributions:

(*i*) We develop an experimental platform realizing an infrastructure assisted UAV system. We focus on navigation tasks based on object detection via DNN models, and equip the UAV with one of the most powerful embedded computers for machine learning to enable a fair comparison with offloading to edge servers.

(*ii*) We develop *Hydra* [14], a middleware architecture enabling the adaptive distribution of computation tasks within infrastructure assisted UAV systems. The modular architecture grants significant flexibility in deploying sensing, analysis, and control pipelines, and includes a logic to dynamically activate-deactivate pipelines in response to changes in the state of their components or environment.

(*iii*) We test the architecture to illustrate its performance against variations of channel gain and contention/interference from other mobile devices.

Our experimental results indicate that offloading complex analysis tasks to edge servers grants a significant reduction in the overall energy intake of operating the UAV, thus prolonging mission lifetime. The proposed Hydra architecture is shown to provide reliable control against fluctuations of the capture-to-control time at different temporal scales based on a tunable tradeoff between energy and delay performance.

The rest of the paper is organized as follows. Section II introduces the experimental platform and provides a preliminary discussion of the problem at hand. In Section III, we present and discuss the architecture and logics of Hydra. Section IV presents and discusses the experimental results. Section V concludes the paper.

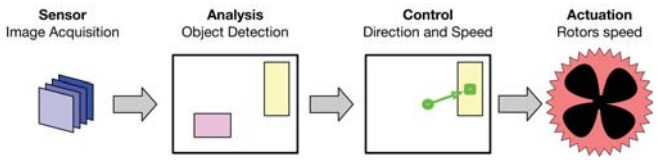

Figure 2: Sensing, analysis and control pipeline where object detection performed on images acquired by the UAV is used to control navigation.

II. Preliminary Discussion

First, we describe the task and experimental platform, and make some preliminary considerations on metrics of interest.

Computation Task - We consider the sensing-analysis-control pipeline illustrated in Fig. 2: the onboard camera acquires images that are analyzed using an object detection algorithm, whose output is a series of labeled bounding boxes. The control module selects a bounding box with a predefined label, and produces steering commands. The objective of steering is to center the bounding box with respect of the vision range of the UAV and match it to a predefined size. Note that our objective is to analyze communication-computation aspects of this class of problems, so we focus on the accuracy, delay and energy consumption associated with the pipeline, rather than on the specific output control.

We use ssdlite mobilenet v2 [15] trained on the Coco dataset [16] and floating Point 32 bits precision for object detection, a highly optimized model designed for mobile devices. Performance in terms of accuracy is expressed in terms of mean Average Precision (mAP), which measures a combination of precision, recall and bounding box intersection with ground truth. The model we use achieves a mAP equal to 22 [17], compared to the $37-43$ of full sized models which are out of reach of most embedded computers.

Experimental Platform - Experiments are performed using a 3DR solo quadcopter customized to mount on an attached plate an additional embedded computer and battery, a GoPro Hero 4 camera, and a Magewell HDMI to USB converter. Specifically, we use the Nvidia Jetson Nano with 4GB RAM, Quad-core ARM Cortex-A57 MPCore processor and 128-core Nvidia Maxwell GPU. We use as edge servers Nvidia Jetson TX2 boards with 8 GB RAM, hex-core ARMv8 64-bit CPU and an integrated 256-core Nvidia Pascal GPU.

The UAV connects to the edge servers using WiFi communications, and specifically IEEE 802.11n, which offers higher data rates (upto 130 Mbps application throughput) compared to IEEE 802.11 a/b/g and can operate both on 2.4 GHz and 5 GHz band. We configure the access points to operate in the 2.4 GHz band on non-overlapping channels.

Preliminary Considerations - The Nvidia Nano is a recently released extremely powerful embedded computer, specifically designed to provide state of the art performance in executing machine learning algorithms. The average time to execute the bare object detection task is 87 ± 6 ms, which almost equal to the 75 ± 8 ms achieved by the Jetson TX2 board. We report that the execution of other models, such as ssd mobilenet v1, took almost the same time on the Nano and half of the time on the TX2 likely due to specific architectural characteristics. We choose v2 to benchmark our system due to its slightly higher accuracy and to provide the most advantageous conditions to the local analysis option. We remark that a more powerful edge server would obviously advantage remote analysis pipelines, and that any other task using the GPU at the UAV

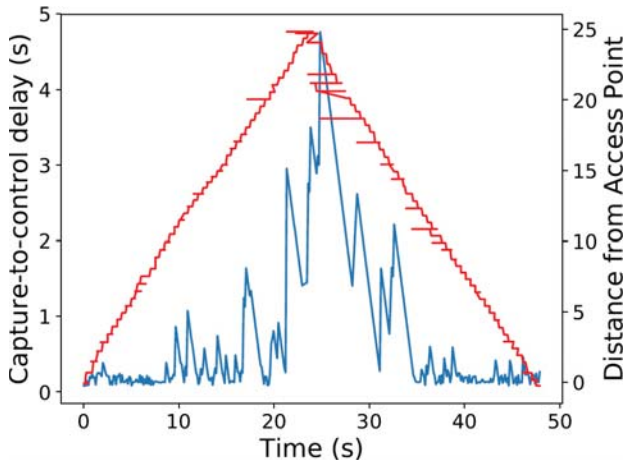

Figure 3: Temporal variation of capture-to-output delay and distance from a reference edge server as the UAV moves away from and toward it.

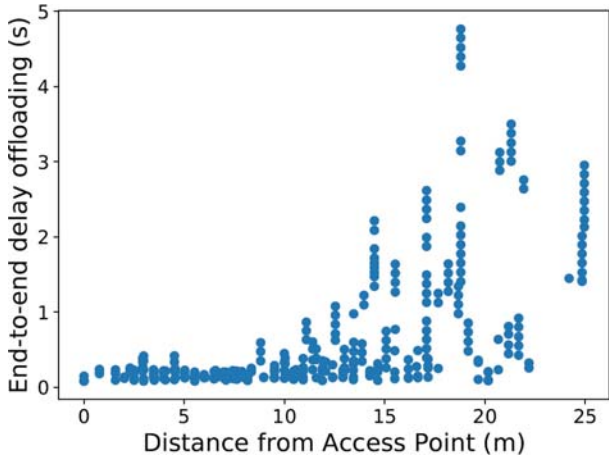

Figure 4: Capture-to-output time as a function of the distance between UAV and the edge server.

could significantly impact the performance of local onboard analysis loops.

However, the extreme performance of the Nano comes at the price of a high energy consumption. Continuous local computing requires 4 ± 0.5 W (measured during flight using the Jetson Nano utilities), which is more that 10% of the 38 W required by the UAV to hover or navigate. This is mostly connected to the use of the GPU, however, our extensive testing on other embedded computers resulted in execution times above half a second.

Continuous offloading to 1 or 2 edge servers requires 2 ± 0.2 W and 2.055 ± 0.1 W, respectively. Therefore, offloading dramatically reduces energy intake, and would prolong mission time. However, the capture-to-control time of pipelines through available edge servers has additional components corresponding to the transfer of the image from the UAV to the server and of steering commands on the reverse path. Importantly, those components are heavily influenced by latent variables such as path loss and channel load, and present large fluctuations at fine-time scale due to fading, as well as channel access and transport protocols' parameters.

Fig. 3 shows an experimental trace obtained as the UAV is flying away from the edge server – a detailed description of the software and experimental parameters used to obtain these results is provided later. Micro and macro scale variations are observed on a general trend of degradation due to path loss. Distance spikes, automatically corrected, are due to wind. Fig. 4 reinforces the notion that distance and connection to edge servers are a rather poor predictor of the delay, with large variations and clustering effect.

As shown in the experimental results section, the onset of heavy-duty data streams has a more abrupt impact on the capture-to-delay time, whose trajectory presents sudden spikes and a more erratic behavior.

The need for a flexible and adaptive strategy is apparent. Importantly, the reaction of the system cannot be exclusively driven by macro-scale parameters such as distance due to sharp delay spikes triggered by micro-scale effects such as the dynamics of the inner state of communication and networking protocols. Moreover, hardly observable variables such as channel load and exogenous traffic emission have a considerable impact on the delay and its dynamics.

III. HYDRA

Herein, we describe the structure of Hydra and its embedded logics, which enables the activation/deactivation of pipelines, both local in-device and through edge servers.

Hydra Architecture

As shown earlier, autonomy pipelines are composed of three main logical blocks, namely sensing, data analysis and control, which transform environmental or internal signals into control. HYDRA allows the construction of flexible pipelines, where the flexibility is both in where the blocks are executed and which blocks are executed. To this aim, the open-source architecture we developed [*citation to repository is omitted to preserve authors' anonymity*] is modular, where the module abstraction corresponds to an encapsulation of data transformation functions. The high level schematics of Hydra is depicted in Fig. 5.

The core of Hydra is a reliable threading of the modules over a distributed system, where the threading itself is controlled by a logic. Every module is characterized by a core function, and is equipped with an input and one or many output queues. The queues are the interfaces between a specific module and all the other – local or remote – modules. Thus, the flow of information, as well as the deactivation of pipelines, is controlled by the routing strategy implemented by the modules. For instance, the deactivation of a pipeline will be realized by disabling an output queue, thus avoiding the summoning of the modules following it.

Some input queues implement filtering to remove replicas of data structures transiting the modules. In our implemen-

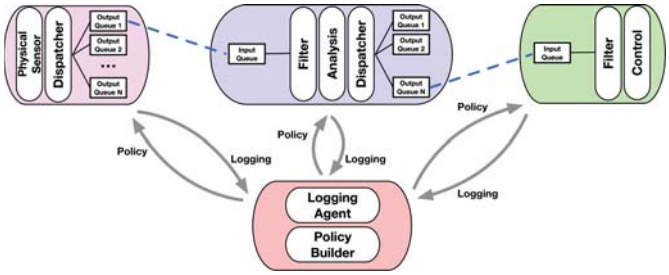

Figure 5: High-level schematics of the modular structure of Hydra.

tation, the input queue of the actuation module at the UAV detects and filters out replica outputs of data analysis modules – that is, outputs corresponding to the same initial data – to avoid the implementation of duplicate steering actions. Moreover, some queues log the activity to monitor the performance of the associated pipelines. In our implementation, the logging driving pipeline selection is delegated to the input queue of the final actuation module at the UAV. The input queue of this module will log the capture-to-control time of active pipelines by tracking the delivery time of control outputs with respect to the generation time of their corresponding frame. Note that both times are generated at the same device.

A module outside those realizing the transformation of the data implements the high-level control logic of HYDRA. Specifically, we concentrate all intelligence in the *logics* module. The module collects the logs from target input queues and determines the routing policies of selected output queues. In this specific implementation, the logics module collects the temporal patterns of frame emission and output reception from the onboard sensing and control modules, respectively. The sequences of delays are, then, used to compute the average driving pipeline activation/deactivation and selection.

The logics module can also control data capture parameters to optimize the flow of information through the pipelines and maximize performance. Herein, we set the image capture rate to match the performance of the fastest pipeline over a moving window. Intuitively, an exceeding capture frequency will overload the active pipelines and create undesirable queueing effects, which may have a substantial impact on the capture-to-control delay. In the current implementation, we enforce a strict queueing policy, where we store only the most recent data structure received from the previous modules. This simple strategy avoids delay accumulation, and results in an "effective" capture rate where all dropped samples are not accounted for.

Hydra Logics

Intuitively, the macro-scale parameters governing temporally local capture-to-output delays are hardly observable. Even if some network interfaces can report the value of some relevant variables, such as channel gain and modulation, many others remain inaccessible unless a complex, and possibly resource consuming, information exchange is established. The most eminent examples include channel load and server load, which would require a direct exchange of information with local access points and available servers. Remarkably, even once these macro features of the environment are known, as shown in the previous section the capture-to-control time still presents complex patterns inducing possibly large performance variations.

We, then, take the simple but effective approach of using directly observable parameters to drive the system, where the selection process is guided by the observation of the actual capture-to-control time offered by available – active – pipelines. An important advantage of this strategy is that of being completely agnostic to the technologies and protocols used within the system, meaning that the state of all the pipelines is represented by a homogeneous set of variables. We remark that local computing at the UAV (local pipeline) is an option available to the UAV possibly with a reduced degree of uncertainty compared to offloading.

Although conceptually trivial, this approach presents an important challenge: only the capture-to-control delays associated with the currently used pipeline are observable. Intuitively, on the one hand, the activation of only one pipeline does not provide any information on other available pipelines. On the other hand, the activation of all the available pipelines – including the local one – would maximize the information available to the UAV, but maximizes the burden imposed to the surrounding networks and servers, possibly decreasing global performance. In fact, any active – non-local – pipeline uses the channel resource to support information exchange, and the server resource to complete the offloaded tasks.

Hydra takes this observation as a starting point to build an adaptive, and parsimonious, strategy for the exploration-exploitation of available resources. In Hydra, pipelines are, thus, activated to: (*i*) use the corresponding resources to generate control outputs; (*ii*) reduce uncertainty in the minimum capture-to-output time – that is, the first available outcome associated with an acquired sample; and (*iii*) update the state estimate of available pipelines. The key, then, is to control the activation of pipelines when necessary to one of these purposes, while minimizing the active time of pipelines whose output will not be used.

In order to control the activation of the pipelines, we define 3 operational modes, namely *Performance* ($\psi(t)=P$), *Exploration* ($\psi(t)=E$), and *Reliability* ($\psi(t)=R$), where $\psi(t)$ is the mode at time t. In the *Performance* mode, the UAV utilizes only one, remote, pipeline, which is achieving the smallest known capture-to-output delay. In the *Exploration* mode, the UAV activates available remote pipelines to update their known "state", and perform an informed selection. In *Reliability*, local computing is activated in addition to all the remote pipelines to guarantee an almost constant capture-to-control time when other options have degraded performance. Note that variations of these modes, where only subsets of pipelines are activated can be included in Hydra.

The selection of the mode is performed based on a recent window of capture-to-control times. Define $\tau_{p,j}$ as the capture-

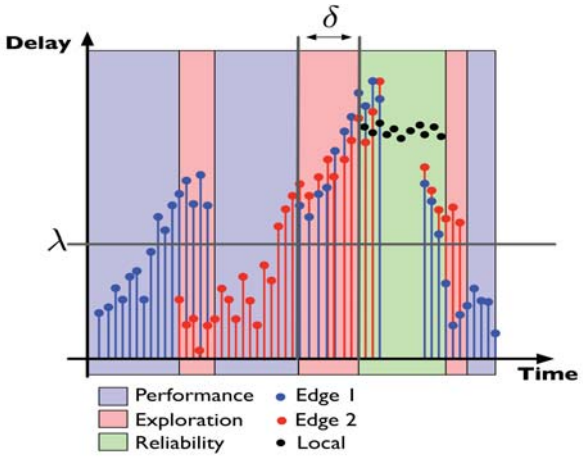

Figure 6: Illustration of the threshold based pipeline activation and selection strategy.

to-control time of image n processed through the pipeline p. At image N, the future mode and edge server selection are determined by the functions

$$E_p = \sum_{j=N-W+1}^{N} \alpha^{W-j} \tau_{(p,j)}, \quad M_p = \max_{j=N-W+1,\ldots,N} \tau_{(p,j)}, \tag{1}$$

respectively corresponding to the moving average window and maximum of the last W images' capture-to-control delay.

Fig. 6 illustrates the modes and the selection strategy. Assuming pipeline p^* through an edge server is being currently used in *Performance* mode, the system switches to *Exploration* if M_p is larger than the threshold λ. In this mode, if for at least one of the pipelines M_p is below λ, then among those the pipeline with the smallest E_p is selected and the system returns to *Performance*. In *Exploration*, the system transitions to *Reliability* after δ samples unless at least one of the pipelines has delay below λ.

IV. Experimental Results

We now presents and discuss experimental results illustrating relevant tradeoffs between accuracy and energy of available computing options. Experiments are performed placing two edge servers at a distance of 25 m (Edge Server 1 and 2, respectively). A fixed image is used to provide a stable performance reference in all the experiments. The image is of size 480×640 RGB pixels and compressed using JPEG to 21 KB. The UAV is set to move for 10 minutes on the line between the two edge servers at a speed of 1 m/s to illustrate the impact of distance from the edge servers and guarantee reproducible experiments across our measurement campaign. Note that in the actual tracking application, variations in the capture-to-control time of different strategies could result in different motion trajectories of the UAV.

Fig. 7 depicts the average capture-to-control time and power consumption over the experiment as a function of the threshold λ. When λ is set to 0.1 s, Hydra almost exclusively chooses local computing over remote computing options. This results

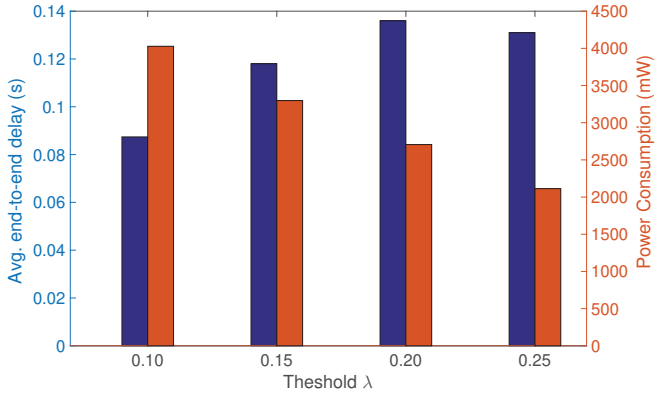

Figure 7: Avg. capture-to-control time and power consumption over the experiment as a function of the threshold λ.

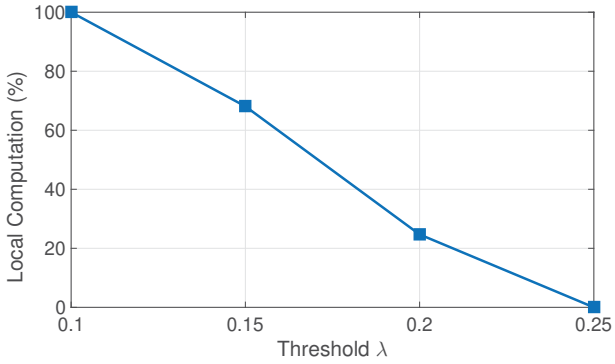

Figure 8: Fraction of time in which local computing is active as a function of the threshold λ.

in a delay of about 0.09 s, and an energy consumption of approximately 4 W. As λ is increased, the system increases the fraction of time in which local computing is deactivated and one – or two – of the pipelines through the edge servers is kept active. At λ is set to 0.25 s, the delay increases to approximately 0.13 s, and the energy decreases to about 2 W. The trend is illustrated in Fig. 8, where the fraction of time in which local computing is active as a function of the threshold λ is shown.

The ability of the system to quickly adapt to delay variations is demonstrated in Fig. 9, which shows a temporal trace of the active pipelines. The UAV is at first connected to Edge Server 2. However, as it moves away from it, the delay degrades and Hydra transitions to *Reliability* mode, activating Edge Server 1. As the connection to Edge Server 1 is still flimsy, both edge-based pipelines have a large delay, and after δ samples local computing is activated. As the link to Edge 2 improves, windows of low delay allow the deactivation of local computing. We remark that when multiple pipelines are active, the first received control is used. Therefore, the smallest delay in the plot is the effective delay perceived by the controller.

In order to further evaluate the system's dynamics, we inject in the channels used by Edge Server 1 and 2 traffic from

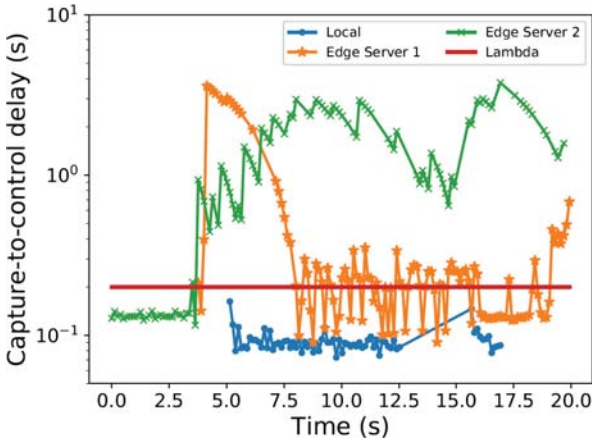

Figure 9: Temporal pattern of the capture-to-control time showing the switching between modes in Hydra.

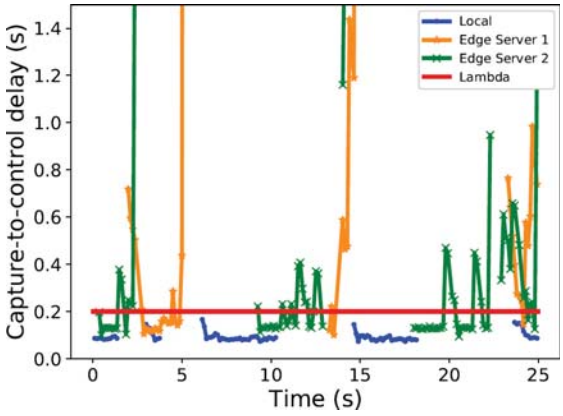

Figure 10: Temporal pattern of the capture-to-control time showing the switching between modes in Hydra.

external data streams. Specifically, we create high-volume traffic with a duty cycle of 20 s – 10 s active followed by 10 s inactive – with an offset of 5 s between the two channels. Thus, within one duty cycle, we have 5 s in which both channels experience congestion, 5 s in which both are congestion-free, and two periods of 5 s in which either one of the channel is congested and the other is congestion-free.

Fig. 10 shows a temporal trace of the capture-to-control time achieved by Hydra under these conditions. The cycles are apparent: the system switches from one edge server to another as the exogenous data streams are activated, and relies on local computing when the channel to both edge servers is congested. We note the erratic behavior of delay due to the interactions between data flows due to channel access and transport layer protocols.

V. Conclusions

This paper presents a testbed platform and a flexible architecture for the distribution of computation tasks within infrastructure assisted UAV systems. The architecture embeds logics to dynamically activate/deactivate pipelines to acquire information on available communication and computation resources and select the most performing options. Our experimental results show that Hydra is capable of counteracting fast variations in the capture-to-control time, and achieve a tunable tradeoff between energy consumption and control performance. Future explorations will consider predictive selection and the integration of macro-scale features in the decision process.

References

[1] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 2009.

[2] Yujin Li and Wenye Wang. Can mobile cloudlets support mobile applications? In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1060–1068. IEEE, 2014.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[4] Xiaowen Cao, Jie Xu, and Rui Zhangt. Mobile edge computing for cellular-connected uav: Computation offloading and trajectory optimization. In *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2018.

[5] Mahadev Satyanarayanan, Grace Lewis, Edwin Morris, Soumya Simanta, Jeff Boleng, and Kiryong Ha. The role of cloudlets in hostile environments. *IEEE Pervasive Computing*, 12(4):40–49, 2013.

[6] Mohamed-Ayoub Messous, Hichem Sedjelmaci, Noureddin Houari, and Sidi-Mohammed Senouci. Computation offloading game for an uav network in mobile edge computing. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[7] Mohamed-Ayoub Messous, Amel Arfaoui, Ahmed Alioua, and Sidi-Mohammed Senouci. A sequential game approach for computation-offloading in an uav network. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.

[8] Davide Callegaro and Marco Levorato. Optimal computation offloading in edge-assisted uav systems. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[9] Seongah Jeong, Osvaldo Simeone, and Joonhyuk Kang. Mobile edge computing via a uav-mounted cloudlet: Optimization of bit allocation and path planning. *IEEE Transactions on Vehicular Technology*, 67(3):2049–2063, 2017.

[10] Fuhui Zhou, Yongpeng Wu, Rose Qingyang Hu, and Yi Qian. Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems. *IEEE Journal on Selected Areas in Communications*, 36(9):1927–1941, 2018.

[11] Nan Cheng, Wenchao Xu, Weisen Shi, Yi Zhou, Ning Lu, Haibo Zhou, and Xuemin Shen. Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities. *IEEE Communications Magazine*, 56(8):26–32, 2018.

[12] Feng Wang, Jie Xu, Xin Wang, and Shuguang Cui. Joint offloading and computing optimization in wireless powered mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, 17(3):1784–1797, 2017.

[13] Mamta Narang, Simon Xiang, William Liu, Jairo Gutierrez, Luca Chiaraviglio, Arjuna Sathiaseelan, and Arvind Merwaday. Uav-assisted edge infrastructure for challenged networks. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 60–65. IEEE, 2017.

[14] Davide Callegaro. Hydra - distributed processing in heterogeneous cloud robotics systems. https://github.com/uci-iasl/HYDRA, 2019.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[17] Tensorflow detection model zoo. https://github.com/tensorflow/models, 2019.