FOCUS: Fog Computing in UAS Software-Defined Mesh Networks

Gokhan Secinti, *Member, IEEE*, Angelo Trotta, *Member, IEEE*, Subhramoy Mohanti, *Student Member, IEEE*, Marco Di Felice, and Kaushik R. Chowdhury, *Senior Member, IEEE*

***Abstract*— Unmanned aerial systems (UASs) allow easy deployment, three-dimensional maneuverability and high reconfigurability, as they sustain communication network in the absence of pre-installed infrastructure. The proposed FOg Computing in UAS Software-defined mesh network (FOCUS) paradigm aims to realize an implementable network design that considers practical issues of aerial connectivity and computation. It allocates UASs to the tasks of data forwarding and in-network fog computing while maximizing number of ground-users in UAS coverage. FOCUS improves efficient utilization of network resources by introducing on-board computation and innovates on top of software-defined networking stack by integrating the capabilities of network and ground controllers to enable simultaneous orchestration of both UASs and communication flows. There are three main contributions of the paper: First, a SDN-based architecture is designed enabling autonomous configuration of computation and communication as well as managing multi-hop aerial links. Second, a global optimization problem to achieve optimal forwarding and computational allocation is formulated using Open Jackson Network model and solved via a heuristic approach with well defined complexity. Third, FOCUS framework is implemented on a small-scale testbed of Intel® Aero UASs performing image analysis with a full software stack. Experiments reveal at least 32% latency improvement in computation service time compared to traditional centralized computation at the end-server or greedy task allocation schemes within the network.**

***Index Terms*— Unmanned aerial vehicles, mobile ad hoc networks, edge computing, software defined networking, heuristic algorithms.**

I. Introduction

UNMANNED aerial system (UAS) applications have grown exponentially over the last five years, presently driving business close to 1 billion USD already within the USA, with an upwards growth targeted to reach 46 billion USD by 2025 [1]. Thus, it is foreseen that they will be one of the key enablers toward smart cities with their applications range from construction to communication and to surveillance. However, most of the existing deployments consider UAS as a mobile wireless sensor, with data processing offloaded to a computational cloud. At the same time, the enhancements in UAS control and communication hardware and mass production at economical price-points are paving the way towards Unmanned Aerial Networks (UANs), composed of swarms of UASs connected to the Internet, capable of limited on-board computation, but also being integrated with the cloud [1].

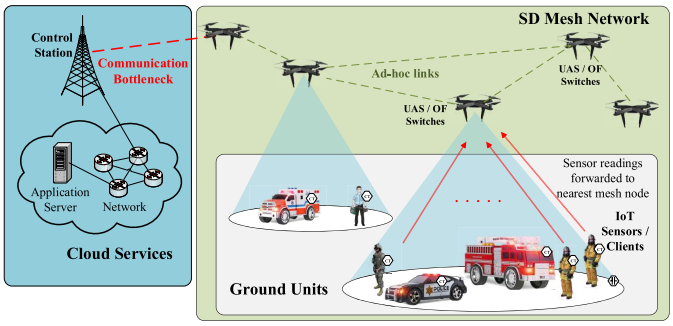

Fig. 1. Network architecture of a UAN connected to computational cloud and ground units.

Different from traditional ad hoc and mobile networks, the design of a UAN poses unique challenges, such as highly dynamic topology, 3D mobility and high energy consumption per unit time [2]. Piloting commercial off the shelf (COTS) UASs requires manual skills, but UAN applications that have rigid objectives and performance constraints amplify the above challenges when they operate in groups. In this paper, we envision the UAN as a mixed sensing, information relaying and computing platform, taking advantage of the entirety of its on-board capabilities. Towards this goal, we wish to adopt the flexibility and structured approach of classical software defined networking (SDN), building on the OpenDayLight [3] architecture that has proven to be successful in the wired networking domain. Thus, each UAS within the larger UAN becomes a network switch that directs data traffic towards the remote cloud as well as towards peer-nodes for in-network processing. To our best knowledge, while many works have pointed towards the trend in COTS UAN to gain increasingly greater computational power and ability to support popular operating systems and processing packages [4], a transformative design that allows the UAN to become a fully capable, aerial SDN has not been implemented in actual systems.

A. Problem and Solution Overview

We consider a scenario where ground sources may generate rich sensing data (e.g. videos, terrain maps, RF spectrum surveys) that needs to be transmitted to the cloud for purposes such as storage, aggregation and analytics as seen in Fig. 1.

Manuscript received January 4, 2019; revised August 29, 2019; accepted November 18, 2019. This work was supported by the Office of Naval Research (ONR) Code 30 Other Transaction Agreement (OTA) #N00014-18-9-0001. The Associate Editor for this article was S. Olariu. *(Corresponding author: Gokhan Secinti.)*

G. Secinti, S. Mohanti, and K. R. Chowdhury are with the Electrical and Computer Engineering Department, Northeastern University, Boston, MA 02115 USA (e-mail: g.secinti@coe.neu.edu; s.mohanti@coe.neu.edu; krc@coe.neu.edu).

A. Trotta and M. Di Felice were with the GENESYS Lab, Northeastern University, Boston, MA 02115 USA. They are now with the Department of Computer Science and Engineering, University of Bologna, 40126 Bologna, Italy (e-mail: angelo.trotta5@unibo.it; marco.difelice3@unibo.it).

Digital Object Identifier 10.1109/TITS.2019.2960305

This generic environment covers several real-world use-cases related to post-disaster recovery, rural broadband access and military operations. In many of these cases, the cloud is not reachable via a direct link, and hence, instead of a single UAS, we envision that a UAN is deployed for data forwarding to the cloud server. We note that several distributed mobility-aware routing and communication-aware mobility control schemes for UASs have been proposed in the literature that address only a subset of factors affecting the UAN deployment [2], [5]. In this paper, we focus on an architecture that leverages SDN, given that it effectively decouples control and user data plane, and has shown great potential in management of wired/wireless networks.

A conceptual view of SDN controllers to compute the UAN topology and routing based on actual link load estimations have been proposed in [6], [7]. In addition, given the highly dynamic network topologies in such networks, there is likelihood of a temporal drift between the actual network state and the virtual state information available at the controller. In such situations, shifting the entire burden of the scenario analysis to the cloud becomes risky; few incorrect or outdated decisions taken by the controller can result in major end-to-end latencies in the mission [8]. Our approach addresses these challenges by considering, in a single theoretical formulation, the optimized assignment of the individual UASs to function as forwarding nodes that relay data towards the sink or to alternate intermediate-UASs that may perform computational tasks closer to the source. This results in significantly lowering the volume of data to be transmitted through the UAN. We look at the problem not only from a theoretical viewpoint but also from that of practical implementation, by (*i*) implementing a robust mesh network formation and routing protocol, (*ii*) defining and architecting the SDN control interfaces, and (*iii*) showing the benefits of intelligently assigning computation tasks to selected UASs in the network.

B. Main Contributions

We make three main contributions in this paper:

- We present the architecture and design of FOCUS, a framework for the deployment of a practical UAN. It autonomously sets up the UAN configuration in terms of network topology (i.e. routing table), and computation (i.e. task allocation over cloud or intra-UAN nodes, also called as *fog* nodes), based on QoS requirements. This is performed via an SDN architecture that handles both computation and communication in multi-hop aerial environments.
- We formulate the problem of joint routing and computation assignment over UAN via an analytical model. Being NP-hard, we decompose it into two sequential tasks, with heuristic solutions for each. We also provide rigorous bounds on the computational complexity.
- We implement the system on a limited testbed of Intel Aero UASs performing data processing of image-data. We also validate FOCUS experimentally and through simulation, starting from the data generation on the ground to the final logging of processed results in a database.

The rest of the paper is structured as follows. In Section II, we review the existing literature on the application of SDN and cloud computing on UAN. Preliminary results supporting the motivation of our work are provided in III. The joint routing-computation problem and algorithms are formally described in IV. FOCUS architecture and implementation details are explained in V. Experimental and simulation results are reported in Section VI. Finally, we conclude in Section VII.

II. Related Work

Recent works have attempted to address UAS requirements of dynamic topology and variable network load [2]. Moving some of these important problems from the physical device plane to the controller of an SDN has gained traction [9], where link status and flight statistics are collected from each UAS and used to compute the routing tables. The core functionality of packet routing is enhanced in [10] by incorporating a centralized energy- and load-aware routing scheme. Reference [6] maps these approaches in context of video-surveillance, while [7] exploits the closely related Network Function Virtualization (NFV) functionalities for telemetry monitoring and anomaly detection. Reference [8] uses SDN controller for motion and location prediction, by utilizing knowledge of the current physical position and trajectory of each UAS to envision how the network may evolve ahead.

Different from these works, FOCUS SDN-enabled architecture jointly performs network routing to both the end cloud as well as in-network computational resources while taking the network topology and the network load requests into consideration. The latter concept, commonly known as *fog computing* is still in a nascent stage. Several papers have investigated the dual problem, i.e., how to offload the computation from the UASs to an edge-server or to a remote server in the cloud [11]. References [12] and [13] describe video-surveillance applications, where data are gathered by UASs, and processed on edge nodes through computational expensive vision algorithms. Instead, in FOCUS, we envision UAS themselves as computationally capable devices. The Intel Aero drones we use have Intel Atom® x7-Z8750 processor, 4 GB LPDDR3-1600 RAM, Intel® Dual Band Wireless-AC 8260, 32 GB eMMC, Altera® Max® 10 FPGA and Ubuntu 16.04 LTS.

Similar to the notion of delegating computational workload to UASs, following studies proposed to utilize vehicles as computational nodes or relays in vehicular ad-hoc networks to improve overall performance of cloud systems [14]–[16]. Reference [14] proposed semi-markov decision process based resource allocation scheme for vehicular clouds, where vehicles in the network increases cloud resource pool via sharing their own computational resources. In [15], the authors developed content dissemination framework by integrating edge computing with vehicular networks where vehicles act as relays to deliver contents efficiently. In addition, an approximation scheme for job completion time in vehicular cloud is proposed in [16]. However, these studies neither considered UAS-specific challenges such as limited on-board resources, 3D mobility and unreliable ad-hoc links while devising their frameworks, nor provided any real-world implementation.

We describe next the works that come closest towards joint consideration of UAS capabilities, including computation.

TABLE I
COMPARISON WITH THE LITERATURE

	Study	Network Aspect	Computational Aspect	Target Domain	Test-bed Imp.
UAS app. with on-board processing	Motlagh et al. [12]	Direct link to LTE infrastructure for communication	Crowd survailence with on-board processing with a single drone	IoT Platform	✓
	Wang et al. [13]	Adaptive relaying mechanism with full-duplex radios on UAS	Real-time video streaming via UAS to a ground server without pre-processing on-board	Multi-UAV network	✓
Vehicular Cloud	Meneguette et al. [14]	Poisson arrival model for vehicles entering to a cloud without stressing network or comms. challenges	Formulating semi-markov decision process to allocate resources based-on availability	Vehicular Networks	✗
	Hui et al. [15]	Probabilistic V2I and V2V link model with two-layer relay selection scheme	Content dissemination framework among edge computation nodes to increase availability	Content Delivery Networks / Vehicular Networks	✗
	Florin et al. [16]	**N/A** Network challenges are not addressed	Approximation of job completion times on distributed systems	Distributed Systems / Vehicular Networks	✗
UAS in Edge	Jeong et al. [17]	Single UAV	Moving cloudlet with the goal of energy optimization	Cellular networks where mobile users demand computation	✗
	Narang et al. [18]	Single or multiple UAVs without interconnection among each other	**N/A** Comms. challenges are addressed	Challenged Networks (CN)	✗
	Kattepur et al [19]	Single hop links to robots/drones	Timing-estimation based deployment of OpenCV applications	Multi-robot environment	✓
	Messous et al [20]	Challenges of aerial networks are not addressed	Game-theory based offloading scheme	Multi-UAV system	✗
FOCUS		Joint formulation and optimization of network and computation resources with two-layer Open Jackson Networks.		Multi-UAV system / Aerial Mesh Networks	✓

Reference [19] estimates the execution time of different tasks when executed locally on ground robot units and remotely in the fog/cloud servers, and defines offloading strategies to optimize the service time. Reference [20] approaches the same problem through game theory. General architectures employing UASs as fog nodes are proposed in [18] and [17]. In the former case, the UASs serve as mobile base stations providing connectivity to the ground units. In the second case, the UASs offer computation offloading opportunities to mobile ground units. However, the goal of this work is on enhancing the uplink/downlink communications and the path planning of the UASs. Table I showcases the differences of FOCUS with the existing studies in the literature where these studies are grouped under three sub-categories based on their scope and are investigated by four different aspects such as network, computation, target domain and test-bed implementation. Moreover, the work that we present in this paper does not explore the limited energy problem of the UASs that is indeed quite important when analysing aerial vehicles. However, several work can be found in literature that proposed solutions for this problem [21], [22] that can be used to obtain continuous operability. FOCUS pushes the envelope further by a joint analytical and systems approach, while opening up for the community the software tools and code to build and deploy aerial SDNs.

III. PRELIMINARY STUDY ON COMPUTATION ON UASS

We conduct two different sets of experiments motivating the use of a UAN as an aerial computing platform: the first one stressing the communication/computing tradeoff on a simple linear topology, the second one showcasing the impact of processing on the UAS battery/flight time. To better understand the advantages of data processing on fog nodes within an UAN as opposed to centralized cloud, we set up a mesh UAN in a linear topology consisting of 3 Intel Aero Ready-To-Fly UASs, as seen in Fig. 2a. Two ground laptops are placed at the two ends of the mesh, one acting as a ground control station (GCS) and application server, and the other as the source. The GCS containing 16GB DDR3 RAM, 7th generation Intel processor and NVIDIA® GTX 1060 graphic card doubles as the centralized cloud entity along with being the controller. The source generates network data as UDP packets and static images for processing. The target location for the processing may vary over time via *computational requests*. Since, we use same workload generation and image processing algorithms during evaluation, further details are explained in Section VI.

Consider four scenarios where: computational requests are handled by UAS in inter-mediate hops, i.e., UAS1, UAS2, UAS3 and when the images reach the GCS/application server. The network load is gradually increased from 0 to 1Mbps and finally to the upper limit of 2Mbps. Under non-loaded network conditions, understandably, it is beneficial to execute the data processing at the remote GCS, as seen in Fig. 2c. However, when the network load begins to increase, the amount of time spent on forwarding information in the network indicates the benefit of processing the image on a fog entity, i.e., the UAS as opposed to the GCS, despite having weaker computational power. As seen for the cases of 1 and 2 Mbps network traffic, processing images on the UAS improves the response time.

As algorithms may vary in processing requirements (for e.g., genetic algorithms and particle swarm optimization run multiple iterations), we define an iteration variable that increases the effort involved in completing the image processing task.

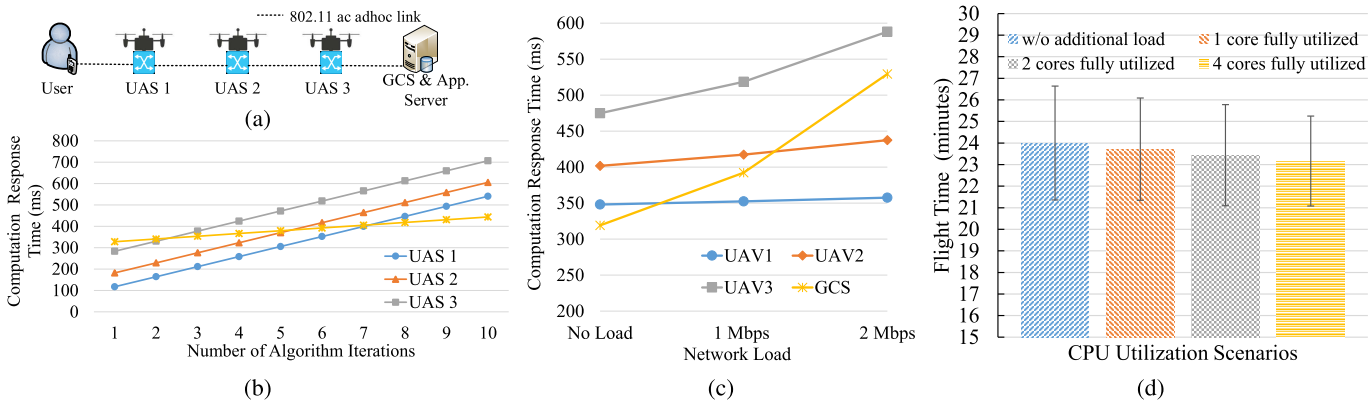

Fig. 2. (a) Linear UAN topology; computation response time w.r.t. (b) algorithm iteration number, (c) network load and; (d) UAS flight time w.r.t. processing complexity.

Each iteration re-analyzes the image, with the goal being to identify the break-even points for switching the processing from fog nodes to the GCS. As seen in Fig. 2b for a constant network load of 1Mbps, algorithms with fewer iterations to completion are more suited for the fog nodes, though every additional forwarding hop impacts the difference from the ground server to a greater extent. These studies indicate practical *handover* points that we use in our optimization approach.

Computation and data relaying both incur energy costs. To study their inter-dependence, we flew a UAS equipped with the standard 4S, 4500mAh battery, under different CPU utilization scenarios, as shown in Fig. 2d, recording the flight time for each setting. These scenarios are defined by the number of CPU cores fully utilized during the entire flight starting with 0 and continue with 1, 2 and 4 cores. During the flights, we use a simple workload generator [23] to fully utilize individual CPU cores on UASs. Increasing CPU utilization drains negligible battery power when compared to the power drawn by the UAS propeller motors. This further motivates us to fully leverage fog nodes for processing within the network. Fig. 2d shows high variations in the upper and lower bounds in the flight time due to uncontrollable flight dynamics (altitude deviation, pitch, yaw, roll) caused by wind, which again impacts the UAS battery consumption rate more than internal data processing. We run the experiment ten times for each scenario with a fully charged battery, and record the flight time until its remaining battery capacity hits a critical level (which is 15%).

IV. Problem Formulation and Optimization Framework

In the following, we formulate the problem of joint assignment of network flows and computational tasks by modeling UAN as a network of queues. Then we describe two-phase solution to this problem, where network flow and computational flow optimizations are solved consecutively. Table II defines the variables and symbols used in the process.

A. System Model

We assume a scenario with N flying UASs, M ground units and one ground control server (*GCS*). The ground units produce two classes of network traffic: (*i*) generic network traffic (denoted by the n apex), i.e. sensor data that must be delivered to the GCS, and (*ii*) computational traffic (denoted by the c apex), i.e. data that must be processed on the cloud or on fog nodes. We introduce the following variables:

- $U = \{u_1, u_2, \ldots, u_N\}$ is the set of available UASs;
- $G = \{g_1, g_2, \ldots, g_M\}$ is the set of the ground units;
- $\Upsilon^{(c)} = \{\mu_1^{(c)}, \mu_2^{(c)}, \ldots, \mu_N^{(c)}, \mu_{N+1}^{(c)}\}$ is the computational capacity (in Kbps) for each UAS $u_i \in U$ and for the *GCS* (defined by $\mu_{N+1}^{(c)}$);
- $\Gamma = \gamma_1, \gamma_2, \ldots, \gamma_M$ is the set of bandwidth requests (in Kbps) for each ground unit $g_j \in G$, where $\gamma_i = \gamma_i^{(n)} + \gamma_i^{(c)}$, i.e. it includes both generic and computational data;
- $\mathbf{E}_{N\times(N+1)}$ is UAN adjacency matrix, where $e_{i,j} \rightarrow \{0, 1\}$ indicates whether there is an active link between the UASs $u_i, u_j \in U$, where also index $N+1$ represents the GCS;
- $\mathbf{A}_{N\times M}$ is Air-to-Ground (A2G) connection matrix, where $a_{i,k} \rightarrow \{0, 1\}$ indicates whether there is an active connection between UAS $u_i \in U$ and ground unit $g_k \in G$.

TABLE II
List of Important Notations

Symbol	Description
N	Number of UASs.
M	Number of ground units.
$\mathbf{E}$	Adjacency matrix for UAS network.
$\mathbf{A}$	Air-to-Ground connection matrix.
$\mu_i^{(t)}$	Service capacity for the traffic type t[1]at UASi.
$\gamma_i^{(t)}$	Bandwidth request of ground unit i belonging the traffic type t.
$\Omega_i^{(t)}$	Sum of arrival rates from ground units assigned to UASi.
$\lambda_i^{(t)}$	Arrival rate for the traffic type t at UASi.
$\mathcal{L}_i^{(t)}$	Expected number of packets in the queue at UASi.
$\mathcal{W}_i^{(t)}$	Average waiting time of type t packet in the queue at UASi.
$\gamma_j^{(t)}$	Bandwidth request of ground unit j.
$\mathbf{R}^{(t)}$	Routing matrix.
$r_{ij}^{(t)}$	Element of $\mathbf{R}^{(t)}$ defining the ratio of the packets routed from UASi to UASj.
$\mathbf{F}, \mathbf{C}$	Auxiliary matrices used in the heuristics to hold pointers to father ($\mathbf{F}$) and child ($\mathbf{C}$) nodes on network flows.
$\mathcal{B}^R, \mathcal{B}^P$	Bipartite graphs holding computational requests (R) and available slots (P)

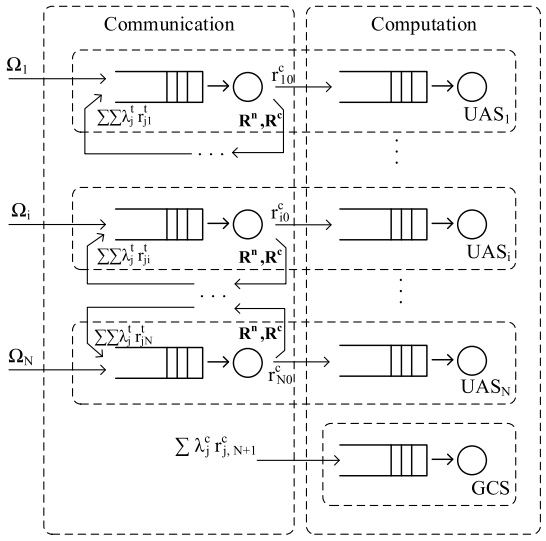

Fig. 3. Two-layer Open Jackson Network.

$\mathbf{E}_{N\times(N+1)}$ and $\mathbf{A}_{N\times M}$ matrices are assumed to be pre-computed based on the position of the ground units. The UAN is modeled as an *Open Jackson Network* [24], and each UAS node is represented with two consecutive M/M/1 queues as shown in Fig. 3. On the left, we depict the network queues used for both traffic types (n and c). On the right, we depict the process queue corresponding to the computational traffic (c), for both the UASs and the GCS. The output of the queues are determined by the routing policies. These latter are formally modeled via the matrices $\mathbf{R}^{(n)}_{N\times(N+2)}$ and $\mathbf{R}^{(c)}_{N\times(N+2)}$, respectively for the n, c traffic classes; $r^{(t)}_{i,j}$ indicates the ratio of traffic of u_i routed to u_j for traffic type $t \in \{n, c\}$. In addition, $r^{(n)}_{i,0}$ and $r^{(c)}_{i,0}$ denotes ratio of the packets left the network and, ratio of packets processed locally at u_i. Similarly, let $r^{(c)}_{i,(N+1)}$ be the ratio of the computational packets that u_i forwards to GCS, to be processed on the cloud. The overall packet arrival rate (λ_i) to u_i is defined as $\lambda_i = \lambda^{(n)}_i + \lambda^{(c)}_i$. Assuming that, the ground units generate requests with the average $1/\gamma$ time difference, we use Markovian queues to obtain closed-form equations for the upper-bounds of the response times in the model.

For each traffic type ($\forall t \in \{c, n\}$), the $\lambda^{(t)}_i$ term can be modeled as $\lambda^{(t)}_i = \Omega^{(t)}_i + \sum_{j=1}^{N} r^{(t)}_{j,i} \cdot \lambda^{(t)}_j$, where $\Omega^{(t)}_i = \sum_{k=1}^{M} a_{i,k} \cdot \gamma^{(t)}_k$ and the variable $\gamma^{(t)}_k$ is the arrival rate of the c and n traffic from the k^{th} ground unit and $\Omega^{(t)}_i$ is the sum of the arrival rates over all ground units assigned to UAS i.

By Little's Law, the average response time for packets belonging to traffic type c ($\mathcal{W}^{(c)}$) is defined as follows:

$$\mathcal{W}^{(c)} = \frac{\sum_{i=1}^{N} \mathcal{L}^{(n)}_i + \sum_{i=1}^{N+1} \mathcal{L}^{(c)}_i}{\sum_{k=1}^{M} \gamma^{(c)}_k} \tag{1}$$

where $\mathcal{L}^{(n)}_i$ and $\mathcal{L}^{(c)}_i$ indicate the expected number of computational packets in the network queue and in the computational queue of UAS i respectively. Let $\Upsilon^{(n)} = \{\mu^{(n)}_1, \mu^{(n)}_2, \ldots, \mu^{(n)}_N\}$ be the average service rate for each UAS $u_i \in U$, with $\mu^{(n)}_i$ expressed again in Kb/s. Hence, the average number of computational (c-type) packets at the first queue is $\mathcal{L}^{(n)}_i = \lambda^{(c)}_i / \left(\mu^{(n)}_i - \lambda_i\right)$. On the other hand, the number of the packets in the second queue is defined as follows $\mathcal{L}^{(c)}_i = \lambda^{(c)}_i r^{(c)}_{i,0} / \left(\mu^{(c)}_i - \lambda^{(c)}_i r^{(c)}_{i,0}\right)$. We can hence rewrite $\mathcal{W}^{(c)}$ as follows:

$$\mathcal{W}^{(c)} = \frac{\sum_{i=1}^{N} \frac{\lambda^{(c)}_i}{\mu^{(n)}_i - \lambda_i} + \sum_{i=1}^{N+1} \frac{\lambda^{(c)}_i \cdot r^{(c)}_{i,0}}{\mu^{(c)}_i - \lambda^{(c)}_i \cdot r^{(c)}_{i,0}}}{\sum_{k=1}^{M} \gamma^{(c)}_k} \tag{2}$$

Similarly, we compute the average response time for the generic network traffic as follows:

$$\mathcal{W}^{(n)} = \frac{\sum_{i=1}^{N} \frac{\lambda^{(n)}_i}{\mu^{(n)}_i - \lambda_i}}{\sum_{k=1}^{M} \gamma^{(n)}_k} \tag{3}$$

The proposed system assumes to have the knowledge of the all data load requests Γ and the network links quality $\Upsilon^{(n)}$. Consequently, also the $\mathbf{A}_{N\times M}$ and the $\mathbf{E}_{N\times(N+1)}$ matrices are assumed to be known. These indexes represent the ground user bandwidth requests and the network links quality, respectively. These two indexes can be known in advances for static scenarios and with full knowledge, but this case is very rare. On the other side, for evolving and/or unknown scenarios, these indexes can be dynamically estimated by the system using link quality estimation within the SDN networks [25] in order to estimate $\Upsilon^{(n)}$ and $\mathbf{E}_{N\times(N+1)}$ and continuous monitoring of the ground user requests to estimate Γ and $\mathbf{E}_{N\times(N+1)}$. The effects of the convergence of the index estimations and their change over time will be analysed in future works.

B. Problem Formulation

Based on this system model, we formally define the joint routing and computation assignment (RC) problem as follows:

$$\min_{\mathbf{R}^{(t)}} \quad \mathcal{W}^{(c)} \tag{4}$$

$$\text{subject to:} \quad \sum_{u_i \in U} a_{i,j} = 1 \quad \forall g_j \in G \tag{5}$$

$$dim\ kernel(Laplacian(\mathbf{E})) = 1 \tag{6}$$

$$\lambda_i < \mu^{(n)}_i \quad \forall u_i \in U \tag{7}$$

$$\lambda^{(c)}_i \cdot r^{(c)}_{i,0} < \mu^{(c)}_i \quad \forall u_i \in U \tag{8}$$

$$\sum_{j=0}^{N+1} r^{(t)}_{i,j} = 1 \quad \forall u_i \in U,\ t \in \{n, c\} \tag{9}$$

$$r^{(t)}_{i,0} + \sum_{j=1}^{N+1} (r^{(t)}_{i,j} \cdot e_{i,j}) = 1 \quad \forall u_i \in U, t \in \{n, c\} \tag{10}$$

$$\mathcal{W}^{(n)} \leq \rho^{(n)} \tag{11}$$

$$r^{(t)}_{i,j}, \lambda^{(t)}_i, \mu^{(t)}_i, \gamma^{(t)}_i \geq 0 \quad \begin{matrix} \forall u_i \in U, \quad t \in \{n, c\}, \\ j \in [0, \mathrm{N}+1] \end{matrix} \tag{12}$$

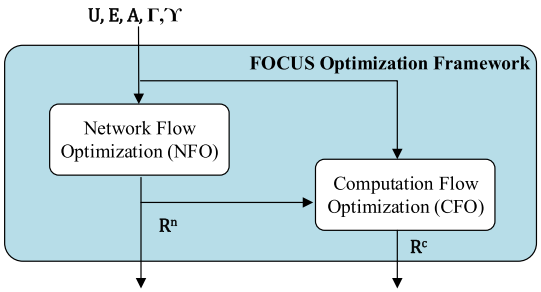

Fig. 4. FOCUS optimization framework.

The goal of the optimization problem, defined by (4), is to minimize the average response time of computational traffic, by determining the optimal routing matrices $\mathbf{R}^{(t)}$, meeting the following constraints: (5) states that each ground unit must be connected to a single UAS; (6) ensures the aerial mesh is connected by analysing dimension of the kernel of the adjacency matrix's *Laplacian*; (7) and (8) guarantee the validity of Open Jackson Network model; (9) and (10) state that all the packets leaving a network queue will flow to another network queue or to a computation queue, but only using active links; finally, (11) ensures the service time for type n traffic does not exceed a user-defined threshold $\rho^{(n)}$.

RC multi-objective optimization problem is NP-hard since it is a generalization of the well-known splittable flow problem [26]. Therefore, we divide the original problem into two phases: first, we compute the routes for traffic type n toward the GCS. Then, based on such allocation, and the estimated network traffic congestion, we compute the routes for the computational data flows (type c). Fig. 4 shows the modules implementing each phase. The first module, namely *Network Flow Optimization (NFO)*, generates the entries of the $\mathbf{R}^{(n)}$ matrix as output. These values are taken as input by *Computation Flow Optimization (CFO)* module producing as output the final $\mathbf{R}^{(c)}$ matrix.

C. Network Flow Optimization (NFO)

NFO algorithm determines the proper values of $r_{i,j}^{(n)}$, with $i, j \leq N$, i.e. the routing entries for the class n traffic directed to the GCS. Given the complexity of the original problem, we relax the constraint of (11), i.e. we determine the routing entries minimizing the average response time $\mathcal{W}^{(n)}$. Formally:

Definition 1: Given the set of available UASs (U), the connection matrix ($\mathbf{E}_{N\times N}$), the air-to-ground active links matrix ($\mathbf{A}_{N\times M}$) and the network requests set (Γ), the goal is to compute the network routing matrix $\mathbf{R}^{(n)}_{N\times(N+2)}$ such that the average response time for class n traffic $\mathcal{W}^{(n)}$ is minimized.

We address the problem by using a modified version of the Dijkstra algorithm for the Shortest Path Problem (SPP) over acyclic weighted graphs. To this purpose, we define the cost function for node u_i ($\xi^{(n)}(u_i)$) as a proxy of the average delay toward the GCS, computed as follows:

$$\xi^{(n)}(u_i) = \frac{1}{\mu_i^{(n)} - \lambda_i^{(n)}} + \sum_{1\leq j\leq(N+2)} f_{i,j} \cdot r_{i,j}^{(n)} \qquad (13)$$

Here, the first fraction represents the average queuing delay at node u_i, while the second one is the average delay of the path toward the GCS. We recall that the Dijkstra algorithm computes the shortest path between a source (or multiple sources) to a single destination. Unfortunately, the original algorithm does not fit our problem because: (*i*) the weight of the nodes/arcs is not static, since it may change over time as a specific node is included in the routing path of a flow toward the GCS; (*ii*) each node can have multiple paths towards the GCS, and exploit all of them concurrently. The proposed solution is described by Alg.1. As in Dijkstra's algorithm, we start building the shortest paths from the destination (the GCS in our case). In addition, we keep two auxiliary matrices as:

1) **F** matrix keeps track of the network flows costs. It includes the forward pointers from one node to its fathers, i.e. $f_{i,j} \in \mathbf{F}$ is greater then zero if u_j is a father for u_i towards the GCS and defines the path cost from u_j;
2) **C** matrix keeps track of the incoming network flows. It includes the reverse pointers from one node to its subtree, i.e. $c_{i,j} = 1$ if node u_j is a child of node u_i, 0 otherwise.

Algorithm 1 NFO algorithm

NFO $(U, \mathbf{E}_{N\times N}, \mathbf{A}_{N\times M}, \Gamma, \Upsilon)$
init $\mathbf{F}_{N+1\times N+2}$ with $f_{i,j} = -1$ $\{f_{i,j} \geq 0$ is the cost$\}$ of j being father of i
init $\mathbf{C}_{N+1\times N+2}$ with $c_{i,j} = 0$ $\{c_{i,j} = 1$ if j is a$\}$ child for i
init $v_i = false$, $\lambda_i^{(n)} = \Omega_i^{(n)}$, $r_{i,j}^{(n)} = 0$, $1 \leq i, j \leq N$
set $f_{N+1,N+2} = 0$ $\{N+1$ is the GCS and $N+2$ is a dummy node$\}$
while $\exists u_k$ s.t. $(v_k = false) \wedge (\exists j$ s.t. $f_{k,j} \geq 0) \wedge (\arg\min_k \xi^{(n)}(u_k))$ **do**
  $v_k = true$
  call *updateLambdas*(u_{N+1})
  call *updateFathersCost*$(N+2, u_{N+1}, 0)$
  call *updateChildren*(u_k)
return $R^{(n)}$
function updateLambdas (u_k)
$\lambda_k \leftarrow \Omega_k$
for all u_i s.t. $c_{k,i} = 1$ **do**
  if $v_i = true$ **then**
    $\lambda_k \leftarrow \lambda_k + (r_{i,k}^{(n)} \cdot$ *updateLambdas*$(u_i))$
return λ_k
function updateFathersCost $(j, u_k, fullcost)$
$f_{k,j} \leftarrow fullcost$
for all u_i s.t. $c_{k,i} = 1$ **do**
  updateFathersCost$(k, u_i, \xi^{(n)}(u_k))$
function updateChildren (u_k)
for all u_i such that $e_{k,i} = 1$ with $e_{k,i} \in \mathbf{E}$ **do**
  if $v_i = false$ **then**
    $c_{k,i} = 1$
for all u_i s.t. $c_{k,i} = 1$ **do**
  $f_{k,i} \leftarrow \xi^{(n)}(u_k)$
  $r_{i,j}^{(n)} \leftarrow \dfrac{\prod_{(m:f_{i,m}\geq 0, m\neq j)} \xi^{(n)}(u_m)}{\sum_{(m:f_{i,m}\geq 0)} \prod_{(w:f_{i,w}\geq 0, w\neq m)} \xi^{(n)}(u_w)}$, $\forall j$: $f_{i,j} \geq 0$

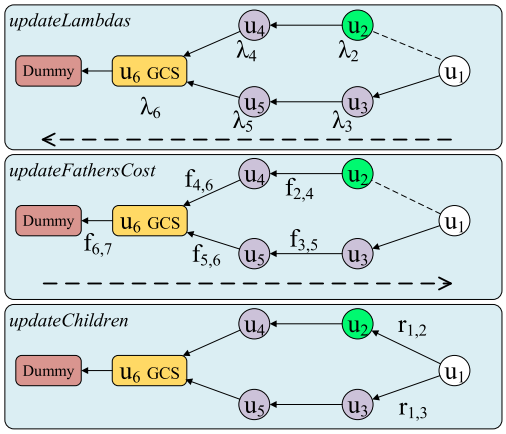

Fig. 5. A schematic view for a single step execution of Alg. 1 (lines 8 - 10). Here $N = 5$ and active node in the step is u_2.

These two matrices are initialized at the beginning of Alg. 1: line 3 for the **C** matrix and line 2 for the **F** matrix; here, as starting point we defined a father connection from the GCS to a dummy node (having index $N+2$) whose cost is set to zero (line 5). At each iteration, the algorithm performs a greedy selection over the nodes, by adding the one having the minimum cost towards the GCS to the solution set (line 6). Then, it updates the $\lambda_i^{(n)}$ and the matrices **F** (lines 8 and 9) and **C** (lines 26-28). Finally, the algorithm updates $\mathbf{R}^{(n)}$ by balancing the outgoing traffic towards all the paths to minimize the total cost (line 28) equalizing $(r_{i,j}^{(n)} \cdot \xi^{(n)}(u_j))$ values, $\forall j$: $f_{i,j} \geq 0$. For space reason, here we omit the check if $r_{i,j}^{(n)} > 1$ in cases, which cost cannot be balanced over all the fathers.

In Fig. 5, we depicted one step execution of Alg. 1. We visualize the three main functions `updateLambdas`, `updateFathersCost` and `updateChildren` in a case where the active node is u_2 and u_1 is not yet visited. During `updateLambdas` method, the flows goes from the leaves to the root (GCS) and all the λ_i are updated accordingly with the tree connections. Then, `updateFathersCost` function updates the links' cost of each connection starting from *Dummy-GCS* connection (having $f_{N+1,N+2} = 0$) and going down towards the leaves. Finally, the `updateChildren` function updates the routing values $r^{(n)}$ for all the children belonging to node u_2. Since each node can have multiple paths towards the destination, the output of the proposed algorithm is a destination oriented directed acyclic graph rooted at the GCS, formally represented by $\mathbf{R}^{(n)}$ matrix.

D. Computation Flow Optimization (CFO)

CFO algorithm allocates tasks to computational resources, represented by the cloud or by UAS fog nodes. Based on the model in Section IV-A, this translates into determining the destination and path for class c packets. We model the problem as a weighted bipartite graph where (*i*) $\mathcal{B}^R = \{b_1^R, b_2^R, \dots\}$ is the set of computational requests (per unit of time) to be executed, with $|\mathcal{B}^R| = \sum_{\gamma_i \in \Gamma} \gamma_i^{(c)}$, and (*ii*) $\mathcal{B}^P = \{b_1^P, b_2^P, \dots\}$ is the set of computational slots (again, per unit of time) available on the fog either on the cloud ($|\mathcal{B}^P| = \sum_{\mu_i^{(c)} \in \Upsilon^{(c)}} \mu_i^{(c)}$). Let $\zeta : \mathcal{B}^R \times \mathcal{B}^P \rightarrow \mathbb{R}$ be the weight function, representing the benefit of assigning a request in $\mathcal{B}^R$ to a computational slot in $\mathcal{B}^P$. We consider an asymmetric assignment problem where the computational resources are strictly greater than the requests (satisfying the requirement of Equation 8), i.e. $|\mathcal{B}^R| < |\mathcal{B}^P|$. Let $req(b_k^R) = u_i$ be the mapping function that returns the UAS generating the request b_k^R; similarly, let $pow(b_l^P) = u_j$ be the function which returns to UAS providing computational slot b_l^P. The aim of the assignment problem is to determine an optimal assignment set $S = \{\left\langle b_i^R, b_j^P \right\rangle : b_i^R \in \mathcal{B}^R, b_j^P \in \mathcal{B}^P\}$, such that the total benefit $\sum_{\left\langle b_i^R, b_j^P \right\rangle \in S} \zeta(b_i^R, b_j^P)$ is maximized, clearly subject to the constraints that each request must be assigned to a single slot and each slot can host at most one computational request.

Algorithm 2 CFO Algorithm

1: **CFO** U, p, $\mathbf{E}_{N\times N}$, $\mathbf{A}_{N\times M}$, Γ, $\Upsilon^{(n)}$, $\Upsilon^{(c)}$, $\mathcal{B}^R$, $\mathcal{B}^P$
2: **for all** $u_i \in U$ **do**
3: **if** $\Omega_i^{(n)} > 0$ **then**
4: **calculate** Dijkstra(u_i, GCS)
5: **update** $r_{j,k}^{(n)}$, $\forall u_k \in U$ and $\forall u_j$ in the calculated path
6: **update** $\lambda_j^{(n)}$, $\forall u_j$ in the calculated path
7: $PL(b_i^R) \leftarrow \{\}$, $\forall b_i^R \in \mathcal{B}^R$
8: **while** S doesn't contains all the assignment for $\forall b_i^R \in \mathcal{B}^R$ **do**
9: **calculate** Dijkstra(u_i, u_j), $\forall u_i, u_j \in U$
10: **update** $\zeta(b_i^R, b_j^P)$, $\forall b_i^R \in \mathcal{B}^R, b_j^P \in \mathcal{B}^P$ based on Dijkstra($req(b_i^R)$, $pow(b_j^P)$)
11: **execute** one forward/reverse step of the auction algorithm
12: **if** $\left\langle b_i^R, b_j^P \right\rangle$ is a new assignment **then**
13: $PL(b_i^R) \leftarrow$ Dijkstra($req(b_k^R)$, $pow(b_l^P)$)
14: **else if** $\left\langle b_i^R, b_j^P \right\rangle$ is removed as an assignment **then**
15: $PL(b_i^R) \leftarrow \{\}$
16: **update** all $\lambda_i^{(c)}$ and $r_{i,j}^{(c)}$ based on the path lists $PL(b_i^R)$, $\forall b_i^R \in \mathcal{B}^R$

The proposed solution (given in Alg. 2) enhances the basic forward/reverse auction scheme described in [27] for the case of weights dynamically changing over time. Indeed, each assignment causes the modification of ζ function due to the alteration of $\lambda_i^{(c)}$ values for UASs residing on the chosen path (line 10). Hence, the algorithm implements a sequence of forward/reverse iterations, where at each iteration a request-slot assignment can be added/removed from the final result. Let $PL(b_i^R)$ denote the list (line 7, 13, 15) containing the calculated path for each b_i^R, the benefit function $\zeta(b_i^R, b_j^P)$ of assigning request b_i^R to slot b_j^P is modeled as follows:

$$\zeta(b_i^R, b_j^P) = \frac{1}{1 + cost(b_i^R, b_j^P)} \tag{14}$$

where $cost(b_i^R, b_j^P)$ is a proxy for the delay of traffic class c from UAS $req(b_i^R)$ to $pow(b_j^P)$, and can be calculated by considering the Dijkstra($req(b_i^R)$, $pow(b_j^P)$) shortest path having the edge $(u_i \rightarrow u_j)$ weight defined as: $\frac{1}{\mu_i^{(n)} - \lambda_i}$. More precisely, let $PL(b_i^R) = \{req(b_i^R), \ldots, pow(b_j^P)\}$ be the path used to reach $pow(b_j^P)$, then:

$$cost(b_i^R, b_j^P) = \left(\sum_{u_k \in PL(b_i^R)} \frac{1}{\mu_k^{(n)} - \lambda_k} \right) + \frac{1}{\mu_{pow(b_j^P)}^{(c)} - \left(\lambda_{pow(b_j^P)}^{(c)} \cdot r_{pow(b_j^P),0}^{(c)} \right)} \quad (15)$$

E. Computational Complexity

We now analyze Computational Complexity (CC) of FOCUS where *NFO* algorithm is followed by *CFO* algorithm.

NFO algorithm is based on the Dijkstra algorithm whose complexity is $O(N^2)$ in its basic form. We see this complexity in the main *while* loop in line 6 of Algorithm 1 where the loop is executed N times and the *argmin* operator is $O(N)$. Inside the loop, the computation is dominated by the functions *updateLambdas* and *updateFathersCost* that visit the whole graph to update the lambdas and cost variables. In conclusion, the *CC* of the *NFO* algorithm is $O(N^3)$.

CFO algorithm copes with the asymmetric bipartite graph problem between two asymmetric sets: the computational requests set having cardinality $|\mathcal{B}^R|$ and the computational slots set having cardinality $|\mathcal{B}^P|$. The auction algorithm solves a generic asymmetric bipartite graph problem in $O(|\mathcal{B}^R||\mathcal{B}^P| \cdot log(n))$, where n is a parametric value [27]. However, in our implementation, we add an extra execution time for updating the cost matrix defined by the function $\zeta(b_i^R, b_j^P)$ (lines 4 and 5 in Alg. 2). Dijkstra's algorithm has a complexity of $O(N^2)$ and the matrix update has complexity $O(|\mathcal{B}^R||\mathcal{B}^P|)$. This brings the total *CC* to $O(|\mathcal{B}^R||\mathcal{B}^P| \cdot log(n) \cdot (|\mathcal{B}^R||\mathcal{B}^P| + N^2))$.

V. Focus Systems-Level Implementation

One of the main contributions within FOCUS is the development of the middleware platform transforming the classical UAN into a joint sensing, forwarding and fog computing architecture. This middleware interacts with existing software blocks, such as those related to coordinating with the ground controller and SDN controller simultaneously. It receives both network and UAS information, and implements the necessary control directives originated from heuristic algorithms that centrally solve RC problem. As shown in Fig. 6, FOCUS is built on top of the OpenDayLight (ODL) SDN controller and DronecodeSDK [28]. The former orchestrates flows in the UAN being controlled by a REST application programming interface (API) and the latter aggregates location information of the UASs and makes this information available to the controller via the telemetry adapter. Through these tightly coupled APIs, the network information required by the FOCUS is aggregated and forwarded to the sub-modules (NFO, CFO).

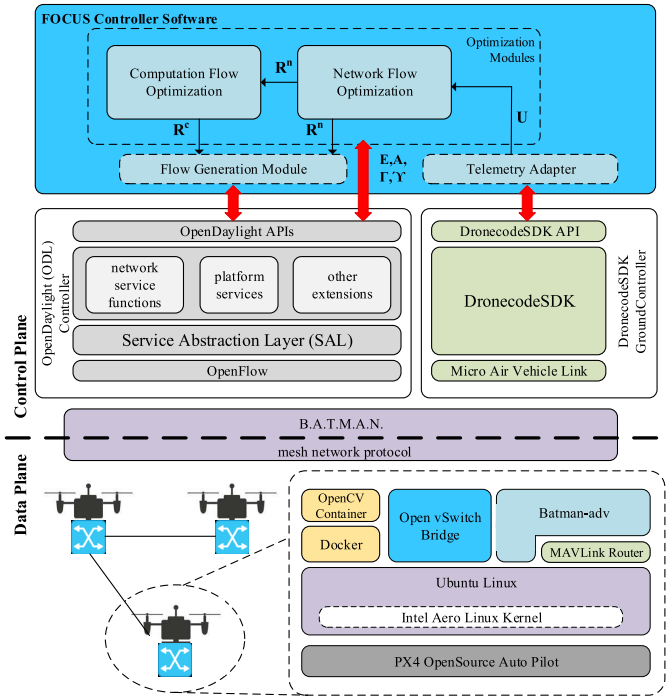

Fig. 6. FOCUS Software Architecture.

These modules, residing in the offsite controller, in turn calculate the optimal allocation of network and computational flows in the UAN and define routing matrices ($\mathbf{R}^{(n)}$ and $\mathbf{R}^{(c)}$). They then initiate control feedback flows via HTTP requests through REST API back to the UAN.

In addition to the control plane design, we also utilize Docker [29] and OpenVswitch [30] (OVS) on UASs at the data plane. Docker hosts a container with OpenCV library to run image processing as computational load (this can be swapped for other applications), while OVS connects to ODL as a traditional Openflow switch. SDNs are classically installed on reliable (often wired) network connections where the control/data planes are not easily impaired. To bring more robustness to the UAN, we utilize a distributed 2nd-layer routing, called 'Better Approach To Mobile Ad-hoc Networking' (BATMAN) [31]. It allows control directives and data to flow over through multiple different pathways to target UAS, even when direct link to the controller is impaired, albeit with an increased latency. Furthermore, it statistically determines the wireless link quality among the nodes and generates numerical values, which are aggregated at the software controller to estimate the throughput capacity on the links. This information is forwarded to FOCUS Optimization framework as seen in Fig. 6, for to be used in constructing of queue-based network model and in solving the optimization problem.

VI. Performance Evaluation

In this section, we validate the performance of FOCUS in terms of overall network traffic and computational response time, in two separate approaches. We conduct our experiments on a small scale UAN testbed, which consists of 4 UASs.

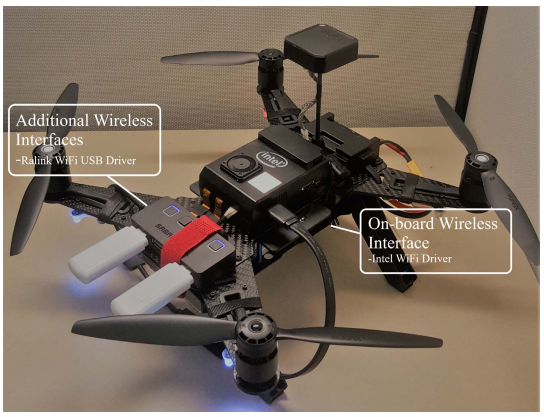

Fig. 7. Intel Aero Ready-to-Fly drone as an aerial OF switch.

The key insights from these experiments then are extended with large-scale simulations consisting of 40 UASs written in C.

A. Results on Small-Scale Testbed Implementation

We use two laptops as ground units and a high performance server as the control station, on where SDN controller (ODL), docker image with OpenCV libraries and dronecode flight controller run. 4 Intel Aero UASs create a mesh network with 4 hops between the ground units and the server, as shown in Fig. 8a. The general hardware specifications are similar to Section III. During the experiment, UASs are positioned in hovering motion at 3-meter above the ground separated by 10-meter distances from each other and from the ground entities in the outdoor drone testing facility at Northeastern University. In our testbed, each UAS is equipped with three wireless interface cards (two RALINK WiFi dongles and one on-board Intel WiFi interface) as seen in Fig. 7, where each interface uses a non overlapping WiFi channel for different tasks. One of them is dedicated to the BATMAN protocol, another is used to create link between UASs and the last one is utilized for the link between ground units and UAS, on channels 1, 6, and 11, respectively. The UASs are positioned in such a way that Ground Unit 1 can only connect with UAS 1 and Ground Unit 2 can connect with UAS 2, creating a separate data path for each ground unit to reach the server. A 780×480 pixels picture of file size 1024 Kilobits is used as payload for image processing. These payloads are created and forwarded with 200 msec average inter-generation time at each ground unit. We used Binary Robust Independent Elementary Features (BRIEF) [32] algorithm as a feature point descriptor on these images. The network load is emulated by creating UDP flows from each ground unit to the server as shown in Fig. 7. We stress the network by gradually increasing the UDP data rate on both flows from 0 to 2Mbps. For comparison, we run other task allocation methods on the central controller:

- *Nearest first*, where the computation is allocated to the nearest neighbor node initially. Based on the load conditions on the nearest neighbor, the task may be allocated to the next-nearest neighbor, and so on.
- *Local-only*, where the computation task is sent only to the UAS, with which the ground unit has an active link.
- *App-Server only*, where the all computation tasks should be done at the server.

From Fig. 8b, we infer that the performance of FOCUS is better than the others under low to medium network load conditions, at around 150ms. It slowly begins to approach the computational response time of *Nearest first* as the network load increases to 2Mbps. This happens because with increasing network load, it becomes more beneficial for FOCUS to allocate computational tasks on the nodes nearest to the ground station (UAS 1 and 2) to mitigate the negative effects of forwarding delay on highly saturated links (e.g., the path from UAS 3 to 4). The *Local-only* approach performs worse because the task allocation is done to only those nodes that are within 1 hop from the ground station. This approach is quite immune to the increase in network load. However, the computation response time is higher than FOCUS because the task is not allocated to the optimal UAS, based on the global network knowledge. Doing the computing task on the server in the *App-Server only* approach is not scalable, since the computation response time increases exponentially with the network load.

The maximum number of computational tasks that are handled per minute in the network under increasing network load is shown in Fig. 8c. FOCUS and *Nearest first* provide higher capacity running the most number of computations per minute. *Local-only* and *App-server only* approaches result in much less capacity in terms of computations per minute, since they are localized to certain specific nodes in the network.

From these experimental results, we see that FOCUS incurs the minimum computation response time while having the capability to run the highest number of computations per minute, when compared to other classical methods.

B. Simulation Results

Next, we evaluate the performance of FOCUS through a numerical simulation to study large scale scenarios. We consider a grid topology in which the UASs are placed at equal distances and are connected in a 'cross formation', where each UAS can have at most four neighbors. We then place the GCS at one corner of the grid using the model described in Section IV. We define P_c and P_n as the probability that each UAS will receive $\Omega_i^{(c)}$ and $\Omega_i^{(n)}$ from the ground units, respectively. There are 40 UASs and we fix the value for the sets $\Upsilon^{(c)}$ and $\Upsilon^{(n)}$. Unless specified otherwise, we use these values for the model parameters: $\mu_{N+1}^{(c)} = 100$, $\mu_i^{(c)} = 5$, $\mu_i^{(n)} = 125$, $\Omega_i^{(c)} = 2$, $\Omega_i^{(n)} = 3$ (in Mbps) and, $P_n = 0.75$, $P_c = 0.8$

Fig. 9a shows $\mathcal{W}^{(c)}$ value generally increases with the only-network traffic. Here, we also show the *Distance-based* method that corresponds to a greedy algorithm in which a node sends its computation requests to the GCS only if its distance to the latter is less then half of the network graph diameter. Else, it shares the requests among its neighbors. The *Local-only* method is not affected by the network traffic; the *App-Server only* method works well with low traffic load

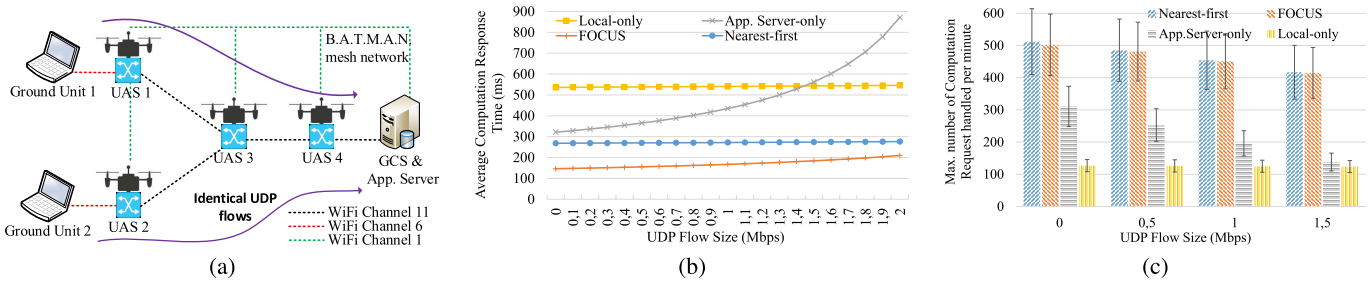

Fig. 8. (a) Testbed with 4 UASs, (b) Average computation time w.r.t. network load, (c) Max. computation capacity of the network.

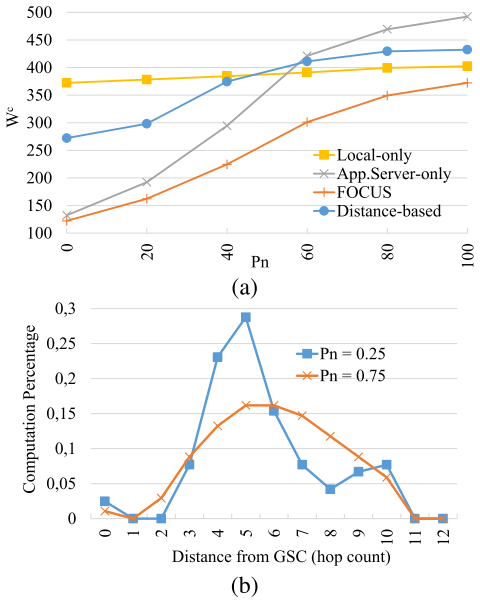

Fig. 9. (a) $\mathcal{W}^{(c)}$ varying network-only traffic P_n, (b) Percentage of computation executed w.r.t. the distance from the *GCS*.

but it is the worst when the traffic load become high. The *Distance-based* method combines both cloud computation and fog computation but as soon as the network become congested close to the GCS, the performance drops. Finally, FOCUS is able to cope with different traffic loads, striking a balance between fog and cloud computation.

In Fig. 9b, we show how FOCUS distributes the computation requests along the UAN. Here we plot two values for P_n: 0.25 and 0.75. We see when network traffic is high, the cloud (point 0 in x-axis) is not preferred. However, with lower P_n some computation occurs in the cloud. With low network load, the computation is largely contained in the middle section of the UAN (as the GCS is at one corner of the grid). At the same time, we have few UASs that are part of the only-network path towards the GCS while some others have their network queue empty. If we increase the P_n, we see the computation is spread uniformly around the network. This is because of a more uniform distribution of the intra-network bound packets. Thus, FOCUS dynamically distributes fog computation tasks around the network based on the traffic conditions.

VII. Conclusion and Future Work

We proposed a fog computing architecture, called FOCUS, for UAS software-defined mesh networks. in the network are utilized to create and operate as fog nodes. We first showed that increasing CPU utilization of UAS has negligible effect on flight time, and characterized the trade-off between computation time/location under different network and computation loads. Then, we formulated the joint problem of network and computation flow optimization, with heuristics having well defined complexity, along with a systems-level implementation. Experiments and simulations validated the approach of allocating computational tasks in a principled manner, revealing over 32% latency improvement compared to greedy or end-server only allocation methods. Finally, we are planing to enhance our framework by including another module, which orchestrates the positioning of UASs optimizing the overall coverage for ground units while sustaining feasible mesh connectivity. Furthermore, we are also going to evaluate the performance of existing low-power radio technologies (such as NB-IoT and LoRa) and analyze the trade-off between energy consumption and the delay in control traffic traffic.

References

[1] P. Cohn, A. Green, M. Langstaff, and M. Roller, "Commercial drones are here: The future of unmanned aerial systems," McKinsey Company, New York, NY, USA, Tech. Rep., 2017.

[2] L. Gupta, R. Jain, and G. Vaszkun, "Survey of important issues in UAV communication networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1123–1152, 2nd Quart., 2016.

[3] OpenDayLight. (Apr. 2018). *Software Defined Network and Network Function Virtualization*. [Online]. Available: https://www.opendaylight.org/

[4] H. Genc, Y. Zu, T. W. Chin, M. Halpern, and V. J. Reddi, "Flying IoT: Toward low-power vision in the sky," *IEEE Micro*, vol. 37, no. 6, pp. 40–51, Nov. 2017.

[5] I. Bekmezci, O. K. Sahingoz, and Ş. Temel, "Flying ad-hoc networks (FANETs): A survey," *Ad Hoc Netw.*, vol. 11, no. 3, pp. 1254–1270, 2013.

[6] C. Rametta and G. Schembra, "Designing a softwarized network deployed on a fleet of drones for rural zone monitoring," *Future Internet*, vol. 9, no. 1, pp. 1–21, 2017.

[7] K. J. S. White, E. Denney, M. D. Knudson, A. K. Mamerides, and D. P. Pezaros, "A programmable SDN+NFV-based architecture for UAV telemetry monitoring," in *Proc. 14th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2017, pp. 522–527.

[8] B. Barritt, T. Kichkaylo, K. Mandke, A. Zalcman, and V. Lin, "Operating a UAV mesh & Internet backhaul network using temporospatial SDN," in *Proc. IEEE Aerosp. Conf.*, Mar. 2017, pp. 1–7.

[9] Z. Yuan, X. Huang, L. Sun, and J. Jin, "Software defined mobile sensor network for micro UAV swarm," in *Proc. IEEE Int. Conf. Control Robot. Eng. (ICCRE)*, Apr. 2016, pp. 1–4.

[10] Z. Zhang, H. Wang, and H. Zhao, "An SDN framework for UAV backbone network towards knowledge centric networking," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 456–461.

[11] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Fog-enabled multi-robot systems," in *Proc. 2nd IEEE Int. Conf. Robot. Comput. (IRC)*, Jan. 2018, pp. 102–105.

[12] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV-based IoT platform: A crowd surveillance use case," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 128–134, Feb. 2017.
[13] X. Wang, A. Chowdhery, and M. Chiang, "Networked drone cameras for sports streaming," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Jul. 2017, pp. 308–318.
[14] R. I. Meneguette, A. Boukerche, and A. H. M. Pimenta, "AVARAC: An availability-based resource allocation scheme for vehicular cloud," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 10, pp. 3688–3699, Oct. 2018.
[15] Y. Hui, Z. Su, T. H. Luan, and J. Cai, "Content in motion: An edge computing based relay scheme for content dissemination in urban vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 8, pp. 3115–3128, Aug. 2018.
[16] R. Florin and S. Olariu, "Toward approximating job completion time in vehicular clouds," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 8, pp. 3168–3177, Aug. 2018.
[17] S. Jeong, O. Simeone, and J. Kang, "Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 3, pp. 2049–2063, Mar. 2018.
[18] M. Narang *et al.*, "UAV-assisted edge infrastructure for challenged networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2017, pp. 60–65.
[19] A. Kattepur, H. K. Rath, and A. Simha, "A-priori estimation of computation times in fog networked robotics," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 9–16.
[20] M. A. Messous, H. Sedjelmaci, N. Houari, and S. M. Senouci, "Computation offloading game for an UAV network in mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, May 2017, pp. 1–6.
[21] H. Shakhatreh, A. Khreishah, J. Chakareski, H. B. Salameh, and I. Khalil, "On the continuous coverage problem for a swarm of uavs," in *Proc. IEEE 37th Sarnoff Symp.*, Sep. 2016, pp. 130–135.
[22] A. Trotta, M. Di Felice, F. Montori, K. R. Chowdhury, and L. Bononi, "Joint coverage, connectivity, and charging strategies for distributed UAV networks," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 883–900, Aug. 2018.
[23] (May 2018). *Stress—Simple Workload Generator for POSIX Systems*. [Online]. Available: http://people.seas.harvard.edu/ apw/stress/
[24] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, 4th ed. New York, USA: Wiley-Interscience, 2008.
[25] E. Bonfoh, S. Medjiah, and C. Chassot, "A parsimonious monitoring approach for link bandwidth estimation within SDN-based networks," in *Proc. 4th IEEE Conf. Netw. Softw. Workshops (NetSoft)*, Jun. 2018, pp. 512–516.
[26] G. Baier, E. Köhler, and M. Skutella, "The *k*-splittable flow problem," *Algorithmica*, vol. 42, no. 3, pp. 231–248, Jul. 2005.
[27] D. P. Bertsekas and D. A. Castañon, "A forward/reverse auction algorithm for asymmetric assignment problems," *Comput. Optim. Appl.*, vol. 1, no. 3, pp. 277–297, Dec. 1992.
[28] (Mar. 2018). *DroneCodeSDK—MAVLink API Library for the Dronecode Platform*. [Online]. Available: https://www.dronecode.org/sdk/
[29] (May 2018). *Docker—Docker Operating System Level Virtualization*. [Online]. Available: https://www.docker.com
[30] (Feb. 2018). *OpenvSwitch—Linux Foundation, Collaborative Projects–OpenvSwitch*. [Online]. Available: https://www.openvswitch.org/
[31] (Feb. 2018). *B.A.T.M.A.N—Better Approach To Mobile Ad-hoc Networking*. [Online]. Available: https://www.open-mesh.org/
[32] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Binary robust independent elementary features," in *Proc. Eur. Conf. Comput. Vis.*, 2010, pp. 778–792.

Gokhan Secinti (S'13–M'18) is currently a Post-Doctoral Research Associate with Northeastern University. His current research interests include aerial networks and software-defined networking. He was a recipient of the IEEE INFOCOM Best Poster Paper Award in 2015 and the IEEE CAMAD Best Paper Award in 2016. He serves as a Reviewer of the *IEEE Communications Magazine*, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS IN WIRELESS COMMUNICATIONS, and *The International Journal of Communication Systems*.

Angelo Trotta received the Ph.D. degree in computer science and engineering from the University of Bologna, Bologna, Italy, in 2017. He was a Visiting Researcher with the Heudiasyc Laboratory, Sorbonne Universities, UTC, Compigne, France, and with the GENESYS-Laboratory, Northeastern University, Boston, MA, USA. He is currently a Post-Doctoral Research Fellow with the Department of Computer Science and Engineering, University of Bologna. His current research interest includes nature inspired algorithms for self-organizing multirobots wireless systems.

Subhramoy Mohanti is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Northeastern University. His current research areas include UAV networking and communication, wireless protocols and networks, and scheduling and optimization techniques. He was a recipient of the IEEE INFOCOM Best Paper Award in 2018 and the Northeastern University Graduate Dissertation Research Grant in 2015.

Marco Di Felice received the Laurea *(summa cum laude)* and Ph.D. degrees in computer science from the University of Bologna, Bologna, Italy, in 2004 and 2008, respectively. He was a Visiting Researcher with the Georgia Institute of Technology, Atlanta, GA, USA, and with Northeastern University, Boston, MA, USA. He is currently an Associate Professor of computer science with the University of Bologna. His research interests include self-organizing wireless networks, unmanned aerial systems, the IoT, and mobile applications.

Kaushik R. Chowdhury (M'09–SM'15) is currently an Associate Professor with the Electrical and Computer Engineering Department, Northeastern University. He received the Presidential Early Career Award for Scientists and Engineers (PECASE) in January 2017, the DARPA Young Faculty Award in 2017, the Office of Naval Research Director of Research Early Career Award in 2016, and the NSF CAREER Award in 2015. His current research areas include networked robotics, dynamic spectrum access, RF energy harvesting sensors, and intra-body implant communication.